Charting Java

Mary Cosway and Andrea di Pietro

August 14, 2002

Special Thanks for the support given by Vincent and Jerome.

Copyright © by Haiko Schmarsow. All rights reserved. This publication or any part of it may not be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission by the owner of the copyright. The copyright and the statements in the following paragraphs extend to the downloadable data offered under the domain-names `www.chartingJava.org` and `www.chartingJava.com`.

The use of general descriptive names, registered names, trade names, trademarks, etc, in this publication, even if the former are not especially identified, does not imply that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

All names and services identified throughout this book, also those which are, or which are related, to trademarks or registered trademarks of their respective companies, are not used to convey endorsement or other affiliation with this publication.

This publication is provided "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. The publisher or owner of the copyright assumes no responsibility of any kind for errors or omissions and ensuing damages resulting from the use of the information contained in this publication.

Written with LyX, the latest version used was 1.1.6fix2, figures and images were realized using Xfig 3.2 and Gimp Version 1.2 on a Linux/KDE platform (SuSE 7.2).

Publisher, holder of copyright and contact for feedback is (June 2002) Haiko Schmarsow (`haiko@schmarsow.de`)

ISBN 3-00-009429-6

Contents

Preface 17

- This Book . . . 18
  - Structure . . . 18
  - Content . . . 18
  - Limitations . . . 20
- The Reader . . . 20
- The Authors . . . 21
- Conventions . . . 21

I. Computer-Models 23

Elementary Computer-Models 27

- A Computer-Model for Programmers . . . 28
  - Computers with Multiple Processors and Networks . . . 30
  - Summary: Computer-Models . . . 30
- A Computer-Model for Hardware Technicians . . . 30

Processor-Commands, Languages 33

- Bits and Bytes used For Coding Human Readable Symbols and Numbers . . . 33
- Commands on Processor-Level: Machine-Language and Assembler . . . 35

Languages for easier Coding 37

- Language-Processing: Interpreting, Compiling, Virtual Machine . . . 38
- Languages: High-Level-Concepts . . . 40

Operating-Systems 43

- File-Systems . . . 46

II. The Java Runtime Environment 49

Installing Java Development Kits (JDKs) 59

- Preparations: Generalities on (Downloading) and Installation . . . 59
- PATH and CLASSPATH for Unix/Linux or Windows-based Operating-Systems . . . 60
- ORGANIZE: Apple Mac OS . . . 61
  - ORGANIZE: Mac OS 9.x: Installing the Java Development Kit (JDK): . . . 61
  - ORGANIZE: Mac OS 9.x: Using the Java Development Kit (JDK) . . . 61
- ORGANIZE: BeOS . . . 62
- ORGANIZE: Linux . . . 62
  - ORGANIZE: Linux: Getting the Java Development Kit (JDK): . . . 62
  - ORGANIZE: Linux: Using the Java Development Kit (JDK): . . . 62
- ORGANIZE: Windows . . . 63
  - ORGANIZE: Windows: Installing the Java Development Kit (JDK): . . . 63

ORGANIZE: Windows: Using the Java Development Kit (JDK): 63
ORGANIZE: Development-Environments 64
Organize: Extending Java 64

How Java Code is Run **65**
Compiling Source-Code 65
Running Programs/Applications 65
Running Applets 67

III. Basic Programming I The Language and Utilities **69**
Hello World Program 71

Data-Storage — Data-Structures **73**
Concept: Literal, Type, Name and Variable 73
Data in Form of Literals — What is stored in the computer memory? 73
Type or Class of Literals — How is it stored into the computer memory? 73
Identifier, Name — Where is it stored in the computer memory? 73
Concept: Declaration, Allocation, Instantiation of a Variable 74
Concept: Statically-, Dynamically-, Runtime-Typed Programming-Languages 75
Java: Declaration, Allocation, Instantiation: Examples and Outlook 75
Java: Literals 77
Integer-Literals 77
Floating-Point-Literals 78
Boolean-Literals 79
Character-Literals 79
String-Literals 81
Java: Identifier, Name 81
Naming Conventions 83
Java: Primitive-Types 83
Logical Type 84
Numeric Type 84
Integral Types 85
Floating-Point Type 86
Java: Declaration, Instantiation of Variables with Primitive-Type 87
Java: Type-Casting I 87

Processing Data **89**
Concept: Expressions, Statements, Control Structures 89
Java: Expressions 90
Operator Expressions — Calculate, Compare and Change 92
Increment/Decrement and Prefix Operators 92
Arithmetic: Multiplicative Operators 93
Arithmetic: Additive Operators 94
Arithmetic: Roots, Exponentials, Logarithms and Powers 94
Non-Arithmetic: String Concatenation 95
Bitwise: Shift-Operators 95
Non-Arithmetic: Relational Expression 96
Un/Equality: Comparison Operators 96
Bitwise and Logical Operators (Single Character: &,^,|) 97

Logical Operators (Double Characters: &&,||) . . . 98
Conditional-Operator . . . 99
Assignment Operators . . . 99
Type-Casting II (Casting of Primitive-Types) . . . 100
Type-Casting implicit . . . 100
Type-Casting explicit . . . 100
Precedence . . . 101
Java: Statements, Control Structures and Programs . . . 103
Java: Statements . . . 103
Block . . . 105
Java: Branching . . . 105
Binary Branching — If-Statement . . . 105
Multiple Branching/Cascading — Switch-Statement . . . 106
Java: Looping, Iteration . . . 108
Post-condition Loops — While, For-Statements . . . 108
Pre-condition Loops — Do-Statement . . . 109
Restricted Jumping — Break, Continue-Statements — Pre- and Post-Condition Mix . 110
Restricted Jumping — Try-Catch-Finally . . . 112

Structuring Code: Proceduralization 115
Concept: Methods are more than Named Blocks of Code . . . 115
Java: Methods . . . 116
Java: Method-Declaration . . . 117
Java: Method-Call . . . 118
Concept: Method-Call Arguments — Transfer By-Value or By-Reference . . . 119
Java: Local Variables - Scoping and Shadowing 1 . . . 121
Scope . . . 122
Shadowing Variables with Methods . . . 123
Concept and Java: Overloading, Recursion, Callback . . . 124
Overloading Methods . . . 124
Recursion of Method-Calls . . . 125
Callback of Methods . . . 127
Concept: Triggerable-Methods Or Who calls the `main()`-Method of an Application? . . . 127
Summary and Outlook: Proceduralization . . . 128

Structuring Code and Data: Object-Orientation 131
Concept: Object/Class-Model . . . 131
Example: Object, Class and Class-Hierarchy . . . 132
Overriding Methods and Shadowing Variables . . . 135
Class-Extension — Inheritance with multiple Super-Classes . . . 136
Communication-Mechanisms . . . 137
Object/Class-Model Keywords . . . 138
Concept: Object/Class-Model within Programming-Languages . . . 140
Java: Classes as Programmer-Definable Types and Programs . . . 142
Java: Reference-Types and their Instantiation with the `new`-Keyword . . . 142
Java: Classes . . . 145
Java: Class-Declaration . . . 145
Java: Class-Instantiation/Object-Creation . . . 149
Java: Defining Constructors . . . 149
Java: Class-Instantiation with Constructors and Factory-Methods . . . 150

Java: Class- and Object Lifespan . . . 151
Java: Introducing and Accessing Class- and Instance-Members . . . 154
Java: Kinds of Classes . . . 157
Java: Anonymous Objects and -Classes . . . 159
Singleton . . . 160
Comparing Objects . . . 161
Java: Identity of Objects . . . 161
Concept: Equality of Objects . . . 162
Java: Comparing Objects (1) . . . 162
Java: Class-Extension . . . 163
The Class `java.lang.Object` . . . 164
Class-Extension: Member-Access . . . 165
Java: Class-Extension with Constructors and Finalizers . . . 166
Java: Overriding Methods and Shadowing Variables (2) . . . 167
Java: Type-Casting III (Casting of Reference-Types) . . . 169
Object to Primitive-Type and Reverse . . . 169
Type-Casting among Identifiers of Reference-Type: Widening and Narrowing . . . 169
Java: Validity of Identifiers: Scoping and Access-Control . . . 171
Un-Shadowing and Un-Overriding . . . 172
Un-Shadowing and Un-Overriding FROM INSIDE AN OBJECT . . . 172
Un-Shadowing and Un-Overriding FROM OUTSIDE AN OBJECT . . . 174
Access-Control . . . 175
Modifiers and Accessibility . . . 180
Java: Other Modifiers . . . 181
Java: Class-Extension with Abstract Classes and Interfaces . . . 183
Java: Abstract Classes and Abstract Methods . . . 183
Java: Interface-Type . . . 185
Java: Comparing (2) and Copying . . . 190
Summary: Object-Orientation and Java Classes . . . 193
Java and the Class-Concept . . . 194
Concept: Polymorphism/Monomorphism . . . 194
Concept: Encapsulation . . . 195

Structuring Code: Multithreading **197**
Concept: Concurrency and Parallel Computing . . . 197
Concept: Threads . . . 198
Concept: Scheduling of Threads (Non-/Preemptive and Priorities) . . . 200
Java: Basic Thread Resource Distribution on a Java Platform . . . 203
Java: Handling Single/Isolated Threads . . . 205
Java: Spawning a Thread . . . 206
`Thread`-Object contains `run()`-Method . . . 207
`run()`-Method in Anonymous Class . . . 208
`Thread`-Object and `run()`-Method in Different Classes . . . 209
`Thread`-Object is Member in Class of `run()`-Method . . . 209
Basics of Controlling a Single Thread . . . 210
Concept: Handling Multiple/Communicating Threads . . . 213
Concept: Atomicity and Exclusive Access (Locking) . . . 213
Concept: Waiting . . . 215
Java: Handling Multiple/Communicating Threads . . . 216
Java: Isolating Code and Data from other Threads . . . 216

Java: Waiting for other Threads . . . 220
Java: Daemon-Threads And Shutting Down of JRE . . . 223
Problems specifically arising with Thread-Communication: . . . 224
Dangers: Starvation . . . 224
Dangers: Race-Conditions . . . 225
Dangers: Deadlock . . . 226
Dangers: Lifelock . . . 226
Concept: Pragmatics in Thread-Specific Programming . . . 227
How can Threads interact elementary on a Level above Scheduling? . . . 228
A Multithreaded Application . . . 228
Concept: Multiprocessing and Multithreading . . . 230
Concept: Blocking and Non-Blocking Methods . . . 231
Summary: Multithreading . . . 232

Concept and Java: Aggregation of Data 233
Java: Arrays and Array-Type . . . 234
Java: Vectors . . . 236
Java: StringTokenizer . . . 237
Java: Collection-Framework . . . 237
Transforming Aggregates . . . 237

Structuring Code: Packaging 239
Java: How to Write and How to Install a Package . . . 240
How to install a pre-written Package . . . 243
How to use an installed Package . . . 244
Package Introspection . . . 245
Compilation, Packages and Compilation Units . . . 246
Summary: Packages . . . 246

Standard Code-Libraries: Pre-programmed Packages 249
Pre-programmed Packages . . . 249
Core-Classes and the Java Runtime System: Location/Content . . . 249
`java.lang`: Some Standard Classes . . . 252
Platform and Runtime Environment . . . 258
Internationalization - Localization . . . 259
Identifying Locality with `java.util.Locale`-Objects . . . 260
Example: `Calendar` and `TimeZone` . . . 261
Example: Resource-Bundle . . . 263
Character Codes and Fonts . . . 265
Java: Encodings and Fonts . . . 266

IV. Basic Programming II Internal Communication 267
Java: Communication: Introducing Classes across Files of Source-Code . . . 269

Communication: Messaging and Streaming 271
Concept: Basic Communication-Models . . . 271
Message-Model . . . 271
Stream-Model . . . 272
Operating-System/Runtime System: Sockets for Input and Output . . . 273
Protocol-Model . . . 274
Initial Triggering . . . 275
Practical Event-driven Communication . . . 277

Code Structuring: Messaging 279
Java: Implemented: Communication by "Throwing Objects" 279
Source: Who throws How from Where? 279
What is thrown? 280
What's the Switchboard-Mechanism and Who catches? 282
Summary: "Throwing Objects" 284
Java: To be Implemented: `Observable`-Class and `Observer`-Interface 284
Java: Event Model — A Mechanism fully to be Implemented 286
Source 287
Event 288
Drain 288
Trigger 288
Trivial Example 288
Remarks about Non-Trivial Extensions 289

Concept: Streams in Java 291

Java: Streams 295
Java: Runtime Input/Output-Streams 297
Java: Piping with Streams 300
Java: More on Java Streams 301
Java: Filtering and Formatting with Streams 303
Java: Serialization of Objects, Persistence 306
Summary: Streams 310

Code-Structure: Software-Components 311
Concept: Software-Components 311
Concept: Object-Oriented Approach to Software-Components 311
Java-Beans: Software-Components in Java 313
Standard Property 313
Indexed Property 313
Event-Communication between Software-Components 314
Bound and Constrained Properties 314
Persistence (of Objects) 318
Software-Design using Components with Reflection and Introspection 318
Summary: Software-Components 318

Persistence 321

Documenting and Archiving Code 323
Comments and Documentation 323
Java: `javadoc` 324
Java: `jar`-Tool for Archiving 325
Java: Running Java Byte-Code 326

Summary: Java Messaging 329
Thread-Communication 329
Exception Throwing 329
Observable/Observer-Mechanism 330
Framework: `java.util.EventObject`, `java.util.EventListener` 331
Java-Beans — Software-Components 331
Abstract Windowing Toolkit (AWT) and Swing 331

Summary: Java Language 333

V. Communication between Computers Networks and Distributed Computing 335

Concept: Classification of Computer-Networks 339
Maximal Number of Hosts and Network-Graph . . . 340
Geographic Extension . . . 340
Quality of Service QoS (Flow Capacity, Load and Reliability) . . . 341
Communication in a Network — Protocols . . . 341
How are the (Host) Computers identified in a Network? (Host Identification) . . . 342
Inside the Network, how to find the Path from the Source of the Data to the Drain? (Pathfinding) . . . 342
How is the Data actually sent, in what Form does the Data pass the Network? (Path Using) . . . 342

Concept: Logical Connections, Virtual Networks 343

Concept: Data-Transfer with Packets and through Channels 345

Concept: Protocols and their Layer-Model 347
Protocol and Type . . . 347
Layers of Protocols . . . 347
More Classifying Criteria for Protocols . . . 349
Client-Server, Peer-to-Peer Communication . . . 352
Communication within the Same Levels of the Layer-Model . . . 353
OSI-related Layer-Model . . . 353

Networking-Protocols 357
The Internet Protocol (IP) — a packet-routing-, connectionless Protocol . . . 357
IP-Addressing of Hosts . . . 357
IP-Addressing of Hosts: Domain Name Service (DNS) . . . 359
TCP/UDP Transport-Protocols . . . 362
TCP - The Transmission Control Protocol — connection- and packet-oriented . . . 363
UDP - The User Datagram Protocol — connectionless and packet-oriented . . . 363
Addressing in TCP/IP-networks: Sockets and URLs . . . 363

Application-Protocols (HTTP, FTP, ...), Types and Network Applications 367
Universally Recognizable Types and Formats . . . 368
HTTP (Hypertext Transfer Protocol) the Protocol of the WWW . . . 369
Hypertext . . . 371
Copying Files over the Network . . . 371
Private Mail . . . 371
Other Applications . . . 371

Triggering: URL- Network-wide Invoking of Application-Services 373
Uniform Resource Locator (URL) — An Introduction . . . 373
URL-Encoding . . . 375

Summary: Internet- and Transport-Protocols, Addresses, Domain-Names 377
ORGANIZE: Your Network-Connection . . . 378

Java: Programming in Networks 381

Java: Relating IP-Address, Host-Name and Owner-Information . . . 381
Java: UDP Sockets and Datagrams . . . 384
Java: TCP-Sockets and Network-Connections . . . 385
Java: URL, Application-Protocol- and Content-Handlers for CLIENTS . . . 388
`java.net.URL` — The Default Mechanism . . . 390
Java-Concept: Uniform Resource Locator in Client-side Connecting to a Network . . . 390
Optional: `java.net.URL` — The Potential . . . 391
Java: Applets as Programs in a Client-Side-Controllable Environment . . . 396
Applet-Concept . . . 397
Applet-Infrastructure . . . 397
A Simple Applet . . . 399
Organize: Browser with Java Runtime Environment: To Do . . . 400
Applet Threading . . . 401
Applet Introspection . . . 402
Applet Communication . . . 403
Java: Servlets / (CGI) . . . 405
Servlet-Concept . . . 406
Servlet-Infrastructure . . . 407
A Simple Servlet . . . 408
Organize: Install Server with Servlet-Container: To Do . . . 409
Organize: Run a Servlet . . . 410
Servlet Threading . . . 411
Servlet Introspection . . . 411
Servlet Communication . . . 412
HTTP-Servlet Extension . . . 413
Java: Remote Method Invocation (RMI) / (RPC) . . . 414
Organize: JRE-Extension: What is needed for a RMI-Connection? . . . 415
A RMI-Connection . . . 416
How is the Remote Method Invocation Service activated? . . . 418
What more? . . . 419
Outlook: Java and Networking . . . 419

Summary: Java Communication 421

Concept: Distributed Computing and Programming 423

VI. Java Runtime Environment and Security 425

Concept/Java: Scope, Access-Areas, Name-Spaces 429

Java: Runtime Environment 431

Class-Loading . . . 433
Class-Loading: The Initial Process . . . 434
Class-Loading: The Secondary Process . . . 435
Class-Loading: Customized Class-Loaders . . . 438
`java.lang.ClassLoader`— Simple Extension . . . 438
`java.net.URLClassLoader` — Simple Instantiation . . . 438
The Full Loading-Process of Instances derived from `java.lang.ClassLoader` . . . 440
Maximally programmer implemented Class-Loader . . . 441

Class-Loaders of Arrays? . . . 443
Simple Class-Loading: `new`-Keyword and the `Class.forName()`-Method . . . 443
Object Storing and Loading: Serialization . . . 443
RAM Recycling: Garbage-Collection and Weak References . . . 444

Security: Basic Concepts 447
Java Language- and Java Compiler-Security . . . 450
Java Runtime Security . . . 451

Security: Policy-Based 453
Java: Access-Control Mechanism . . . 453
Secure Class-Loaders and Protection-Domains . . . 454
From File to `Policy`-Object . . . 455
Security-Manager and Access-Controller . . . 455
Java User and Administrator . . . 457
Text-File: `java.security` . . . 457
Policy Text-Files . . . 458
Selecting Policies for Running Applications and Applets . . . 460
Some Practical Hints . . . 461
Java Programmer . . . 462
Java Security-Manager . . . 462
The `check...()`-Methods of the Security-Managers . . . 463
User and Administrator Use of a Security-Manager . . . 465
Why a Security-Manager if there is an Access-Controller for consulting an External Policy? . . . 465
Java Access-Controller: Policy . . . 465
Java Access-Controller: Custom Permissions . . . 467
Java Access-Controller: Protection-Domain . . . 467
Java Access-Controller: Method-Call History . . . 470
Java Access-Controller: Privileged Code . . . 471
Java Access-Controller: Policy-Secured Environment extends to Objects . . . 474
Applets and Security . . . 475
Summary: Policy-Based Access-Control Mechanisms in Java . . . 475

Security-Related Coding of Data 477
Basic Concepts of Security through Data-Processing . . . 477
Social Aspects: TRUST . . . 478
Concepts: Some Results of the Mathematical Theory . . . 478
Cryptography with a Secret Key (Symmetric Key) . . . 478
Cryptography with a Public/Private Key-Pair (Asymmetric Key) . . . 479
Hashcode - Message-Digest - Digital Fingerprint - Checksum . . . 481
Signing/Authentifying a Message Using Encrypted Message Digests . . . 482
Securely Associating an Identity with the Owner of a Key . . . 483
Variants of an Encryption Algorithm (Modes and Padding) . . . 484
More Concepts . . . 484
Java Provider-Infrastructure: Concept and Implementation . . . 485
User/Administrator: Key-Management and Signing Files . . . 488
Access-Controlling with Signed Data . . . 492
User/Administrator: How to encrypt Files . . . 492
Application-Programmer: Signing and Encrypting . . . 492
Installing the Java Cryptography Extension (JCE) . . . 496
Using the JCE . . . 496
Infrastructure-Programming: Signing and Encrypting . . . 497
Concept: Network-Security . . . 500

VII. Single User Interface 503

Concept: User Interfaces 507
Hardware User Interface . . . 507
Graphical User Interfaces (of Operating-Systems) . . . 508
Graphical User Interface of Operating-Systems: Desktop - Icons - Windows . . . 508
Graphical User Interface of Operating-Systems: Screen-Components . . . 510
OS-GUI: File-System: Opening Files and Folders . . . 510
GUI: Giving Commands: Clicking - Drag and Drop (D&D, dnd) . . . 510
GUI: Giving Commands: MenuBars - PopupMenus - Toolbars . . . 511
GUI: Windows as Containers of Screen-Components . . . 512
Graphical User Interface of Application-Software . . . 512
Design of Graphical User Interfaces (GUIs) . . . 513

Concept: Programmer's View of User-Interface (UI) 515
Program-Structure: Model View Controller (MVC) . . . 515

Screen Output 519
Concept: Computers and Images . . . 520
The Concept of Image-Formats . . . 521
Concept: Colors . . . 522
The RGB Color-Model . . . 523
The CMY Color-Model . . . 524
The CMYK Color-Model solves the Printing-Problem: C+M+Y≠Black . . . 525
The HSV and HSB Color Models . . . 525
Two Basic Image- and Color-Formats . . . 525
Direct Color-Model: ARGB . . . 526
Indirect Color-Models . . . 526
Java: Transparency, Partial Transparency and Opaqueness . . . 527
Java: `java.awt.Image` . . . 527
Java: Simple Bitmapped Images . . . 530
Concept: Graphical-Objects . . . 534
Painting Graphical-Objects . . . 536
Java: Object-Painting on Screen-Components . . . 538
Concept: Screen-Components . . . 540
Java: Basic Swing Screen-Components . . . 543
Lightweight-/Heavyweight Components — Java: AWT versus Swing . . . 548
Platform and Java Runtime Points-of-Contact . . . 551
Java: Basics of Screen-Components . . . 551
Java: `java.awt.Component` . . . 552
Java: `java.awt.Container` . . . 556
Java: `javax.swing.JComponent` . . . 559
Java: `Frame` and `JFrame` . . . 561
AWT and Swing Class-Hierarchies . . . 564
Java: `ContentPane` and `JLayeredPane` . . . 566
Java: Fonts . . . 569

Event-Communication - Mouse and Keyboard 571
Event Delegation within the Abstract Window Toolkit (AWT) . . . 571
Concept: Mouse Interaction . . . 571
Concept: Keyboard Interaction . . . 572
Java: The AWT-Event-Communication . . . 572
Java: Delegation Event-Handling . . . 574
Java: Triggering . . . 574
Java: Switchboarding/Event-Scheduling . . . 574
Java: Event Types . . . 575
Java: Sources, Event-Objects, Sinks (Listeners) . . . 576
Summary: Java Variants of Event-Listening . . . 579
Java: Adapter-Classes . . . 580
Java: Examples for Mouse- and Keyboard Event Handling . . . 581
AWT-Switchboarding and Java . . . 585
Using the Event-Dispatch-Thread . . . 586
Java Programmer: Sources of `AWTEvent`-Objects . . . 590
Java: `processEvent()`-Method as Drain and Source of `AWTEvent`-Objects . . . 592

Interaction: Screen, Keyboard and Mouse 595
Java: Changing Components and Images On-Screen . . . 595
Java: Painting-Methods . . . 595
Pixel-Based Images . . . 596
Graphics-Objects . . . 597
Screen-Components . . . 597
Summary: Drawing On-Screen . . . 599
Java: User Changes User Interface . . . 599
Java: Animation . . . 602
Concept: Time-Delayed Processing . . . 605
Java: Time-Delayed Data . . . 606
Java Overview: Interfaces and Classes for Time-Delay . . . 606
The Workings of an `ImageObserver`-Object . . . 607
`java.awt.image.ImageProducer` . . . 610
`java.awt.image.ImageConsumer` . . . 610
Java Swing Desktop . . . 610

Java: Applets - Applet Runtime-Environment 613
Triggering And Configuring an Applet-Run . . . 614
Class-Hierarchy and Class-Structure of Applets . . . 615
A Simple AppletViewer . . . 619

Summary: Graphical User Interfaces (GUIs) 621

Outlook: Single User Interface 623

VIII Appendix 625

Reflection, Introspection and Profiling 629
Classes . . . 629
Objects . . . 630
System . . . 631
Applets . . . 631
Servlets . . . 631

How to read Programs? 633

How to write and check Programs? 637

If Something goes Wrong . . . 637

How to make Programs faster? 639

Further Questions 641

List: Prototypes of Programs 645

Primitive-Types and Control Structures . . . 645
Code Structuring: Proceduralization-Methods . . . 645
Code- and Data-Structuring: Object-Orientation: Class-Object . . . 646
Code Structuring: Object-Orientation: Class-Extension . . . 646
Code Structuring: Multithreading . . . 646
Code Structuring: Aggregation: Arrays and Container-Classes . . . 647
Code Structuring: Packaging and Libraries . . . 647
Communication: Messaging . . . 647
Communication: Streams and Pipes, Serialization (package `java.io`) . . . 647
Communication: Component/Bean-Based Programming, Documentation . . . 648
Communication: Computer-to-Computer: IP (Internet Protocol)-Networks (package `java.net`) . . . 648
Java Runtime Environment, Class-Loading and Security: Class-Loading . . . 649
Java Runtime Environment, Class-Loading and Security: Policy-Based Security . . . 649
Java Runtime Environment, Class-Loading and Security: Security by Data-Processing649
Single User Interface: Graphical User Interface (GUI) (package `java.awt`, `javax.swing`) . . . 650
Appendix . . . 651

List: Mentioned Packages, Classes and Interfaces 653

List: Syntax-Diagrams 655

Backus-Naur Form (BNF) . . . 655
Syntax- or Railroad-Diagrams . . . 655
List . . . 656

Little Classification of some Notions 657

Sources 659

Sources On Java . . . 659
Sources on the Internet . . . 659
Sources on Programming . . . 659
Literature . . . 660

Preface

First are presented some questions and some attempts to answer them:

What is a computer? A machine that can be ordered to process data. The machine gets data and commands from keyboard, mouse, storage-disks, network-connections (like the Internet). The machine stores the processed data (a fax sent, an email received) on storage-disks and can show it on output devices like screen or printer. For more see the text beginning with page 28.

What is a program? A program is given by a sequence of commands which tell the computer what to do. Programmers store programs by writing those sequences of commands into text-files, as if writing a letter or an email. First examples of a program are given on page 37 and on page 71.

What is an operating-system? A program that organizes technical-related tasks of hardware-management (allocating storage-space, accessing hard-drives (also called hard-disks), disk-drives establishing fully-functional network-connections ...). For more see page 43.

What is a platform? The computer hardware together with the working software of an operating-system is called a computer-platform (this is considered more extensively in the context of operating-systems).

What is Java? The term Java may be heard to be describing the entire **virtual platform**[1] given by the Java Runtime Environment (JRE, past page 51). Or the name Java may be used for the entire Java Development Kit, a **software-kit** that enables a programmer to write programs in the Java programming-language. But the term Java may also denote the Java **programming-language**. (For more information on programming-languages see pages around page 37.)

There are several overlapping, sometimes also rather fragmented, approaches to a multi-faceted piece of knowledge like the Java programming-language:

Handbook view: Notions are presented as far as possible in their strict logical dependence, assuming that the reader furnishes some familiarity with the concepts. The topics are usually covered extensively in an analytic, often rigid way. The ➡▤ ➡▤ Java Language Specification [JLS2] may be an example of a publication with such a view.

Reference view: References collect notions, more or less commented, in a dictionary-like presentation, sometimes in the form of an index or a glossary. Examples are the ➡▤ ➡▤ Java Platform API Specification [JPAPIS] or the Java Developers Almanac [JDA].

Practical/Textbook view: Learning with examples that introduce and illustrate concepts. Synthetic experiences and experimenting becomes possible as part of homework problems. The Online Tutorials on *Sun*'s Java-related Internet-site (`http://java.sun.com`) can be seen as examples for this kind of approach.

Programmed Learning (of practical skills): Learning in a textbook way by predominantly solving tasks under written guidance. Some certification books may represent examples for this approach. Search for "Java certification" in the Internet.

Theory-Oriented This means predominantly learning concepts of programming. In this context practical applications would distract from the central idea. Example: Algorithms (sorting, cryptography, routing packages of bytes through networks) can be formulated in many different programming-languages, their ideas remains the same. Many books with the notion "Algorithms" in their title may fit into this category.

This classification could make a reader aware of the choices and possibilities when searching for information. This present text could be classified as a textbook, although within the sections titled "Concept: ... " this text attempts a theory-related introduction to the notions.

[1]A computer-platform runs a program, called the Java Runtime System (JRS) that makes the computer mimic another platform: the so-called Java Runtime Environment (**JRE**).

This Book

This book tries to present a reasonable broad approach to computers, networks and Java programming (Java 2 Standard Edition, Version 1.4. The Java 2 Micro Edition (J2ME) and the Java 2 Enterprise Edition (J2EE) haven't been covered here.). As a result of some reading of this book, a beginner should be able to categorize notions of that field and be ready for learning advanced programming techniques. The text works as a collection of concepts and their transformation into the Java language. Marginal or advanced topics have been deliberately omitted. The Java programs presented are prototypes, maybe inviting to be changed and to be extended by the reader. All in all, the text should give a fast practical access to the Java language and its basic concepts. This book represents no atlas, rather a collection of charts.

Structure

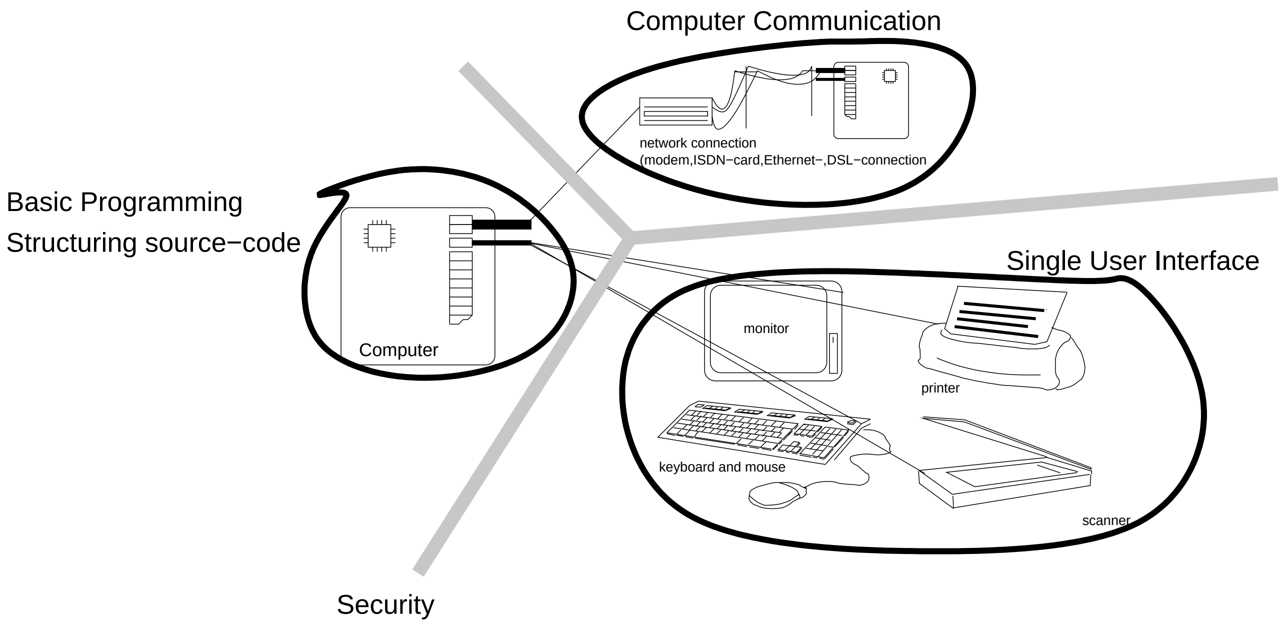
The text has been structured into *conceptual sections and Java-specific sections*. Conceptual sections describe general notions and give an overall idea of the content. The knowledge of these sections may be also applicable in non-Java contexts. The *Java-specific sections* demonstrate the realization of those concepts in the case of the Java programming-environment. — Headers of the parts and sections of the book denote the main view-point of these passages. Nevertheless *some* extensions of those main ideas can be found in *other* parts of the book. But these extensions are bound to their respective main passage by cross-references, so that they should be easily relatable. — Many programs have their descriptions added to the program's (source-)code as comments. This may better integrate the programs into the text and may make it straightforward to identify program-specific information. — Concepts and principles are introduced predominantly without referring to any special programming-language. Those paragraphs should furnish the theory that makes much of the ➡▤ ➡▤ Java Platform API Specification self-explaining. (See the concept of color on page 522 which gives some of the basic information used in the Java-class `java.awt.Color`) — The text presents simple procedural programs first; the structuring of code with methods, classes and threads comes later. — To keep the ideas above in a good order, some notions of the Java language are anticipated at times (especially the construct of a `main()`-method inside a class-declaration, being indispensable when writing regular programs.). — All the examples can be refined and extended and to get practice in Java; the reader may benefit from doing some creative work with them. — The text allows the reader several lines of access to spot specific information:

- Table of Content
- Index
- Chapter and Section Introductions/Summaries should make a small text-booklet of an overview
- Images as a picture book of ideas
- Most of the gray frames are part of a glossary. Related details can be found in the context of these gray frames.
- List of classes or interfaces introduced (in the Appendix)
- List of Programs (in the Appendix)
- List of Syntax-diagrams (in the Appendix)

Another approach may be taken by browsing through the text, then checking and maybe even extending the programs, thereby referencing the text if necessary. A text on the Java Application Programming Interface (API) available for parallel reading may give details which have been deliberately omitted in this text. Some of those details also may have been changed at the time this text has been published.

Content

The central parts of that book are basic programming, structuring source-code, communication between computers, security and single user interfaces:



The first chapters describe the building elements of the Java language-expressions and statements. These building elements do not differ very much from the basic structures in languages like C/C++, Pascal or COBOL. — Then chapters titled Proceduralization, Object-Orientation, Multithreading, Packaging and to a certain extent the chapter about communication (messaging, streaming, component-programming) consider the structuring of code.

A computer-model is introduced to envision the hardware- and software-context in which Java is running. Basic programming introduces to basic programming notions (variables and their creation, expressions like assignment) and their realization in Java. Control structures essentially constitute the acting source-code (if-then- and loop-statements), the rest of the items describe methods to structure and organize the code:

Proceduralization means sorting out a relatively independent sub-task and giving that sub-task a name of its own. This introduces the notion of a Java-method (in Pascal or C/C++ a function or a procedure). The idea is to **give reusable chunks of code a name**, and to call that piece of code again as needed (page 115).
Object-Orientation (page 131) develops the idea of code-reuse in different directions: One aspect means furnishing **data together with the methods to manipulate the data**; this also means extending the notion of a type of a variable. Another aspect means **designing code for extensibility as another way to organize code for reuse**. A third aspect means organizing or **indexing code in a hierarchy** for easier reference. If larger programming tasks have to be solved in a collaborative approach by several persons, then object-oriented structuring of code may prove to be an advantageous approach.
Multithreading (page 197) means running **independent sub-tasks parallel** on multiple processors; and letting them communicate with each other: This means, for example, that some threads wait for information another thread generates.
Packaging (page 239) describes collecting code in a software-library, this just means collecting several meaningfully-related entities of code (Java-classes) and giving them a common surname for easier reference.

Leaving the elements of the core language-functionalities opens a field of selected issues: The part about Communication (page 271) includes a description of how to realize a delegation event-model consisting of source, message and drain. Throwing objects, especially so-called exceptions or errors, happens within a Java-internal communication-infrastructure. The concept of a stream may describe a one-way buffer of incoming- or outgoing data. Wrapping streams includes the idea of “on-the-fly” processing of stream-data (after reading in-, before writing out the data).
Basics of the CPU-to-CPU communication (computer-networks, page 337) stress TCP/IP-communication. Java-specifics like Remote Method Invocation (RMI) are introduced.

Part of the text also considers how to turn a file on the hard-drive, maybe downloaded from a network, into a program ready to be run. This part of activation of program-code is also called “class-loading” (page 431). The area of computer-related security (page 447) is considered by trying to answer questions like the following: How to control a user’s access or a fellow program’s access to hardware and software on a computer-platform, especially in the Java Runtime Environment (JRE), the Java virtual platform.
Finally, single user interfacing (page 507) introduces concepts and examples of how monitor, keyboard,

mouse interact with Java and the end-user. There, the Model-View-Controller (MVC)-concept describes another way of structuring Java source-code. The Appendix may be seen as collecting the remainder.

Limitations

This book tries to get the concepts clear and give a basic introduction that helps the reader to get own programming experiences. The book does not provide exhaustive information on the Java language like any handbook or reference book. For in-depth systematics see the ➡▤ ➡▤ Java Language Specification [JLS2] or other Java Language References. Thus, ramifications of the Java language and Application Programming Interfaces (APIs) are left to be found in the ➡▤ ➡▤ Java Platform API Specification [JPAPIS] and inside handbooks on this subject as mentioned in the Appendix. This especially concerns information which Java version (1.0.1, 1.1, 1.2, 1.3, 1.4) offers which options.

This text presents no in-depth description of standard software-libraries of Java (Application Programming Interfaces (APIs) of Java). For exhaustive information see the books from Addison-Wesley or O'Reilly. For an index-like compilation of the software-libraries see the online-documentations from *Sun Microsystems Inc.* or the Java Developer's Almanac [JDA].

The book is based on the **second edition of the ➡▤ ➡▤ Java Language Specification** [JLS2] and covers the **Java 2 Standard Edition (J2SE)** (not the Java 2 Micro Edition or the Java 2 Enterprise Edition). Only the Java 2 Standard Edition (J2SE) has been considered in this book, aspects of the Java 2 Micro Edition (J2ME) and the Java 2 Enterprise Edition (J2EE) have been omitted. The implementation and some aspects of the Java design may be subject to criticism; this book has been written in the belief that the drawbacks are well outweighed by the advantages when comparing Java to other existent programming-environments.

Algorithms (roughly identifiable as the language-independent version of a program), such as sorting algorithms or cryptographic algorithms, are not described in this text. See the Appendix for hints on sources of algorithms.

The writing of source-code as presented here, does not show how to organize the writing of a **large software-project**. This field is usually treated in books about Object Oriented Design of programming tasks. No extended discussion of component-software is to be found; neither are mentioned other means of structuring code independently of a programming-language, for example the UML (Unified Modeling Language).

Documents, **database**s, their formats, production and handling, and **image-manipulation** are not covered in this text. This means NO DOCUMENT PROCESSING, NO DATABASES, NO IMAGE-PROCESSING. This text provides no information on how to program text-editors and related tasks (character encodings surpassing the ASCII[2]-format, the Unicode-format or a Unicode Transformation Format (UTF) for Unicode-characters, font-management, document-types like the Portable Document Format (PDF), Extended Markup Language (XML), drag and drop implementation). — Relational- or object-databases are not mentioned, neither has been introduced the Java DataBase Connection (JDBC), a Java software-library for accessing data in relational databases with the Structured Query Language (SQL).

References to hard- and software are limited to those products that are likely to be used in a more or less average home-environment: Mouse, keyboard, monitor. The text assumes that the reader has comfortable access to a computer, an installed operating-system, at least a dial-up Internet-connection and maybe two computers connected with each other over, for example, an Ethernet connection. The additional Java software should be downloadable for free from the Internet. Differences between platforms (Unix-type like Linux, WindowsNT/95/98, Mac OS) are not treated exhaustively.

Many of the introduced programs have been tested on 233MHz G3-Mac and Intel-type machines. Generally the programs are written reduced to the essentialities, so that they should present the gist of the functionality they intend to demonstrate.

The Reader

A reader might have the following interests or goals: Know what the Java language means and to be introduced to some of its central capabilities. Wants to get an overview and a short introduction. Wants to check the prototype programs and use them as a basis for own programming efforts. Know how to install a working, cost-free Java Runtime System (JRS) and development-environment on Apple, Linux and Windows platforms. Use another book for learning programming algorithms and other advanced programming-techniques (for example text- and image-processing or database-programming).

[2]American Standard Code for Information Interchange

The Authors

Almost every view on a subject cannot be complete or ignore the descriptor's point of view. The non-completeness aspect clearly can be seen by the limited size of the book. The authors point-of-view clearly sets its limits and preferences. The goal was to establish a foundation from which the wealth of the Java programming system can be explored without getting lost in too many details.

Conventions

The names of subjects are chosen to avoid unnecessarily specialized jargon. Nevertheless in some contexts the usage of notions may vary from that chosen in this book. The index and the explanations in the gray rectangles try to take account of that. The symbol ➡▤ means, there exists an online-source for further information and ➡▤ ➡▤ indicates that this online-source can be considered as highly recommendable for further reading.

Source-code of the (Java) programming language and the content of text-files are presented in `typewriter-font`.

Parts of the text, that represent commands or programs (in `typewriter-font`), may contain passages written in *`italics`* or passages parenthesized by "smaller than" (<) and "larger than" (>) symbols. This indicates that these passages are placeholders for values and have to take a concrete value to give a valid expression. Often these placeholder-passages indicate what values they expect. For an example see the graphic past the program-code on page 71. And see the code fragments at the beginning of the section on the switch-statement on page 107. In the part of the text about networking and HTML-formats, the "smaller than" (<) and "larger than" (>) symbols are also used as so-called HTML-tags (page 370). In Bachus-Naur expressions, which should be clearly discernable from HTML-text or terminal-commands or programs, the "smaller than" (<) and "larger than" (>) symbols indicate that the enclosed entity occurs one or multiple times (page 655).

Arrows ($\rightarrow$), within the text of the gray boxes, denote notions which have been referenced by the index with a capital letter. In other context, arrows within the text can be substituted by the words "is super-class of" (see page 615, following the graphic).

Part I.

Computer-Models

Computer-Models — Wanting to know all about a computer may resemble wanting to know all about a car. Most people want to use the car for driving, not as an object of study to become a car-mechanic. So, the average user of a car has limited skills for repairing or servicing the car. To the average user the car represents just an object that helps to accomplish personal purposes. Therefore, every motorist has a well-developed set of ideas about a car's behavior, maybe not as refined as that of a test-driver, but enough to serve the daily needs well.

The relation between a user of a computer-system and the computer exhibits aspects of the same kind. The user of a computer-system confines her- or himself to those few skills which are necessary to get the computer-system's support in accomplishing the daily real-world tasks. The notion of a computer-model intends to give this collection of skills and ideas a consistent environment.

The following part of the text considers simpler computer-models, computer-languages, operating-systems and file-systems. Extensive information about end-user interfaces of various operating-systems can be found in the last part of this text beginning with page 507.

Elementary Computer-Models

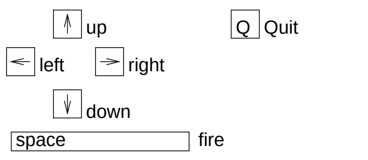
Elementary Computer-Models — First, this table shows different ways to use a computer:

recreationalist	uses a computer-game	with joystick, CD-ROM, monitor
writer	uses a text-formatter	with keyboard, monitor, black and white printer
layouter in press	uses graphics-software	with scanner, printers
application-programmer	uses interpreter, compiler, debugger, component-construction software to program: graphics-software, text-formatters or computer games for an operating-system environment	with keyboard and monitor, modems
utility-programmer infrastructure-programmer system-programmer	programs drivers, assembler, interpreter, compiler, debugger, low-level network-software, operating-systems for a specific hard- or software	with keyboard and monitor, modems
person, task	uses software	with additional hardware

Different users have different ideas how they access their computers and receive services from their computers, they have different ideas about their computer's physical components and software functionalities. End-users, who want to be helped by the computer in solving some of their real-life-tasks cannot be required to have the knowledge of a programmer. To be able to discern pure technicalities from skills for effectively solving real-life-problems with the computer, often requires a considerable knowledge in advance.

Depending on the software or hardware the user faces, each user assigns the computer-system a different set of functionalities. Such a set of functionalities can be described coherently by constructing an appropriate computing-model. In that sense the model can be used to "explain" the specific computing system (to the user).

A part of a game player´s computer model

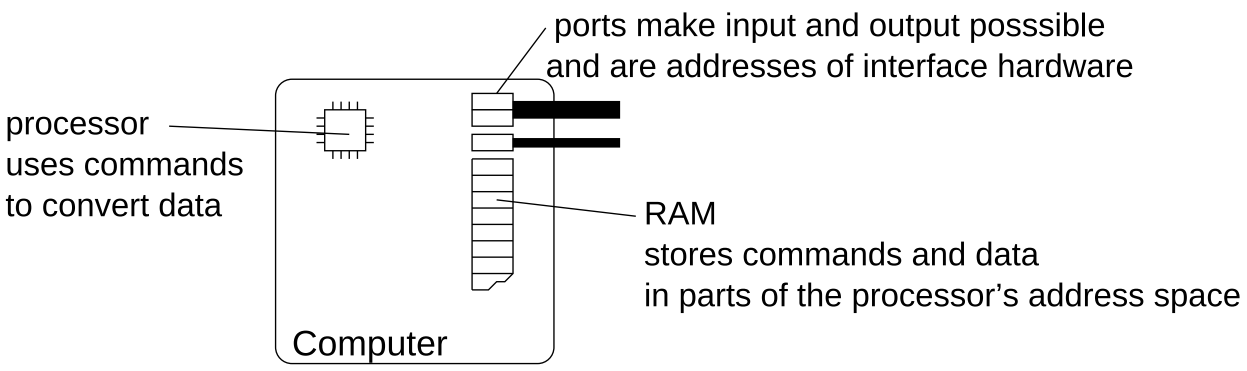


Some computing models are simple, like a computing model for a video game. Although the computing-model for the video game tells what can be done, it doesn't reflect complexity and richness of the video display and it does not tell how to acquire the skill to master the game.

The more functionalities the software furnishes, the longer the list of commands and the more difficult it may be to remember them. Even well-designed models have to reconcile contradicting perspectives: Simplicity of design makes learning and standard work fast, complexity of the models slows the learning-process but adds flexibility, opening ways to more efficient work.

A Computer-Model for Programmers

A Computer-Model for Programmers — For programmers an introductory model of a computer can be made fairly simple:


ports make input and output posssible
and are addresses of interface hardware
processor
uses commands
to convert data
RAM
stores commands and data
in parts of the processor's address space
Computer

The processor, its address-space, within that space the Random Access Memory (RAM) and the ports are given here as central notions of a computer-model. Input-data appears in the Random Access Memory (RAM) of the computer via ports, that occupy address-space which the RAM does not use. Output-data leaves the computer's Random Access Memory (RAM) by transferring the data to addresses occupied by output-ports. The Random Access Memory (RAM) stores the data. Part of the data gets recognized by the processor as sequences of processor-commands. Thus the processor can convert, manipulate or process data autonomously. A processor[3] manipulates, changes data which has been stored in the Random Access Memory (RAM).

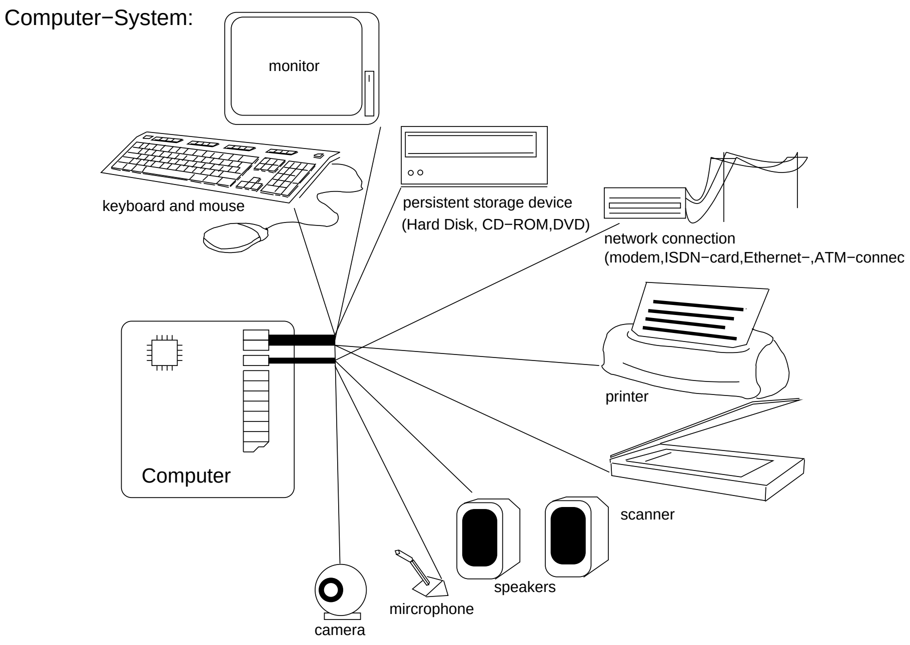
Processor: The piece of hardware inside a computer that manipulates the data it fetches from the Random Access Memory or hardware-ports. After the processor has fetched and manipulated the data, the new data is put back into the Random Access Memory or written into a port that represents a connection to a hard-disk or a computer-network. Multiprocessor-computers have multiple processors built into them; their function has to be coordinated by appropriately designed hard- and software.
CPU: Acronym for Central Processing Unit. 1. The processor(s) of a computer; the part of a computer-hardware that does the main data-manipulation. 2. A Central Processing Unit consists of the microprocessor(s), the Random Access Memory (RAM) and busses that connect both with Input/Output interface-hardware. 3. An entire personal computer, excluding keyboard and monitor.

Again: The acronym CPU stands sometimes for the processor alone, sometimes for the processor and the RAM and more seldom even for an entire personal computer (which includes an internal hard-drive but without monitor and keyboard).

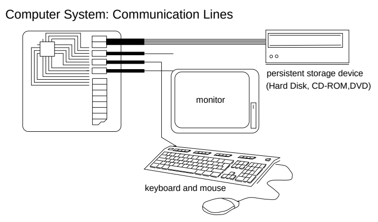
The command-sequences, that make the processor work on the data, are stored together with the data-to-be-manipulated in the Random Access Memory (RAM).

RAM: Acronym for Random Access Memory. Often simply denoted "memory". Stores data, which includes coded information and sequences of processor-commands. Generally, the processor can access each unit of information of the Random Access Memory directly. Access to data usually happens fast (in comparison with devices like hard-disks and network-connections). But data residing in Random Access Memory usually is stored non-persistently, that means the data is lost if not saved to a hard-disk before turning off the computer's power.

[3]Most processors in personal computers are, technically speaking, special integrated circuits – so-called chips – that have been given a set of very general functions. An integrated circuit is made of a collection of connected transistors and other electronic elements built into a chip primarily made of crystalline silicon. ("silicon*e*" denotes a rubbery compound of silicon)



A **computer-system** is made of the computer itself and additional devices like low data-rate input/output (I/O) devices, for example keyboard and mouse, and higher data-rate I/O devices like monitor, storage devices (hard-drive, CD-ROM), printer, scanner, microphone, loudspeakers, camera, modem or network-card (usually of the Ethernet-technology). The maximal data-rate of the communication-lines between components of a computer and the maximal data-rate of the components itself determine the performance of parts or the overall computer-system. (Analogously to a network of several computer-systems which draws its overall performance from the data-rate of the data-lines and the rate with which the individual computers can process the data.) That's much like a system of pumps and pipes for transporting water: The pipes' capacities have to measure up to the pump's power and vice versa. computer-system


Computer System: Communication Lines
persistent storage device
(Hard Disk, CD-ROM,DVD)
monitor
keyboard and mouse

A communication-line, given by a bundle of wires (more than two usually around fifty or hundreds), is called "**bus**". Those lines are usually more efficient than the single lines. Busses generally are used between hardware-components of a computer-system that have to maintain a high data-rate. Communication lines of different technologies are connected either by a "**bridge**" or by an "**interface**" (**card**); that's an electrical circuit for establishing the connection between otherwise unreconcilable types of communication-lines. If the user is not concerned with modifying the hardware of the computer, then there won't occur many occasions for handling hardware-interfaces.

Above have been described some conceptual similarities of different computer-systems. But different types of computer-systems differ significantly in technical details. For example, different makes of processors take different sets of commands.

Especially the technical development, which furnishes faster and hopefully easier-to-use hardware, makes older technical architectures obsolete. Consider some averaged performance-data of computer-systems:

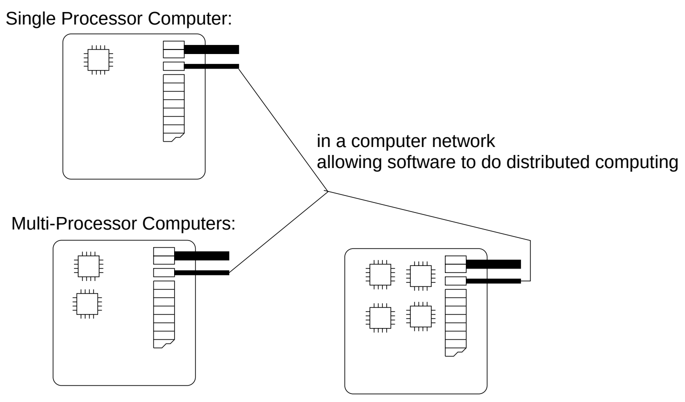
year	processor-speed (roughly in commands per second):	Random Access Memory size:
1990	20MHz	4MB
1998	200MHz	40MB
2002	2000MHz	400MB

Next to the computer's internal bus that connects the processor and the Random Access Memory, there are communication-"lines" *within* the computer: The PCI (Programmable Communication Interface) with up to 64 parallel data-lines yielding a data-flow of up to 132MB per second. The EIDE (Enhanced Integrated Device Equipment) bus with up to 33MB per second. Also, predominantly for adding external hardware to a given computer, are mentioned the SCSI (Small Computer Systems Interface) bus with up to 40MB per second for connecting up to seven external devices to the Central Processing Unit. The IEEE1394-line (also called FireWire or iLink) with up to 50MB per second and the USB (Universal Serial Bus) with up to 15MB per second as well as the Ethernet hardware with up to 100MB per second. The technological data is changing and may have changed when this has been published.

The acronym MHz (Mega-Hertz) indicates the processing-speed and the acronym MB (MegaByte) indicates the volume of the data. For what purposes may be possibly needed such information? When buying a computer, various computer-systems can be roughly compared. Then, using a computer, performance limits can be assessed, thus making it possible to quantify the volume of the Random Access Memory (RAM) or to estimate the dimension of a needed storage-device. For example, error-messages like “out of memory” may get a clearer meaning and may actually result in adding RAM to the computer. For more on these units see the subsections past the next for “MHz”, and for more on the unit of the data-“volume” (MB) see the page 34.

Computers with Multiple Processors and Networks

Computers with Multiple Processors and Networks — Multiprocessor-computers consist of several processors, connected with each other by the fast computer-internal bus-system. These multiple processors (usually 2, 4, 8, or 16) use more or less the same Random Access Memory (RAM) and usually can access the same hardware-ports. Non-expensive consumer computers usually are equipped with only one processor. But those one-processor-computers can emulate a multiprocessor-environment to allow multi-threaded programming too (see page 197 for multi-threaded programming in Java).


Single Processor Computer:
in a computer network
allowing software to do distributed computing
Multi-Processor Computers:

Networks connect entire computers mostly by external wiring, for example with versions of Ethernet-hardware, or even by wireless radio-connections. These external connections among computers usually do transfer data slower than the internal bus-systems of the individual computer. The difference between a network and a multiprocessor-computer also can be formulated as follows: Each computer in a network has its own Random Access Memory (RAM). In a multiprocessor computer the processor have to share the Random Access Memory. For more about networks see the part of this text, beginning on page 337.

Summary: Computer-Models

Summary: Computer-Models — There are imaginable different computer-models; one of those models, that gives many details independently of the technology, has been presented more extensively: The processor takes commands and data and returns transformed data. The Random Access Memory stores data, part of that data are sequences of processor-commands. Communication-lines like busses transfer the data between components of the computer-system. Why, next to Random Access Memory (RAM), are needed other storage-devices such as hard-drives (magnetic disks) or CD-ROMs? RAM works fast, is expensive and looses the data (and programs) in the moment the computer-system’s power is turned off. Compared to RAM, hard-drives are inexpensive and hold the data permanently (persistently) until deleted by the user. For storage-purposes their slowness doesn’t matter. CD-ROMs are what the name indicates, Read Only Memories on compact-disks of about 600MB. They are easy to transport and their data cannot be changed, which can be seen as a security advantage. The same applies to DVD-RAMs and DVD-ROMs, but they can store about 5000MB, about ten times the data-volume of a CD-ROM.

How to start a computer? The model described above lacks any related description. In the technical context, this process requires additional features to be included into the model. An end-user may answer that question saying: “By pressing the power button.” This turns out to be a workable and rather sensible answer on the level of the end-user’s model and working-requirements.

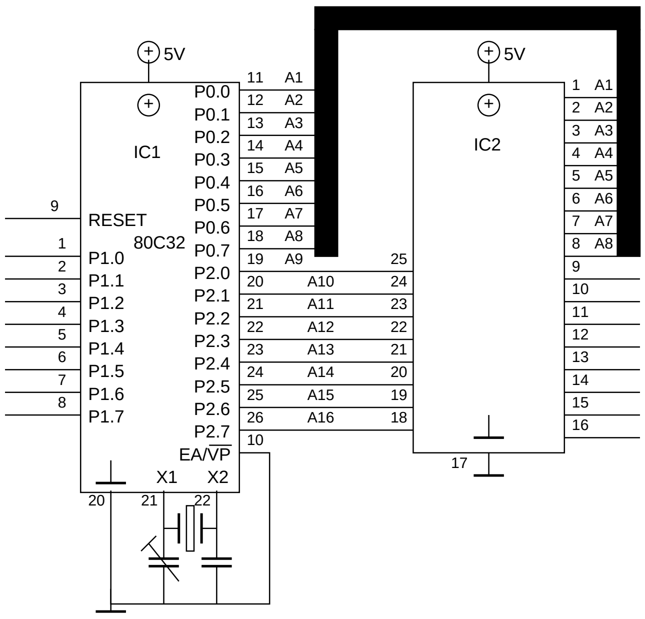
A Computer-Model for Hardware Technicians

A Computer-Model for Hardware Technicians — Even hardware technicians use computer-models. Their models are much more refined towards the technical aspect of computing.

The graphic below is meant only for getting a feeling for the technicalities involved. The large rectangles represent individual silicon-chips. Each line represents a single electrical connection. The symbol that looks like a little sandwiched rectangle, in the lower left part of the graphic, near the number 22, represents the oscillating crystal, which gives the frequency (also called clock-rate) according to which the computer works. The number of times an event reoccurs per time-interval is called frequency. The crystal inside the computers wobbles millions of times per second. This frequency of oscillation (millions of times per second) of such a crystal is called one Mega-Hertz. An average car-engine turns about hundreds of times per second (thousands of rotations per minute), which is slower about a factor ten-thousand than the frequency of a computer's oscillating crystal.

frequency

Mega-Hertz



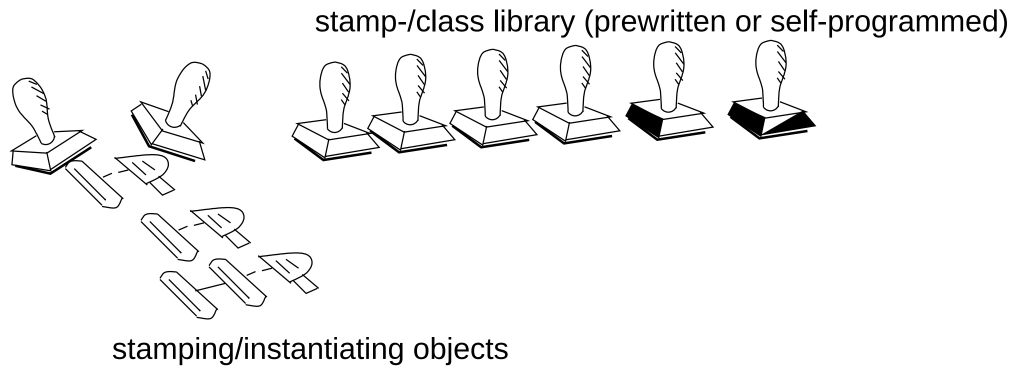
Structuring Code and Data: Object-Orientation

Structuring Code and Data: Object-Orientation — Methods and, if those methods properly encapsulate their functionality, the associated Application Programming Interfaces (APIs) are useful for structuring source-code. If properly designed, even large pieces of source-code, which have been structured with the means of proceduralization, do work and can be maintained. Nevertheless there are many (real-life) programming problems that are open to another way of structuring: Object-Oriented Structuring of source-code.

Historically the object/class-model emerged from trying to simulate physical real-life objects and their interactions[45]. There, classes characterize sets of objects with common properties; this approach can be seen as the more general one.

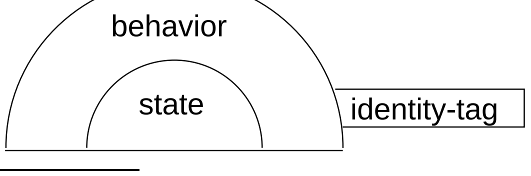
Another approach starts by developing programmer-defined types: A number may be thought as being characterized by its data AND the methods for its manipulation, like addition and subtraction, multiplication and division.

Since real-life objects often can be seen as displaying a considerable complexity, their relevant properties have to be isolated. That process of model-building is also called abstraction. Object-Orientation has a considerable conceptual overhead, but a readily approachable aspect can be shown with the stamp-analogy: Classes within object-oriented structuring are like stamps (which the programmer can carve or take from a library). With these classes the programmer can instantiate objects, much like the stamps can be used to produce several prints, into which it is possible to write data by hand. There even may be little stamp collections, which appropriately combined ("extended") produce an new type of printed matter.


stamp-/class library (prewritten or self-programmed)
stamping/instantiating objects

Concept: Object/Class-Model

Concept: Object/Class-Model — This section just introduces the central notion of the Object/Class-Model: An object is defined as an entity having a state, a behavior and an identity[46].



[45]Nygaard, Kristen and O. J. Dahl. *The Development of the Simula Languages* in History of Programming Languages. Academic Press. New York, NY. 1981.

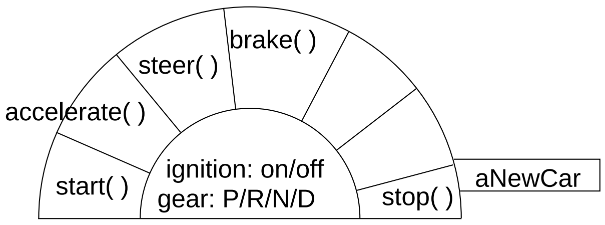
[46]In object-oriented programming-languages the "state" of an object usually is given by variables and the "behavior" is given by methods. Though the identity often may be thought as being given by a variable-name, this may not be true when variable-names are pointers, that may change their pointing-direction! This is the case in Java; there, an identity-tag has to be retrieved by a special `hashCode()`-method, as introduced on page 161. CAUTION: Variable-names in Java do not fix the identity of an object!

In the graphical representation the identity-tag often is omitted, because of self-evidence. But the identity gives the means to discern two objects with the same state and behavior. Two objects with the same state and behavior are called equal (but they are not the same due to their different identities).

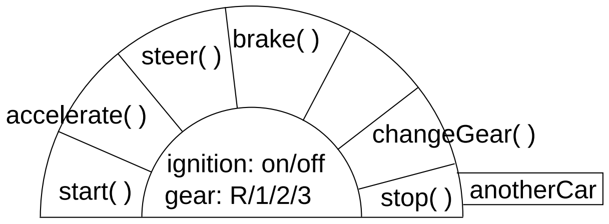
Later, the object-model is supplemented by an equally generally defined class-model, see page 140.

Example: Object, Class and Class-Hierarchy

As an example of such a general object-notion may be taken a new car coming from a production line: Although the cars look alike, the car under consideration is a unique one, it has an identity. It has also a state describable by the gauges for gasoline, motor oil, braking fluid, gear box fluid, cooling water, battery voltage, engine on/off, lights on/off, blinker right on/off, blinker left on/off, warning blinker on/off and other dashboard indicator positions. The behavior of that car, starting the engine, driving forward, driving backward, changing the direction of driving is described by more or less complex sequences of actions as found in the handbook of the car (or to be learned by driving it). The graphic below presents two objects which can be seen as simplified representations (so-called abstractions) of a real car:



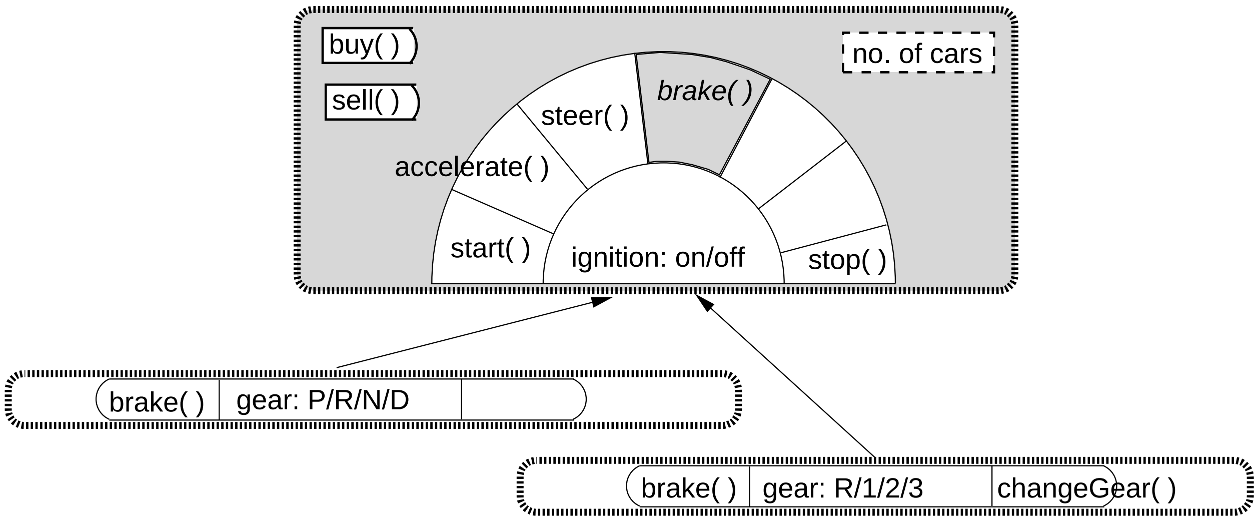
car with automatic transmission



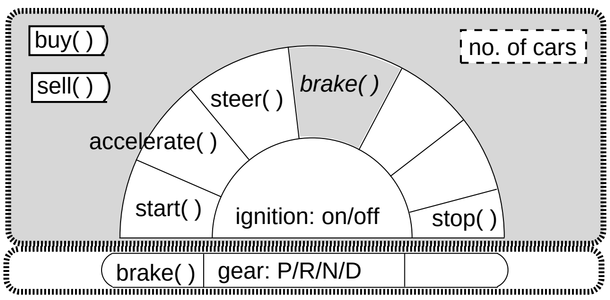
car with mechanical gearbox

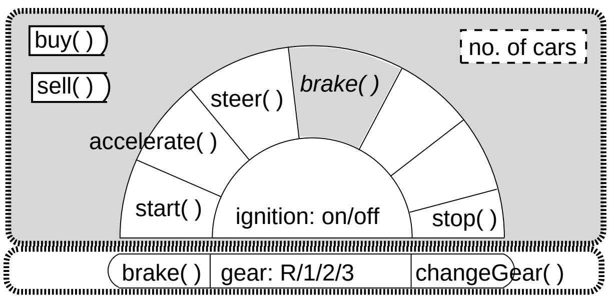
One car may have an automatic transmission. Another car may have a mechanical gearbox, this changes the possible state of the car (possible gear-positions are 1/2/3 instead of P/N/D) and this even extends the behavior of the car by an explicit complex changeGear()-procedure. So far, the structure of the above objects can be overseen easily. This view may change in cases where ten or hundred cars have to be considered, abstracted in the way above. If more cars are to be collected, then the different structure of the objects would become clearer by isolating common states and behaviors to get the pure structure of the collection of objects:

Car- Class Structure



(The various features, like rounded frames and grayed backgrounds, of the graphic above will be described successively in the following paragraphs.) Of course, the pure structure above does not relate directly to any individual object, but any given individual object can be classified fast and easily by using the structure above. The structure just collects all the different features used to describe the two kinds of cars above. The iglu-shaped part of that analysis describes a sort of basic car structure giving no information about what gearbox-system such a car contains. The extension of that structure produces two new structures: One of those extensions describes a car with an automatic transmission, and the other describes a car with a mechanical gearbox. The rounded frame with the iglu-like figure inside represents a structure, called a **class**. That **class can be extended** by adding one of the little frames: class class-extension





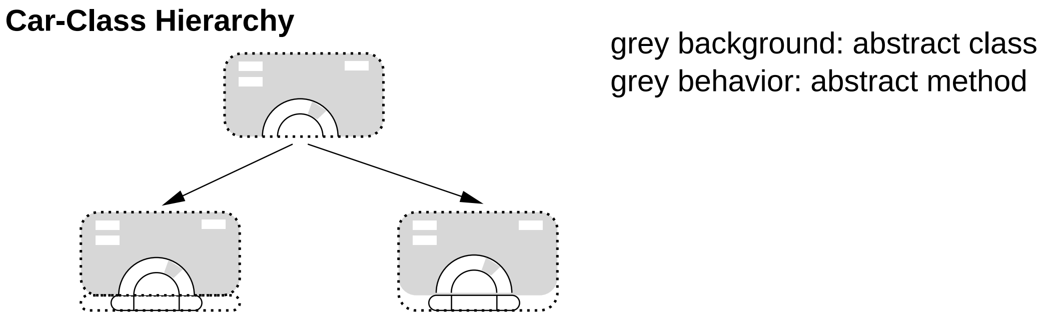
The resulting new classes above can be used as templates for producing objects (Car-objects[47] with automatic transmission and Car-objects with a mechanical gearbox). The frame around the iglu-shape, or around the burger-like shape indicates that it is a class or part of a class-structure. Producing objects from a class is also called instantiating the class or **generating an instance of the class** or **deriving the object from the class**. This instantiating process in a way resembles the stamping-act: The class is used as a stamp for printing out one or more objects. instantiating a class

The structural analysis of the Car-objects produced a **class-hierarchy** given by the abstract Car-class, without any gear-specification, and the two extended classes, above: A class for cars with a mechanical gear-box and a class for cars with an automatic gearbox. class-hierarchy

Buying a car can be seen as instantiating the car in your living area. Selling removes it. Buying, selling or scrapping the same car more than once in the time of your ownership would be unusual. So buying, selling, scrapping can be seen as actions that are part of the behavior of the Car-class rather than reformulating them for individual Car-objects (The sales-man or the car-crunching machine are not integral parts of the car's state or behavior!). These behaviors can be assigned to the Car-*class* and then are called **class-behavior**. class-behavior

The number of Car-objects, which are described with this class-model, neither characterizes the individual Car-object; but that number describes the *set of Car-objects* which is considered. So, the "number of Car-objects" can be seen as a **class-state**, to be associated with the Car-*class*. In the frame that fixes the class-structure, the class-states and class-behavior are collected as squares or square-like boxes in the upper corners. class-state

The graphics above allow another interesting observation: The base-class, that can be extended, cannot be sensibly instantiated; the Car-object would have an undefined gearbox and therefore couldn't describe an average real-world car. – Such classes which cannot be sensibly instantiated are also called **abstract classes**. But, as can be seen above, abstract classes can be used for devising clearer structures and abstract classes can have a defined behavior and a definable state which they transfer onto the extended classes (the ignition on/off state and the other behaviors) and instantiated objects thereof. Abstract classes here are written with a gray-underlaid frame, to make them discernable from non-abstract, instantiable classes. (The Java-implementation of the notion "abstract class" is given on page 183 and an elementary example in Java can be found on page 181.) abstract class


Car-Class Hierarchy
grey background: abstract class
grey behavior: abstract method

methods and variables

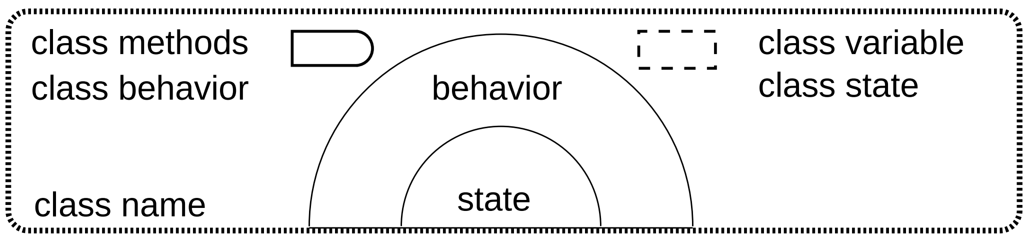
As could be seen above, the very general notion of "behavior" usually is realized as a set of actions or procedures also called **methods**. Whereas the notion of a "state" usually is realized as a set of **variables**. There can be discerned so-called **instance-variables** (ignition with the values on/off, gear position with the values P/R/N/D) used for describing the state of an object and so-called **class-variables** (number of Car-instances) describing the state of the class. instance-**variables** and class-variables

[47]The upper-case letter "C" indicates that "Car-object" denotes a mental image — the model — of a car, not the real thing!

instance-**methods** and class-methods

The same classification can be applied to the *behavior* of classes and objects: There are **instance-*methods*** (start(), accelerate(), ...) that describe the behavior of the individual object. These instance-methods can be invoked for each object individually (car1.accelerate(slow) and car2.accelerate(fast)). – But there are also **class-*methods*** which describe behavior of the entire class: Like Car.buy(car1) and Car.buy(car2). The act of buying is taken as a class-method, because the buying-process can be described independently from the individual car, whereas for example the price of the car describes an aspect of the car's individual state. Later, in the context of programming-languages, the equivalent to producing and buying a car can be seen as the instantiation of an object by using a class-specific constructor-method.

These remarks make it possible to identify the various abstracted notions in the graphical representation of the class-model:



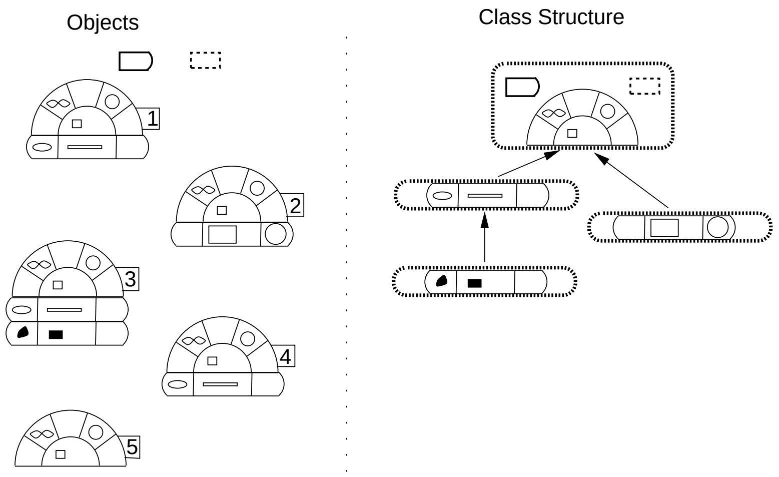
abstract method

Finally, consider a reason why even unimplemented methods, so-called **abstract methods** may be a sensible idea: The brake()-method is a method of the abstract Car-class. But the braking-process differs in a car with a mechanical gearbox from that in a car with an automatic gearbox (The clutch should be released during a heavy braking-process.). So, specifying the braking process for the entire Car-class becomes difficult because the brake()-method has to be implemented differently for the two derived classes. But still it may be useful to have at least the idea of a brake()-method within the Car-class, because braking is what every car should be able to do. This necessity can be indicated with a non-implemented (abstract) brake()-method in the Car-class. (The Car-class itself couldn't be instantiated, because there exists no conventional car without a transmission of any kind, and therefore the entire Car-class has to be called abstract.) Of course, a class with at least one abstract (unimplemented) method automatically has to be declared abstract, because objects with unimplemented methods violate the idea of a defined behavior.

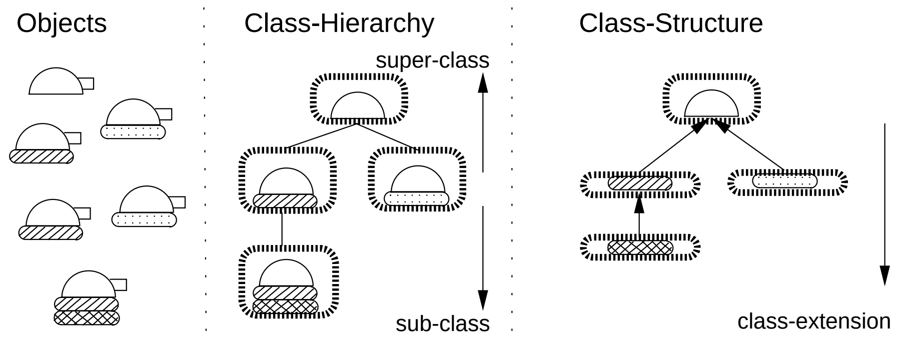
abstract class

What has done with the brake()-method could have been done also with the accelerate()-method, assuming that acceleration is thought of involving a changing of gears. Why the difference? There exists no particular reason for that difference: Class/object-structures are models chosen by an onlooker for describing an area of interest or a problem according to the onlooker's necessities. THERE IS NO INHERENT ULTIMATE TRUE STRUCTURE TO BE FOUND, the choice above was taken just to illustrate the class/object-model. Of course this freedom, or seemingly arbitrary approach, can be used to properly design classes to depict real-world-problems; with the intention of letting this class/object-model be re-used as much as possible. — Making class-design, problem-oriented as well as anticipating future needs, can turn out to be more difficult than expected!

The relation between a collection of objects, class-methods or class-variables and a given class-structure is restated in the following graphic:



Observe that all objects share the class-variables and the class-methods. Objects Number 1 and Number 4 are equal but not the same! (Their state and behavior are the same but they have been given different identities.)



extension-hierarchy

A given class-structure, like that of the Car-class, makes it easy to construct sub-classes by adding class-extensions. In the (resulting) class-hierarchy, also called extension-hierarchy, those classes that have been extended often are called **super-classes**, their extensions consequently are also called **sub-classes**. Observe also that THE MOST COMPACT DESCRIPTION OF THE CLASSES CONSIDERED GIVES THE ORGANIZATION OF THE CLASS-STRUCTURE. Therefore in programming and documentation, classes are economically declared or described by using the more fragmented but least redundant representation given by the class-structure (and avoiding the organization as a class-hierarchy or worse as a collection of objects).

super-classes and sub-classes

class-structure: most compact description, thus widely used!

The notions “object” and “class” thus can be seen from two sides:

(Class as Template for Creating Objects) A class can be seen as a collection variables and methods that can be of the instance- or class-kind. A class-hierarchy is given by a structured collection of these variables and methods. The class then serves as a template for objects; and the hierarchy of classes represents a tree-structure of successively refined templates. Class-methods and class-variables can be used independently from the existence of an object.

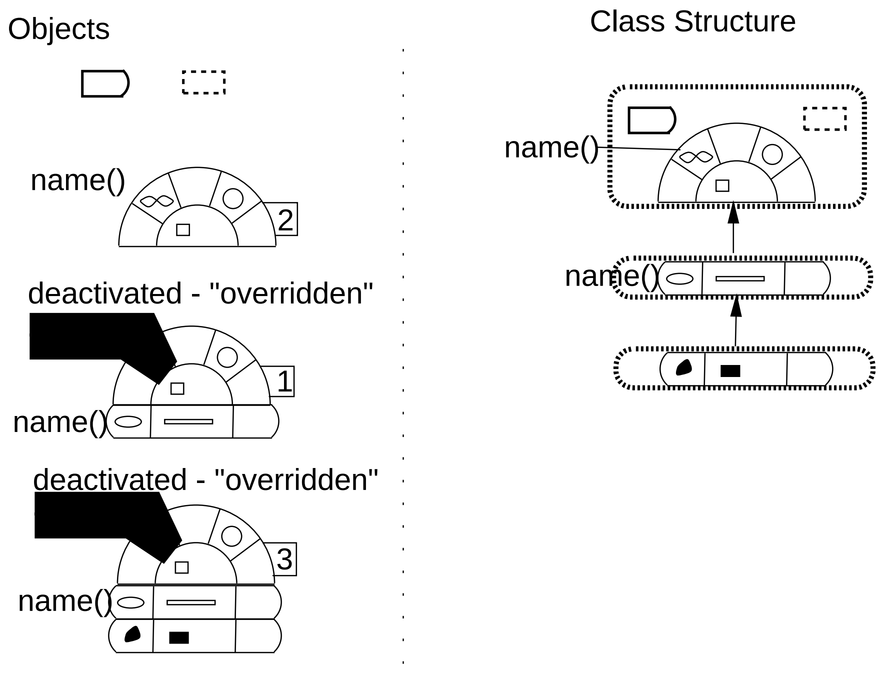
(Object Collection is abstracted to Class) Consider several similar objects *together* with methods that describe how to construct and how to destruct an object: The methods that exist independently from the existence of any object are called class-methods. There may be a variable that has the same value for all objects under consideration, that variable is likely to become a class-variable. Another variable that occurs in some of the objects and contributes to the object’s individuality by having different values from object to object needs to be an object’s state- or instance-variable. Depending on the variety of the given objects, there are compilable different collections of variables and methods. These collections constitute the classes that can be arranged into a class-hierarchy.

So both approaches, the one starting with the class-notion, or the one starting from the object-notion are part of - or result in the same object/class-model.

OUTLOOK: The class-structure is what will be used to define class-hierarchies in Java on page 163. An example of a larger class-hierarchy is given by the graphic on page 564 in the context of arranging a Graphical User Interface; details of the emerging class-structure of the three base-classes (`Component`, `Container`, `JComponent`) are given successively in the lists on pages 553, 556 and on page 559.

Overriding Methods and Shadowing Variables

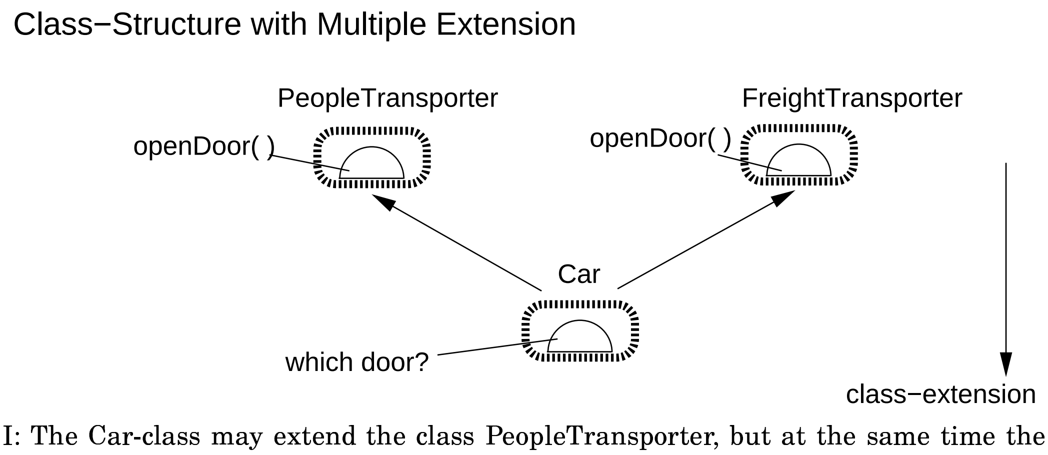
Overriding Methods and Shadowing Variables — Consider the brake()-method in the graphics around page 132; there the abstract method has been redefined differently in both sub-classes. What about re-defining non-abstract methods of super-classes? Or formulated conversely, what if a method or a variable in a class-extension has the name of a method or variable in the super-class? First, objects derived from (one of) the super-classes use the original method, BUT all objects derived from the class-extension or its sub-classes use the newly defined (instance-)method or (instance-)variable. The overriding of a method has been visualized in the graphic below:



Remember the shadowing of variables by local variables of Java methods as introduced on page 123, shadowing of variables by class-extension can be imagined likewise. Overriding class-methods and shadowing class-variables would produce *class*-specific variants. For an example consider the Java source-code on page 167 and the corresponding footnote following that source-code.

Class-Extension — Inheritance with multiple Super-Classes

Class-Extension — Inheritance with multiple Super-Classes — Another approach to the process of extending classes may be thought of by imagining a class having *several* super-classes. Example: Below the Car-class extends the class PeopleTransporter and the class FreightTransporter. Each instance of a Car may be used to transport people as well as freight:



Example I: The Car-class may extend the class PeopleTransporter, but at the same time the Car-class may extend the class FreightTransporter. This makes it possible to construct an ambiguity: Both classes may use a method openDoor(), but as can be seen with the example of a standard car, such a method usually means different doors (doors to the area with seats or door of the car's trunk). Though, here, the ambiguity may seem constructed, a general method for structuring contexts should be designed to spare the programmer or the user unnecessary complexities. (Sketch the class-structure for this example, the graphic above may serve as an orientation.)

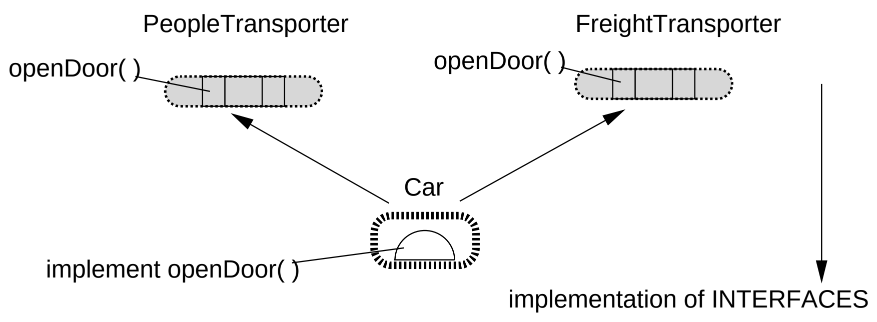
From above emerges a BASIC PROBLEM: MULTIPLE INHERITANCE BECOMES AMBIGUOUS IF TWO BASE-CLASSES HAVE METHODS OR VARIABLES OF THE SAME NAME. The question may arise about which member takes precedence when they both have the same name. Therefore, using classes that extend multiple super-classes should be considered rather careful.

again abstract methods

A tricky workaround concerning this predicament can be formulated as follows: Consider a class having only named, but unimplemented methods, also called **abstract methods**. This makes such a class automatically abstract, because it obviously cannot be sensibly instantiated. This kind of class, with no apparent functionality, furnishes a kind of template and this seems to be the best one can get, to formulate multiple inheritance without an inherent source of contradictions or unnecessary considerations about precedence.

Example I (continued): So, using the example above, PeopleTransporter's openDoor() and FreightTransporter's openDoor() may be unimplemented. When both are extended to the Car-class, the openDoor()-method has to be implemented considering the two types of doors for loading people and for loading freight.

Class−Structure with Interfaces (fully abstract, multiply implementable)



"Classes" which contain only unimplemented, so-called abstract, methods AND allow other inheriting "classes" of the same kind at their side are called "interfaces" in Java. The name "interface" signals the option to realize a kind of multiple inheritance as described above. Java[48] classes can implement *multiple* "interfaces" and thus furnish a version of multiple inheritance that proves relatively free of inherent complexities. For more see page 185.

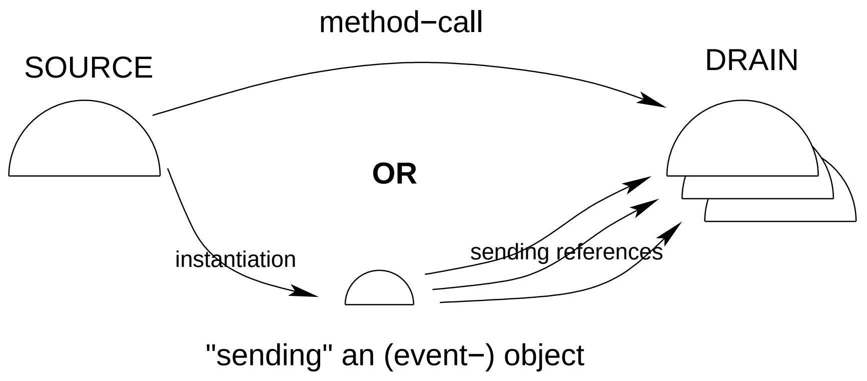
Java's "interfaces"

Communication-Mechanisms

Communication-Mechanisms — The structuralizing of objects by classes and class-hierarchies may be considered as useful, but coexisting objects should be able to communicate to simulate real-life interaction or just to make the object-model more efficient:

Method-Activation or method-calling represents the natural means of communication between objects. This requires the objects to know each other well, but this knowledge of each other may also involve an unwanted specialization of the objects.

Communication by method−call or by dispatching an Event−Object



Message-Objects exchanged between communicating objects generally requires the objects to know very little about each other: The only knowledge, that remains necessary, is to know how to interpret the message-objects, how to send them and how to listen for them. The communication is thus realized by exchanging messages between objects, this "uncouples" objects. Then, the methods that have to

[48]The programming-language C++ allows multiple inheritance which has to cope with making inherited behavior non-ambiguous.

be called, are only listening- and dispatching methods, which, by following conventions, generally simplifies matters! For an object in a message-exchange context, a carefully-designed set of message-objects furnishes a relatively lean interface to its outside world. The rest of the implementation of the object could be changed, if necessary, without changing the behavior of the object. This isolation of the inner workings of an object exemplifies the concept called **encapsulation**. Like in proceduralization, encapsulation of object-functionalities means hiding the details of the implementation.

For more Java-related information on general messaging see pages after page 271. If the interaction between objects is realized by exchanging message-objects rather than by issuing object-specific method-calls, then this is one first step towards what is often called **component-model**. There, objects usually send and filter highly normed event-objects. For more information on the area of software-components, see page 311.

Object/Class-Model Keywords

Object/Class-Model Keywords — The following paragraph gives a small glossary on the main notions of the object/class-model:

abstract- or virtual class A CLASS THAT CANNOT BE INSTANTIATED, this means that no instances of that class can be created. Abstract- or virtual classes are collections of common states and behaviors; but these become only part of an object that is an instance of some sub-class of the abstract- or virtual class. Some of the methods may be implemented and others may be declared abstract. Example: A Car-class which lacks the specification of the transmission type is an abstract class, the brake()-method may be abstract but the accelerate()-method may already be implemented.

abstract method A named method with no functionality defined. A class which contains at least one such unimplemented method automatically cannot be instantiated; that means a class with a single abstract method becomes implicitly abstract itself! A standard use of abstract methods can be found in multiple inheritance, where these method-declarations (in Java “interfaces”) are used as templates. But as could be seen with the example of the Car-class, when considering the brake()-method non-abstract, classes can be considered to be abstract without having abstract methods. A car has to have either an automatic transmission or a mechanical gearbox! (Unless it is an electrical car; this option has been ignored as the Car-model was introduced.)

class A collection of possible states and behaviors, either gathered from a set of similar objects or established from a given problem or area of interest **(class as collection of states and behavior), (class as an abstraction of a collection of objects)**. In programming, a class can ALSO be taken as a programmable type-concept **(class as a customizable type)**. A class can be seen as a template that furnishes the fields (state) and methods (behavior) of an object **(class as a template)**. A class gives the behavior and the possible states; an object of that class has the same behavior but only one fixed state and an identity. Due to the given identity (a registration-number or some other tagging-instrument) of an object, there can be several objects with the same state, instantiated from the same class.

class-extension Defining a new (sub-)class by adding other kinds of states and behaviors, respectively kinds of variables and methods to an existent class.

class-hierarchy A collection of classes, arranged according to their dependencies in form of sub- and super-classes. These kind of dependencies usually form a tree-like structure.

class-methods Methods which are associated with the class, not with an individual object. For example, methods (also called constructors) for creating instances of that class. In Java, beside constructors, so-called **static methods** are the class-methods.

class-variables Variables that are associated with the class, not with an individual object. For example the number of objects that have been instantiated during the use of the class can be counted by using a class-variable. In Java, class-variables are also called **static variables**.

container-object An object which contains a list of references pointing to other objects. Container-objects typically implement buffers, stacks or little non-persistent databases. Java supplies container-objects like instances of the class `java.util.Vector`. CAUTION: Though Java array-types can index objects, array-types are something different than class-types (See the syntax-diagram on page 143, and the introduction on page 234.). Only from Java class-types can be instantiated objects!

encapsulation Hiding, often complex, self-contained solutions of problems and giving them a much easier programmer- or user-interface. This allows a relative ease of use without having to grasp inner complexities. Proceduralization offers a means of encapsulation by needing only to know the method's signature and return-value to use it. Object-Orientation can be made to work similarly on the level of the entire object. Encapsulated code may document its functionality in a so-called (Application) Programming Interface (API). Especially in the context of networking, the effects of simplifying things, by encapsulation of code, are circumscribed by the notion "transparency".

inheritance All sub-classes (or class-extensions) receive the methods and the variables furnished by their super-classes (sometimes also called parent-classes). INHERITANCE DESCRIBES THE SAME AS CLASS-EXTENSION; but inheritance focuses on the sub-classes whereas extension stresses its starting-point being the super- or parent-classes.

instance of a class An object derived or instantiated from that class. The class represents a template for state and behavior; an object bears a concrete state, a concrete behavior and and identity(-tag). Also refer to the item "object" below.

instance-method A method associated with an individual object (in contrast to a class-method which is associated with the entire class). The nature of that association, for example, makes the method change one of the object's variables. Instance-methods are taken to represent the behavior of their object.

instance-variable Instance-variables are associated with an individual object (in contrast to a class-variable which is associated with the entire class). Instance-variables describe the state of their object.

method A method can be called (for example by other objects) to alter the object's state or to send messages to other objects. In programming practice, a method (also called function, procedure, subroutine) usually represents a kind of named piece of code associated with an object or class. Usually methods are the means by which another object can change the object's variables (alter the object's state).

message A signal from a source-object to a target-object, usually requesting a method-invocation of the target-object. Sometimes, simply calling the method of another object and transferring information within the method's arguments is considered as sending a message! But, often messages are objects themselves, then they bear a standardized structure, which is known to all communicating instances. Then, the method-calls used for communication of objects become highly standardized. Usually a message-object contains the name of the sending object, the name of the receiving object, the name of the method to invoke and some variables that are used as the invoked method's arguments. This concept has been developed further beginning with page 271 and on page 571.

multiple inheritance The case where a class inherits state and behavior of multiple super-classes (In Java, multiple super-interfaces). To avoid ambiguities of the super-classes with same variable- or method-names, only abstract methods and unassigned variables should be used. The Java `interface`-construct furnishes this restriction.

object An entity with a **state**, a **behavior** and an **identity**. In programming, an object is given as an entity of variables and methods together with an identity. This generality often reduces to an object being seen as a variable that has been instantiated from a class-definition, where the class is taken to be a sort of generalized type-definition.

overloaded methods Methods are identified by their object's name, their own name and their parameter-list. So even in their own class, different methods can have the same name but a differing parameter-list to remain valid. Such methods are called overloaded. Remember the section beginning with page 124.

overridden methods Overridden methods always occur in the context of a class-extension: A class-extension may define a method with the same name and the same list of parameters as a method given by the super-class. The newly defined method becomes the default method of any objects instantiated from this extension or any further extensions. An analogon on the level of variables presents itself: Two variables with the same name but different overlapping scopes produce the following effect: The variable with the limited scope overshadows the other. (Why are there no overridden variables? Because overriding variables would amount to redefining the type of the variable-name, nothing more.)

Pragmatics: Constructing class-hierarchies, to describe a problem efficiently, usually turns out to be demanding and even may need redesign efforts. But there are some simple rules and conventions that may ease the task: When describing the real-life problem, using the everyday language,

nouns indicate an object or a class to be used in the object-oriented analysis,

verbs indicates a method's functionality and the formulation

is a *name* indicates something to be an object or a sub-class.

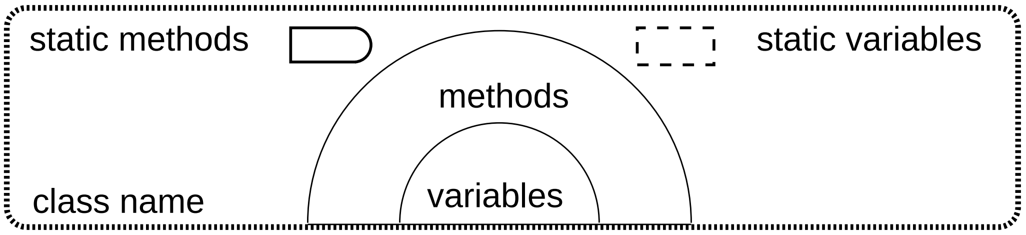
Object-oriented analysis of problems can happen without using a programming-language. Object analysis represents a problem-oriented method of program-design and therefore usually happens *before* constructing source-code. But object-oriented design usually addresses itself to EXTENSIVE real-life problems with less dominant algorithmic complexities. The wealth of approaches and methodologies of object-oriented-analysis and -design is described in books or related Internet-sites.

Concept: Object/Class-Model within Programming-Languages

Concept: Object/Class-Model within Programming-Languages — In many programming-languages, aggregates of variables are called `records` or `structs`. Aggregates of command-sequences and variables are described by functions, methods or subroutines. Here aggregates of variables AND methods are called **class**es.

```
class name
{
  variables;
  methods;   // usually for manipulation of the variables
}
```

In that context, remember the object-model on page 131 and the graphic of the general class-model on page 134. The general class-model, introducing state and behavior, changes in the context of a programming language to a class-model given by an aggregate of methods and variables:



Classes in programming-languages can be given another variant of definition: Classes just furnish a way for the user to aggregate data (variables) and methods (functions, subroutines). Object-oriented programming-languages also furnish ways to introduce static class-members and ways to derive individual objects from a given class.

For practical programming purposes the notion of a class can be used

- to introduce programmer-defined types,
- to use a class as a self-contained program and
- to use a class-structure as an index for reference when re-using software:

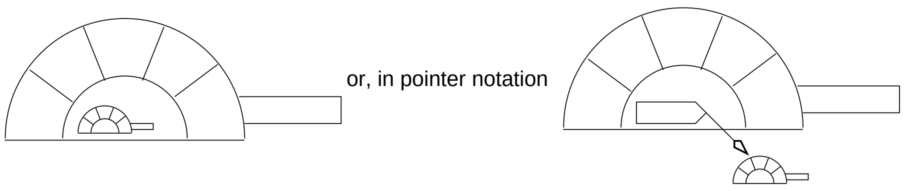
Classes as Custom-Programmed Types **Classes can be used to define programmer-specified types** within the programming-language. These programmer-defined types give templates for the data and furnish the methods to manipulate the data. The central idea aims at presenting the data and methods for the data-manipulation as an entity. Thus, in other contexts, methods for data-manipulation do not have to be re-written. To the programmer, both, the data and the methods, are related intrinsically; just like any integer type furnishes mathematical functions like addition, subtraction, multiplication and division for its manipulation:

```
class Integer // this is a concept-class and does not work in Java
{
  int i;
  addTo(int j){return i+j;}; multiplyWith(int j){return i*j;}; // etc...
}
```

If the standard types of the given programming-language can be supplemented by types defined by classes, then variables of class-type even can be declared inside classes:

```
class ClassName1
{ ClassName2 identifierOfObject; // declaration of variable of class-type
}
```

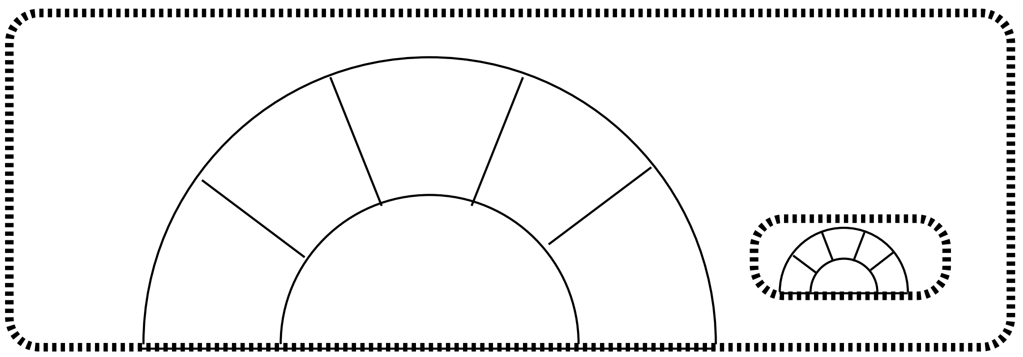
Thus a variable of class-type — an object itself — can be seen as part of the state of an object:



So the state of an object is given by objects (pre-programmed or programmed by the application-programmer) or by variables of primitive-type. Objects of standard classes given by the programming-language (Java: `String`) or variables of primitive-types (Java: `int`) finish the recursion indicated by the diagram above (objects in objects in objects . . .). For example, in the concept-class above, named Integer, the `int`-variable of primitive-type, given by the programming-language, defines the state. A practical example of this idea is given on page 142, where a newly-defined type (VarInteger) is used for declaring and instantiating a variable (there the variable can be seen as an equivalent to an object).

Instead of containing only a variable-declaration, a class even may contain a class-definition, a so-called inner class, giving a kind of local type:

```
class ClassName1
{ class ClassName2 // class-definition
  { ...
  }
  ...
  ClassName2 name; // declaration of a variable of that local type (inner class)
}
```



Inner classes are described further beginning with page 157.

Class as a Program A class can be a flexible type, or **if a class is furnished with a `main()`-method, then a class can be a program**:

```
class Program // this is a concept program and does not work in Java
{
  main(){ statements }
}
```

If the source-code inside the main()-method references other classes or creates and references other objects, then the class with the main()-method does represent the central piece of a program! Conventionally, classes with a main()-method enable an end-user to issue some command on the level of the operating-system, eventually triggering a call of the main()-method (the java-command entered into a terminal-window of the native operating-system triggers a call of the main()-method). This peculiarity gives classes with a main()-method all the characteristics of a regular program. Thus, in most object-oriented programming-languages, the class-notion becomes more than simply a flexible way to define new types.

Class-Structure as Index to Software-Library But the perspective of the class/object-model gives another important functionality to object-oriented programming-languages: As can be seen in the graphic on page 135, class-extension lets the code be positioned in a sort of programmer created hierarchical index-structure for easier identification. Successive class-extensions and the ensuing class-hierarchy can be used to **realize a structure of the involved objects**. Since the super-classes may be used multiply (remember the car example above), that kind of structuring also makes it possible to easier re-use code.

Java: Classes as Programmer-Definable Types and Programs

Java: Classes as Programmer-Definable Types and Programs — Java classes can be more than templates for objects. A Java class with a main()-method constitutes the center of a program. The fully-functional example below defines two Java-classes: The first class introduces a programmer-defined type and the second class acts as a program due to its main()-method. (The use of the modifiers public, static and void is described elsewhere: public on page 175, static on page 147 and void beginning with page 117)

```
// a Java class: a programmer-defined type or a program
class VarInteger                                // programmer-defined type
{ int i;                                           // state: instance-variable
  VarInteger( int x ){ i = x; };                   // class-method: constructor
  public void set( int x ){ i = x; }               // behavior: instance-methods
  public int get(){ return i; }
  public void increment(){ i = ++i; }
  public void print(){ System.out.println( i ); };
}
public class Program                             // program or application
{ public static void main( String unused[] )     // class-method: static main()
  { VarInteger theObject = new VarInteger( 0 );
    String theOtherObject;
    theOtherObject = new String( "Hello" );
    theObject.print();
    System.out.println( theOtherObject );
} }
/*
[localhost:~/JavaCodeFiles/Basics1/SCObjectOrientation1] andreadipietro% javac Program.java
[localhost:~/JavaCodeFiles/Basics1/SCObjectOrientation1] andreadipietro% java Program
0
Hello
[localhost:~/JavaCodeFiles/Basics1/SCObjectOrientation1] andreadipietro%
*/
```

Another example of a programmer-defined type is shown in the source-code on page 76, where the Pair-class has been introduced. These two approaches for using Java classes (program and custom-defined type) are the most typical and elementary ones. – A class may also be used to just furnish a collection of methods; as does the java.lang.Math-class which just offers a range of mathematical functions to the Java programmer (for more see pages 94 and 257).

A class with a main()-method remains instantiable!

An instantiation of a class with a main()-method has no special meaning with respect to the main()-method and its program-character: The main()-method just remains one of the static methods, except that it can be activated by an external event (the issuing of the “java”-command). A first example of such a case is given on page 159.

Java: Reference-Types and their Instantiation with the new-Keyword

Java: Reference-Types and their Instantiation with the new-Keyword — Remember the introduction of primitive-types on page 83. Variables of primitive-type have their identifier fixed to an address in the computer’s Random Access Memory (RAM). These types, with identifiers being an unchangeable reference, thus being glued to the RAM-address and exhibiting a fixed behavior (addition, division with number types), are called atomic- or primitive-types. Primitive-types constitute an integral part of the Java language and prescribe the data-format (`int`: 4bytes) and the possible manipulations (Addition, Subtraction, Multiplication, Division). With variables of primitive-type assignment happens by-value, this means that an assignment copies the value to the variable-name's part of the RAM. Usually a computer processes variables of primitive-types faster than Java objects of class-type. remember: assignment by-value

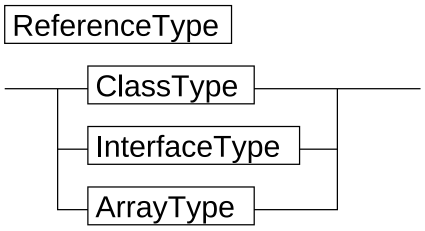
Java offers two kinds of basic non-primitive-types, the so-called reference-types :

class-type Currently discussed. A variable of that type is instantiated by prefixing a constructor-call with a new-keyword; for more see page 145.

array-type Realizes a simple form of aggregation of values or object-references. Variables of class-type as well as variables of primitive-type always can be aggregated as an array. Variables of array-type are recognizable by the brackets [ ] used in declaration and referencing, for more see page 234.

An example of introducing a class-type has been given in the preceding program on page 142. There has been defined a class(-type) "VarInteger" and from this class has been instantiated an object (referenced by an identifier "theObject").

The interface-type gives a variant of the class-type for cases of multiple-inheritance, for more see page 185.



ReferenceType := *ClassOrInterfaceType* | *ArrayType*

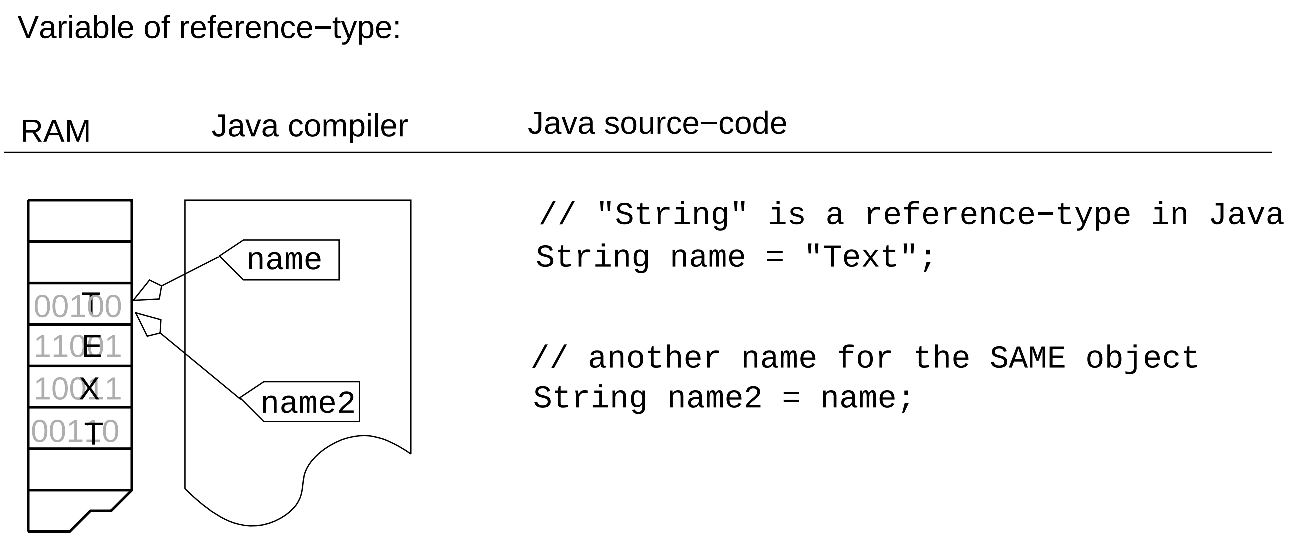
ClassOrInterfaceType := *ClassType* | *InterfaceType*

ClassType := *TypeName*

InterfaceType := *TypeName* — ClassType 146, InterfaceType 187, ArrayType 234

(ClassOrInterfaceType is used in ClassInstanceCreationExpression on page 151)

Like variables of primitive-type, variables of reference-type have identifiers that point to an address in RAM-space. But, the RAM-space-address of variables of reference-type is changed by assignment (=), NOT the value stored there, as it is the case with variables of primitive-type. IMPORTANT TO NOTE: IDENTIFIERS OF VARIABLES OF REFERENCE-TYPE CAN BE SEEN AS ALIASES AND THEIR POINTING-TARGET CAN BE CHANGED BY ASSIGNMENT:



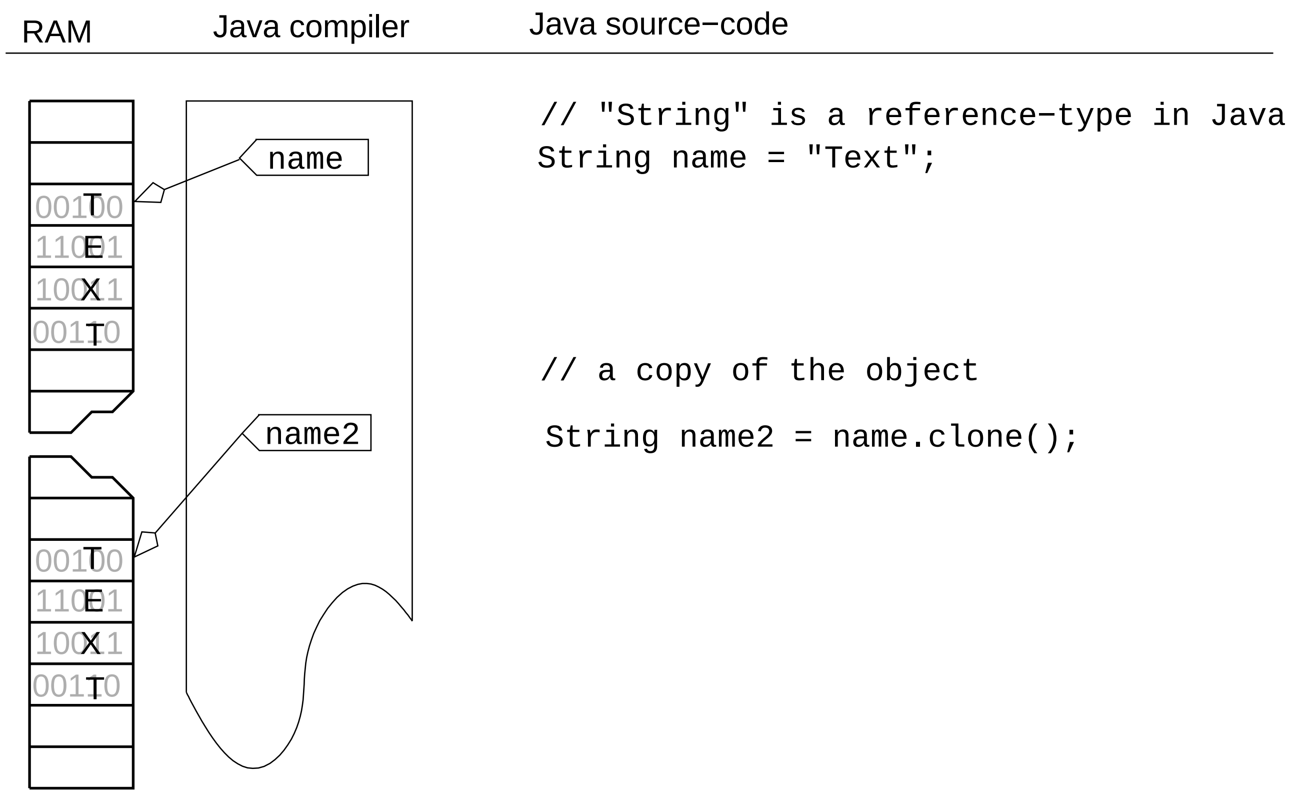
```
// "String" is a reference−type in Java
String name = "Text";

// another name for the SAME object
String name2 = name;
```

A reference is a container for a RAM-address which can be changed by the programmer by assignment. That's an important difference to Java variables of primitive-type where the references are fixed to an address of the RAM-space. And the act of assigning variables of primitive-type to each other with the equal sign (=) means copying. (see the graphics on page 83) For variables of reference-type assignment assignment by-reference

(**=**) means: If objects and associated names are given, reassignment of names by using the equal sign (=) only exchanges references, not the entire object, as indicated within the graphic above.

Copying of an object, a variable of reference-type, is done with a `clone()`-method as described below:

Copying a variable of reference−type with the clone()−method:

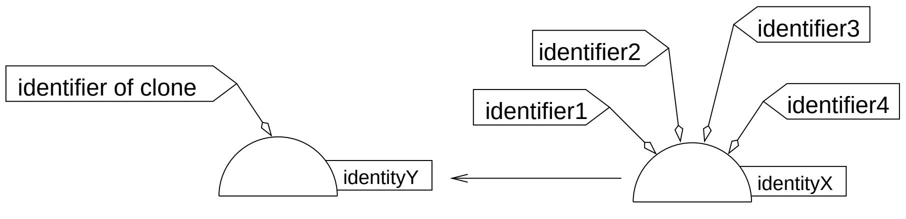


```
// "String" is a reference−type in Java
String name = "Text";

// a copy of the object
String name2 = name.clone();
```

The `clone()`-method PRODUCES A NEW OBJECT that contains the same data AND RETURNS A REFERENCE TO THAT NEW OBJECT! (It does NOT "return the new object" itself!)

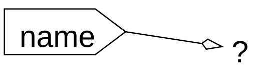
This statement is true in all generality: Java-methods with return-types of the primitive kind return the variable-value itself. But if the RETURN-TYPE OF ANY JAVA METHOD IS OF REFERENCE-TYPE, then, like the `clone()`-method, the method returns only a reference to the object!

The name of every reference-type could be suffixed with the name "Pointer" (StringPointer instead of `String`), to reflect the difference to the named value of data that has a primitive-type. Data of primitive-type has only one name, assignment with the "="-operator copies that value. Whereas data of reference-type can have multiple names, meaning that multiple pointers are referencing the same data; here assignment with the "**=**"-operator just makes the pointers equal. The assignment-operator, used with identifiers of reference-type, does not copy the data!



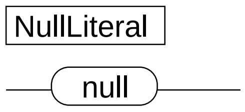
```
identifierOfClone = identifier1.clone();
```

Any name of reference-type can be assigned the literal `null`, which lets the name reference nothing.



```
name = null;
```

```
    variableName = null; // an identifier of reference-type not yet assigned
```



NullLiteral := null

Instances of reference-types (to be more precise, instances of non-abstract classes and arrays) are created by prefixing so-called **constructor-method**s with the **new**-keyword or by calling so-called **factory-method**s, described later on page 151. (The `clone()`-method is a factory-method, because calling this method creates a new object and this method returns a reference to that object.)

constructors and factory-methods

```
    ClassName variableName = new ClassName();
    // Class-methods with the class-name are called constructors.
```

The `new`-keyword in conjunction with a constructor allocates the memory, creates and initializes the object. Then the variable-name (identifier) represents a reference to the object; the variable-name does not represent the object itself, contrary to names of variables with primitive-type!

Below the processes of declaration, memory allocation and instantiation are compared for variables of primitive-type and variables of reference-type:

variables of type-		Declaration	Allocation	Instantiation	
-primitive		`PrimitiveType name`	=	`value;`	
-reference		**`ClassType name`**	**= new**	**`ClassType();`**	← **a constructor-call**
-reference (array)		`PrimitiveType [] name`	= new	`PrimitiveType[];`	(arrays, see page 234)
-reference (array)		`ClassType [] name`	= new	`ClassType[];`	

Java's **`String`** **exception**

In Java, the `String`-class has been allowed an exception to the general class-usage; a Java `String`-object can be instantiated like a variable of primitive-type:

`String s ="text";`

Only in this special case of the `String`-class, Java allows an instantiation like those used with variables of primitive-type. Actually the `String`-class also allows every `String`-object to be instantiated using the `new`-keyword prefixing the `String`-constructor:

`String s = new String("text");`

This standard way of instantiating objects of class-type has been used multiply in the previous programs.

Java: Classes

Java: Classes — The following sections use the class-concept together with the idea of a reference-type to describe the Java-specific realization of programmer-defined classes. This includes, for example, inner classes, anonymous classes and anonymous objects, supplemented by scoping- or accessibility considerations.
In the previous chapter, the description of proceduralization by using methods had more implications than could be forseen by the pure concept[49], the same is true with the implementation of the class-concept by the Java programming-language.

Java: Class-Declaration

Class-declaration may also be called class-definition or class-implementation. The standard Java-form can be seen below:

```
    class ClassName { classMembers }
```

[49]Starting with considering named blocks of code, ending up with local variables making possible overloading, recursion and call-back.

Index

* in policy-file, 472
- in policy-file, 472
-1, default TCP-port, 388, 627
-1, end of file, 441, 627
-1, end of stream, 296, 627
.java.policy (text-file), 458, 461, 462, 467, 469
100 Percent Java, 72

Absolute Path-Name, 48
absolute path-name (example), 510
absolute positioning and layout-manager, 542
Abstract Class, 184
abstract class (non-instantiable class), 184
abstract class (non-instantiable class, concept), 134, 138
abstract class and static variables, 184
abstract class-type, type-casting identifiers to-, 552
Abstract Method, 184
abstract method (non-implemented method), 184
abstract method (non-implemented method, concept), 134, 136, 138
abstract modifier, 181
Abstract Window Toolkit (AWT), applets, 613
Abstract Window Toolkit (AWT), as API, 52, 67, 274
Abstract Window Toolkit (AWT), as package, 53, 184, 246, 251, 266, 505
Abstract Window Toolkit (AWT), class-hierarchies, 564
Abstract Window Toolkit (AWT), description, 520
Abstract Window Toolkit (AWT), event-handling, 276, 331, 440, 572
Abstract Window Toolkit (AWT), image drawing, 519, 595
Abstract Window Toolkit (AWT), image processing, 520
Abstract Window Toolkit (AWT), image processing (extensive), 527
Abstract Window Toolkit (AWT), peer-classes, 353
Abstract Window Toolkit (AWT), screen-components, 538, 547, 551
Abstract Window Toolkit (AWT), time-delayed data, 606
Abstract Window Toolkit (AWT), versus Swing, 520, 548
AbstractButton (javax.swing), 574
AbstractList (java.util), 237
Access (security-related), 449
access, random-, 233, 291
access-area, 430
access-area and scope in an Hierarchical File System (HFS), 172
Access-Area of an identifier, 172
access-area, extended by overriding, 175, 286
access-control, in security context, 449
access-control, locking data in threads, 213
access-control, with modifiers, 171, 175
access-modifier, 177, 179, 180, 429
access-modifier (protocol character), 345
AccessControlContext (java.security), 470
AccessController (java.security), 427, 453, 458, 462, 463, 465, 470, 476
accessing class-members, 155
accessing methods by overriding, 192, 286
accessor (of a software-component), 312
accessor-method (of a software-component), 312
ActionEvent (java.awt.event), 544, 573, 574, 578, 603
ActionListener (java.awt.event), 578
actionPerformed()-method (java.awt.event.ActionListener), 127, 544, 576, 591
activated window, 577
activated window, concept of-, 509
active graphics-card (hardware), 198
active modem (hardware), 198
adapter, 272, 289
adapter-class, 187, 190, 580, 584, 601
adapter-class with software-components, 318
additive-operators, 94
address of a computer in a network, 342
address-space and port, 231, 271, 337
address-space of a processor, 28, 291, 522, 535
address-space, in threading, 217
addressing resources network-wide: URL, 373
administrator, 458, 475
administrator (key-management and signing files), 488
administrator (using a security-manager), 465
administrator and jar-archive, 326
administrator and security, 448, 453, 457
administrator and tools, 54
administrator, cannot change a security-manager, 463
administrator, selecting policies, 460
aggregates to streams, 237
algebra, boolean-, 97
algorithm, 20, 41, 140
algorithm, compression-, 534
algorithm, cryptographic-, 478
algorithm, cryptographic- or encryption-, 480
algorithm, ROT-encryption-, 478
algorithm, Run Length Encoding (RLE)-compression-, 522
algorithm, sorting-, 195, 249
algorithm, strength or quality of a cryptographic-, 479
algorithm, ZIP-compression-, 326, 522
AlgorithmParameterGenerator (java.security), 485

AlgorithmParameters (java.security), 485
algorithms, implementation of cryptographic-, 485
algorithms, modes of encryption-, 484
algorithms, names of encryption-, 479
algorithms, names of key-pair-, 481
algorithms, names of- for digital signatures, 483
algorithms, names of- for hashfunctions, 482
AllPermission (java.security), 459, 461
alpha-value/-channel, 526, 527, 556
ambiguity and multiple inheritance, 136
ambiguous identifier, 187
ambiguous variable-name, 187, 239
AND-operator (&&), 98
AND-operators (&, &&), 84
anonymous class, 159
anonymous constructor-call, 159
anonymous object, 159, 279, 581
anonymous package, 241
anonymous screen-window objects, 159
API (Application Programming Interface), 51, 55, 129, 131, 190, 249
API (Application Programming Interface) as Java software library, 54, 59
API (Application Programming Interface, definition), 43
append()-method (java.lang.StringBuffer), 120, 257
applet, 67, 205, 396, 430, 613
Applet (java.applet), 68, 396, 397
Applet (java.applet) heavyweight, 565
applet and application, difference of-, 460, 613
applet and distributed computing, 338, 406, 423
applet communication, 403
applet HTML-tag, 68, 397, 399, 401, 431, 614
applet introspection, 402
applet methods, 529, 615
applet methods, triggerable-, 397, 400, 614, 617
applet runtime-environment, 67, 398, 564, 618
applet runtime-environment as security-sandbox, 463
applet runtime-environment, example of a-, 619
applet runtime-environment, start, 401, 432, 614
applet security, 460, 475
applet source-code, 399, 615
applet threading, applicational-, 401
applet threads, supporting-, 205, 232
applet with a main()-method, 619
applet's triggerable methods, 618
applet's uniqueness, 67, 333, 460, 613
applet's window, 400
applet's window (security mark), 475
applet, concept of an, 67, 397
Applet, description, 68
applet-container (= applet-sandbox), 398
AppletContext (java.applet), 404, 617
appletviewer, 67, 397, 431, 475
appletviewer as a Java program, 619
appletviewer, controlling security externally with-, 457, 461, 463
Application, 38, 507, 512, 613
application and applet, difference of-, 460, 613
application and main()-method, 38, 65, 127, 141, 142, 154
application icon, 512
Application Layer (OSI), 355
Application Programming Interface (API), 51, 55, 129, 131, 190, 249
Application Programming Interface (API) as Java software library, 54, 59
Application Programming Interface (API, definition), 43
application protocol and TCPport-number, 363
Application User Interface, 512
application with Graphical User Interface, 515
application, HelloWorld-, 71
application, inside named packages, 241, 246
application, inter-application-communication, 353
application, MIME-type, 369
application, security control by programmer, 462
application, security control by user, 457
application, starting an-, 65, 431
application, stopping an-, 432
Application, Web-Application, 406
application-development, 313
application-programmer, 27, 448
Application-Programming, 38
application-protocol, 348, 367, 378
application-protocol and servlet, 405
application-protocol and URL, 373
application-protocol and URL-class, 388
application-specific programming interface of Javabeans, 311
application-triggered-programming, 599
applications with system tasks, 423
applications, multiple in one JRE, 443, 630
applications, security-related-, 488
Archiving-Tool: jar, 325
archiving-tool: jar, 54, 242, 243, 250, 489, 492
ARGB-color-coding applied, 530
ARGB-pixel-format, 526
arguments of a method-call, 116
ArithmeticException (java.lang), 92, 280, 281
array, 234
Array (java.lang.reflect), 250
array, dimension of an -, 234
array-initializer, 235
array-type, 143, 234
array-type and class-loader, 236, 443
array-variable, 234
arraycopy()-method (java.lang.System), 236, 254, 630
ArrayIndexOutOfBoundsException (java.lang), 235, 281, 451
Arrays (java.util), 236, 238, 249
ASCII (American Standard Code for Information Interchange), 20, 33, 34, 73, 265
ASCII-format and HTTProtocol, 369
ASCII-format and policy-file, 451, 458
ASCII-format and security-files, 457
ASCII-format and security/policy-files, 455
ASCII-format and URL encoding, 375
assignment, 74

assignment by-reference, 143
assignment by-value, 83, 143
assignment operators, 99
assignment-operator (=), 99
assignment-operator (primitive- and reference-type), 144
associative, left-/right-, 101
asymmetric-key-cryptography, 479
asynchronous method, 231
atomic block of code, 214, 216
Atomic Operation, 214
AUI (Application User Interface), 512
Authentication, 477
authentication, 477, 488
AWT (Abstract Window Toolkit) versus Swing, 520
AWT (Abstract Window Toolkit), applets, 613
AWT (Abstract Window Toolkit), as API, 52, 67, 274
AWT (Abstract Window Toolkit), as package, 53, 184, 246, 251, 266, 505
AWT (Abstract Window Toolkit), class-hierarchies, 564
AWT (Abstract Window Toolkit), description, 520
AWT (Abstract Window Toolkit), event-handling, 276, 331, 440, 572
AWT (Abstract Window Toolkit), image drawing, 519, 595
AWT (Abstract Window Toolkit), image processing, 520
AWT (Abstract Window Toolkit), image processing (extensive), 527
AWT (Abstract Window Toolkit), peer-classes, 353
AWT (Abstract Window Toolkit), screen-components, 538, 547, 551
AWT (Abstract Window Toolkit), time-delayed data, 606
AWT (Abstract Window Toolkit), versus Swing, 548
AWTEvent (java.awt), 331, 555, 573
AWTEvent, class-hierarchy, extensions, 575
AWTEvent, event-dispatch-thread, 586
AWTEvent-objects, triggering dispatch, 590
AWTEventListener (java.awt.event), 586

Backus-Naur Form (BNF), 76, 655
Base-Class, 55
base-classes, 54, 247, 249, 389
base-classes (loading), 434
base-classes, peeking inside-, 243, 244
BasicPermission (java.security), 467
BasicPermission (java.security) in footnote, 467
Bean, 251, 313
Bean Class, 313
BeanInfo (java.beans), 318
Beans, 311
behavior and method (in object-orientation), 133
behavior of an object, 131, 161
big-endian order of bytes, 533
binary number-system, 34
binary operators, 93
bit, 33, 78, 79, 87
bit-length of words in the Java Virtual Machine, 216
Bit-Mask, 533
bit-mask, 287, 555, 563, 576, 579, 581, 591, 593
bit-redundancy checks, 341
Bitmap of image, 521
bitmap of image, 96, 519, 525, 530, 535
bitmap of image and screen-component, 540
bitmap of image in rendering, 535
bitmap of image, updated loading, 563
bits for monitor screen-image, 537
bits of IP-addresses, 359
bits of IP-addresses (in footnote), 377
bitwise operators, 95, 97
bitwise shift-operators, 95
bitwise-negation-operator ( ), 97
blank-final, 122, 182, 186, 190
block, 105
block, (machine-)synchronized-, 216
block, atomic-, 214, 216
block, empty-, 184, 187, 190, 581
block-cipher, 484
Blocked thread, 199, 211
Blocking Method, 129
blocking method, 231
BNF (Backus-Naur form), 76, 655
Boolean (java.lang), 169, 250, 256
boolean (primitive-type), 84, 93, 96, 97, 106, 108, 169, 253, 312
boolean algebra, 97
boolean literal, 79
boolean negation-operator (!), 93, 97, 98
BorderLayout (java.awt), 543, 558
bound property (of software-component), 315
branching, 89, 283
branching (if-statement), 105
branching (switch-statement), 106
branching (switch-statement), example, 401
branching (try-catch clause), 112
branching in Java, 105
break (Java-keyword), 106
break-keyword, 110
break-statement, 110
break-statement and try-statement, 282
breaking a cryptographic algorithm, 478
bridge, 29, 353, 354
broadcasting, 277, 343
Browser, 371
browser, 38, 197, 205, 243, 274, 324, 370, 373, 430, 520
browser and applets, 67, 68, 72, 396, 397, 431, 564, 614
browser and servlets, 406
BufferedImage (java.awt.image), 520, 530
BufferedInputStream (java.io), 304, 441
BufferedOutputStream (java.io), 304
BufferedReader (java.io), 300, 304
BufferedWriter (java.io), 304, 386
builder-tool (for software-components), 311
bus (communication-line), 29, 87, 337
bus (network topology), 340

Button (java.awt), 566
by-reference, assignment-, 143
by-reference, passing arguments-, 120, 256
by-value, assignment-, 83, 143
by-value, passing arguments-, 120, 225, 256
byte, 33, 73, 74, 78
Byte (java.lang), 169, 181, 250, 256
byte (primitive-type), 73, 85, 169
byte (primitive-type), behavior in a cast, 87, 100, 101
byte (primitive-type), byte-code as array of-, 441
byte (primitive-type), key-data as array of-, 493
byte (primitive-type), length of an array, 234
byte order, big-endian- or network-, 533
byte, 16 byte IP-address, 358
byte, 8 to 16 bytes as hashcode, 481
byte, four byte int-value, 388
byte, four byte int-value as ARGB-format, 526
byte, four byte IP-address, 357
byte, most significant-, least significant- of a given byte order, 533
byte, one byte ASCII-format, 33, 265, 374
byte, one byte unit of stream-data, 296
byte, represented by a pair of hexadecimal digits, 36
byte, two byte port-address, 231, 363
byte, two byte Unicode-format, 34
byte, two byte unit of stream-data (=char), 296, 302
Byte-Code, 65
byte-code-verifier, 281, 441, 450, 451
ByteArrayInputStream (java.io), 238
ByteArrayOutputStream (java.io), 238

CA (certification authority), 484
Calendar (java.util), 262
callback, 228, 288, 631
callback of methods, 127
callback-model, 276, 585
Canvas (java.awt), 536, 538, 597
CardLayout (java.awt), 543, 559
case (Java-keyword), 106
case of letters, 34, 66, 81, 82, 491
case of letters (in domain-names), 361
case of letters (in URIs), 375, 394
case of letters, in integer-literal of type long, 78
case-sensitive identifiers, 81
case-sensitive, not-(applet's parameter-name), 616
cast potential, testing type-, 253
cast, implicit type- (example), 544, 580
cast, narrowing type-, 169
cast, widening type-, 169
casting identifiers to an abstract class-type, 552
casting identifiers to interface-type, 194, 404
casting of variable of primitive-type, 88
casting of variable of reference-type, 169
casting, implicit type-, 86, 87, 100, 106, 169, 194, 253
Casting, Type-, 88
casting, type-, 257, 450
CD-ROM, 298
certificate, 489
Certificate (java.security.cert), 494
Certificate interface (java.security) in footnote, 494
Certificate Revocation List (CRL), 484
Certificate Signing Request (CSR), 484
certificate, a signed public key, 484
certificate, self-signed, 484
CertificateFactory (java.security.cert), 485
certificates in key-store, 488
certification authority (CA), 484
chain-model, 276, 440, 574
ChangeEvent (javax.swing.event), 547, 574, 577
ChangeListener (javax.swing.event), 547, 577
channel, virtual- (in network), 346
channel-switching, 277, 345
char (primitive-type), length of an array, 234
char (primitve-type), 85, 169
char (primitve-type), as TCPport, 385
char (primitve-type), as UDPport, 384
char (primitve-type), character-encoding, 86, 584
char (primitve-type), stream-data, 296, 302, 309
Character (java.lang), 82, 169, 257
character-literal, 73, 79
CharArrayReader (java.io), 238
CharArrayWriter (java.io), 238
charAt()-method (java.lang.String, StringBuffer), 257
check...()-method (java.lang.SecurityManager), 128, 256, 456, 462, 464, 465
check...()-methods, triggerable-, 462
checkbox, 510, 521, 548
Checked Exception, 281
checked exception, 280, 330
checked exceptions and throws-keyword, 281
checkPermission()-method (java.security. AccessController), 128, 453, 456, 465, 467
checksum, 481
child-component, 542, 543, 551
child-component, layout, 558
chime-like counting (NON-), 458
chime-like counting in random-access-filing, 299
chime-like counting of character-positions in strings, 257
chime-like counting of coordinates, 537
chime-like counting, (0,1,2,...) in footnote, 234
Cipher, 478
Cipher (javax.crypto), 497
cipher, block-, 484
ciphers, names of-, 496
class, 138
Class (java.lang), 151, 250, 252, 255, 256, 435, 437, 629
Class (java.lang) aquired for reflection, 252, 318
Class (java.lang) as class-loader, 443
Class (java.lang) its origin, 441
Class (java.lang) references ProtectionDomain, 468
Class (java.lang) references super-class, 558
Class (java.lang) references the class-loader, 435
class as program/application, 141
class as type, 73, 140
class declaration/-definition, 145

class final and constructors non-public: no instance, 255, 257, 382, 417
class locked by one of its static synchonized methods, 214
class without a super-class (java.lang.Object), 164
class, abstract- (non-instantiable class), 184
class, abstract- (non-instantiable class, concept), 134, 138
class, abstract-, and static variables, 184
class, adapter-, 187, 190, 580, 584, 601
class, adapter-, with software-components, 318
class, anonymous-, 159
class, constant-, 183
class, final- (non-extensible), 146, 183
class, final- (non-extensible) example, 257
class, inner-, 157
class, interface- (RMI), 416
class, loading of a-, 152, 433
class, local-, 157
class, member-, 157
class, nested, 157
class, public-, extended over different packages, 438
class, public-, has to be in file of same name, 146, 241, 242, 245, 246
class, runtime-representation of a-, 54, 255, 433, 441, 443, 629
class, signed- = signed class-file, 494
class, skeleton- (RMI), 416
class, static member-, 158, 182
class, stub- (RMI), 416
class, top-level-, 157
class, virtual- (non-instantiable class), 138
class, wrapper-, 169, 171
class-extension, 138
class-hierarchy, 133, 135, 138, 164, 247
class-hierarchy (AWT/Swing), 564
class-hierarchy and casting, 171
class-hierarchy and multiple extension, 186
class-literal, 252, 437
class-loader (secure), 469
class-loader delegation, 440
class-loader's name-space, 436, 469
class-loader, secure-, 454, 458, 471
Class-Loading, 433
Class-Member, 148, 166
class-member, 147, 148, 154
class-member and static-keyword, 154
class-member, static-, 140, 148, 154
class-members and interfaces, 190
class-members, accessing-, 155, 156, 171
class-method, 138, 154, 182
Class-Name, 146
Class-Structure, 135, 164
class-structure, 134, 135, 164, 653
class-structure (example), 399
class-structure and class-hierarchy, 135, 166, 194
class-structure and interface, 190
class-structure and java.lang.reflect-package, 250
class-structure and programmer, 164
class-structure as index, 142, 194
class-structure of JApplet, 615
class-type, 120, 143, 146
class-variable (static variable), 138, 148, 154, 182
class/object-model, 131
class/object-model (in programming language), 140
ClassCastException (java.lang), 171, 253, 281
classes, list of mentioned-, 653
classes.zip, 243, 244, 247, 249, 389, 434, 641
ClassLoader (java.lang), 250, 255, 435, 436
ClassLoader (java.lang) extensive, 440
ClassLoader (java.lang) referenced, 629
ClassLoader (java.lang) simple, 438
ClassNotFoundException (java.lang), 281, 436
CLASSPATH-variable (operating-system), 61, 66, 243, 254, 436, 451, 629
Client, 276
client-side computing, 406
Clonable interface (java.lang), 189
Clone, 191
clone, 192
clone()-method (java.lang.Object), 144, 165, 252
clone()-method, a factory-method, 151
clone()-method, needs Cloneable-interface, 192, 258, 637
clone()-method, unoverridden, copies shallow, 193
clone()-method, usually overridden, 168
Cloneable interface (java.lang), 192, 252, 281
CloneNotSupportedException (java.lang), 192, 281
clones are shallow equal, 162, 191, 193
cloning a variable of primitive-type, 83
cloning a variable of reference-type, 144
Cloning, Deep-, 191
Cloning, Shallow-, 191
close()-method (java.io.FilterInputStream, java.io.FilterReader), 302
close()-method (java.io.InputStream, java.io.Reader), 296, 305, 456, 493
close()-method (java.io.OutputStream, java.io.Writer), 296, 456, 493
close()-method (java.io.RandomAccessFile), 299
close()-method (java.net.Socket), 383, 385
close()-method (javax.servlet.ServletOutputStream), 409
CMY-model, 524
CMYK-model, 525
code, machine-, 37
code, object-, 38
code, P-, 39
code, Pseudo-, 39
code, Source-, 38
code, source-, 37
code, standalone- (=program), 160
codeBase in a policy-text-file, 459–461, 472
codebase of applet's byte-code, 403, 614
CodeSource (java.security), 454, 455, 465, 501
coding- and content-transformation, 35, 520
coding-transformation, 534, 535
Collection (java.util), 237
Color (java.awt), 550, 556
Color and transparency, 527
color concept, 522

color-cube, 523
color-model, 522
color-model, CMY-, 524
color-model, CMYK-, 525
color-model, direct-, 526
color-model, HSV and HSB, 525
color-model, HSV-, ..., 525
color-model, index-, 526
color-model, indirect-, 526
color-model, RGB-, 523
color-space, 523
ColorModel (java.awt.image), 520, 556, 596
Command-Line, 44
command-line, 44, 250, 324, 400, 407, 430, 431, 496
command-line of browser, 68, 373, 614
comment in source-code, 37, 71, 323
communication infrastructure, three-tiered-, 277, 352, 406, 414
communication suggesting concurrency, 231
Comparable interface (java.lang), 189
compareTo()-method (java.lang.Comparable) abstract, 189
comparison operators (==, ! =), 96
Compilation Unit, 246
compileClass()-method (java.lang.Compiler), 255
Compiler, 38, 39
compiler (byte-code to native code), 255
Compiler (java.lang), 250, 255
compiler, cross-, 45
compiler, Java-, 52, 53, 124, 184, 257, 280, 431
compiler, Java-, and -generated names, 83
compiler, Java-, and character-encoding, 71, 79
compiler, Java-, and comparison-operator, 163
compiler, Java-, and compilation unit, 246
compiler, Java-, and nested classes, 159
compiler, Java-, and security, 450
compiler, Java-, and unreachable statements, 106
compiler, Java-, import-on-demand, 244
compiler, Java-, in JDK, 54
compiler, Java-, late-binding, 168, 183
compiler, Java-, letter-case sensitive, 66
compiler, Java-, searching files, 63, 65
compiler, rmic- (RMI), 415
Compiler-tool: javac, 72, 431
compiler-tool: javac, 60, 246, 333, 397, 407, 473
Component (java.awt), 538, 551, 559, 563
Component as drawing surface, 528
Component as source of event-objects, 577, 591
Component method isLightweight(), 550
Component methods, 553
Component's paint()-method, 595
Component, as source of event-objects, 331, 573
Component, class hierarchy, 552, 565
component, screen-, 331, 399, 509, 519
component, screen- as a painting-surface, 538
component-model, 138, 311, 381, 542
ComponentEvent (java.awt.event), 573, 575, 577, 598
ComponentListener (java.awt.event), 577
compression, 240, 325, 368
Compression, destructive-, 522
Compression, non-destructive-, 522
compression, RLE-, 522
compression, ZIP-, 326, 522
compression-algorithm, 534
Computer, 17
computer-model, 19, 23
computer-platform, 44
computer-system, 29
computing, distributed-, 423
concatenation, string-, 95
conditional compilation, 106
conditional-operator (?), 99
connection, logical-, 343
connection, multipoint-, 343
connection, point-to-point-, 277, 343
connection-oriented network, 345
Connection-Oriented Protocol, 350
connection-oriented protocol, 275, 351
connectionless network, 345
connectionless-protocol, 350
constant (final variable in interfaces), 180, 187
constant (final variable), 83, 85, 86, 106, 122, 182, 256, 450, 497, 563, 577, 581, 584
constant class, 183
constant object, 183
constant, String-, 383
constrained property (of software-component), 315
constructor, 145, 165, 642
Constructor (java.lang.reflect), 250, 253
constructor (object-orientation), 134, 138
constructor of core-class and security behavior, 463
constructor without arguments, 642
constructor, declaration, 149
constructor, default-, 149, 166
constructor, default-accessible-, 181
constructor, not a class-member, 148, 166
constructor, overloaded-, 151, 174
constructor, private-, 150, 181, 182, 382, 417
constructor, private-, and singleton, 160
constructor, protected-, 151, 181
constructor, public-, 180
constructor, super()- (class-extension), 166
constructor, this()-, 167
constructor-call, 145, 151
constructor-call chaining (class-extension), 166
constructor-call sequence, 152
constructor-call, anonymous-, 159
constructor-callback, 631
constructors and factory-methods, 151
constructors non-public and class final: no instance, 255, 257, 382, 417
Container (java.awt), 551, 556, 559
container and multiple layouts, 559, 566
Container as content-pane, 536, 538
Container as content-pane (Swing), 567
Container as drawing surface, 528
Container as layered-pane (Swing), 567
Container as source of event-objects, 578

container of screen-components, 512, 542, 551, 556, 576, 598
Container's paint()-method, 595
container, applet- (=applet-sandbox), 398
Container, class hierarchy, 556, 564, 565
container, fifo-, 293
container, servlet- (=servlet-sandbox), 406, 407, 409, 430
container-file (=jar-file), 491
container-object, 138
ContainerEvent (java.awt.event), 576, 578, 598
ContainerListener (java.awt.event), 578
containment-hierarchy (screen-components), 440, 541, 543
content- and coding-transformation, 35, 520
Content-Handler, 369
content-stream, 389, 394
content-type, 368, 405
content-type and encoding, 368
content-type and portability, 265
ContentHandler (java.net), 394
ContentHandlerFactory (java.net), 394
context-free-/generic event-object, 572, 573
context-independent-event, 575
contextual event, 575
continue-keyword, 110, 112
continue-statement, 110
Control Structure, Statement and Expression (concept), 89
Controller and MVC, 515
conventions, 83, 140
Cookies, 351, 369, 413
coordinate-system for colors, 525
coordinate-system for graphics-objects, 519, 534, 536
coordinate-system of devices (pixel-oriented), 536
coordinates, generic-, 536
coordinates, pixel-, 536, 542
copy — deep copy of an object-graph, 162
copying a variable of primitive-type, 83
copying a variable of reference-type, 144
Core-Class, 55
core-classes, 54, 247, 249, 389
core-classes (loading), 434
core-classes, peeking inside-, 243, 244
country-code top-level-domains (DNS), 361
CPU (Central Processing Unit), 28
cross-compiler, 45
cryptographic algorithm, 478, 480
cryptographic algorithm, strength or quality of an-, 479
cryptographic algorithms, implementations of-, 485
cryptographic-engine, 486
cryptography, 477
cryptography with key-pairs or asymmetric keys, 479
cryptography with secret- or symmetric keys, 478
Current Directory, 48
currentThread()-method (java.lang.Thread), 207, 212, 232, 401
currentTimeMillis()-method (java.lang.System), 254, 261
cursor (GUI-pointer), 509, 536, 542, 556, 605

Daemon-thread, 223
daemon-thread, 215, 432, 586
Data Link Layer (OSI), 354
Database, 321
database, 20, 40, 138, 193, 235, 246
database and resource-bundle, 263
database, distributed, 360
database, file-system as, 46
database, key-store-, 489
database, non-persistent-, 233
Datagram, 351
DatagramPacket (java.net), 384
DatagramSocket (java.net), 384, 421
DataInput (java.io) interface, 300, 307
DataInputStream (java.io), 382
DataOutput (java.io) interface, 300, 307
DataOutputStream (java.io), 382
Date (java.util), 261
deadlock (thread-specific problem), 226
decimal number-system, 34, 79, 257
declaration, 74, 84
declaration-assignment-sequence, 122, 155
decrement operators (pre-, post-), 92
Deep Cloning, 191
deep copy of an object-graph, 162
deep equal object-graphs, 162, 192
default access, 177
default assignment, 86, 87
default assignment of member-variables: YES, 148
default assignment with local variables: NONE, 121
default constructor, 149, 166
default type of number literals: int (primitve-type), 86
default-accessible constructor, 181
default-keyword (switch-statement), 107
default-package, 180
default-package (idea), 241
default-package, in footnote, 193
Delegation, 440
delegation event-model, 19, 276, 314, 440, 571, 573, 576, 585
delegation event-model (Concept in Java), 278
delegation, class-loader-, 440
design-pattern, 251, 311, 312
design-pattern (event-framework), 287
desktop, 44, 509
desktop in Swing, 610
destroy()-method (java.applet.Applet), 205, 398, 400, 618, 641
destroy()-method (java.lang.Process), 255
destroy()-method, triggerable-, 398
development-environment, 64, 65, 311, 313, 318, 450
Dialog (java.awt), 565
Dialog (java.awt) top-level screen-component, 564, 598

dialog window, 540
dialog, modal-, 564
difference of applet and application, 460
digest, message-, 481
digital fingerprint, 481
digital signature, 483
digital signature and certification, 484
digital signatures, names of algorithms for-, 483
Dimension (java.awt), 555
dimension of an array, 234
dimension, length of a - (array), 234
dining humans (thread-specific problem), 215
direct color-model, 526
DirectColorModel (java.awt.image), 526
directory-/folder-/file-separator, 46, 254
directory-/folder-/file-separator (in packaging), 242
directory-/folder-/file-separator (policy-files), 458, 462
directory-/folder-/file-separator (URL), 364, 388
disableEvents()-method (java.awt.Component), 287
disassembler-tool: javap, 250, 642
Discrimination (security-related), 449
dispatchEvent()-method (custom-defined), 287, 288, 331
dispatchEvent()-method (java.awt.Component), 592
dispatchEvent()-method (java.awt.Component, ...), 591
disposing of a top-level screen-component, 598
Distinguished Name (DN), 481, 488, 489
Distributed Computing, 338
distributed computing, 423
distributed computing (example), 362
distributed database, 360
distributed garbage-collector, 251, 415
distributed programming, 419, 423
dithering, 535
division by zero with integers: ArithmeticException, 86
division-operator, 93
DN (Distinguished Name), 481, 488, 489
DNS (Domain Name Service), 348, 354, 359, 362, 381
do-statement (usually a pre-condition loop), 109
documentation-tool: javadoc, 324
Domain Name Service (DNS), 348, 354, 359, 362, 381
Domain-Name, 362
domain-name (case of letters), 361
dots per inch (dpi) resolution, 536, 537
Double (java.lang), 86, 169, 181, 194, 250, 256
Double (java.lang) constants (NaN), 256
double (primitive-type) and JVM, 87, 216
double (primitve-type), 169
double (primitve-type) and equality operator, 97
double (primitve-type) and strictfp modifier, 118, 146, 183
double (primitve-type), argument in mathematical functions, 94
double (primitve-type), behavior in a cast, 169
double (primitve-type), default type of floating-point-literals, 79, 86
double (primitve-type), return-value of mathematical functions, 106
double (primitve-type), value of a StreamTokenizer, 301
double buffering (= off-screen rendering), 535, 542, 621
double-click (GUI concept), 510
downloading a Java Runtime System (client FROM server), 59
Downloading files (client FROM server), 371
downloading files (client FROM server), 19, 372, 374
downloading files (client FROM server) and distributed computing, 338
downloading files (client FROM server) and monitoring, 606
downloading files (client FROM server) and security, 57
downloading mail (client FROM server), 371
downloading package-files (client FROM server), 243
downloading the Java Cryptography Extension (client FROM server), 496
downloading the Java Servlet Web Development Kit (client FROM server), 409
dpi (dots per inch) resolution, 536, 537
drag and drop (GUI concept), 510
dragging (GUI-concept), 510
drain of message-model, 271
drawing-method, 536
drawing-surface, 528, 536
drawing-surface, a JComponent, 528, 530, 531, 538, 602
drawing-surface, a JPanel, 608
drawing-surfaces, 538
driver (system-software), 38, 43, 45, 273, 378, 508, 512, 516, 522, 572
drivers and operating-system, 508
duplicating objects, 191
dynamically-typed language, 75

Early-Bound Method, 168
editor, hex-, 449, 489
editor, hex-, see Java byte-code, 642
editor, text-, for editing security- and policy-files, 453, 457, 458, 496
editor, text-, for writing source-code, 53, 54, 57, 62, 63, 65, 323
email-client, 371
empty statement, 103
empty-block, 184, 187, 190, 255, 581
enableEvents()-method (java.awt.Component), 287, 579, 591, 593
encapsulation, 115, 122, 138, 139, 377
encapsulation and non-local variables, 195
Encoding, 35
encoding, 40, 73
encoding and content-type, 368
encoding of image-information, 521
encoding transformation with images, 535
encoding, ASCII-, 34
encoding, Unicode-, 34

encoding, URL-, 375
encodings, 265
Encryption, 477
encryption algorithm, 478, 480
encryption algorithms, implementations of-, 485
encryption algorithms, modes of-, 484
encryption algorithms, names of-, 479
encryption-engine, 478
endian-order of bytes, little-/big-, 533
Engine, cryptographic-, 486
engine, encryption-, 478
environment, development-, 64, 65, 311, 313, 318, 450
equal — deep equal object-graphs, 162, 192
equal — shallow equal object-graphs, 162, 192
equal sign (=), assignment-operator, 99
equal sign (=), copying or referencing, 143
equal sign (=), tool option, 461
equality operator (==), 161–163
equality operator (==) and reference-type, 162
equality operator (==) with float and double, 97
equals()-method (java.lang.Object), 165, 190, 252, 256
equals()-method (java.lang.String), 162
Error (java.lang), 281
escape-sequence, 80
Ethernet (network hardware), 20, 29, 30, 190, 339–341, 348, 354, 360
Ethernet (network hardware) -address, 342
Ethernet (network hardware) installation hint, 378
event, context-independent-, 575
event, contextual-, 575
event-delegation model (Concept in Java), 278
Event-Dispatch-thread, 585
event-dispatch-thread, 205, 228
event-dispatch-thread, and painting-methods, 595
event-dispatch-thread, running code on the-, 589
event-example (footnote), 275
event-handler, 287, 575, 577, 585
event-mask, 287, 555, 576, 579, 581, 591, 593
event-messages (message-model), 272
event-messaging, unicast-, 287, 289, 633
event-model, 278, 329
event-model, delegation-, 19, 276, 314, 440, 571, 573, 576, 585
event-model, subscriber-, 276
event-queue, 586
event-scheduling, 572, 574
event-type-hierarchy, 578
EventListener interface (java.util), 189, 287, 288, 314, 331, 332, 573
EventListenerList (javax.swing.event), 579
EventObject (java.util), 250, 276, 287, 288, 331, 585, 633
EventQueue (java.awt), 332, 571, 586, 587
EventQueue reference, 586
EventQueue-object, 595
Exception, 113
Exception (java.lang), 280, 281, 283, 330, 637
Exception, Checked-, 281
exception, checked-, 280, 330
exception-handler, 283
exec()-method (java.lang.Process), 255
exec()-method (java.lang.Runtime), 254
exit()-method (java.lang.Runtime), 254
exit()-method (java.lang.System), 254, 432, 581, 651
exit()-method (java.lang.System), example, 226, 382
exp()-method (java.lang.Math), 94
expansion, property- with text-files, 458, 459
explicit type-casting, 100
exponential function (mathematical function), 94
expression, 90
Expression, Statement and Control Structure (concept), 89
ext-folder, 61, 243, 244, 247, 250, 410, 429, 487, 496, 499, 641
extended file-name, 47
extending access-area by overriding, 175, 286
extends-keyword, 146, 187
extends-keyword (example), 163, 186
eXtensible Markup Language (XML), 410, 414
extension, class-, 138
extension-hierarchy, 135

Factory-Like-Method, 185, 551
factory-like-method, 56, 266, 527, 551, 569, 596, 641, 642
Factory-Method, 151
factory-method, 145, 185, 273, 381, 392, 485, 629, 642
factory-method (static), 151
factory-method (static) in footnote, 382
factory-methods and constructors, 151
Field, 148
Field (java.lang.reflect), 250, 253
field, final- (quasi-constant), 148
field, final-, example, 160
field, transient- (non-serializable), 183, 309
field, volatile- (uncached), 148, 183
field-variable, 155
fifo-container, 293
fifo-container (stream), 310
fifo-storage (first-in-first-out), 233
File (java.io), 259, 463
File Transfer Protocol (FTP), 349, 354, 371, 374, 389
file, container- (=jar-file), 491
file-/folder-/directory-separator, 46, 254
file-/folder-/directory-separator (in packaging), 242
file-/folder-/directory-separator (policy-files), 458, 462
file-/folder-/directory-separator (URL), 364, 388
file-hierarchy, 510
file-name, extended-, 47
File-System, 46
file-system as database, 46
file-system, flat-, 46
FileInputStream (java.io), 297, 298
FileInputStream (java.io) security, 463
FileOutputStream (java.io), 297, 298

FilePermission (java.io), 459–461, 463, 472
FileReader (java.io), 298, 456, 463
FileReader (java.io) security, 453
FileWriter (java.io), 298, 456
filter-stream, 292
FilteredImageSource (java.awt.image), 610
final class (non-extensible), 146, 183
final class (non-extensible) example, 257
final field (quasi-constant), 148
final field, example, 160
final local variable, 122
final method called faster, 118, 168, 182
final method, unoverridable, 168, 182
final modifier, 117, 182
final variable (quasi-constant), 182
final, implicitly - variables in interfaces, 186
finalize()-method (java.lang.Object), 127, 152, 165, 232, 252, 415
finalize()-method and destroy()-method, 398
finalize()-method with own thread, 228, 232
finalize()-method, triggerable-, 127, 153
finalize-thread, 228
finalizer, not a class-member, 148, 166
finally-keyword, 113, 282, 330
fingerprint, digital-, 481
fire...()-method (custom-defined), 288, 331
fire...()-method (Java beans, Swing, ...), 287, 312, 314, 331, 574, 590
fireActionPerformed()-method (javax.swing. AbstractButton), 574
firePropertyChange()-method (java.awt.Component), 315, 319, 331, 578, 603
fireStateChanged()-method (javax.swing.AbstractButton, ...), 574, 577
fireVetoableChange()-method (javax.swing.JComponent, java.beans.VetoableChangeSupport), 315, 331
flat file-system, 46
Float (java.lang), 86, 169, 181, 250, 256
Float (java.lang) constants (NaN), 256
float (primitve-type), 86, 169
float (primitve-type) and equality operator, 97
float (primitve-type) and strictfp modifier, 118, 146, 183
float (primitve-type) as color-value, 556
float (primitve-type), behavior in a cast, 101, 169
float (primitve-type), postfixing literals, 100
floating-point-literal, 78
FlowLayout (java.awt), 543, 558
FlowLayout, default layout of many containers, 559
focus, concept of-, 509
focus, screen-component on-, 542, 552, 575, 577
focus-cycle, 509
FocusEvent (java.awt.event), 575, 577
FocusListener (java.awt.event), 577
folder-/file-/directory-separator, 46, 254
folder-/file-/directory-separator (in packaging), 242
folder-/file-/directory-separator (policy-files), 458, 462
folder-/file-/directory-separator (URL), 364, 388
font, 34, 265
Font (java.awt), 569
fonts, Java and, 266, 569
for-statement (usually post-condition loops), 109
Format, 35
format, 521
format, text-based, 414
forName()-method (java.lang.Class), 185, 252, 318, 394, 435, 437, 443, 468, 619, 629, 642
FP-strict, 118
Frame (java.awt), 562
Frame (java.awt) top-level screen-component, 564, 598
frame (message-model), 272
Frame (of packet-oriented protocol), 349
frame or image (of a movie, animation-context), 602
frame or packet (networking-context), 349
frame or window (as modal screen-component), 540
frame or window (GUI container), 512, 573
frame or window (GUI-context), 509
frame or window (size-change), 573
frame- or window-states (concept), 509
Framework, 237
framework, 287
freeMemory()-method (java.lang.Runtime), 254, 258, 445
frequency, 31
front-end, 507
front-end and kernel, 515
front-end, a Java desktop, 610
front-end/kernel model, 515, 516, 542
FTP, 364
FTP (File Transfer Protocol), 349, 354, 371, 374, 389
fully-qualified class-name, 244
Function, Hash-, 482
Functions, One-Way-, 480

gamma-correction (screen-hardware related), 523
garbage-collection (Concept), 444
garbage-collection (distributed), 415
garbage-collection-thread, 223, 228, 444
garbage-collector, 153, 223, 254, 415, 450, 618
garbage-collector (distributed), 251
gateway, 353
GB (GigaBytes) more than a billion bytes, 35
gc()-method (java.lang.Runtime), 254
gc()-method (java.lang.System), 153, 223, 254, 445, 598
generic coordinates, 536
generic top-level-domains (DNS), 361
generic-/context-free event-object, 572, 573
get...()-method, 312, 313, 318
getByName()-method (java.net.InetAddress), 151, 382, 385
getClass()-method (java.lang.Object), 245, 252, 318, 437, 630, 633, 638
getClassLoader()-method (java.lang.Class), 435, 437, 439, 468, 629

getContentPane()-method (javax.swing.JRootPane), 528, 536, 567
getDefaultToolkit()-method (java.awt.Toolkit), 185, 528, 579, 596, 604, 608
getDefaultToolkit()-method (java.awt.Toolkit, Component), 551
getErrorStream()-method (java.lang.Process), 255
getImage()-method (java.awt.Toolkit, java.applet.Applet, ...), 232, 528, 529, 596, 608
getInputStream()-method (java.lang.Process), 255
getInputStream()-method (java.net.Socket), 297, 302, 383, 385, 386, 395
getInputStream()-method (java.net.URLConnection), 393, 395
getInputStream()-method (javax.servlet.ServletRequest), 412
getInstance()-method (custom-defined), 160
getInstance()-method (java.security.KeyFactory), 494
getInstance()-method (java.security.KeyPairGenerator), 493
getInstance()-method (java.security.KeyStore), 495
getInstance()-method (java.security.SecureRandom), 493
getInstance()-method (java.security.Signature), 493
getInstance()-method (java.util.Calendar), 262
getInstance()-method (javax.crypto.Cipher), 497
getInstance()-method (javax.crypto.KeyGenerator), 497
getInstance()-method (some engine-class), 485, 487
getMessage()-method (java.lang.Throwable), 280
getMessage()-method (java.lang.Throwable, ...), 282
getName()-method (java.io.File), 259
getName()-method (java.lang.Class), 630
getName()-method (java.lang.Thread), 211
getOutputStream()-method (java.lang.Process), 255
getOutputStream()-method (java.net.Socket), 297, 302, 382, 385, 386, 395
getOutputStream()-method (java.net.URLConnection), 393
getOutputStream()-method (javax.servlet. ServletResponse), 409, 412
getPackage()-method (java.lang.Package), 245
getPriority()-method (java.lang.Thread), 211, 224, 232
getProperties()-method (java.awt.Image), 607
getProperties()-method (java.lang.System), 254, 258, 556
getProperty()-method (java.awt.Image), 608
getProperty()-method (java.lang.System), 254, 259, 456
getRuntime()-method (java.lang.Runtime), 254, 258, 445
getSecurityManager()-method (java.lang.System), 254, 255
getSource()-method (java.awt.Image), 607
getSource()-method (java.util.EventObject), 288, 315, 545, 546, 611
GIF (Graphics Interchange Format), 369, 522, 529
GigaBytes (GB) more than a billion bytes, 35
glyph of a font, 34, 82, 265
graph of a network, 340
graph of references, 161, 162, 191, 233, 307, 441, 444
graph of references (cloning), 191
graph, processing-, 520
Graphical User Interface (GUI), 44, 190, 507, 508, 520
Graphical User Interface and applets, 615
Graphical User Interface and applications, 512
Graphical User Interface and Java-Beans, 313
Graphical User Interface and locale, 263
Graphical User Interface and MVC, 515
Graphical User Interface and operating-system, 508
Graphical User Interface and policytool, 458
Graphical User Interface, design of, 513
graphical-object, 507, 519
graphical-object, concept, 534
Graphics (java.awt), 400
Graphics (java.awt) as software-library, 505, 519, 534, 536, 540, 556
Graphics (java.awt) typed reference to native-code, 185, 527, 556
Graphics Interchange Format (GIF), 369, 522, 529
graphics toolbox, 538
graphics toolbox/-context/-library, 519, 536, 540, 599
graphics-card (hardware), 522
graphics-card, active- (hardware), 198
graphics-library, 538
graphics-object, 505, 597
GraphicsEnvironment (java.awt), 266, 551, 569
GregorianCalendar (java.util), 262
GridBagLayout (java.awt), 543, 559
GridLayout (java.awt), 543, 558, 569
GuardedObject (java.security), 474
GUI (Graphical User Interface), 44, 190, 507, 508, 520
GUI and applets, 615
GUI and applications, 512
GUI and Java-Beans, 313
GUI and locale, 263
GUI and MVC, 515
GUI and operating-system, 508
GUI and policytool, 458
GUI, design of, 513

Handler, Content-, 369
handler, exception-, 283
Handler, Protocol-, 349
Handler-code, 113
handler-method, 113
hard-disk (hard-drive), 17
hard-disk (hard-drive) and persistent storage, 28, 30, 46, 238, 306, 311, 321, 433, 447, 510
Hardware, 36
Hash-Function, 482
Hashcode, 481
hashcode, 319
hashCode()-method (java.lang.Object), 161, 165, 191, 252, 256, 482

hashCode()-method and identity of an object, 131, 161, 419
hashcode-value of type int, 161
hashcode-value: 10 bytes, 483
hashtable, 233, 237
Hashtable (java.util), 254, 486, 634
heavyweight screen-component, 559, 562, 564, 566, 595, 597
heavyweight screen-component (concept), 353, 548
Heavyweight-Code, 550
HelloWorld-application, 71
Hertz (unit of frequency), 31
hex-editor, 449, 489
hex-editor, see Java byte-code, 642
hexadecimal number-system, 34, 100
hexadecimally coded, 36, 78, 80, 87, 96, 100, 342, 375, 556
HFS (Hierarchical File System), scope and access-area, 172
hierarchical file system, 298
Hierarchical File System (HFS), scope and access-area, 172
hierarchical structures, their importance, 543
hierarchy of a domain-name, 361
hierarchy of classes, 133, 135, 164, 247
hierarchy of classes (AWT/Swing), 564
hierarchy of containment (screen-components), 440, 541, 543
hierarchy of event-types, 578
hierarchy of files, 247, 510
hierarchy of packages, non-existent, 240
hierarchy of sub-types of Throwable, 281
hierarchy, class-, 138
hierarchy, domain-name, 361
hierarchy, of exceptions, 113
host of a network, 339
host-application (of an applet), 613
Host-Name, 339
HSB-model, 525
HSV-like color-models, 525
HSV-model, 525
HTML (HyperText Markup Language), 414
HTML-tags, 21, 324
HTML-tags (applet), 68, 402, 431, 614
HTML-tags (servlet), 406, 414
HTTP, 364
HTTP (HyperText Transfer Protocol), 369
HTTP (Hypertext Transfer Protocol), 348, 349, 354, 363, 373, 413
http://webopedia.internet.com, 660
hyperlink, 371
hypertext, 371
HyperText Markup Language (HTML), 414
HyperText Transfer Protocol (HTTP), 369
Hypertext Transfer Protocol (HTTP), 348, 349, 354, 363, 373, 413

icon, 44, 508
IDE (Integrated Development Environment), 64, 313, 323, 411
identifier, 81
identifier, (variable-), 40
identifier, access-area of-, 172
identifier, ambiguous-, 187
identifier, case-sensitivity, 81
identifier, scope of-, 122, 171
identifier, variable-, 74
identity of a person, 477, 479, 481–483
identity of an object, 131, 161
identity of an object and hashCode()-method, 131, 161, 419
identity of key-pair-owner, 481
IEEE 754 Standard (for floating-point-numbers), 85
if-statement (branching), 105
IllegalAccessError (java.lang), 281
IllegalAccessException (java.lang), 281
Image (java.awt), 527, 530, 556
Image typed reference to native code, 184, 528
Image, manipulation, 596, 607
ImageConsumer (java.awt.image), 563, 610
ImageObserver (java.awt.image), 607
ImageObserver interface (java.awt.image), 563
ImageProducer (java.awt.image), 610
imageUpdate()-method (java.awt.image.ImageObserver), 530, 563, 607, 608
imageUpdate()-method, triggerable-, 563
implements-keyword, 146, 187
implements-keyword (example), 186
implements-keyword and multiple super-interfaces, 188
import-keyword, 244
import-keyword, invisible-, 244
Inactive (=Interrupted) thread, 200
increment operators (pre-, post-), 92
incremental development (class-extension), 164
incremental development (trying and retrying), 599, 637
index color-model, 526
IndexColorModel (java.awt.image), 526
IndexOutOfBoundsException (java.lang), 281
indirect color-model, 526
inequality operator (! =), 96
InetAddress (java.net), 151, 381, 421
information-hiding, 122, 129, 377
infrastructure, three-tiered communication-, 277, 352, 406, 414
infrastructure-programmer, 27
Infrastructure-Programming, 38
infrastructure-programming (security), 455, 486, 497
infrastructure-programming (switchboard example), 633
inheritance, 139
inheritance, multiple-, 139
init()-method (java.applet.Applet), 205, 398, 400, 618, 619, 641
init()-method (javax.crypto.Cipher), 497
init()-method (javax.servlet.GenericServlet), 411, 641
init()-method, triggerable-, 398
Initialized thread, 211
initializer, array-, 235

initializer, static- and instance- (member sequence matters), 122
initializer, static-/instance-, 152
inner class, 157
input-stream, 296
InputEvent (java.awt.event), 579, 581, 582
InputStream (java.io), 254, 310, 385, 429
InputStream (java.io) abstract, 296
InputStreamReader (java.io), 304
insert()-method (java.lang.StringBuffer), 257
Inset (java.awt), 558
instance, 139
instance and object and variable, 149
instance-initializer, 152, 441, 641
instance-initializer and blank-finals, 182
instance-initializer, not a class-member, 148, 166
instance-method, 139, 182
instance-variable, 139, 148, 182
instance-variable, unshadowing an-, 172
instanceof-keyword, 191, 498
instantiation, 74
int (primitive-type), 85, 95, 169, 642
int (primitive-type), argument in switch-statement, 106
int (primitive-type), behavior in a cast, 87
int (primitive-type), bitmask, 563, 581
int (primitive-type), buffer-size, 296
int (primitive-type), coding-constants, 563, 575, 584
int (primitive-type), default type of number literals, 86
int (primitive-type), identity-tag of objects, 161
int (primitive-type), JRE-exit-status, 254, 432
int (primitive-type), pixel-data, 526
int (primitive-type), priority of thread, 211
int (primitive-type), stream-data, 296
int (primitive-type), TCPport, 388
int (primitve-type) as color-value, 556
Integer (java.lang), 169, 181, 252, 256
integer-literals, 77
Integrated Development Environment (IDE), 64, 313, 323, 411
Inter Process Communication (IPC), 230, 231
inter-application communication, 353
Interface, 190
interface, peer-, 59, 251
interface-class (RMI), 416
interface-type, 143, 185
interface-type, arguments of-, 190
interface-type, type-casting identifiers to-, 194, 404
interfaces and class-members, 190
interfaces, list of mentioned-, 653
InternalError (java.lang), 281
internationalization, 259
Internet, 344
Internet (definition), 372
Internet Protocol (IP), 344, 348, 349, 354, 357, 359, 372, 377
Internet Protocol Version 4 (IPv4), 357
Internet Protocol Version 6 (IPv6), 357
Internet, its defining protocol-stack, 348
Internet-Address, 359
Internet-address (=IP-address), 273, 358
Interpreter, 39
Interrupt, 212
interrupt()-method (java.lang.Thread), 204, 212, 213, 232
Interrupted (=Inactive) thread, 200
interrupted()-method (java.lang.Thread), 212, 232
intialized by assignment: local variable of primitive-type, 121
intialized to default value: member-variable of primitive-type, 148
Intranet, 344, 372, 377
introspection, 629
introspection (software-components), 311, 318
introspection: applet, 402
introspection: package, 245
introspection: servlet, 411
invisible extends-Object-clause, 164, 169
invisible file (in Unix-like operating-systems), 63, 458, 469, 492
invisible import-java.lang.-clause, 244
invokeAndWait()-method (javax.swing.SwingUtilities), 589
invokeLater()-method (javax.swing.SwingUtilities), 589
IOException (java.io), 281, 318, 330
IP (Internet Protocol), 344, 348, 349, 354, 357, 359, 372, 377
IP-Address, 359
IP-address, 339, 357, 360, 381, 410, 421
IP-address (assignment organization), 359
IP-mask, 358, 378
IPC (Inter Process Communication), 230, 231
IPv4 (Internet Protocol Version 4), 357
IPv6 (Internet Protocol Version 6), 357
is...()-method, 312, 313
isAlive()-method (java.lang.Thread), 211, 218, 226, 227, 232
isArray()-method (java.lang.Class), 253
isAssignableFrom()-method (java.lang.Class), 253
isDaemon()-method (java.lang.Thread), 224, 232
isEventDispatchThread()-method (javax.swing.SwingUtilities), 538, 586, 588, 600
isInstance()-method (java.lang.Class), 253
isInterface()-method (java.lang.Class), 253
isInterrupted()-method (java.lang.Thread), 212, 232
isPrimitive()-method (java.lang.Class), 253
ItemEvent (java.awt.event), 547, 575, 578
ItemListener (java.awt.event), 547, 578
iteration, 108

J2EE (Java 2 Enterprise Edition), 18, 20, 277, 319, 352, 407, 414, 501
J2ME (Java 2 Micro Edition), 18, 20
J2SE (Java 2 Standard Edition), 18, 20, 381
JApplet (javax.swing), 68, 396, 397
JApplet (javax.swing) heavyweight, 549, 550, 562, 565, 597
JApplet as AWT-component, 618

JApplet is a RootPaneContainer, 566
jar, Archiving-Tool, 325
jar, archiving-tool, 54, 242, 243, 250, 489, 492
JAR- (Java ARchive) file-format, 326
JAR-File, 326
JAR-file, 54, 246, 326
JAR-file (and applet), 403, 614
JAR-file (and directory-path), 247
JAR-file (and ext-directory), 64, 179, 243, 245, 487
JAR-file (and package), 240
JAR-file (and Provider Infrastructure), 487
JAR-file (and running applications), 327
JAR-file (signed), 491
JAR-file (with prototypes of Java source-code), 645
JAR-file, signed-, 491
jarsigner, signing-tool, 489, 492
Java, 51, 71
Java 2 Enterprise Edition (J2EE), 18, 20, 277, 319, 352, 407, 414, 501
Java 2 Micro Edition (J2ME), 18, 20
Java 2 Standard Edition (J2SE), 18, 20, 381
Java application, 38, 51, 613
Java ARchive (JAR) file-format, 326
Java compiler, 52, 53, 124, 184, 257, 280, 431
Java compiler and -generated names, 83
Java compiler and character-encoding, 71, 79
Java compiler and comparison-operator, 163
Java compiler and compilation unit, 246
Java compiler and nested classes, 159
Java compiler and security, 450
Java compiler and unreachable statements, 106
Java compiler searching files, 63, 65
Java compiler, import-on-demand, 244
Java compiler, in JDK, 54
Java compiler, late-binding, 168, 183
Java compiler, letter-case sensitive, 66
Java DataBase Connectivity (JDBC), 321
Java Development Kit (JDK), 54
Java Development Kit (JDK), installing-, 59
Java Foundation Classes (JFC), 505, 623
Java keywords, 81
Java Runtime Environment (JRE), 51, 52, 250, 431
Java Runtime Environment (JRE) and abstract references accessing native code, 184, 551
Java Runtime Environment (JRE) and applets, 397, 613
Java Runtime Environment (JRE) and application, 72
Java Runtime Environment (JRE) and daemon-threads, 223
Java Runtime Environment (JRE) and Java Runtime System (JRS), 55
Java Runtime Environment (JRE) and security, 128, 448
Java Runtime Environment (JRE) and servlets, 406
Java Runtime Environment (JRE) and threading portability, 204
Java Runtime Environment (JRE) and triggerable methods, 128
Java Runtime Environment (JRE) as sandbox, 463
Java Runtime Environment (JRE) communication through streams, 297
Java Runtime Environment (JRE) DEFINITION, 55
Java Runtime Environment (JRE) extended by packages, 240
Java Runtime Environment (JRE), class-loading, 433
Java Runtime Environment (JRE), communication by method-calls, 414
Java Runtime Environment (JRE), communication with event-objects, 571
Java Runtime Environment (JRE), errors and exceptions, 279, 329
Java Runtime Environment (JRE), garbage-collection, 444
Java Runtime Environment (JRE), introspection, 254
Java Runtime Environment (JRE), multiple- on one native platform, 231
Java Runtime Environment (JRE), tools for single user interfacing, 505
Java Runtime Environment, Creation-tool: java, 51, 431
Java Runtime Environment, creation-tool: java, 54, 55, 64, 65, 72, 231, 236, 246, 434, 435, 451, 457, 461, 465, 473, 505
Java Runtime System (JRS), 51, 53, 54, 65, 249
Java Runtime System (JRS) and Java Runtime Environment (JRE), 55
Java Runtime System (JRS) DEFINITION, 56
Java Virtual Machine, 40
Java Virtual Machine (JVM), 36, 40, 52, 54, 55, 444
Java Virtual Machine (JVM) and threads, 203, 216, 232
Java Virtual Machine (JVM), 32bit bus, 87
Java Virtual Machine (JVM), How to start?, 431, 641
java, tool for Creating a Java Runtime Environment, 51, 431
java, tool for creating a Java Runtime Environment, 54, 55, 64, 65, 72, 231, 236, 246, 434, 435, 451, 457, 461, 465, 473, 505
Java-Bean, 251, 313
Java-Beans, 311
java-tool as trigger for a method-call, 127
java.beans, switchboarding, 314
java.lang package automatically imported, 244
java.policy (text-file), 458, 462, 467, 469, 501
java.security (text-file), 455, 457, 458, 460, 462, 465, 467, 487, 489, 501
javac, Compiler-tool, 72, 431
javac, compiler-tool, 60, 246, 333, 397, 407, 473
javadoc, documentation-tool, 324
javap, disassembler-tool, 250, 642
JavaScript, 338

JButton (javax.swing), 516, 544, 576, 580, 621
JComponent (javax.swing), 536, 538, 551
JComponent (javax.swing) as super-class, 565
JComponent (javax.swing) methods, 559
JComponent as drawing-surface, 528, 530, 531, 538, 602
JComponent as source of event-objects, 590
JComponent method isLightweightComponent(), 550
JComponent method paintComponent(), 552
JComponent opacity, 550
JDBC (Java DataBase Connectivity), 321
JDesktopPane (javax.swing), 610
JDialog (javax.swing), 565, 566
JDialog (javax.swing) heavyweight, 549, 550, 562, 597
JDialog (javax.swing) top-level screen-component, 531, 564, 598
JDK (Java Development Kit), 54
JDK (Java Development Kit), installing-, 59
JFC (Java Foundation Classes), 505, 623
JFrame (javax.swing), 528, 561, 563, 565, 566, 568, 597, 621
JFrame (javax.swing) heavyweight, 549, 550, 562, 597
JFrame (javax.swing) top-level screen-component, 531, 598
JFrame (javax.swing) top-level-screen-component, 564
JInternalFrame (javax.swing), 566, 567, 598, 610
JLabel (javax.swing), 544, 566, 621
JLayeredPane (javax.swing), 567, 568
JList (javax.swing), 545, 566
JMenu (javax.swing), 546, 566
JMenuBar (javax.swing), 546, 567
JMenuItem (javax.swing), 546
join()-method (java.lang.Thread), 211, 213, 216, 219, 221, 227, 232, 255, 329
Joint Photographic Experts Group (JPEG), 369, 522, 529
JPanel (javax.swing), 566–568, 597, 599, 621
JPanel as drawing-surface, 608
JPEG (Joint Photographic Experts Group), 369, 522, 529
JPopupMenu (javax.swing), 598, 610
JProgressBar (javax.swing), 547, 548, 566
JRE (Java Runtime Environment), 51, 52, 250, 431
JRE (Java Runtime Environment) and abstract references accessing native code, 184, 551
JRE (Java Runtime Environment) and applets, 67, 397, 613
JRE (Java Runtime Environment) and application, 72
JRE (Java Runtime Environment) and applications, 65
JRE (Java Runtime Environment) and daemon-threads, 223
JRE (Java Runtime Environment) and security, 128, 448
JRE (Java Runtime Environment) and servlets, 406
JRE (Java Runtime Environment) and threading portability, 204
JRE (Java Runtime Environment) and trigger-able methods, 128
JRE (Java Runtime Environment) as sandbox, 463
JRE (Java Runtime Environment) DEFINITION, 55
JRE (Java Runtime Environment) extended by packages, 240
JRE (Java Runtime Environment), class-loading, 433
JRE (Java Runtime Environment), communication by method-calls, 414
JRE (Java Runtime Environment), communication through streams, 297
JRE (Java Runtime Environment), communication with event-objects, 571
JRE (Java Runtime Environment), errors and exceptions, 279, 329
JRE (Java Runtime Environment), garbage-collection, 444
JRE (Java Runtime Environment), introspection, 254
JRE (Java Runtime Environment), multiple- on one native platform, 231
JRE (Java Runtime Environment), tools for single user interfacing, 505
JRE, Creation-tool: java, 51, 431
JRE, creation-tool: java, 54, 55, 64, 65, 72, 231, 236, 246, 434, 435, 451, 457, 461, 465, 473, 505
JRootPane (javax.swing), 561, 566, 568
JRS (Java Runtime System), 51, 53, 54, 65, 249
JRS (Java Runtime System) DEFINITION, 56
JSlider (javax.swing), 547, 548, 574
JToolTip (javax.swing), 560
JVM (Java Virtual Machine), 36, 40, 52, 54, 55, 444
JVM (Java Virtual Machine) and threads, 203, 216, 232
JVM (Java Virtual Machine), 32bit bus, 87
JVM (Java Virtual Machine), How to start?, 431, 641
JWindow (javax.swing), 545, 565, 566, 599, 610, 621
JWindow (javax.swing) heavyweight, 549, 550, 562, 597
JWindow (javax.swing) top-level screen-component, 531, 564, 598

kB (kiloBytes) 1024 bytes, 35, 537
kernel and front-end, 515
kernel/front-end model, 515, 516, 542
Key (java.security), 497
key of hashtable, 233, 263
key-chain, 484
key-pair algorithms, names of-, 481
key-pair-cryptography, 479
key-store (definition), 488

key-store (for public keys), 488
key-store database, 489
KeyAdapter (java.awt.event), 581
Keyboard, 572, 584
keyboard, 29, 271, 314, 507
KeyEvent (java.awt.event), 577, 581, 584
KeyFactory (java.security), 485, 493
KeyListener (java.awt.event), 577, 581
keymanagement-tool: keytool, 489, 492
KeyPair (java.security), 493
KeyPairGenerator (java.security), 485, 493, 497
keyPressed()-method (java.awt.event.KeyEvent), 584
KeyStore (java.security), 492, 494
keytool, keymanagement-tool, 489, 492
keywords, Java-, 81
kiloBytes (kB) 1024 bytes, 35, 537

Label (java.awt), 566
labeled statement, 111
labeled while-statement (a pre-post-condition loop), 111
language, JavaScript-, 338
language, non-procedural-, 40
language, procedural-, 40
late-binding language, 168
late-binding of code, 75
Late-Bound Method, 168
Latency, 587
layer-model, 353
layer-model (RMI), 417
layer-model of computer-platform, 52, 353, 550
layer-model of protocols, 348
layer-model, OSI-, 353
layers for painting and screen-components, 567, 615
layers of card-layout, 559
layout-manager, 543
layout-manager and absolute positioning, 542
LayoutManager interface (java.awt), 556, 558
layouts, multiple- in one container, 559, 566
least significant byte, 533
left-associative, 101
length of a dimension (array), 234
length()-method (java.io.File), 259
length()-method (java.io.RandomAccessFile), 299
length()-method (java.lang.String), 227, 257, 384, 387, 394, 456
length()-method (java.lang.StringBuffer), 257
letter, case of, 34, 66, 81, 82, 491
letter, case of (in domain-names), 361
letter, case of (in URIs), 375, 394
letter, case of, in integer-literal of type long, 78
lifelock (thread-specific problem), 226
lifo-storage (last-in-first-out), 233
lightweight process, 230
lightweight screen-component, 559, 565, 566
lightweight screen-component (concept), 353, 549
Lightweight-Code, 550
line-separator, 254, 304
List (java.awt), 566, 578
List (java.util), 237, 238
List (java.util) (java.awt), 239
list of listener-objects (with Swing-components), 579
list()-method (java.io.File), 259
list()-method (java.rmi.Naming), 417
list()-method (java.security.Provider), 486
list()-method (java.util.Properties), 254, 258
listenerList (variable of javax.swing.JComponent), 579
ListSelectionEvent (javax.swing.event), 547
ListSelectionListener (javax.swing.event), 547
literal, boolean-, 79
literal, character-, 73, 79
literal, class-, 252, 437
literal, floating-point-, 78
literal, integer-, 77
literal, literal form, 40, 73, 74
literal, null-, 144
literal, number-, 73
literal, number- (default type: int), 86
literal, string-, 81, 257
literals in Java, 77
literate programming, 323
little-endian-order of bytes, 533
loadClass()-method (java.lang.ClassLoader), 255, 438–441
loading of a class, 152, 433
local class, 157
local type, 124
Local Variable, 121
local variable, 121
local variable (and field), 148
local variable of primitive-type: initialized by assignment, 121
local variable, final-, 122
local variable, scope of-, 122
local variable-declaration, 122
local variables and threads, 217
Locale (java.util), 260–263, 402, 617
locale of screen-component, 542
localization, 259
Locking data, 214
log()-method (java.lang.Math), 95
logical connection, 343
Long (java.lang), 169, 181, 256
long (primitive-type), 85, 95, 169
long (primitive-type) and JVM, 87, 216
long (primitive-type), bitmask, 287, 555, 593
long (primitive-type), byte-count, 296
long (primitive-type), milliseconds, 223, 254, 261
long (primitive-type), NEVER length of an array, 234
long (primitive-type), postfixing literals, 100
loop/code, post-condition-, 90
loop/code, pre-condition-, 90
looping, 90, 108
looping (do-statement), 109
looping (for-statement), 109
looping (while-statement), 108
lower-case letter, 34, 82, 257, 394

machine-code, 37, 38

machine-synchronization, 216
main()-method characterizes an application, 38, 65, 127, 141, 142, 154
main()-method, explicitly called-, 630
main()-method, static-, 71, 142, 154
main()-method, triggerable-, 72, 127, 185
main- or main()-thread, 205, 207, 228, 232
Map (java.util), 237
Mask, 533
mask, 287, 555, 563, 576, 579, 581, 591, 593
mask, IP-, 358, 378
mask, subnet-, 358, 378
Math (java.lang), 95, 106, 182, 194, 227, 250, 257
Math (java.lang) private constructor, 150, 181
math (java.math package), 251
mathematical function: exponential function, 94
mathematical function: natural logarithm, 95
mathematical function: power function, 95
mathematical function: square root, 94
MB (MegaBytes) more than a million bytes, 35, 537
MegaBytes (MB) more than a million bytes, 35, 537
MegaHertz (unit of frequency), 31
Member of a class, 148
member, accessing class-, 171
Member, Class-, 148, 166
member, class-, 147, 148, 154
member, package-, 240
member-class, 157
member-variable, 148
member-variable of primitive-type: initialized to default value, 148
members, accessing class-, 156
members, interfaces and class-, 190
MemoryImageSource (java.awt.image), 530, 596, 610
menu, 511, 546
Menu (java.awt), 566
menubar, 511
MenuBar (java.awt), 566
message, 113
message (in message-model), 271
message (in object-orientation), 139
Message-Digest, 482
message-digest, 481
message-model, 271, 606
message-objects, 137
MessageDigest (java.security), 485
method, 139
Method (java.lang.reflect), 250, 253, 629, 630, 635
method and behavior (in object-orientation), 133
method, (static) factory-, 151
method, (static) factory- (in footnote), 382
method, abstract- (non-implemented method), 184
method, abstract- (non-implemented method, concept), 134, 136, 138
method, accessor- (of a software-component), 312
method, asynchronous-, 231
Method, Blocking-, 129
method, blocking-, 231
method, class-, 138, 182
method, content-handler-, 369
Method, Early-Bound- and Late-Bound-, 168
method, event-handler-, 287, 575, 577, 585
Method, Factory-, 151
method, factory-, 145, 185, 273, 381, 392, 485, 629, 642
Method, Factory-Like-, 185
method, factory-like-, 56, 527, 551, 569, 596, 641, 642
method, final-, called faster, 118, 168, 182
method, final-, unoverridable, 168, 182
method, handler-, 113
method, instance-, 139, 182
method, native-, 183
Method, Non-Blocking-, 231
method, non-blocking-, 231
method, overloaded-, 139, 168
method, overridden-, 139, 168
Method, Overriding-, 168
method, protocol-handler-, 349
method, static-, 154, 182
method, strictfp-, 183
method, synchronized-, 183
method, the caller of the, 642
Method, Triggerable-, 128
method-call, 118
method-call history, 470
method-declaration, 117
method-Signature, 117, 124
method-signature (in abstract methods), 184
method-signature (in interfaces), 187
method-signature (in overriding methods), 168
methods for rendering (an image), 536
methods in sub-classes overriding methods in super-classes, 168
middleware, 353
MIME-type, 389, 393
MIME-types, 368
minus-operator (binary), 94
minus-operator (unary), 93
modal dialog, 564
modal screen-component, 540, 542
Model and MVC, 515
Model Delegate model, 516, 542
model of thread-states, 210
Model View Controller model, 515
model, callback-, 276
model, chain-, 276, 440, 574
model, color-, 522
model, component-, 138, 381, 542
model, computer-, 19, 23
model, delegation-event-, 19, 276, 314, 440, 571, 573, 576, 585
model, event-, 278, 329
model, event-delegation- (Concept in Java), 278
model, kernel/front-end-, 515, 516, 542
model, layer, 353
model, layer- of computer-platform, 52, 353, 550
model, message-, 271, 606
model, Model View Controller, 515

model, object/class-, 131
model, object/class- (in programming language), 140
model, processor-, 39
model, protocol-, 274
model, pull-, 274, 520
model, push-, 520, 574
model, stream-, 272
model, subscriber-event-, 276
model, three-tier-, 352, 406
model-variable (MVC-concept), 516
modifier, 117
Modifier (java.lang.reflect), 250
modifier, abstract-, 181
modifier, access-, 177, 179, 180, 429
modifier, access- (protocol character), 345
modifier, final-, 182
modifier, native- (for methods), 183
modifier, no-, 176, 181
modifier, private-, 176, 181
modifier, protected-, 176, 180
modifier, public-, 176, 180
modifier, static-, 182
modifier, strictfp- (for classes and methods), 183
modifier, synchronized- (for methods), 183
modifier, transient- (for fields), 183, 309
modifier, volatile- (for fields), 183
Modularization, 240
modulo-operator, 93
most significant byte, 533
Mouse, 508, 571, 581
mouse, 29, 44, 127, 212, 271, 314, 371, 507
MouseAdapter (java.awt.event), 581, 584, 599
mouseClicked()-method (java.awt.event.MouseListener), 582, 584, 586, 595, 599, 600
MouseEvent (java.awt.event), 576–578, 581
MouseEvent constants, 577
MouseListener (java.awt.event), 577, 581
MouseMotionListener (java.awt.event), 577, 581, 601
mouseMoved()-method (java.awt.event. MouseMotionListener), 582, 587, 595, 601, 602
multi-processing, 199
multi-threading, 199
multicasting, 277, 343
multiple inheritance, 139
multiple inheritance and ambiguity, 136
multiple super-classes (concept), 136, 139, 187
multiplication-operator, 93
multipoint connection, 343
Multipurpose Internet Mail Extension-types, 368
multithreading (long and double), 87
MVC, Model View Controller, 515

name, ambiguous variable-, 187, 239
Name, Variable-, 40, 74
name, variable-, 73, 83, 119
name, variable-, representing a reference, 145
Name-Space, 123, 429
name-space, 430
name-space and security, 450
name-space considerations, 472
name-space of class-loader, 436, 469
name-space of package, 240
name-space, flat- (of packages), 241, 246
Naming (java.rmi), 417
naming conventions, 83, 140
NaN (Not a Number) constant of Float- and Double-class, 92, 257
NaN (Not a Number) usage, 86
NaN-expression (Not a Number), 86, 96
narrowing an identifier's type, 170
narrowing cast, 169
native method, 183
native modifier (for methods), 183
Native Software, 53
natural logarithm (mathematical function), 95
negation-operator, bitwise- ( ), 97
negation-operator, boolean- (!), 93, 97, 98
NEGATIVE_INFINITY usage, 86
NEGATIVE_INFINITY, constant of Float- and Double-class, 257
nested class, 157
Network Layer (OSI), 354
network, connection-oriented-, 345
network, connectionless-, 345
network, sub-, 343
network, virtual-, 343, 357
network-order of bytes, 533
network-protocol, technology-specific-, 348
network-topology, 340
new-keyword and anonymous object of interface-type Runnable, 198
new-keyword and anonymous objects, 159
new-keyword and array-type, 234
new-keyword and factory-methods, 151
new-keyword and reference-type, 77, 142, 145, 150, 253
newInstance()-method (java.lang.Class), 151, 253, 439, 499, 619, 642
no-name-package, 241
NoClassDefFoundError (java.lang), 61, 171, 281, 436
node of a network, 339
Non-Blocking Method, 231
non-blocking method, 231
non-local variable, 123
non-local variable (definition), 117
non-local variables and encapsulation, 195
non-localized switching-mechanism, 283
Nonvisible (state of screen-component), 598
nonvisible (state of screen-component), 541
NOT-operator (!), 97, 98
notify()-method (java.lang.Object), 211, 213, 216, 221, 223, 232, 252, 329
notify...()-method (custom-defined), 287, 331
notify...()-method (Swing, ...), 314
notifyAll()-method (java.lang.Object), 223, 232, 252
notifyObservers()-method (java.util.Observable), 285, 330
null-literal, 144
NullPointerException (java.lang), 281, 289

Number abstract (java.lang), 181, 256
number-literal, 73
number-literal (default type: int), 86
number-system, binary-, 34
number-system, decimal-, 34, 79, 257
number-system, hexadecimal-, 34, 100
number-system, octal-, 100

object, 131, 139
Object (java.lang), 161, 164, 252
Object (java.lang) array of -, 235
Object (java.lang) as message-object's type, 284, 285
Object (java.lang) as super-class, 164, 166, 252, 564
Object (java.lang) class without a super-class, 164
Object (java.lang) clone()-method, 151, 168, 192, 258
Object (java.lang) equals()-method, 163
Object (java.lang) finalize()-method, 127, 153, 398
Object (java.lang) getClass()-method, 437
Object (java.lang) hashCode()-method, 161, 482
Object (java.lang) in casting, 169
Object (java.lang) super-class of Throwable, 280, 281
Object (java.lang) wait() and notify()-methods, 223, 329
object and instance and variable, 149
object locked by one of its synchronized instance-methods, 214
object's state, behavior and identity, 131, 161
object, anonymous-, 159, 279, 581
object, constant-, 183
object, container-, 138
object, graphics-, 505, 597
object-code, 38
object-graph, 161, 191, 233, 307, 441
object-graph (cloning), 191
object-graph (in cloning), 191
Object-Orientation, 131
object-orientation, 637
object-orientation (event-messaging), 278
object-orientation (software-components), 311
object-orientation and Java, 193
object-orientation in Java, 142
object-orientation, behavior and method, 133
object-orientation, variable and state, 133
object/class-model, 131
object/class-model (in programming language), 140
ObjectInput interface (java.io), 308
ObjectInputStream (java.io), 298, 308, 318
ObjectOutput interface (java.io), 308
ObjectOutputStream (java.io), 298, 307, 308, 318
Observable (java.util), 284, 330
Observable-Observer switchboarding, 285
Observer interface (java.util), 285, 330
octal number-system, 100
off-screen rendering (=double buffering), 535, 542, 621
on-screen rendering, 535
One-Time-Server, 386
One-Way-Functions, 480

open protocol, 347
open-source, 659
Operating-System (OS), 17, 44
operating-system (OS), 338, 516
operating-system and drivers, 508
operator, AND- (&&), 98
operator, assignment- (=), 99
operator, assignment- (primitive- and reference-type), 144
operator, bitwise-negation- ( ), 97
operator, boolean negation- (!), 93, 97, 98
operator, conditional- (?), 99
operator, division-, 93
operator, equality- (==), 161–163
operator, inequality- (! =), 96
operator, minus- (binary), 94
operator, minus- (unary), 93
operator, modulo-, 93
operator, multiplication-, 93
operator, NOT- (!), 97, 98
operator, plus- (binary), 94
operator, plus- (unary), 93
operator, pre-deprecation-, 127
operator, qualification-, 182
operator, remainder-, 93
operator, shift- (example), 533
operator, string-concatenation-, 95
operators, additive-, 94
operators, assignment-, 99
operators, binary-, 93
operators, bitwise shift-, 95
operators, shift-, 95
order of bytes, big-endian-, 533
order of bytes, little-endian-, 533
order of bytes, network-, 533
OS (Operating-System), 44
OS (operating-system), 338, 516
OSI layer-model, 353
OutOfMemoryError (java.lang), 281, 444
output-stream, 295
OutputStream (java.io), 255, 310, 385
OutputStream (java.io) abstract, 296
OutputStreamWriter (java.io), 304
overflow/underflow warning with floating-point operations: NONE, 86
overflow/underflow warning with integers: NONE, 86
overloaded constructor, 151, 174
overloaded method, 139, 168
Overloading Methods, 125
overloading methods and polymorphism, 194
overridden method, 139, 168
overriding a static method, 168
overriding has to extend accessibility, 168
overriding makes methods accessible, 192, 286
Overriding Method, 168
overriding methods and final modifier, 168
overriding methods and polymorphism, 194
overriding variables?, 139

P-code, 39
pack()-method (java.awt.Window), 562, 598–600

Package, 240
Package (java.lang), 245
package introspection, 245
package naming, 241
package, anonymous-, 241
package, default-, 180
package, default- (idea), 241
package, default-, in footnote, 193
package, install a -, 242
package, no-name-, 241
package, unnamed-, 241
package, write a -, 240
package-hierarchy non-existent, 240
packages, application inside named-, 241, 246
packages, list of mentioned-, 653
packet (message-model), 272
Packet (of packet-oriented protocol), 349
packet or frame (networking-context), 349
packet-example (footnote), 275
Packet-Oriented Protocol, 349
packet-oriented protocol, 275, 351
packet-routing, 345
packet-routing, IP-, 358
packet-segmentation in networks, 351
paint()-method (java.awt.Component, ...), 128, 185, 205, 397, 527, 536, 539, 552, 595
paint()-method (javax.swing.JComponent, ...), 559
paint()-method importing a reference to native code, 185, 527
paint()-method of applet, 397
paint()-method, triggerable-, 397, 536, 552, 576, 595, 597
paint()-method, triggerable- (fixes view), 540
paint()-method, triggerable- calls the paintComponent()-method, 539, 559
paintComponent()-method (javax.swing.JApplet) NONE, 615
paintComponent()-method (javax.swing.JComponent, ...), 128, 185, 527, 528, 531, 539, 540, 551, 559, 576, 595–597, 621
paintComponent()-method importing a reference to native code, 185, 527
paintComponent()-method, triggerable-, 528, 531, 536, 539, 551, 559, 576, 595, 596
paintComponent()-method, triggerable- (fixes view), 540
PaintEvent (java.awt.event), 573, 576, 577, 595, 598
painting screen-components, 597
painting-methods, 595
Pairing-Quandary, 480
pairing-quandary, 483
Panel (java.awt), 559, 597
Panel heavyweight, 566
Panel, super-class of Applet, 251, 614, 615
Parallel-Server, 386
parameter of a method-declaration or -definition, 116
parent in containment-hierarchies, 543
parent of screen-component, 543
parent-class, 139
parser, 648
partial transparency, 527, 550
passing arguments by-reference, 120, 256
passing arguments by-value, 120, 225, 256
path in a network, 342
Path-Name, 47
path-name and package-name, 240, 242, 244
path-name and property expansion, 458
path-name and property expansion (example), 459
path-name and URI, 409, 410
path-name in URL, 364, 374
path-name separator, 48, 254
Path-Name, Absolute-, 48
path-name, absolute- (example), 510
Path-Name, Relative-, 48
path-name, relative- (example), 393
path-separator, 48, 254
PATH-variable (operating-system), 60, 409
PDF (Portable Document Format), 369, 414
Peer, 353
peer, 353, 548
peer-based screen-components (concept), 548
peer-code, 598
peer-interface, 59, 251
peer-to-peer communication, 275, 352, 516
peer-to-peer connections, 273
peer-to-peer HTTProtocol, 369
Permission (java.security), 453, 455, 459, 465, 476, 501
PermissionCollection (java.security), 455, 466
Permissions (java.security), 455, 469
Persistence, 238, 321
persistence and serialization of objects, 306, 318
persistent database, non- (aggregation), 138, 233
persistent path-name-setting (operating-system), 63
persistent storage (example), 298
persistent storage and communication over time-intervals, 231
persistent storage and databases, 321
persistent storage and file-systems, 46, 238
persistent storage and hard-disk (hard-drive), 28, 30, 46, 238, 306, 311, 321, 433, 447, 510
persistent storage and packages, 249
persistent storage and policy-files, 455
persistent storage and random-access-files, 299
persistent storage and serialization of objects, 443
persistent storage and software-components, 311, 318
persistent storage and transient member-variables/fields, 148, 183
persistent storage of cryptographic keys (key-store), 488
persistent storage of registered security-providers, 487
persistent storage, non- (Random Access Meme-ory), 28
persistent storage-device and class-loading, 433
physical connection, 340
Physical Layer (OSI), 354
pipe, 292

piped streams, 292
PipedInputStream (java.io), 300
PipedOutputStream (java.io), 300
PipedReader (java.io), 300
PipedWriter (java.io), 300
Pixel (picture-element), 519
pixel's partial transparency, 527
pixel, screen-, 523
pixel-coordinates, 536, 542
pixel-format, ARGB, 526
pixel-format, indirect, 526
PixelGrabber (java.awt.image), 610
pixelgrabbing, 535
Platform, 17, 44
platform, virtual-, 17, 36, 51, 55, 151, 333
plug-in (handler), 113, 325, 369
plus-operator (binary), 94
plus-operator (unary), 93
Point (java.awt), 555
point-to-point connection, 277, 343
Policy (java.security), 455, 457, 458, 460, 462, 465, 469
policy-file and public key, 459, 492
policy-file, conventions for the, 458
policytool (footnote), 458
polymorphism, 194
POP, 364
POP (Post Office Protocol), 371, 373
popup-menu, 511
PopupMenu (java.awt) top-level screen-component, 564
Port, 231
portability, 51
portability (and Java language), 333
portability (AWT and Swing), 548
portability and 100 Percent Java (footnote), 72
portability and multithreading, 203
portability and native methods, 118, 183
portability and pre-written packages, 249
portability and runtime environments, 55
portability of content-types, 265
portability-problem, 45, 47, 51
Portabiliy, 53
Portable Document Format (PDF), 369, 414
Portable Software, 53
portable source-code, 45
POSITIVE_INFINITY usage, 86
POSITIVE_INFINITY, constant of Float- and Double-class, 257
Post Office Protocol (POP), 371, 373
post-condition loop/code, 90, 109
postEvent()-method (java.awt.Component, java.awt.EventQueue, ...), 586, 590, 592
pow()-method (java.lang.Math), 95
power function (mathematical function), 95
pre-condition loop/code, 90, 109
pre-condition loop/code (example), 297
pre-deprecation-operator, 127
precedence, 101
preemptive thread-scheduling, 200, 203, 213
prepareImage()-method (java.awt.Component), 232, 529, 596
Presentation Layer (OSI), 354
primitive-type, 83, 87, 142, 191
primitive-type and atomic operations, 214
primitive-type and comparison-operator (==), 162
primitive-type, type-casting of-, 100
println()-method (java.lang.System.out), 37, 71, 254, 637
println()-method (java.lang.System.out) (footnote), 72
printStackTrace()-method (java.lang.Throwable), 282
PrintStream (java.io), 253
PrintWriter (java.io), 310, 413
priority-based thread-scheduling, 200, 202, 213, 641
private access, 176
private class-member, 148
private constructor, 150, 181, 182, 382, 417
private constructor and singleton, 160
private key (for decrypting), 481
private key (for signing), 482, 488, 492
private key (for signing), an example, 490
private key format (PCKS8), 481
private key in key-store, 488
private key producing a one-way-function, 480
private modifier, 176, 181
PrivateKey (java.security), 493
PrivilegedAction interface (java.security), 470, 471
PrivilegedExceptionAction (java.security), 471
problem tutoring, 512
Process (java.lang), 56, 250, 255
process and applet, 613
process and thread, 230, 254, 273, 293, 321
process...()-method, triggerable-, 287
processEvent()-method, triggerable-, 591, 592
processing-graph, 520
Processor, 28
processor's address-space, 28, 291, 522, 535
processor-model, 39
profiling, 629
Program, 17, 35
program, 160
program and applet, 67
program and application, 38
program and compiler, 39
program and interpreter, 39
program and operating-system, 43, 44
program and runtime-environment, 55
program, Java-, 51
program, native-, 53
programmer, application-, 27
programmer, infrastructure-, 27
programmer, system-, 27
programmer, utility-, 27
Programming, 38
programming language, non-procedural-, 40
Programming, Application-, 38
programming, distributed-, 419, 423
Programming, Infrastructure-, 38

programming, infrastructure- (security), 455, 486, 497
programming, infrastructure- (switchboard example), 633
programming, literate-, 323
programming, symbol-oriented-, 40, 71
programming, visual- (with software-components), 311, 313
programming-language, JavaScript-, 338
programming-language, procedural-, 40
Properties (java.util), 254, 258, 444, 486
properties and variables, 312
property (of a software-component), 312
property expansion with text-files, 458, 459
property, bound- (of software-component), 315
property, constrained- (of software-component), 315
Property-Expansion, 458
PropertyChangeEvent (java.beans), 314, 331, 578, 603
PropertyChangeListener (java.beans), 314, 319, 331, 578
PropertyChangeSupport (java.beans), 314, 319, 331
PropertyPermission (java.util), 469
protected access, 177
protected constructor, 151, 181
protected modifier, 176, 180
Protection (security-related), 449
protection-domain, 454, 466, 469, 471
ProtectionDomain (java.security), 453, 454, 458, 465, 468, 469, 501, 629
Protocol, 274, 275
protocol, 122, 190, 231, 279, 289, 347
Protocol (message-model), 275
protocol and type, 347
protocol, application-, 348, 367, 378
protocol, application-, and servlet, 405
protocol, application-, and TCPport-number, 363
protocol, application-, and URL, 373
protocol, application-, and URL-class, 388
Protocol, Connection-Oriented-, 350
protocol, connection-oriented-, 275, 351
protocol, connectionless-, 350
protocol, DNS, 382
protocol, open-, 347
Protocol, Packet-Oriented-, 349
protocol, packet-oriented-, 275, 351
protocol, service- (or application-), 378
protocol, signaling-, 342
protocol, state-bound-, 351
protocol, stateless-, 351
protocol, transport-, 342, 348
protocol, virtual-network-, 348
Protocol-Handler, 349
protocol-model, 274
protocol-stack, 348, 512
protocols, layer-model of, 348
Provider (java.security), 486, 499
Provider Infrastructure, 487
provider-infrastructure, 485
Pseudo-code, 39
public access, 177
public class extended over different packages, 438
public class has to be in class of same name, 146, 241, 242, 245, 246
public constructor, 180
public key (distribution problem), 481, 483
public key (for authentication), 482
public key (for encrypting), 481
public key and identity of owner, 481
public key and policy-file, 459, 492
public key producing a one-way-function, 480
public keys in key-stores, stored as certificates, 488
public modifier, 176, 180
public-key-file (example), 493
PublicKey (java.security), 493, 494
pull-model, 274, 520
push-model, 520, 574

QoS (Quality of Service), 345, 351, 354, 423, 444
QoS (Quality of Service) concept, 447
qualification-operator, 182
quality of a cryptographic algorithm, 479
Quality of Service (QoS), 339, 345, 351, 354, 423, 444
Quality of Service (QoS) concept, 447
queue, event-, 586

race-condition (thread-specific problem), 225
railroad-diagram, 76
RAM (Random Access Memory), 28, 73, 76
RAM-address, 73, 143
Random (java.util), 257
Random Access Memory (RAM), 28, 73, 76
random()-method (java.lang.Math), 106, 112, 219, 229, 257, 531
random-access, 233, 291
random-access-file, 299, 321
RandomAccessFile (java.io), 299, 321
rasterization, 535
read()-method (java.io.FilterInputStream,
  java.io.FilterReader), 302
  java.io.FilterReader) triggerable, 306
read()-method (java.io.InputStream, java.io.Reader), 127, 232, 291, 295
read()-method (java.io.InputStream, java.io.Reader) abstract, 296
Reader (java.io), 304, 310
Reader (java.io) abstract, 296
readLine()-method (java.io.BufferedReader), 300, 387, 390, 394, 456
readObject()-method (java.io.ObjectInputStream), 308–310, 318
real-life problems, 512
Rectangle (java.awt), 555, 607
Recursion, 125
recursive data-structures (not treated), 233
redundant information, 522
reference, passing arguments by-reference, 120, 256
Reference-Type, 143

reference-type, 143, 191
reference-type and atomic operation, 214
reference-type and equality operator (==), 162
reference-type, type-casting of-, 169
reference-type, variable of-, 143
references, concept of weak-, 445
reflection (Java language), 252, 442, 629
reflection (software-/screen-components), 542
reflection (software-components), 251, 318, 319
registry (message-model), 272
Relative Path-Name, 48
relative path-name (example), 393
remainder-operator, 93
Remote interface (java.rmi), 189, 417
Remote Method Invocation (RMI), 19, 64, 251, 273, 310, 338, 373
Remote Method Invocation (RMI), introduction, 416
rendering an image, 519
rendering an image (off-screen), 535, 542, 621
rendering an image (on-screen), 535
rendering as coding-transformation, 535
rendering of images, 587
rendering-method, 536
repaint()-method (java.awt.Component), 128, 405, 596–598, 601, 602
repaint()-method (java.awt.Component), example, 602
RepaintManager (javax.swing), 560, 599
repository (message-model), 272
resolution in dpi (dots per inch), 536, 537
Resource, 44
ResourceBundle (java.util), 260, 263
return-keyword, 118
return-statement, 118
reusability (of software-components), 313
RGB-model, 523
right-associative, 101
RLE-compression (Run Length Encoding), 522
RMI, 414
RMI (Remote Method Invocation), 19, 64, 251, 273, 310, 338, 373
RMI (Remote Method Invocation), introduction, 416
RMI layer-model, 417
RMI-compiler-tool: rmic, 415
RMI-registry-tool: rmiregistry, 415
rmic, RMI-compiler-tool, 415
rmiregistry, RMI-registry-tool, 415
RootPaneContainer (javax.swing), 563, 566, 615, 618
round-robin-scheme of scheduling threads of same priority, 202
router, 290
rt.jar, 54, 247, 249, 389
rt.jar (loading), 434
rt.jar, peeking inside-, 243, 244
run()-method (java.lang.Runnable), 160, 198, 204, 205, 207–210, 255, 256, 258, 329, 386, 401, 589, 641
run()-method (java.lang.Runnable) abstract, 189
run-time typed language, 75
Runnable interface (java.lang), 189, 209, 228, 232, 255, 256, 258, 386, 401, 589
Runnable interface (java.lang) instantiated anonymously, 208
Runnable interface (java.lang), introduction, 205
Runnable thread, 200, 211
Running thread, 211
Runtime (java.lang), 56, 250, 254, 258, 402, 432, 445, 641
Runtime Environment, 55
runtime-representation of a class, 54, 255, 433, 441, 443, 629
Runtime.exit()-method (java.lang), 254
Runtime.gc()-method (java.lang), 254
RuntimeException (java.lang), 279, 281
RuntimePermission (java.lang), 468, 472

Safe Channel, 483
safe channel, 449, 479, 481, 483
Sandbox, 463
sandbox, 68, 333, 398, 406, 432, 475, 613
scheduling of events, 572, 574
scheduling of threads, 53, 200, 213, 216, 228, 230
scheduling threads in Java, 203
SCOPE, 122
scope, 430
scope and access-area, 172
scope and access-area in an Hierarchical File System (HFS), 172
scope and name-space, 123
scope of identifier, 122, 171
scope of local variable, 122
screen-component, 331, 399, 509, 510, 519
Screen-Component (concept), 540
screen-component and its GUI-related states, 509, 598
screen-component as a painting-surface, 538
screen-component as container, 542
screen-component on focus, 542, 577
screen-component, heavyweight-, 559, 562, 564, 566, 595, 597
screen-component, heavyweight- (concept), 353, 548
screen-component, lightweight-, 559, 565, 566
screen-component, lightweight- (concept), 353, 549
screen-component, modal-, 540, 542
screen-component, visible, showing, nonvisible, 598
screen-pixel, 523
Scrollbar (java.awt), 548, 559
ScrollPane (java.awt), 559
ScrollPaneLayout (javax.swing), 543
secure class-loader, 454, 458, 469, 471
secure stream, 501
SecureClassLoader (java.security), 465, 469, 629
Security (java.security), 485–487, 499
security-manager, 462
security-manager (and access-control-mechanism), 454, 455
security-manager (and applets), 475
security-manager (and class-loading), 440

security-manager (autistic and free), 255
security-manager (its various uses), 475
security-manager (run Java code with and without a-), 460, 465
security-manager and (security-)sandbox, 463
security-related-applications, 488
Security-Sandbox, 463
security-sandbox, 68, 475
SecurityException (java.lang), 281, 463
SecurityManager (java.lang), 250, 254, 255, 427, 440, 453, 456, 476
SecurityManager (java.lang) core, 462
SecurityPermission (java.security) in footnote, 465
segmentation of packets in networks, 351
self-signed certificate, 484
semantic event, 578
semantic manipulation, 35, 73
semi-transparency, 550
separator, folder-/directory-/file-, 46, 254
separator, folder-/directory-/file- (in packaging), 242
separator, folder-/directory-/file- (policy-files), 458, 462
separator, folder-/directory-/file- (URL), 364, 388
separator, line-, 254, 304
separator, path-, 48, 254
separators, 641
separators, table of-, 48
sequence of constructor-calls, 152
Sequential-Server, 386
sercret-key-cryptography, 478
Serializable interface (java.io), 189, 307, 312, 318, 563
Serialization, 307
Server, 276
Server, One-Time-, 386
Server, Parallel-, 386
Server, Sequential-, 386
server-side computing, 406
ServerSocket (java.net), 273, 385, 421
service()-method (javax.servlet.Servlet, ...) abstract, 128, 407, 408, 411
service()-method, triggerable-, 407
service- (or application-) protocol, 378
Servlet, 406
servlet introspection, 411
servlet's triggerable method, 407
servlet-container (=servlet-sandbox), 406, 407, 409, 430
servlets and web-applications, 414
Session, 277
Session Layer (OSI), 354
set...()-method, 312, 313, 318
setContentHandlerFactory()-method (java.net.URLConnection), 190, 393, 394
setDaemon()-method (java.lang.Thread), 224, 232
setName()-method (java.lang.Thread), 212
setPriority()-method (java.lang.Thread), 212, 224, 232
setSecurityManager()-method (java.lang.System), 254, 255, 463–465
setURLStreamHandlerFactory()-method (java.net.URL), 190, 391, 394
setVisible()-method (java.awt.Component), 528, 552, 595, 597, 598
shadowing a static variable, 168
shadowing variables, 123, 126, 167, 168
Shallow Cloning, 191
shallow equal object-graphs, 162, 192
shift operators, 95
shift-operator (example), 533
Short (java.lang), 169, 181, 256
Short (java.lang), un/wrapping, 256
short (primitive-type), 85, 169
short (primitive-type), length of an array, 234
short (primitive-type), un/wrapping, 256
Showing (state of screen-component), 598
showing (state of screen-component), 541, 598
signaling-protocol, 342
Signature (java.security), 485, 492, 494
signature, digital-, 483
signature, digital- and certification, 484
Signature, method-, 117, 124
signature, method- (in abstract methods), 184
signature, method- (in interfaces), 187
signature, method- (in overriding methods), 168
signature-file (example), 493
signed class = signed class-file, 494
signed data (example), 493
signed data and authentification (concept), 482
signed data and policy-files, 459, 492
signed data and trust, 478
signed data in class-loading, 469
signed data-files, by user, 488
signed data-files, byte-code, classes, objects, 494
signed JAR-file, 491
SignedObject (java.security), 494
signing data by programmer, 492
signing public keys (certification), 483
signing-tool: jarsigner, 489, 492
Simple Mail Transfer Protocol (SMTP), 371, 373, 389
Singleton, 160
singleton, 463
skeleton-class (RMI), 416
sleep()-method (java.lang.Thread), 204, 211, 212, 223, 232
SMTP, 364
SMTP (Simple Mail Transfer Protocol), 371, 373, 389
Socket, 273
Socket (java.net), 273, 382, 385
socket, TCP-, 273, 383, 385
socket, TCP/UDP-, 364
SocketPermission (java.net), 459
Software, 36
Software, Native-, 53
Software, Portable-, 53
software-component (concept), 311
software-components: introspection, 311
source of message-model, 271
Source-Code, 38

source-code, 37
source-code written with a text-editor, 53, 54, 57, 62, 63, 65, 323
source-code, portable-, 45
source-code, programmer comment, 37, 71, 323
space, address-, 28
space, address-, and port, 231, 271, 337
space, address-, of a processor, 291, 522, 535
spawn a thread, 206
SQL (Structured Query Language), 20, 40, 252, 321
sqrt()-method (java.lang.Math), 94
square root (mathematical function), 94
stack (aggregate of protocols), 348, 512
stack (aggregate), 138, 233, 292
stack (aggregate, processor-state), 217, 271, 281
stack of cards, 202
stack, protocol-, 348
StackOverflowError (java.lang), 281
standalone code (=program), 160
start()-method (java.applet.Applet), 205, 398, 400, 618, 641
start()-method (java.lang.Thread), 204, 206–209, 211, 212, 232, 386, 401
start()-method, triggerable-, 398
starvation (thread-specific problem), 224
state and variable (in object-orientation), 133
state of an object, 131, 161
state-bound protocol, 351
stateless protocol, 351
statement, 103
statement, break-, 110
statement, continue-, 110
statement, do- (looping), 109
statement, empty-, 103
Statement, Expression and Control Structure (concept), 89
statement, for- (looping), 109
statement, if- (branching), 105
statement, labeled-, 111
statement, return-, 118
statement, switch- (branching), 106
statement, synchronized- (multithreading), 216
statement, throw- (messaging with Throwable-objects), 279
statement, try- (messaging with Throwable-objects), 282
statement, unreachable-, 106
statement, while- (looping), 108
states of a screen-component, conceptual-, 509, 598
static blank-final, 182
static class-member, 140, 148, 154
static class-members remain unserialized, 309
static factory-method, 151
static factory-method (footnote), 382
static member-class, 158, 182
static method, 138, 154, 182
static method, overriding a-, 168
static modifier, 182
static synchonized methods lock their class, 214
static variable (class-variable), 138, 148, 154, 182
static variable in an abstract class, 184
static variable, unshadowing a-, 173
static-initializer, 152, 441, 641
static-initializer and static blank-finals, 182
static-initializer, not a class-member, 148, 166
static-keyword and class-member, 154
statically-typed language, 75
stop()-method (java.applet.Applet), 205, 398–400, 618, 641
stop()-method, triggerable-, 398
Stream, 273
stream, 19, 233
Stream (Java), 295
stream (message-model), 272
stream, content-, 389, 394
stream, filter-, 292
stream, I/O-, 292
stream, input-, 296
stream, output-, 295
stream, secure-, 501
stream, wrapper-, 292
stream-example (footnote), 275
stream-model, 272
stream-objects and -classes, 295
streams and variables, 292
streams in Java, 291
streams to aggregates, 237
streams, piped-, 292
strength of an cryptographic algorithm, 479
strictfp, 118
strictfp method, 183
strictfp modifier (for classes and methods), 183
String (java.lang), 81, 194, 250, 257, 520, 642
String (java.lang) casting, 170
String (java.lang) comparing, 162
String (java.lang) exceptionality, 77, 145
string-concatenation, 95
string-concatenation-operator, 95
string-literal, 81, 257
StringBuffer (java.lang), 120, 250, 257, 642
StringBufferInputStream (java.io), 237
StringReader (java.io), 291, 310
StringTokenizer (java.util), 233, 237, 301, 386
StringWriter (java.io), 310
Structured Query Language (SQL), 20, 40, 252, 321
stub-class (RMI), 416
sub-class (concept), 135
sub-classes with methods overriding methods in super-classes, 168
sub-network, 343
subdomain (DNS), 360
subnet-mask, 358, 378
subscriber event-model, 276
super()-constructor, 149
super()-constructor (class-extension), 166
super()-constructor, example, 304
super-class, 164
super-class (concept), 135

super-class, class without a- (java.lang.Object), 164
super-classes and casting, 169, 253
super-classes and class-loading, 441
super-classes containing methods overidden by methods of sub-classes, 168
super-classes in graphical representation, 562
super-classes, multiple- (concept), 136, 139, 187
super-interfaces and implements-keyword, 188
super-keyword, 153, 173, 593
super-keyword for un-overriding from OUTSIDE a class, 175, 192, 286
super-keyword for un-overriding from WITHIN a class, 174
super-keyword undefined in static methods, 182
super-keyword used for unshadowing, 173
Swing, 505
Swing (basics), 548
Swing as API, 52
Swing versus AWT (Abstract Window Toolkit), 520, 548
Swing, applets, 613
Swing, class-hierarchies, 564
Swing, desktop, 610
Swing, image drawing, 519, 595
Swing, screen-components, 551
SwingUtilities (javax.swing), 538, 587–589
switch-statement (branching), 106
switch-statement (branching), example, 401
switchboard (message-model), 337
switchboard (OSI layer-model), 354
switchboard and pairing-quandary, 480
Switchboard routes messages (message-model), 272
switchboard, custom-written, 633
switchboard-mechanism, advanced AWT-, 594
switchboard-mechanism, AWT-, 571
switchboarding, java.beans, 314
switchboarding, Observable-Observer-, 285
switchboarding, try-catch clause, 282
switchboarding-structures, 276
switching-mechanism, non-localized-, 283
symbol-oriented-programming, 40, 71
symmetric-key-cryptography, 478
synchonized block, 216
synchronization, machine-, 216
synchronized instance-methods lock their object, 214
synchronized method, 183
synchronized modifier (for methods), 183
synchronized-statement (multithreading), 216
Syntax Diagrams, 655
syntax-diagram, 76
System (java.lang), 56, 250, 253, 254, 258, 402, 505
System (java.lang) stream, 274
system-programmer, 27
System-Property, 259
system-property and property-expansion, 458
System.arraycopy()-method (java.lang), 236, 630
System.currentTimeMillis()-method (java.lang), 261
System.exit()-method (java.lang), 254, 432, 581, 651
System.exit()-method (java.lang), example, 226, 382
System.gc()-method (java.lang), 153, 223, 254, 445, 598
System.getProperties()-method (java.lang), 258
System.in (java.lang) stream, 297
System.out (java.lang) stream, 297
System.out.println()-method (java.lang), 71, 254, 637
System.out.println()-method (java.lang) (footnote), 72
System.setSecurityManager()-method (java.lang), 255, 463
SystemColor (java.awt), 527

TCP (Transmission Control Protocol), 363, 378, 385
TCP-port, 643
TCP-port 80 (HTTP), 371
TCP-port in URL, 374
TCP-socket, 273, 383, 385
TCP/UDP-port, 363
TCP/UDP-socket, 364
technology-specific network-protocol, 348
TELNET, 364, 371, 372
terminal-based user interface, 507
terminal-window, 44, 54, 55, 64
terminal-window (and appletviewer-command), 396, 431, 613, 641
terminal-window (and CTRL-C-command), 432
terminal-window (and jar-command), 326, 645
terminal-window (and java-command with jar-option), 327
terminal-window (and java-command with security-option), 462
terminal-window (and java-command with String-arguments), 236
terminal-window (and java-command with verbose-option), 250, 434
terminal-window (and java-command with version-option), 59, 65, 250
terminal-window (and java-command), 51, 65, 431, 505, 619
terminal-window (and java-command, triggers main()-method), 116, 127, 142
terminal-window (and javac-command), 65, 72, 431
terminal-window (and javadoc-command), 324
terminal-window (and javap-command), 642
terminal-window (and keytool-command), 489
terminal-window (and policytool-command), 458
terminal-window (and RMI-related commands), 418
terminal-window (and servlet-related commands), 407
terminal-window (and Uniform Resource Locator (URL)), 373
terminal-window (as single user interface), 508
terminal-window (its current directory), 48
terminal-window (Linux operating-system), 62

terminal-window (Mac OS X), 61
terminal-window (multiple- for multiple JREs), 52, 231
terminal-window (typical java-commands), 37
terminal-window (Windows operating-system), 63
text-based document format, 414
text-based user interface, 507
text-editor for editing security- and policy-files, 453, 457, 458, 496
text-editor for writing source-code, 53, 54, 57, 62, 63, 65, 323
text-editors and text-model, 543
this()-constructor, 167
this()-constructor, example, 568
this-keyword, 173, 207, 217, 223, 580, 596, 630
this-keyword undefined in static methods, 182
this-keyword used for unshadowing, 172
this-keyword with little sense, 227
Thread (java.lang), 206, 210, 216, 232, 256, 329
Thread (java.lang) cast, 442
thread communication, with objects, 225
thread, Blocked-, 199, 211
thread, Daemon-, 223
thread, daemon-, 215, 432, 586
thread, Event-Dispatch-, 585
thread, event-dispatch-, 205, 228
thread, event-dispatch-, and painting-methods, 595
thread, event-dispatch-, running code on the-, 589
thread, finalize-, 228
thread, garbage-collection-, 223, 228, 444
thread, Inactive (=Interrupted), 200
thread, Initialized-, 211
thread, Interrupted- (= Inactive), 200
thread, main- or main()-, 205, 207, 228, 232
thread, method-call history of a-, 470
thread, Runnable-, 200, 211
thread, Running-, 211
thread, spawn a -, 206
thread-context-switch, 230
thread-scheduling, 53, 200, 213, 216, 228, 230, 641
thread-scheduling in Java, 203
thread-scheduling, preemptive, 200, 203, 213
thread-scheduling, priority-based-, 200, 202, 213, 641
thread-scheduling: round-robin-scheme, 202
thread-scheduling: time-slicing, 201, 203, 213, 641
thread-specific problem (deadlock), 226
thread-specific problem (dining humans), 215
thread-specific problem (lifelock), 226
thread-specific problem (race-condition), 225
thread-specific problem (starvation), 224
thread-states, 210
ThreadGroup (java.lang), 250, 256
threads and local variables, 217
three-tier-model, 352, 406
three-tiered communication infrastructure, 277, 352, 406, 414
throw-statement, 279, 330
throw-statement (messaging with Throwable-objects), 279
throw-statement (security-exception), 463
Throwable (java.lang), 256, 280, 284, 329
Throwable (java.lang) hierarchy, 281
throws-keyword and checked exceptions, 281, 330
throws-keyword in constructor-declaration, 150
throws-keyword in method-declaration, 117, 280
tiered communication infrastructure, three-, 277, 352, 406, 414
time-slicing, a form of thread-scheduling, 201, 203, 213, 641
Timer (javax.swing), 576, 603
toCharArray()-method (java.lang.String), 291, 495
tool, 54
tool tutoring, 512
tool(Archiving): jar, 325
tool(archiving): jar, 54, 242, 243, 250, 489, 492
tool(Compiler): javac, 72, 431
tool(compiler): javac, 60, 246, 333, 397, 407, 473
tool(disassembler): javap, 250, 642
tool(documentation): javadoc, 324
tool(Java Runtime Environment Creation): java, 51, 431
tool(Java Runtime Environment creation): java, 54, 55, 64, 65, 72, 231, 236, 246, 434, 435, 451, 457, 461, 465, 473, 505
tool(keymanagement): keytool, 489, 492
tool(RMI-compiler): rmic, 415
tool(RMI-registry): rmiregistry, 415
tool, application as tool for solving real-life-problems, 507
tool, Bean-Box, 313
tool, builder- (for software-components), 311
tool: jarsigner, 489, 492
toolbar (tool-bar), 511
Toolkit (java.awt), 254, 332, 529, 556, 586, 603, 607
Toolkit typed reference to native-code, 185
Toolkit typed reference to native-code, getting a-, 528, 551, 555, 579, 596
Tooltip, 601
tooltip, 559, 599
top-level window, 577
top-level window, concept of-, 509
top-level windows and heavyweights, 550, 565
top-level-class, 157
top-level-domains, country-code- (DNS), 361
top-level-domains, generic- (DNS), 361
topology of a network, 340
toString()-method (java.lang.Object), 161, 165, 252, 256
toString()-method (java.lang.StringBuffer), 257
toString()-method (java.lang.Throwable), 282
toString()-method (java.util.EventObject), 288
totalMemory()-method (java.lang.Runtime), 254, 258, 445
toUppercase()-method (java.lang.String), 102
transformation, coding-, 35, 520, 534, 535
transformation, content-, 35, 520
transient field (non-serializable), 183, 309

transient modifier (for fields), 183, 309
Transmission Control Protocol (TCP), 363, 378, 385
transparency, 139, 377, 526, 550, 556
transparency close to encapsulation, 139, 367, 377, 382
transparency, partial, 527
Transparent, 367
Transport Layer (OSI), 354
transport-protocol, 342, 348
tree-structure, 233, 543
tree-structure of a class-hierarchy, 135
tree-structure of a Hierarchical File System, 46, 326
tree-structure of texts, 516
triggerable check...()-methods, 462
triggerable destroy()-method, 398
triggerable finalize()-method, 127, 153
triggerable imageUpdate()-method, 563
triggerable init()-method, 398
triggerable main()-method, 72, 127, 185
Triggerable Method, 128
triggerable method of servlets, 407
triggerable methods of applets, 397, 614, 618
triggerable paint()-method, 397, 536, 552, 576, 595, 597
triggerable paint()-method (fixes view), 540
triggerable paint()-method calls the paintComponent()-method, 539, 559
triggerable paintComponent()-method, 528, 531, 536, 539, 551, 559, 576, 595, 596
triggerable paintComponent()-method (fixes view), 540
triggerable process...()-method, 287
triggerable processEvent()-method, 591, 592
triggerable read()-method, 306
triggerable service()-method, 407
triggerable start()-method, 398
triggerable stop()-method, 398
triggerable validate()-method, 598
trim()-method (java.lang.String), 257, 387, 394
TRUST, 478
try-catch clause, 113, 279
try-catch clause (branching), 112
try-catch clause, switchboarding, 282
try-statement (messaging with Throwable-objects), 282
try-statement and break-statement, 282
tunneling (data in network), 346
tutoring, problem-, 512
tutoring, tool-, 512
Type, 73
type, 35, 40, 71, 74, 75, 521
type and protocol, 347
type safety, 450
type, array-, 143, 234
type, class-, 143, 146
type, content-, 368, 405
type, interface-, 143, 185
type, local-, 124
type, MIME-, 368, 389, 393
type, primitive-, 83, 87, 142, 191
type, programmer-defined-, 131, 140, 142
Type, Reference-, 143
type, reference-, 143, 191
type-cast potential, testing-, 253
type-cast, implicit- (example), 544, 580
Type-Casting, 88
type-casting, 257, 450
type-casting identifiers to an abstract class-type, 552
type-casting identifiers to interface-type, 194, 404
type-casting of primitive-types, 100
type-casting of reference-type, 169
type-casting, explicit-, 100
type-casting, implicit-, 86, 87, 100, 106, 169, 194, 253
typed, dynamically-typed language, 75
typed, run-time typed language, 75
typed, statically-typed language, 75

UCS (Unified Coding Standard), 265
UDP (User Datagram Protocol), 354, 363, 378, 384
UDP (User Datagram Protocol) and RMI, 415
un-overriding from OUTSIDE a class with super-keyword, 175, 192, 286
un-overriding from WITHIN a class with super-keyword, 174
underflow/overflow warning with floating-point operations: NONE, 86
underflow/overflow warning with integers: NONE, 86
unicast as one-way-flow, 343
unicast event-messaging, 287, 289, 633
Unicode Transformation Format (UTF), 20, 265, 308
Unicode-format, 34, 265
Unified Coding Standard (UCS), 265
Uniform Resource IDENTIFIER (URI), 365
Uniform Resource Locator (URL), 365
Uniform Resource Name (URN), 365
UnknownError (java.lang), 281
unnamed package, 241
unreachable statement, 106
UnresolvedPermission (java.security) in footnote, 467
unshadowing a static variable, 173
unshadowing an instance-variable, 172
update()-method (java.awt.Component), 552, 595
update()-method (java.security.Signature), 493
update()-method (java.util.Observer), 285, 286, 330
update()-method (java.util.Observer) abstract, 285
Uploading files (client TO server), 371
uploading files (client TO server), 372, 374
upper-case letter, 34, 102
upper-case letter, in integer-literal of type long, 78
upper-case letters in jar-file manifest, 491
upper-case letters, keyboard signals for-, 572
URI (Uniform Resource IDENTIFIER), 365
URL (java.net), 388, 390–392, 405, 439
URL (Uniform Resource Locator), 365

URL-encoding, 375
URL-Rewriting, 413
URLClassLoader (java.net), 438, 469, 629
URLConnection (java.net), 389, 392
URLStreamHandler (java.net), 391, 392
URLStreamHandlerFactory (java.net), 391
URN (Uniform Resource Name), 365
user (key-management and signing files), 488
user (using a security-manager), 465
user and applet-configuration, 475
user and jar-archive, 326
user and means of interaction, 56
user and operating-system, 43
user and security, 448, 453, 457
user and tools, 54
user calling Java's main()-method, 127
User Datagram Protocol (UDP), 354, 363, 378, 384
User Datagram Protocol (UDP) and RMI, 415
user interface, 507
user interface, terminal-based-, 507
user interface, text-based-, 507
user interfaces, 507
user, cannot change a security-manager, 463
user, selecting policies, 460
users and applications, 38
UTF (Unicode Transformation Format), 20, 265, 308
utility-programmer, 27

validate()-method, triggerable-, 598
value, passing arguments by-value, 120, 225, 256
valueOf()-method (java.lang.Boolean, Byte, ...), 256
valueOf()-method (java.lang.Integer), 642
valueOf()-method (java.lang.String), 257
Variable, 40, 74
variable and member, 148
variable and object and instance, 149
variable and state (in object-orientation), 133
variable of primitive-type, casting of-, 88
variable of primitive-type, copying/cloning, 83
variable of primitive-type, type-casting of-, 100
variable of primitive-type, wrapped-, 256
variable of reference-type, 143
variable of reference-type, casting of-, 169
variable of reference-type, copying/cloning, 144
variable of String-class, 145
variable or field, 148
variable, array-, 234
variable, assignment by-value, 83
variable, class- (static variable), 138, 148, 154, 182
variable, CLASSPATH- (operating-system), 61, 66, 243, 254, 436, 451, 629
variable, data-equality with primitive-type, 162
variable, dynamically typed-, 75
variable, field-, 155
variable, final local-, 122
variable, final- (quasi-constant), 182
variable, instance-, 139, 148, 182
Variable, Local-, 121
variable, local- (and field), 148
variable, member-, 148
variable, model- (MVC-concept), 516
variable, non-local-, 123
variable, non-local- (definition), 117
variable, overridden?, 139
variable, passed by-reference to a method, 120
variable, passed by-value to a method, 120
variable, PATH- (operating-system), 60, 409
variable, pointer-equality with reference-type, 162
variable, shadowing a static-, 168
variable, static- (class-variable), 138, 148, 154, 182
variable, static-, in an abstract class, 184
variable, statically typed-, 75
variable-declaration, 147
variable-declaration, local-, 122
Variable-Name, 40, 74
variable-name, 73, 83, 119
variable-name, ambiguous-, 187, 239
variable-name, representing a reference, 145
variables and properties, 312
variables and streams, 292
variables, implicitly final in interfaces, 186
variables, shadowing-, 123, 126, 167, 168
Vector (java.util), 238, 445
Vector (java.util), as buffer, 586
Vector (java.util), as package-member, 250
Vector (java.util), as type of container-object, 138, 236
Vector (java.util), example, 219, 634
Vector (java.util), example, as buffer, 228
verifier, byte-code-, 281, 441, 450, 451
VetoableChangeListener (java.beans), 316, 331
VetoableChangeSupport (java.beans), 316, 331
View and MVC, 515
virtual channel (in network), 346
virtual class (non-instantiable class), 138
Virtual Machine, 40
virtual machine, 39
virtual network, 343, 357
virtual platform, 17, 36, 51, 55, 151, 333
virtual-network-protocol, 348
VirtualMachineError (java.lang), 281, 444, 445
Visible (state of screen-component), 598
visible (state of screen-component), 541, 598
visual programming (with software-components), 311, 313
Void (java.lang), 117, 257
volatile field (uncached), 148, 183
volatile modifier (for fields), 183

wait()-method (java.lang.Object), 211, 213, 216, 221, 223, 232, 252
waitFor()-method (java.lang.Process), 255
weak references, concept of-, 445
Web-Application, 406
web-applications and servlets, 414
web-server (see server), 407
while-statement (a pre-post-condition loop), 110
while-statement (usually a post-condition loop), 108

while-statement, labeled- (a pre-post-condition loop), 111
WHOIS, 362, 364, 382
widening cast, 169
widening cast and polymorphism, 195
widening of the identifier's type, 170, 638
Window (java.awt), 562, 598
Window (java.awt) heavyweight, 565
Window (java.awt) top-level screen-component, 564, 598
Window as source of event-objects, 578
window of an applet, 400
window of an applet (security-mark), 475
window or frame (as modal screen-component), 540
window or frame (GUI container), 512, 573
window or frame (GUI-context), 44, 263, 509
window or frame (size-change), 573
window, activated-, 577
window, concept of activated-, 509
window, concept of top-level-, 509
window, dialog-, 540
window, terminal-, 44, 54, 55, 64
window, terminal- (and appletviewer-command), 396, 431, 613, 641
window, terminal- (and CTRL-C-command), 432
window, terminal- (and jar-command), 326, 645
window, terminal- (and java-command with jar-option), 327
window, terminal- (and java-command with security-option), 462
window, terminal- (and java-command with String-arguments), 236
window, terminal- (and java-command with verbose-option), 250, 434
window, terminal- (and java-command with version-option), 59, 65, 250
window, terminal- (and java-command), 51, 65, 431, 505, 619
window, terminal- (and java-command, triggers main()-method), 116, 127, 142
window, terminal- (and javac-command), 65, 72, 431
window, terminal- (and javadoc-command), 324
window, terminal- (and javap-command), 642
window, terminal- (and keytool-command), 489
window, terminal- (and policytool-command), 458
window, terminal- (and RMI-related commands), 418
window, terminal- (and servlet-related commands), 407
window, terminal- (and Uniform Resource Locator (URL)), 373
window, terminal- (as single user interface), 508
window, terminal- (its current directory), 48
window, terminal- (Linux operating-system), 62
window, terminal- (Mac OS X), 61
window, terminal- (multiple- for multiple JREs), 52, 231
window, terminal- (typical java-commands), 37
window, terminal- (Windows operating-system), 63
window- or frame-states (concept), 509
window-closing-box, 581
window-object, 263
window-object, anonymous-, 159
WindowEvent (java.awt.event), 573, 575, 578
WindowListener (java.awt.event), 578, 581
World Wide Web (WWW), 372
wrapped variable of primitive-type, 256
Wrapper, 169
wrapper-class, 169, 171
wrapper-class (Float, Double), 86, 92
wrapper-class (GuardedObject), 474
wrapper-class (Integer), 85
wrapper-class (Number), 256
wrapper-class (Void), 257
wrapper-stream, 292
wrapping and unwrapping, 256
write()-method (java.io.FilterOutputStream, java.io.FilterWriter), 302
write()-method (java.io.OutputStream, java.io.Writer), 127, 295
write()-method (java.io.OutputStream, java.io.Writer) abstract, 296
writeObject()-method (custom-defined), 318
writeObject()-method (java.io.ObjectOutputStream), 308, 309
Writer (java.io), 304, 310
Writer (java.io) abstract, 296
www (as subdomain), 360, 374
WWW (World Wide Web), 372

XML (eXtensible Markup Language), 410, 414
XOR-operator (^ , eXclusive OR), 84

yield()-method (java.lang.Thread), 204, 211–213, 232

ZIP, 326
ZIP-compression, 326, 522
ZIP-file, 240
ZIP-file (and applet), 403, 614
ZIP-file (and directory-path), 247
ZIP-file (in Java API), 250
ZipFile (java.util.zip), 250