

# **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](http://www.chartingJava.org) and [www.chartingJava.com](http://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](mailto:haiko@schmarsow.de))

ISBN 3-00-009429-6



# Contents

<b>Preface</b>	<b>17</b>
This Book . . . . .	18
Structure . . . . .	18
Content . . . . .	18
Limitations . . . . .	20
The Reader . . . . .	20
The Authors . . . . .	21
Conventions . . . . .	21
 <b>I. Computer-Models</b>	 <b>23</b>
<b>Elementary Computer-Models</b>	<b>27</b>
A Computer-Model for Programmers . . . . .	28
Computers with Multiple Processors and Networks . . . . .	30
Summary: Computer-Models . . . . .	30
A Computer-Model for Hardware Technicians . . . . .	30
 <b>Processor-Commands, Languages</b>	 <b>33</b>
Bits and Bytes used For Coding Human Readable Symbols and Numbers . . . . .	33
Commands on Processor-Level: Machine-Language and Assembler . . . . .	35
 <b>Languages for easier Coding</b>	 <b>37</b>
Language-Processing: Interpreting, Compiling, Virtual Machine . . . . .	38
Languages: High-Level-Concepts . . . . .	40
 <b>Operating-Systems</b>	 <b>43</b>
File-Systems . . . . .	46
 <b>II. The Java Runtime Environment</b>	 <b>49</b>
<b>Installing Java Development Kits (JDKs)</b>	<b>59</b>
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
<b>How Java Code is Run</b> . . . . .	<b>65</b>
Compiling Source-Code . . . . .	65
Running Programs/Applications . . . . .	65
Running Applets . . . . .	67
 <b>III. Basic Programming I</b>	
<b>The Language and Utilities</b> . . . . .	<b>69</b>
Hello World Program . . . . .	71
 <b>Data-Storage — Data-Structures</b> . . . . .	<b>73</b>
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
 <b>Processing Data</b> . . . . .	<b>89</b>
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
<b>Structuring Code: Proceduralization</b> . . . . .	<b>115</b>
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
<b>Structuring Code and Data: Object-Orientation</b> . . . . .	<b>131</b>
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 <code>java.lang.Object</code> . . . . .	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
<b>Structuring Code: Multithreading</b> . . . . .	<b>197</b>
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 <code>run()</code> -Method . . . . .	207
<code>run()</code> -Method in Anonymous Class . . . . .	208
Thread-Object and <code>run()</code> -Method in Different Classes . . . . .	209
Thread-Object is Member in Class of <code>run()</code> -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
<b>Concept and Java: Aggregation of Data</b>	<b>233</b>
Java: Arrays and Array-Type . . . . .	234
Java: Vectors . . . . .	236
Java: StringTokenizer . . . . .	237
Java: Collection-Framework . . . . .	237
Transforming Aggregates . . . . .	237
<b>Structuring Code: Packaging</b>	<b>239</b>
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
<b>Standard Code-Libraries: Pre-programmed Packages</b>	<b>249</b>
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
<b>IV. Basic Programming II</b>	
<b>Internal Communication</b>	<b>267</b>
Java: Communication: Introducing Classes across Files of Source-Code . . . . .	269
<b>Communication: Messaging and Streaming</b>	<b>271</b>
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

<b>Code Structuring: Messaging</b>	<b>279</b>
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
<b>Concept: Streams in Java</b>	<b>291</b>
<b>Java: Streams</b>	<b>295</b>
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
<b>Code-Structure: Software-Components</b>	<b>311</b>
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
<b>Persistence</b>	<b>321</b>
<b>Documenting and Archiving Code</b>	<b>323</b>
Comments and Documentation	323
Java: javadoc	324
Java: jar-Tool for Archiving	325
Java: Running Java Byte-Code	326
<b>Summary: Java Messaging</b>	<b>329</b>
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

<b>Summary: Java Language</b>	<b>333</b>
<b>V. Communication between Computers Networks and Distributed Computing</b>	<b>335</b>
<b>Concept: Classification of Computer-Networks</b>	<b>339</b>
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
<b>Concept: Logical Connections, Virtual Networks</b>	<b>343</b>
<b>Concept: Data-Transfer with Packets and through Channels</b>	<b>345</b>
<b>Concept: Protocols and their Layer-Model</b>	<b>347</b>
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
<b>Networking-Protocols</b>	<b>357</b>
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
<b>Application-Protocols (HTTP, FTP, . . . ), Types and Network Applications</b>	<b>367</b>
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
<b>Triggering: URL- Network-wide Invoking of Application-Services</b>	<b>373</b>
Uniform Resource Locator (URL) — An Introduction . . . . .	373
URL-Encoding . . . . .	375
<b>Summary: Internet- and Transport-Protocols, Addresses, Domain-Names</b>	<b>377</b>
ORGANIZE: Your Network-Connection . . . . .	378

<b>Java: Programming in Networks</b>	<b>381</b>
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
<b>Summary: Java Communication</b>	<b>421</b>
<b>Concept: Distributed Computing and Programming</b>	<b>423</b>
 <b>VI. Java Runtime Environment and Security</b>	 <b>425</b>
<b>Concept/Java: Scope, Access-Areas, Name-Spaces</b>	<b>429</b>
<b>Java: Runtime Environment</b>	<b>431</b>
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
<b>Security: Basic Concepts</b>	<b>447</b>
Java Language- and Java Compiler-Security . . . . .	450
Java Runtime Security . . . . .	451
<b>Security: Policy-Based</b>	<b>453</b>
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
<b>Security-Related Coding of Data</b>	<b>477</b>
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/Authenticating 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

<b>VII. Single User Interface</b>	<b>503</b>
<b>Concept: User Interfaces</b>	<b>507</b>
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
<b>Concept: Programmer's View of User-Interface (UI)</b>	<b>515</b>
Program-Structure: Model View Controller (MVC) . . . . .	515
<b>Screen Output</b>	<b>519</b>
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

<b>Event-Communication - Mouse and Keyboard</b>	<b>571</b>
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
<b>Interaction: Screen, Keyboard and Mouse</b>	<b>595</b>
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
<b>Java: Applets - Applet Runtime-Environment</b>	<b>613</b>
Triggering And Configuring an Applet-Run . . . . .	614
Class-Hierarchy and Class-Structure of Applets . . . . .	615
A Simple AppletViewer . . . . .	619
<b>Summary: Graphical User Interfaces (GUIs)</b>	<b>621</b>
<b>Outlook: Single User Interface</b>	<b>623</b>
<b>VIII Appendix</b>	<b>625</b>
<b>Reflection, Introspection and Profiling</b>	<b>629</b>
Classes . . . . .	629
Objects . . . . .	630
System . . . . .	631
Applets . . . . .	631
Servlets . . . . .	631

<b>How to read Programs?</b>	<b>633</b>
<b>How to write and check Programs?</b>	<b>637</b>
If Something goes Wrong . . . . .	637
<b>How to make Programs faster?</b>	<b>639</b>
<b>Further Questions</b>	<b>641</b>
<b>List: Prototypes of Programs</b>	<b>645</b>
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 <code>java.io</code> ) . . . . .	647
Communication: Component/Bean-Based Programming, Documentation . . . . .	648
Communication: Computer-to-Computer: IP (Internet Protocol)-Networks (package <code>java.net</code> ) . . . . .	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-Processing	649
Single User Interface: Graphical User Interface (GUI) (package <code>java.awt</code> , <code>javax.swing</code> ) . . . . .	650
Appendix . . . . .	651
<b>List: Mentioned Packages, Classes and Interfaces</b>	<b>653</b>
<b>List: Syntax-Diagrams</b>	<b>655</b>
Backus-Naur Form (BNF) . . . . .	655
Syntax- or Railroad-Diagrams . . . . .	655
List . . . . .	656
<b>Little Classification of some Notions</b>	<b>657</b>
<b>Sources</b>	<b>659</b>
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**<sup>1</sup> 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.


---

<sup>1</sup>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

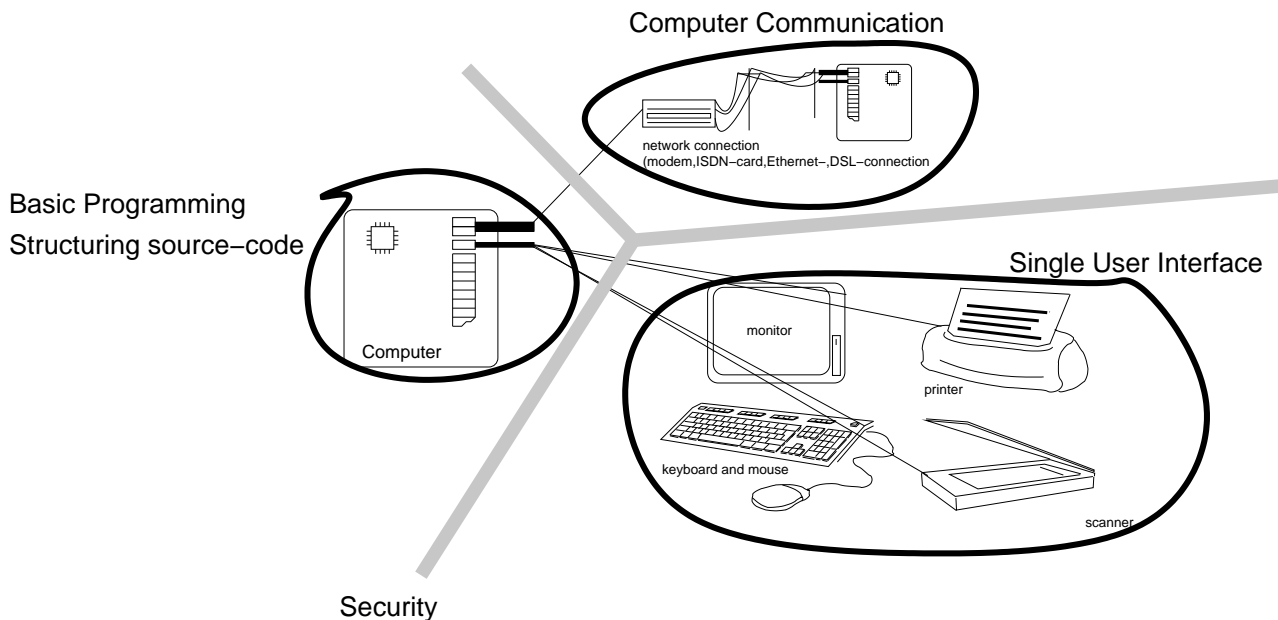
The text has been structured into *conceptual sections* and *Java-specific sections*. Conceptual sections describe general notions and give an overall idea of the content. The knowledge of these sections may be also applicable in non-Java contexts. The *Java-specific sections* demonstrate the realization 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, databases**, 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<sup>2</sup>-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).

<sup>2</sup>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 (→), 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

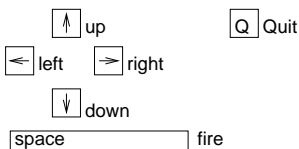
Elementary Computer-Models — First, this table shows different ways to use a computer:

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

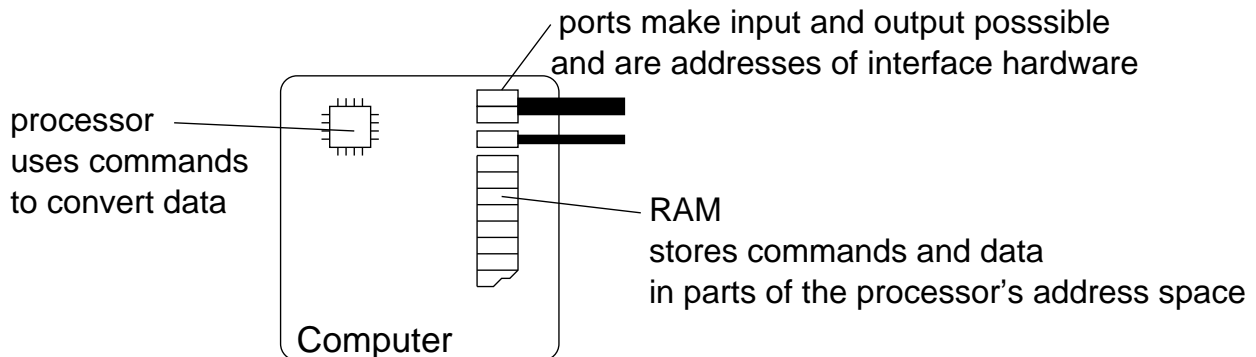


Some computing models are simple, like a computing model for a video game. Although the computing-model for the video game tells what can be done, it doesn't reflect complexity and richness of the video display and it does not tell how to acquire the skill to master the game.

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:



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<sup>3</sup> 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.

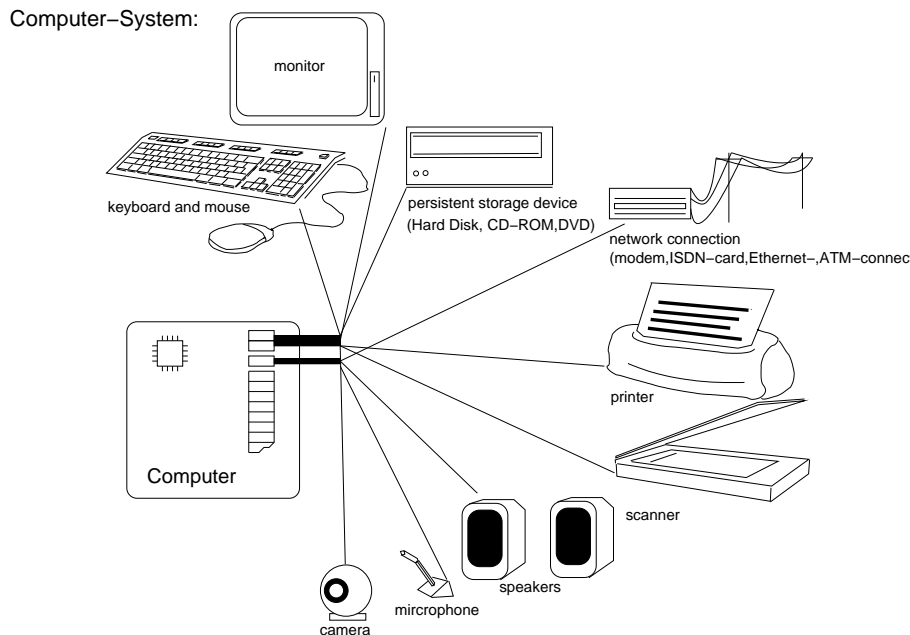
**CPU:** Acronym for Central Processing Unit. 1. The processor(s) of a computer; the part of a computer-hardware that does the main data-manipulation. 2. A Central Processing Unit consists of the microprocessor(s), the Random Access Memory (RAM) and busses that connect both with Input/Output interface-hardware. 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).

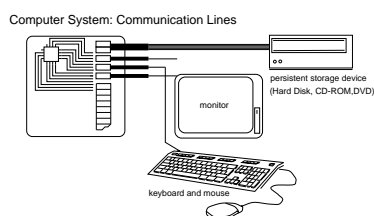
The command-sequences, that make the processor work on the data, are stored together with the data-to-be-manipulated in the Random Access Memory (RAM).

**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.

<sup>3</sup>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. ("silicone" denotes a rubbery compound of silicon)



A **computer-system** is made of the computer itself and additional devices like low data-rate input/output computer- (I/O) devices, for example keyboard and mouse, and higher data-rate I/O devices like monitor, storage system 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.



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

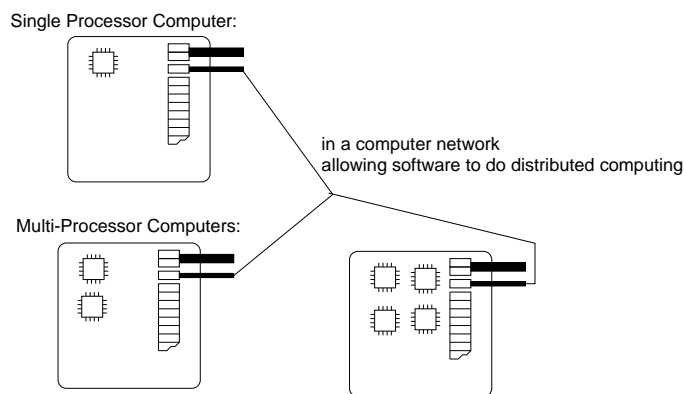
Next to the computer's internal bus that connects the processor and the Random Access Memory, there are communication-"lines" *within* the computer: The PCI (Programmable Communication Interface) with up to 64 parallel data-lines yielding a data-flow of up to 132MB per second. The EIDE (Enhanced Integrated 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).



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 loses 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 frequency 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 Mega-Hertz 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.

