Charting Java

Mary Cosway and Andrea di Pietro

August 14, 2002

Special Thanks for the support given by Vincent and Jerome.

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

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

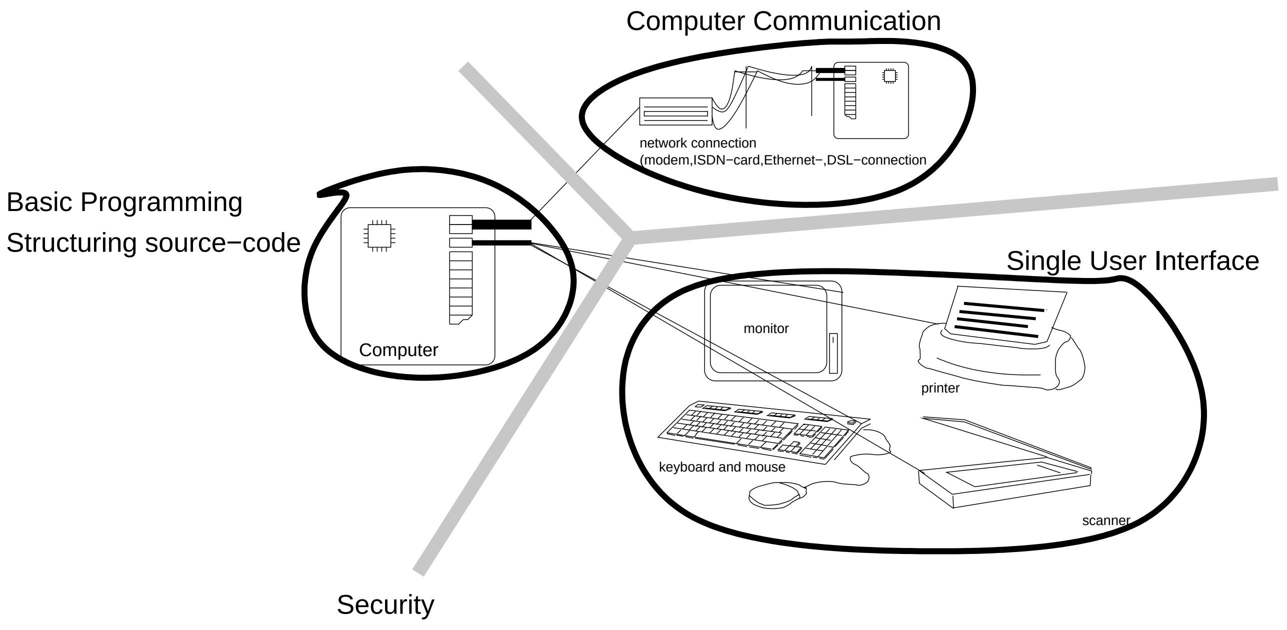
The text has been structured into *conceptual sections and Java-specific sections*. Conceptual sections describe general notions and give an overall idea of the content. The knowledge of these sections may be also applicable in non-Java contexts. The *Java-specific sections* demonstrate the realization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

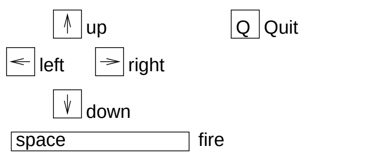
Elementary Computer-Models — First, this table shows different ways to use a computer:

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

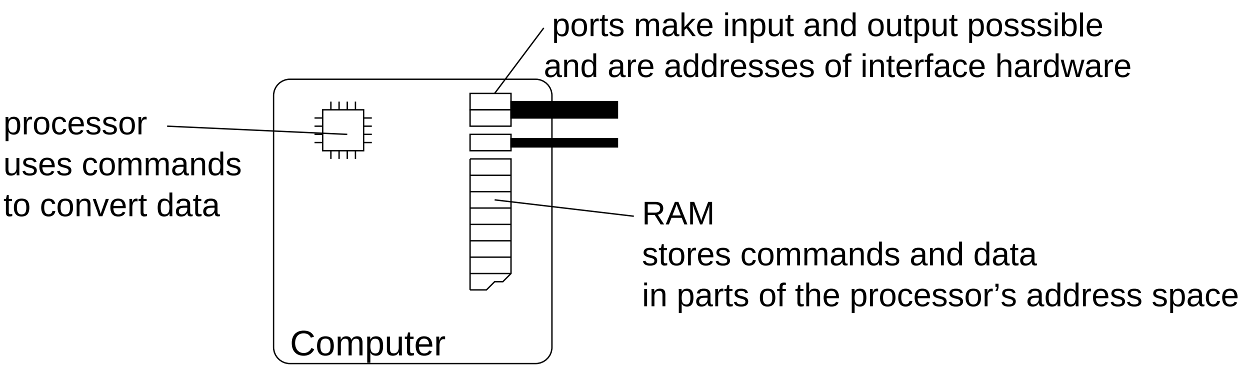


Some computing models are simple, like a computing model for a video game. Although the computing-model for the video game tells what can be done, it doesn't reflect complexity and richness of the video display and it does not tell how to acquire the skill to master the game.

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).

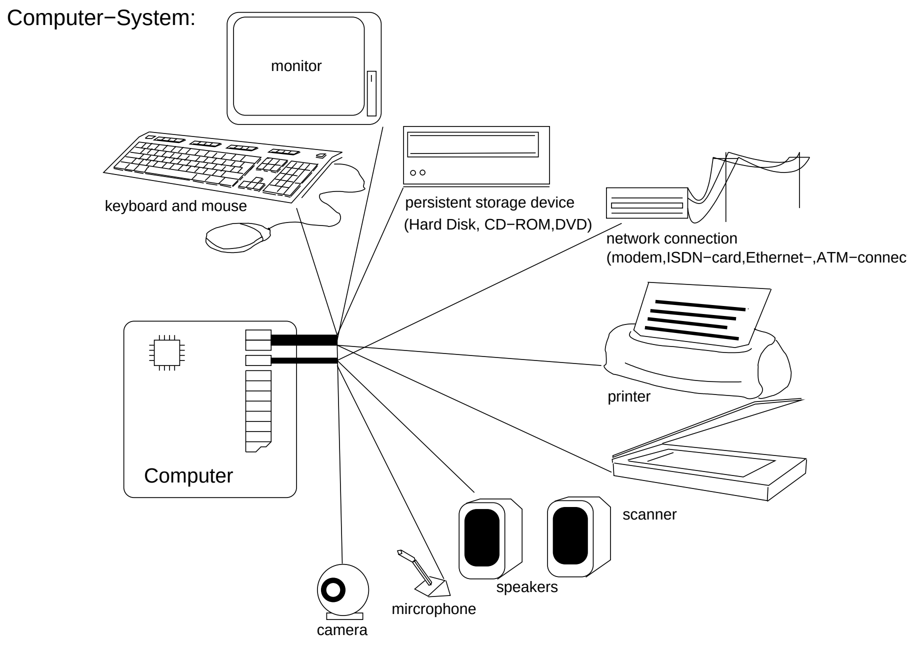
Processor: The piece of hardware inside a computer that manipulates the data it fetches from the Random Access Memory or hardware-ports. After the processor has fetched and manipulated the data, the new data is put back into the Random Access Memory or written into a port that represents a connection 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).

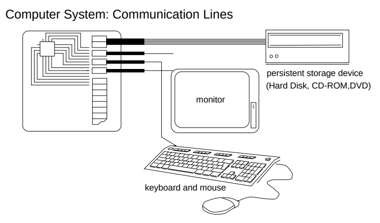
RAM: Acronym for Random Access Memory. Often simply denoted "memory". Stores data, which includes coded information and sequences of processor-commands. Generally, the processor can access each unit of information of the Random Access Memory directly. Access to data usually happens fast (in comparison 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:

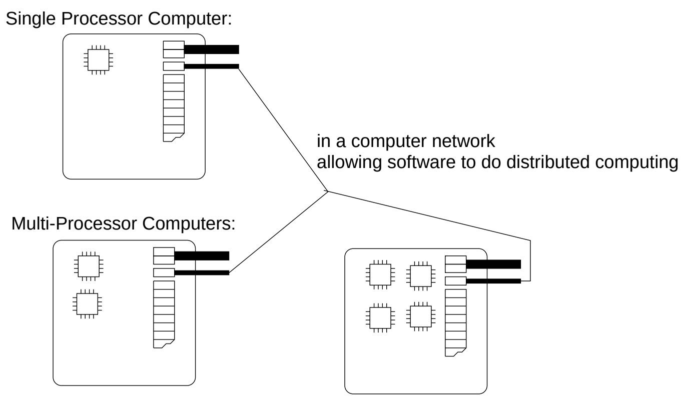
year	processor-speed (roughly in commands per second):	Random Access Memory size:
1990	20MHz	4MB
1998	200MHz	40MB
2002	2000MHz	400MB

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 multithreaded 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.

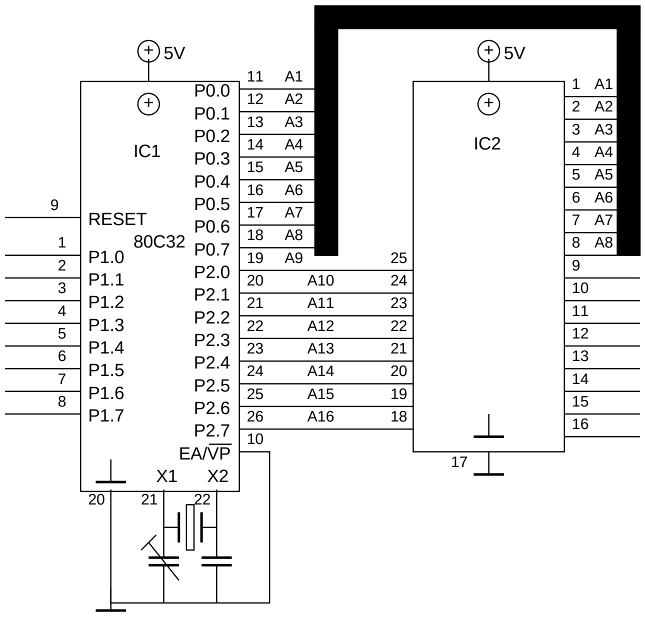
A Computer-Model for Hardware Technicians

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


5V
IC1
80C32
RESET
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
P2.0
P2.1
P2.2
P2.3
P2.4
P2.5
P2.6
P2.7
EA/VP
X1
X2
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
A16
IC2
5V