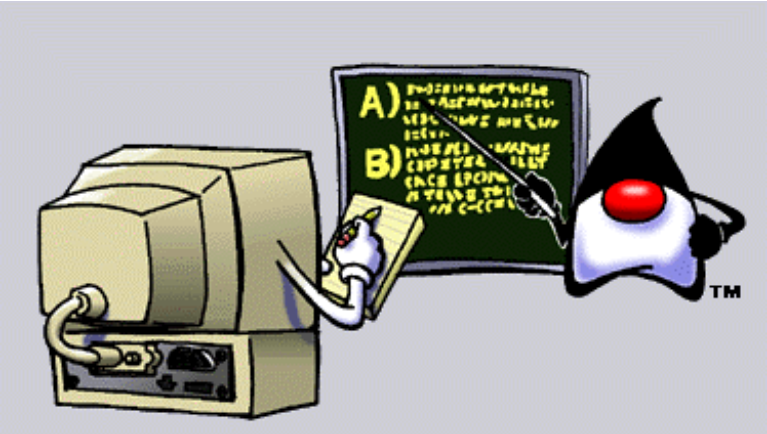
In diesem Kapitel

Rechner und Programmier Grundlagen

1.1. Über diesen einführenden Teil des Kurses

Computer und Grundlagen der Programmierung dient sozusagen dem Warmlaufen: mit möglichst wenig Voraussetzungen werden wir (im Schnellzugstempo) die Grundlagen



© Sun Microsystems

wiederholen, die wichtig sind für das Verständnis der modernen Objekt Orientierten Programmierung, wie sie in Java praktiziert wird.Dieser Teil kann also für folgende Personen sinnvoll sein:

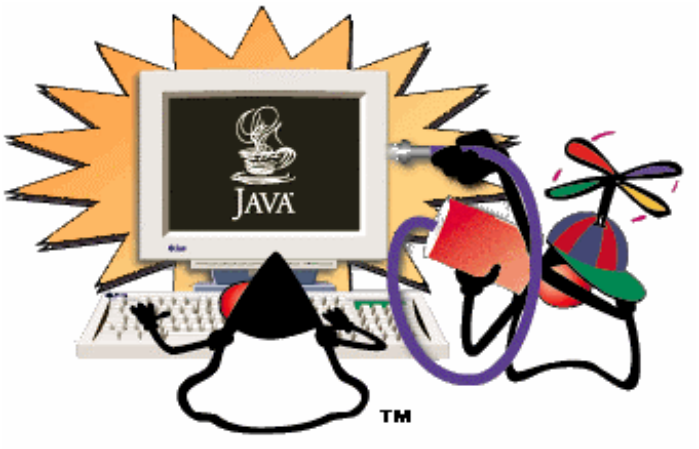
- für Personen mit wenig oder keinen Vorkenntnissen
- für Personen, welche die Grundlagen wiederholen möchten oder müssen
- für Personen, die sich intensiver mit der Programmierung befassen möchten, aber zur Zeit nicht "in der heissen Suppe" löffeln.

Bemerkung 1 Ziel dieses Teils des Kurses

ist es nicht, bereits Expertenwissen zu vermitteln, sondern eventuell fehlendes Wissen zu ergänzen.

1.2. Einleitung

Herzlich willkommen zum Kurs "Rechner- und Programmier-Grundlagen".
Dieser Teil des Kurses dient der Abstimmung der Terminologie, also der Schaffung einheitlicher Begriffe und wiederholt sehr kurz die grundlegenden Konzepte des Rechneraufbaus und der Programmierung, erklärt die Funktionsweisen und Bedeutung der einzelnen Komponenten.

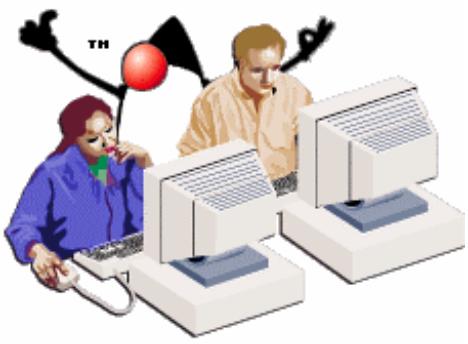


In diesem Teil werden wir folgende Themen behandeln:

- Standard- Komponenten eines Rechners
- Zusammenspiel von Hardware und Software
- Programmiersprachen
- grundlegende Programmierkonzepte
- Übersicht über die Programmierung
- Java™ als Programmiersprache
- Software Entwicklung
- Objekt Orientierte Konzepte
- Software Entwicklungsmodelle

1.2.1. Lernziele

Nach dem Durcharbeiten dieser Kurseinheit sollten Sie



- die Grundfunktionen und das Zusammenspiel von Hardware und Software verstehen
- eine Idee haben, wie sich die Programmiersprachen entwickelt haben
- unterscheiden können zwischen Objekt Orientierten und prozeduralen Programmiersprachen
- erkennen können, dass Java™ einen echten Fortschritt zu andern Programmiersprachen darstellt.
- den Software Lebenszyklus und unterschiedliche Methoden der Software Entwicklung kennen und verstehen
- erkennen, warum der Objekt Orientierte Weg der vielversprechendere ist, nach heutigem Wissensstand.

1.3. Modul 1 : Rechner-Komponenten und -Prinzipien

In diesem Modul wiederholen wir die Grundlagen des Rechneraufbaus, benennen also die grundlegenden Komponenten eines Rechners, seine Hardware und seine Software. Aber wir gehen auch auf Java ein und beleuchten, warum diese Programmiersprache und die gesamte Java Technologie so populär ist und welche Vorteile sie gegenüber anderen Programmiersprachen besitzt.

Falls Sie bereits Kenntnisse haben, sollte es sich dabei um eine einfache Wiederholung handeln. Sonst sollten Sie allenfalls aktiv werden und sich melden, damit Sie nicht bereits in dieser einführenden Einheit ins Strudeln kommen.

Als erstes müssen wir die grundlegenden Operationen eines Rechners verstehen. Wie empfängt ein Rechner Informationen, wie verarbeitet er sie, wie speichert er Informationen und wie stellt er diese dar.

Ohne diese Grundkenntnisse werden Sie kaum in der Lage sein, je ein sinnvolles Programm schreiben zu können und damit Probleme zu lösen.

1.3.1. Module Zielsetzungen

Nach dem Durcharbeiten diese Moduls sollten Sie in der Lage sein,

- die wichtigsten Komponenten eines Rechners eindeutig zu identifizieren
- beschreiben zu können, wie die wichtigsten Komponenten des Rechners zusammenarbeiten, um die Aufgabe des Rechners erfüllen zu können.
- beschreiben zu können, wie die Rechnerkomponenten mit der Software zusammenarbeiten, wo die Software gespeichert wird, wie sie transportiert wird und dadurch einsetzbar wird.
- einige Vorteile von Java gegenüber konventionellen Programmiersprachen wie Delphi oder C zu erkennen, insbesondere den Vorteil, der sich dadurch ergibt, dass Java Programme auf unterschiedlichen Betriebsystemen lauffähig sein können.

1.3.1.1. Orientieren Sie sich selbst

Bereiten Sie sich für das Durcharbeiten des Moduls mit folgenden Fragen vor:

- sind Sie in der Lage zu beschreiben, wie die verschiedenen Rechnerkomponenten zusammenarbeiten?
- welche Rolle spielt das Betriebsystem beim Erledigen folgender Aufgaben:
  - Memory Management
  - Dateiverwaltung
  - Eingabe / Ausgabe
  - Einlesen der Anwendungssoftware und wechseln zwischen unterschiedlichen Applikationen?
- welche Rolle spielt die konkrete Wahl der Programmiersprache beim Zusammenspiel zwischen den unterschiedlichen Komponenten?
- welchen Vorteil bietet Java im Vergleich zu anderen Programmiersprachen?
- inwiefern können unterschiedliche Komponenten einfacher miteinander zusammenarbeiten, wenn die Software in Java geschrieben wurde?

Was wissen Sie bereits? Versuchen Sie die obigen Fragen vor dem Weiterlesen bereits aus Ihrem jetzigen Wissenstand heraus zu beantworten. Sie können damit leicht herausfinden, was Sie konkret noch lesen müssten oder was Ihnen bereits sonnenklar ist.

Lesen Sie die nächsten Seiten, falls überhaupt, mit den obigen Fragen im Kopf und versuchen Sie die Antworten aus dem Text herauslesen zu können.

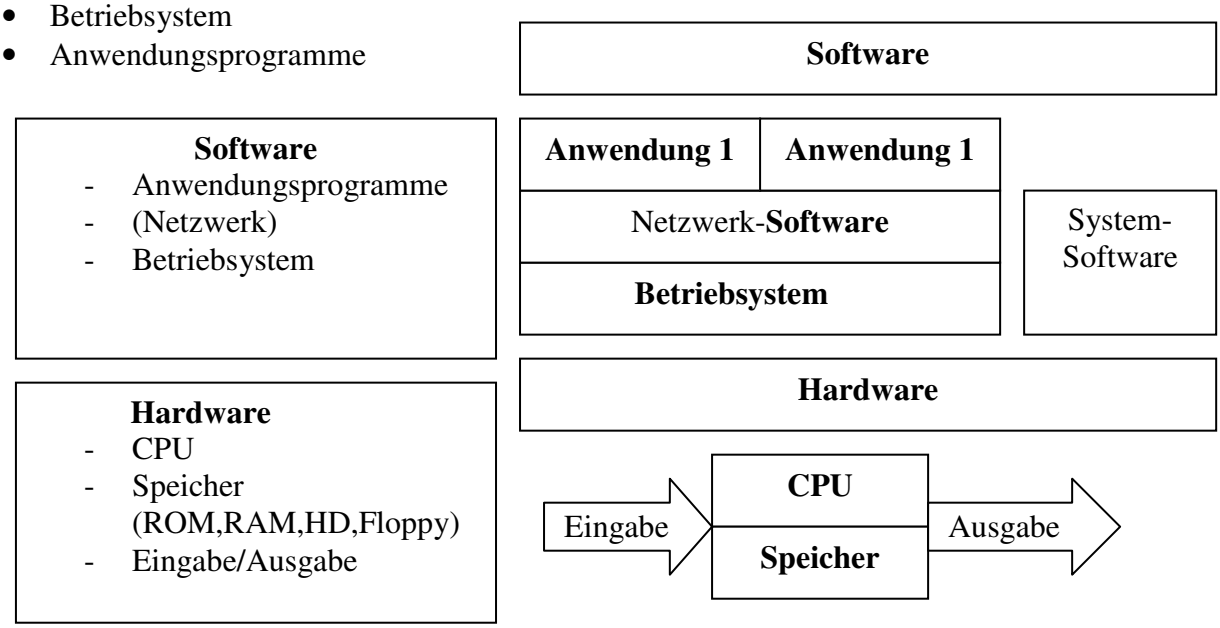
1.3.2. Komponenten eines Rechners

Ein Rechner besteht aus verschiedenen Hardware und Software Komponenten.

1. **Hardware Komponenten umfassen:**
   - Eingabegeräte
   - (Haupt-) Speicher
   - Verarbeitungseinheit / CPU
   - Ausgabegeräte

Ein Rechner benötigt Stimulation / Anregung in Form von Software - Instruktionen, Prozeduren, Objekte, Daten und so weiter - um zu funktionieren und zusammenzuarbeiten.

2. **Software Komponenten umfassen:**
   - Betriebsystem
   - Anwendungsprogramme



Hardware
Die Hardware Komponenten kann man in vier Kategorien einteilen.

1. **Eingabegeräte (Input)**
   Maus; Tastatur, Scanner und Touch Screen

2. **Ausgabegeräte (Output)**
   Bildschirm, Drucker, Plotter und irgendwelche Robotergeräte (Messroboter)

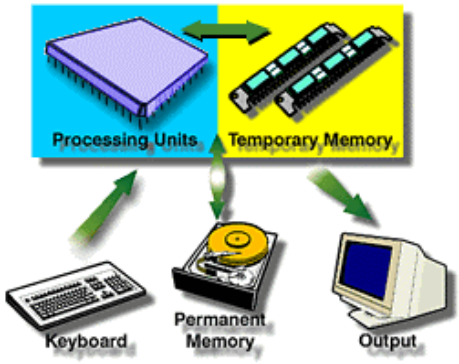
3. **Speicher**

   Kurzzeitspeicherung (temporary memory)
   (gespeicherte Information geht beim Ausschalten verloren)
   Speicher mit Zufallszugriff (random access memory : RAM),

   Langzeitspeicher (permanent memory)
   (gespeicherte Informationen stehen langfristig zur Verfügung)
   Festplatten, Floppy- Disks CD-ROM und optische Platten

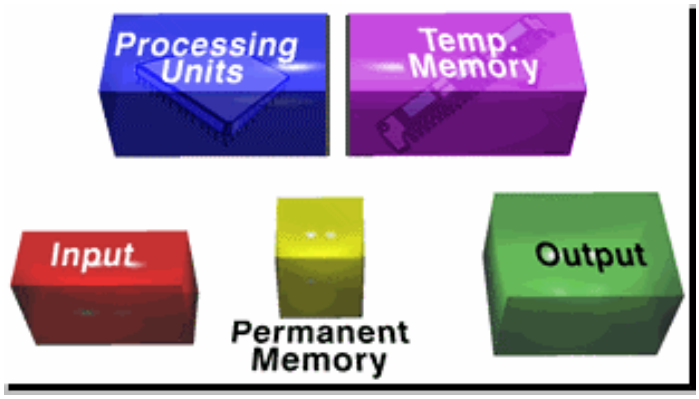
4. **Verarbeitende Einheiten (processing unit)**
   Die intern vorhandenen Recheneinheiten (central processing unit: CPU) eventuell mit speziellen Zusatzprozessoren (für Eingabe / Ausgabe- Optimierung, Grafikprozessoren, Spezialprozessoren für schnellere Arithmetik)

   Das folgende Bild zeigt das Zusammenspiel dieser Komponenten:



1.3.3. Selbsttest

Betrachten Sie das folgende Bild und ergänzen Sie den Text unten (die Fragen).



Die Lösungen stehen jeweils in den entsprechenden Fussnoten.

1. **Identifizieren Sie die korrekte Komponente:**
   Ein Rechner verfügt normalerweise über eine Art [1]....., wie Festplatten, um Daten zu speichern, welche nicht gelöscht werden sollen, falls die Stromversorgung ausgeschaltet wird.

2. **Identifizieren Sie die korrekte Komponente:**
   Die CPU, ALU (arithmetisch logische Einheit) und die CU (control unit) sind Bestandteil der [2].....

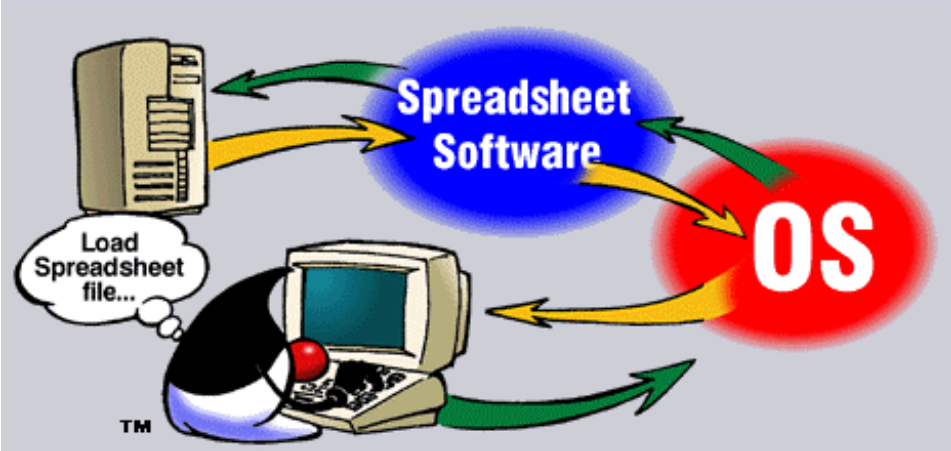
3. **Identifizieren Sie die korrekte Komponente:**
   Der Monitor, ein Bildschirm oder ein Plotter werden aus [3]..... Komponenten bezeichnet.

[1] temporary memory
[2] processing units
[3] output

1.3.4. Rechnersoftware

Software Programme bestehen aus Instruktionen, welche der Programmierer kreiert, um den Rechner kontrollieren zu können. Die Software wird normalerweise im permanenten Speicher de Rechners gespeichert. Für die schnellere Ausführung wird die Software jeweils ins RAM geholt, steht also für den schnellen Zugriff temporär zur Verfügung. RAM Zugriffe sind wesentlich schneller als Festplattenzugriffe, da die Festplatte als mechanisches, magnetisches und / oder optisches Gerät im Vergleich zu den RAM Chips deutlich langsamer ist.

Software wird in verschiedene Kategorien eingeteilt:



1. **Betriebsystem (operation system : OS)**
   Software, welche als Schnittstelle zwischen den Anwendungen und der Hardware eingesetzt wird, wie beispielsweise Solaris™ oder Windows™ . Diese Software koordiniert die Aktivitäten der unterschiedlichen Hardware Komponenten. Das OS kontrolliert beispielsweise die Eingabe- und Ausgabe- Einheiten und ist für die Zuteilung der Prozessorleistung zuständig.
   Nach dem Starten (booten) des Rechners läuft das Betriebsystem dauernd, und wird zur Schnittstelle zwischen der Hardware und der Software und teils des Users. Sie kommunizieren mit dem Betriebsystem mit Hilfe von Systembefehlen *(system commands*).

   Beispielsweise muss das Betriebsystem entweder einen Befehl ausführen oder ein Programm starten, falls Sie
   in Windows den Befehl `dir` (Verzeichnis anzeigen; directory)
   in Unix ® den Befehl `ls` (list)
   eintippen und an das Betriebsystem schicken.

   In diesem Beispiel ist der Befehl (in Windows) Teil des Kerns des Betriebsystems, es muss also kein spezielles Programm gestartet werden. Bei komplexeren Befehlen muss häufig ein spezielles Hilfsprogramm, mit dem dieser Befehl implementiert wurde, gestartet werden.

2. **Applikation**
   Software oder Rechnerprogramme laufen "auf" dem Betriebsystem. Beispiele kennen Sie genug: Textverarbeitung, Tabellenkalkulation und vieles mehr.

   Eine Applikation ist ein selbständiges Programm, welches von einem Betriebsystem ausgeführt, gestartet werden kann. Das Betriebsystem kontrolliert die der Anwendung zugeteilten Ressourcen (CPU, RAM, ...) sowohl falls das Programm interaktiv oder im Hintergrund (Batch) abläuft. Batch Programme laufen in der Regel immer dann, wenn die Rechnerressourcen nicht durch Echtzeitprozesse benötigt werden. Je nach Konfiguration des Betriebsystems kann sich dies jedoch ändern (ein System kann als Batch System ausgelegt werden, speziell während der Nacht).

1.3.4.1. Client Server Software

Client Server Software beschreibt eine spezielle Art und Weise, wie Applikationen und Betriebsystem(e) miteinander zusammenarbeiten.

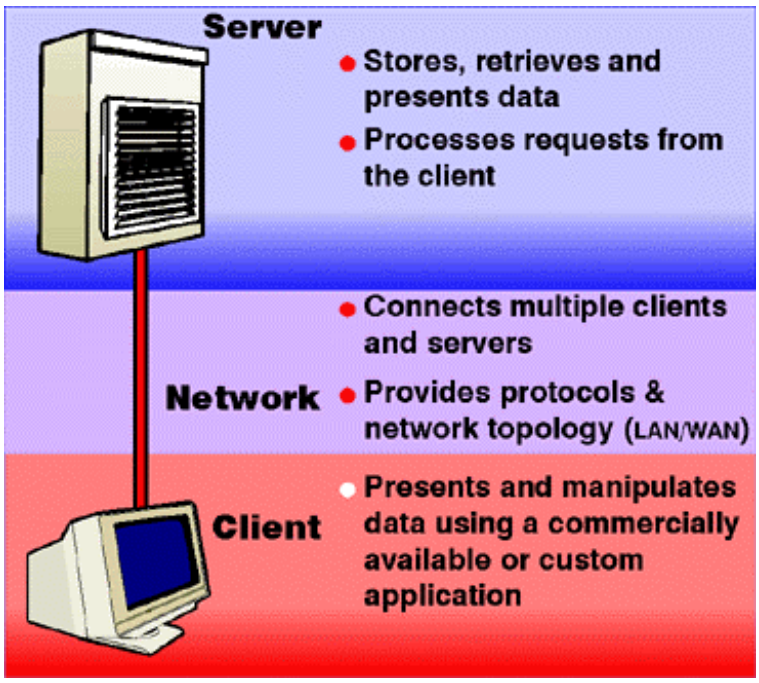
Server

Der Server speichert, holt und schützt die Daten. Ein Server läuft im Hintergrund ist immer aktiv, wartet auf Clients, also Kunden, die mit ihm kommunizieren wollen, in der Regel weil sie seine Ressourcen benutzen möchten.

Client

Ein Client kann eine Workstation oder ein PC sein. Auf Clients laufen in der Regel Echtzeit-Programme, mit denen der Benutzer arbeitet. Beispielsweise wartet ein Web Browser auf dem Client auf Eingaben des Benutzers; diese werden dann weitergeleitet an einen Web Server, der die Anfragen bearbeitet und falls nötig dem Client einen Feedback liefert, beispielsweise eine Web Seite.

Das Client Server Modell behandelt Server und Workstation als intelligente programmierbare Geräte und nutzt deren Rechnerkapazitäten. Die Ausführung einer Aufgabe wird dort gemacht, wo sie am effizientesten ausgeführt werden kann (GUI: beim Client; Datenbankzugriffe: beim Server). Die Arbeitslast wird also aufgeteilt, je nach Anforderungen.



Server
- speichert, holt und bereitet Daten auf
- verarbeitet Anfragen vom Client

Netzwerk
- verbindet mehrere Clients und Server
- stellt Protokolle und Netzwerktopologien (LAN, WAN) zur Verfügung

Client
- präsentiert und manipuliert Daten mit handelsüblichen oder speziell entwickelten Applikationen.

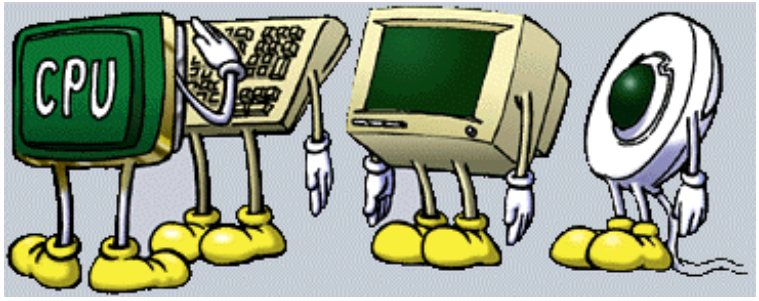
1.3.4.2. Grundlegendes Zusammenarbeiten der Komponenten

Eine typische Rechnernutzung hilft verstehen, wie die einzelnen Komponenten eines Rechners zusammenarbeiten.

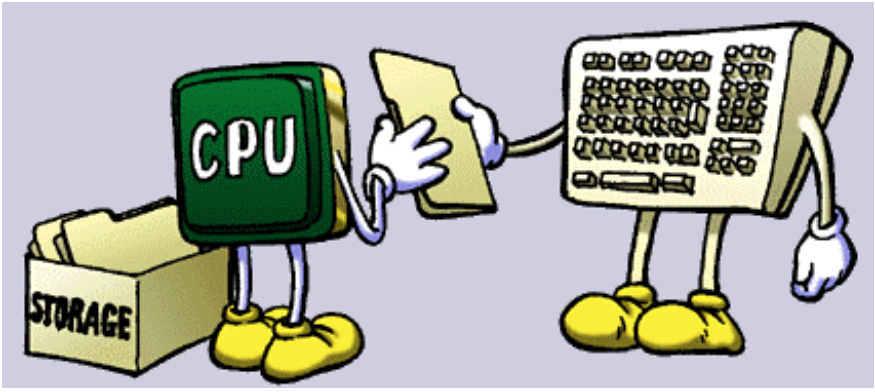
1. nehmen wir an, der Rechner sei bereits eingeschaltet und das Betriebsystem (OS) geladen (aus dem permanenten Speicher ins RAM, wenigstens teilweise).

   Jetzt führt die CPU das Betriebsystem (vermutlich) im "waiting for input" Modus. Dieser Modus überprüft letztlich lediglich die Eingabegeräte (Tastatur, Maus und ähnliche) auf neue Eingaben und kontrolliert die Bildschirmausgabe.

   Falls das Betriebsystem komplexe Berechnungen durchführen muss, wird es dafür die ALU (arithmetic logic unit) einsetzen und eine entsprechende Antwort aufbereiten.



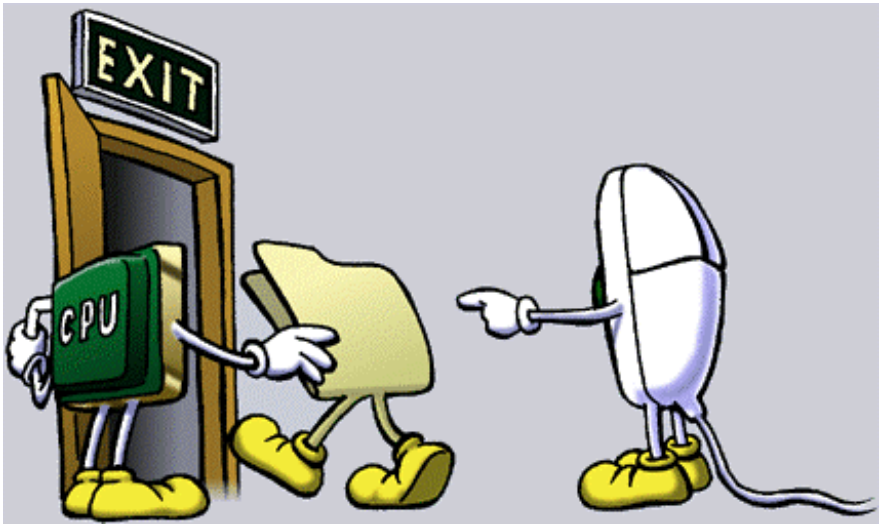
2. nun gibt der Benutzer einen Befehl mit Hilfe der Tastatur ein
   Diese neue Information wird vom Eingabegerät gelesen. Das Betriebsystem analysiert die Informationen und führt die entsprechenden Befehle, Funktionen aus.
   Beispielsweise könnte der Benutzer verlangen, dass eine bestimmte Applikation gestartet werden soll. Das Betriebsystem lädt die Applikation vom permanenten Speicher in den dynamischen Speicher, das RAM. Anschliessen teilt das Betriebsystem der CPU mit, dass die Applikation gestartet werden soll.



3. nehmen wir an die CPU ist ausgelastet, der Benutzer drückt die EXIT Taste.

   Die CPU führt gerade eine Anweisung nach der andern der bestimmten Applikation aus. Da will der Benutzer die Ausführung abbrechen und drückt beispielsweise eine EXIT Taste.

   Das Betriebsystem stellt diesen Tastendruck fest und leitet diese Eingabe an die Applikation weiter. Die Applikation verarbeitet diese Eingabe:
   benötigte Daten werden in Dateien gespeichert und die Applikation verlassen.
   Zusätzlich sendet die Applikation eine Nachricht an das Betriebsystem.
   Das Betriebsystem gibt alle für die Applikation reservierten Ressourcen frei.



Diese dreistufige Übersicht lässt viele Details weg, die im Einzelnen wesentlich sind (ein Betriebsystem ist nicht so banal); aber als Kurzwiederholung der Konzepte für diesen Kurs reicht diese Übersicht, welche sich auf einige wenige Prinzipen beschränkte, um die Wechselwirkung der Komponenten aufzuzeigen.

1.3.5. Programmiersprachen

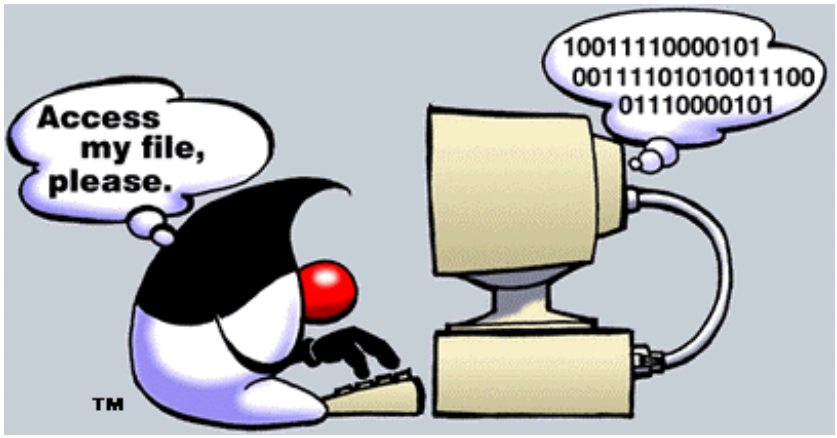
Menschen kommunizieren mittels unterschiedlicher Sprachen und Dialekte. In der Regel kommunizieren wir am besten, falls wir dieselbe Sprache sprechen. Falls wir mit jemandem sprechen wollen, der nicht unsere Sprache spricht, benötigen wir vielleicht einen Dolmetscher, einen Übersetzer.

Rechner kommunizieren auch mittels unterschiedlicher Sprachen. Aber der Rechner selbst versteht lediglich die ***Maschinensprache***, dargestellt durch eine Serie von 0en und 1en (0 steht für aus, 1 steht für ein, offen oder geschlossen, durchlässig oder nichtdurchlässig,...).

Jede Software und alle Daten werden innerhalb des Rechners in der Form von ***Bits*** (8 Bits sind ein Byte) gespeichert und manipuliert. In den ersten Rechnern entsprach on einem geschlossenen Relais, off einem offenen. Moderne Chip-basierte Rechner haben dieses Schema letztlich beibehalten - ein Bit kann zwei Werte speichern (entweder ein oder aus).

Maschinencode ist sehr mühsam zu lesen und in Maschinensprache zu programmieren grenzt an Masochismus. In der Regel muss man ausweichen und die Generierung des Maschinencodes einem Programm überlassen. Falls jemand mit einem Rechner direkt kommunizieren möchte, müsste er lernen, auf die selbe Art und Weise zu denken, wie dieser bestimmte Rechner - und mit hohem Aufwand seine Gedanken in korrekte binäre Sequenzen umwandeln. Da Maschinencode die einzige Sprache ist, die ein Rechner versteht, haben Programmierer "Hochsprachen" entwickelt (mehrere Generationen: Generation 1 : Maschinensprache; Generation 2 : Assembler; Generation 3 : die üblichen Programmiersprachen wie Java, C, Delphi und ähnliche; Generation 4 : Hochsprachen oder Makrosprachen). Damit lassen sich Applikationen besser, zuverlässiger, lesbarer und in der Regel zuverlässiger entwickeln.

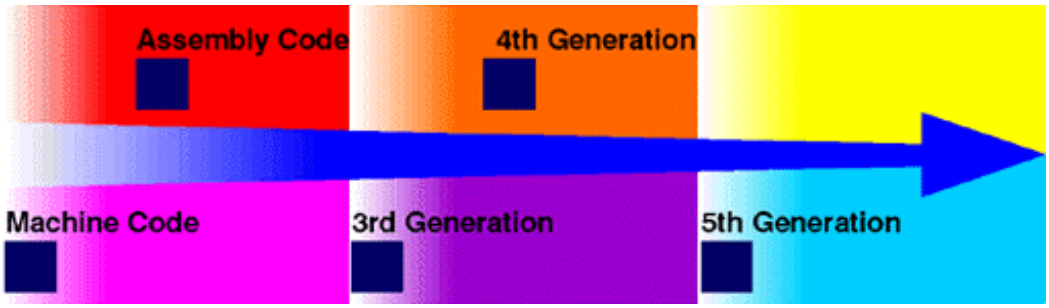
Die Schnittstelle zwischen den Hochsprachen und dem Maschinencode stellt der Compiler dar. Diese übersetzen die Hochsprache in den Maschinencode, der dann vom Rechner interpretiert bzw. verstanden werden kann. Maschinencode ist CPU spezifisch, also von der konkreten Hardware abhängig (Intel, Motorola, Sparc, ...). Die einzelnen Maschinensprachen unterscheiden sich beträchtlich und sind inkompatibel (Intel sorgt in der Regel für eine bestimmte Rückwärtskompatibilität). Jede Prozessorarchitektur benötigt also eine spezifische binäre Sequenz für die Kommunikation (der Code Generator des Programmiersystems generiert Code für die jeweilige Zielplattform, für den speziellen Prozessor).



1.3.5.1. Evolution der Programmiersprachen

Über die Jahre wurden unterschiedliche Sprachen entwickelt, um die Programmierung effizienter zu machen. Auf dem folgenden Chart sehen Sie schematisch die Entwicklung bis zur sogenannten 5th Generation, einem Ansatz bei dem man versuchte (speziell in Japan) die Programmierung durch natursprachliche Kommunikation zu ersetzen. Das MITI, das Japanische Wirtschaftministerium, finanzierte viele Projekte, ohne allzu grossen Erfolg zu erzielen.

Auf dem Chart verläuft die Zeitachse von links nach rechts.



Jede neue Generation entfernte sich jeweils weiter weg vom Maschinencode. Nachteil ist der zunehmende Speicherbedarf, eventuelle Leistungseinbussen kann man durch stärkere Prozessoren kompensieren. Zudem wird die Portabilität erhöht.

1. Maschinensprache
   Maschinencode bezeichnet man auch als erste Generation der Programmiersprachen. Es ist die tiefste, also hardwarenächste Programmiersprache und *ist die einzige Sprache, welche von Rechnern verstanden wird*. Maschinencode kann man allerdings kaum lesen und verstehen, da er lediglich aus 0en und 1en besteht.

2. Assembler
   Als nächstens wurden Assembler entwickelt. Assembler verwendet einfache Abkürzungen für Befehle (LDA für load accumulator; MOV für move, also verschieben der Daten). Ein Assembler Befehl fasst in der Regel mehrere Maschinenbefehle zusammen. Aber die Abbildung von Assembler in die Maschinensprache ist sehr direkt und damit auch sehr maschinenabhängig.

   Neuere Varianten unterstützen auch Modularisierung und Wiederverwendung; aber Assemblersprachen werden immer sehr direkt die Daten und Register manipulieren und globale Daten zulassen.

   In der Regel werden jedoch nur einfache Kontrollstrukturen unterstützt, wie Schleifen oder Sprunganweisungen. Diese Art der Programmierung eignet sich speziell für die hardwarenahe Programmierung, bei der Spezialitäten der Hardware optimal ausgenutzt werden sollen.

3. Die dritte Generation
   Diese Hochsprachen erlaubten eine in der Regel daten- oder funktions-orientierte Zerlegung des Problembereichs. Allerdings hatten die meisten Sprachen ungute Nebeneffekte, wie beispielsweise globale Daten, Daten die irgendwo im Programm verändert werden konnten und Seiteneffekte haben konnten, also Effekte in Bereichen, in denen man keinerlei Effekte erwartete.

   Programme waren in der Regel wenig flexibel und schwierig zu erweitern. Strukturierte Programmierung und Sprachen wie Pascal, Modula, Oberon, Eiffel und ähnliche Sprachen, sorgen für einige Besserung.

4. Die vierte Generation
   Moderne blockstrukturierte und objektorientierte Sprachen ermöglichen die Modularisierung der Software sowohl aus Datensicht wie auch aus Funktionssicht.

   Mit diesen Programmiersprachen kann man abstrakte Datentypen definieren, so dass komplexe Objekte modelliert werden können.

   Viert Generation Sprachen werden in der Regel interpretiert. Das heisst, ihnen steht eine Laufzeitumgebung zur Verfügung, welche das Programm auf portable Art und Weise auf unterschiedlichen Hardware- und Software- Plattformen ausführen kann.

5. Die fünfte Generation
   Bei der 5^{th} Generation verwendet man visuelle oder grafische Schnittstellen, um Programmcode zu generieren, welcher dann in der Regel in einer der dritt Generationen Sprachen erzeugt wird.

   IBM, Borland und viele mehr haben solche visuellen Programmierumgebungen entwickelt. Diese Systeme gestatten eine leichte Visualisierung der Objekte und Klassen.

   Eine weitere Entwicklung im Bereich der 5^{th} Generation waren die Anstrengungen in Japan, bei denen versucht wurde direkt natursprachliche Kommunikation zuzulassen. Weitere Ansätze, die im Moment untersucht werden, sind die Programmierung auf Grund von Gesten oder Gedanken (6^{th} und 7^{th} Generation). Diese Entwicklungen stecken noch im Anfangsstadium.

6. Portabilität
   Unter Portabilität versteht man die Fähigkeit ein Programm auf unterschiedlichen Plattformen einzusetzen, ohne dass der Quellcode modifiziert werden muss.

1.3.5.2. Die Java Programmiersprache

Entgegen vielen anderen Programmiersprachen, welche übersetzt oder maschinenspezifisch interpretiert werden, wird Java in einen Standard Instruktionsset (den Bytecode) übersetzt. Der Bytecode ist unabhängig von der Prozessorarchitektur oder dem Betriebsystem.

Dieser Bytecode kann von jedem Rechner gelesen werden, auf dem ein Java Programmiersprache Interpreter, die **Java Virtual Machine (JVM)** läuft. Diese JVM konvertiert den Bytecode in maschinenspezifischen Code, der dann auf dem Zielrechner ausgeführt wird. JVM's sind für meisten Betriebsysteme verfügbar (Windows, Linux, Solaris, AS/400, IBM Grosrechner, Lego Mindstorm, PalmPilots,...).

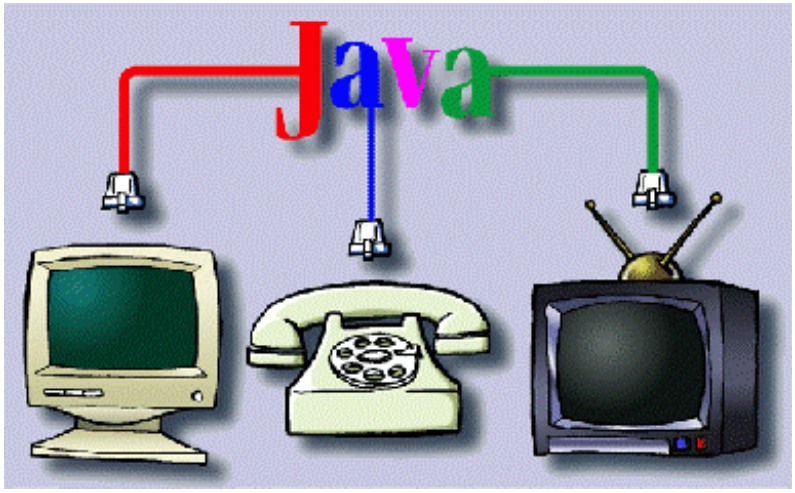
Die Idee ist einfach - Software, welche in Java geschrieben wurde, läuft fast überall, vom grössten Rechner bis zum kleinsten Gerät oder auf einem Supercomputer - ohne dass die Applikation jeweils neu für die entsprechende (Architektur-) Plattform geschrieben werden muss. Programme, welche in Java geschrieben wurden, laufen auf Windows NT, Windows 9x, dem MacIntosh oder anderen Betriebsystemen.

Das ist der Kern der Aussage **Write ONCE, Run Anywhere** für die Java Technologie.

Java Technologiekomponenten sind so konzipiert, dass sie auf einer grossen Anzahl geräte lauffähig sind, vom Telefon über den Fernseher, die Waschmaschine oder ein bestimmtes Betriebsystem.

Die Java Architektur ist:

- eine Programmier-sprache
- eine Entwicklungs-umgebung
- eine Applikations-umgebung



Die Java Platform ist ein grundlegend neuer Weg Informatik zu betreiben, basierend auf Netzwerken und der Idee, Applikationen einmal zu entwickeln und sie auf einer beliebigen Zielplattform (Rechner, Gerät, Spielzeug, Telefonen, Handies oder erst noch zu entwickelnden Geräten) laufen zu lassen.

1.3.6. Selbsttest

Die folgenden Fragen können mit wahr oder falsch beantwortet werden. Nach der Beantwortung der Aufgaben sollten Sie die Zusammenfassung lesen und zum nächsten Modul übergehen.

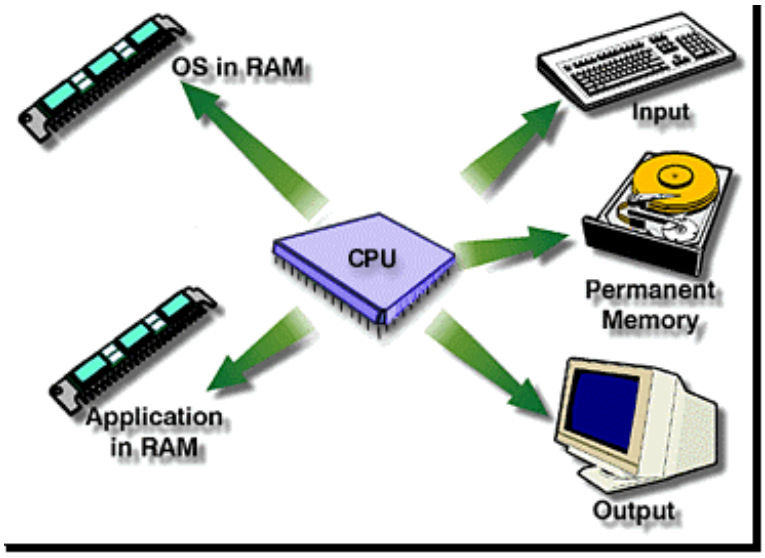
1. Welche der folgenden Komponenten ist keine Hardware- Komponente?

   a) Eingabegerät
   b) temporärer Speicher
   c) die Java Virtual Machine (JVM)
   d) Prozessoren

[4]

2. Die Komponenten der folgenden Darstellung können alle korrekt miteinander kommunizieren

   a) wahr
   b) falsch



[5]

3. Programmiersprachen können mit der menschlichen Kommunikation verglichen werden, weil Menschen miteinander (schriftlich oder mündlich) kommunizieren, und sich die Sprache im Verlaufe der Zeit verändert.

   a) wahr
   b) falsch

[6]

[4] c) die JVM
[5] a) die Tastatur kann nicht direkt kommunizieren
[6] a) wahr

4. Welche der folgenden Charakterisierungen der 4th Generation Programmiersprachen ist korrekt?

   a) es handelt sich um Hochsprachen, welche eine funktionale Zerlegung der Probleme gestatten.
   b) diese Sprachen sind blockstrukturiert, objektorientiert und gestatten eine Zerlegung der Probleme sowohl datenoriemtiert als auch funktionsorientiert.
   c) diese Sprachen haben zum Ziel eine natursprachliche Kommunikation mit dem Rechner zu ermöglichen.
   d) diese Sprachen verwendet Abkürzungen, welche sich leicht auf die Maschinensprachen abbilden lassen

   [7]

5. Programme, die in Java geschrieben wurden, laufen ohne grosse Anpassungen auf unterschiedlichen Betriebsystemen (Windows, Solaris).

   a) wahr
   b) falsch

   [8]

[7] b) OO Konzepte, blockstrukturiert
[8] a) Java Zielsetzung = Portabilität

1.3.7. Zusammenfassung

In diesem Modul haben wir einige stark vereinfachte Strukturen und Konzepte kennen gelernt, Hardware und Software, sowie Programmiersprachen.

Die Entwicklung der Programmiersprachen zeigt, dass mit Java eine neuartige Sprache entwickelt wurde, welche wesentlich auf die Vernetzung setzt und die Kommunikation unterschiedlicher Geräte als Designprinzip verwendet.

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein:

- die wichtigsten Komponenten eines Rechners zu identifizieren
- zu beschreiben, wie die wichtigsten Komponenten eines Rechners im Zusammenspiel Aufgaben erfüllen können.
- das Zusammenspiel zwischen den Rechnerkomponenten und Applikationssoftware zu erklären.
- die Vorteile von Java als Programmiersprache zu verstehen und klar zu machen, inklusive der Portabilität auf unterschiedliche Betriebsysteme.

Überlegen Sie sich:

1. welche Trends zeichnen sich in Bezug auf Speicher (RAM, Festplatten) ab?
2. welche CPU Leistungen kann man innerhalb der nächsten Jahre erwarten?
3. warum ist es überhaupt wichtig, Applikationen portabel zu entwickeln?

1.4. Module 2 : Grundlagen der Programmierung

1.4.1. Einleitung

In diesem Modul sollten Sie sich mit den Grundlagen der Programmierung vertraut machen - mit dem, was ein Programmierer tut, wie Programmiersprachen funktionieren.

Wir werden uns mit diesen Konzepten noch vertieft beschäftigen, hier handelt es sich also nur um einen Start. Dieser Modul dient also lediglich als Einstieg.

1.4.1.1. Lernziele

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein:

- zu beschreiben, wofür man eine Programmiersprache benötigt und warum man ein Programm übersetzen muss.
- die zwei gängisten Programmierparadigmen oder Modell zu identifizieren.
- zu erläutern wie traditionelle, Objekt orientierte und die Java Programmiersprachen funktionieren.
- die Hauptkomponenten eines Java Programms zu identifizieren.

1.4.1.2. Selbstbesinnung

Versuchen Sie vor dem Durcharbeiten dieses Moduls folgende Fragen zu beantworten:

- wie entwickelt und designed ein Programmierer ein Programm?
- welche typischen Probleme treten beim Entwickeln von Programmen auf?
- welche Probleme kann man mit einer Programmiersprache überhaupt lösen?
- wie funktioniert eine Programmiersprache?
- warum sind Programmierer so begeistert über Java? Was macht Java so erfolgreich?
- wie wird Software entwickelt?

Versuchen Sie sich klar zu werden, was Sie bereits wissen! Sie werden dann viele Punkte, die wir im folgenden nur skizzieren, besser und umfassender verstehen oder den Modul einfach überspringen!

Sie sollten die Fragen beim Durcharbeiten im Kopf behalten und jeweils fragen, inwiefern der Text oder eine Frage Ihnen helfen kann, die obigen Fragen zu beantworten.

1.4.2. Übersicht über die Programmierung

Langsam kommen wir unserem Hauptthema näher.

1.4.2.1. Wie Programmierer arbeiten

Haben Sie sich je gefragt, "Was *ist es*, was ein Programmierer tut?". Programmierung kommt vielem Anfängern fast mystisch vor - speziell Anfängern. Offensichtlich programmieren die Programmierer! Aber was bedeutet dies eigentlich? Warum es es nötig? Wie wird es gemacht?

Im Wesentlichen nutzen die Programmierer ihre Erfahrung mit einer *Programmiersprache*, um Anwendungen zu (be-)schreiben, welche ein spezifisches Problem lösen oder eine Aufgabe erfüllen, um Ziele zu erreichen (Datenbanken, Computerspiele, Textverarbeitung,...).

Ursprünglich wurden Programme in Maschinensprache geschrieben (also in 0en und 1en). Mit der Zeit erfanden die Programmierer höhere Programmiersprachen, welche Befehle verwenden, welche einem Menschen leichter verständlich sind. Heute sind diese höheren Programmiersprachen üblich und jeder Programmierer verwendet sie, um Programme zu erstellen. Maschinensprache wird höchstens noch für Spezialaufgaben eingesetzt.

Um eine Applikation auf einem spezifischen Rechner ausführen zu können, muss der Programmcode übersetzt werden, also in die Maschinensprache konvertiert werden. Diese Konversion wird als Interpretation oder Übersetzung bezeichnet (wir werden darauf zurück kommen).

1.4.2.2. Effektive Programmierprinzipien

Wie in anderen Gebieten und Berufen wurden bei der Programmierung bestimmte Prinzipien und Standards entwickelt, damit man qualitativ hochstehende und effiziente Programme erstellen kann.

Eine effektives Programm, eine Applikation sollte:

- **zuverlässig** (engl. *reliable*) sein - Software, welche zuverlässig ist, wird auch eine ganze Menge anderer Attribute haben: Sicherheit, Ausfallsicherheit.
  Eine zuverlässige Software wird bei einem allfälligen Ausfall idealerweise nicht gleich zu einem Totalausfall führen.

- **benutzbar** (engl. *usable*) **-** Software, welche benutzbar sein soll, muss über ein brauchbares Benutzerinterface und aktuelle Dokumentation verfügen

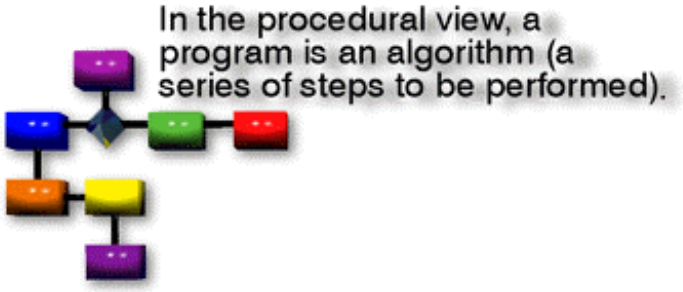
- **effizient** (engl. *efficient*) **-** Software, welche effizient ist, vermeidet unnötigen Programmcode und nutzt die Systemressourcen (CPU, Speicher) optimal.
- **wartbar** (engl. *maintainable*) **-**Software, welche wartbar ist, kann auch im Laufe der Zeit erweitert, ausgebaut werden und neuen Anforderungen des Kunden angepasst werden

Sie benötigen die englischen Ausdrücke im Selbsttest weiter hinten!

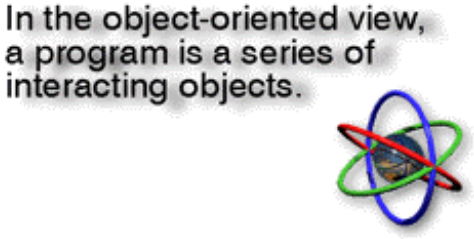
1.4.2.3. Programmier- Paradigmas

Programmiersprachen werden eingeteilt in Gruppen, welche jeweils durch ein Modell, ein *Paradigma*, definiert werden.

Die bekanntesten Paradigmas sind das prozedurale (welches auch funktional genannt wird) und das Objekt orientierte (OO).



Die prozedurale Sicht besteht aus einem Algorithmus, also einer Squenz von Anweisungen, die nacheinander ausgeführt werden.



In der objektorientierten Sicht besteht das Programm aus mehreren Objekten, die miteinander wechselwirken.

Vermutlich haben Sie schon einiges über die OO Technologie gehört, sonst würden Sie sich nicht für diesen Kurs und Java interessieren.

Nach den ersten Wellen mit viel Versprechungen und Marketing, beginnt man nun zu verstehen, welchen Nutzen man aus dem OO Paradigma ziehen kann.

Java ist eine objektorientierte Sprache und objektorientierte Methoden bieten viele Vorteile, in der Entwicklungsphase und der Wartung - aber man muss sich intensiv mit den Konzepten befassen. Das Verständnis wird nach einiger Zeit zunehmen. Aber es kann einige Zeit dauern.

Im Folgenden werden wir uns intensiv mit OO beschäftigen.

1.4.2.3.1. Das Prozedurale Paradigma

In diesem Paradigma zerlegt man die Programme in unabhängige Funktionen, welche jederzeit auf (fast) alle Variablen zugreifen können. Dieses Paradigma ist auch heute noch sehr populär. Es ist das traditionelle Paradigma der Programmierung. Begriffe wie Datenstrukturen und Funktionsbibliotheken sind gängige Begriffe in diesem Umfeld. Es gibt mehrere wichtige Unterschiede zwischen der objektorientierten und der prozeduralen Denkweise. Beispielsweise werden in prozeduralen Sprachen die Zugriffe auf Daten nicht geschützt, ganz im Gegensatz zum OO Verfahren, in dem die Zugriffe auf den Objektzustand, also die Datenfelder eines Objekts, nur mit Hilfe von speziell definierten Zugriffsmethoden möglich sein sollte. Der ungeschützte Zugriff auf Programmdaten kann zu Problemen führen und macht in der Regel prozedurale Programme fehleranfälliger und schwieriger wartbar.

Kleinere Korrekturen können zu grossen Problemen führen, neue Fehler ins Programm bringen und alte Fehler aufdecken.

1.4.2.3.2. Das objektorientierte Paradigma

In diesem Paradigma betrachtet man die Daten als den wichtigsten Bestandteil des Programms. Anstelle einer Gruppierung von Funktionen werden die Daten zu Objekten zusammengefasst und dann passende Methoden, Funktionen definiert, welche auf die Daten zugreifen können.

Diese Methoden dürfen aber nur auf (Objekt-) lokale Daten zugreifen.

Das eigentliche Programm beschreibt eine Simulation, die Wechselwirkung mehrere Objekte.

Objektorientiertes Software- Engineering beschreibt, wie man solche Objektklassen bestimmen kann. Klassen sind allgemeiner als Objekte: eine Person Meier ist ein Bibliothekar, besitzt also bestimmte Kennzeichen, wie Namen, Adresse etc. und hat bestimmte Fähigkeiten, Methoden; diese Fähigkeiten haben vermutlich alle.
Es bietet sich also an, diese allgemeinen Fähigkeiten zusammenzufassen, zu einer Klasse Bibliothekar; der Bibliothekar Meier ist eine konkrete Version, eine Instanz der Klasse Bibliothekar, ein Bibliothekarobjekt.

Sie können auch ein Buch nehmen:
jedes Buch hat einen Titel, einen Autor, eine ISBN Nummer
das Buch "Use Your Head" hat den Titel "Use Your Head", den Autor "Tony Buzan" und die ISBN Nummer 0 563 16552 9, also haben beim Objekt alle Felder konkrete Werte.

1.4.3. Objekt Orientierte Programmierung

Stellen Sie sich vor, Sie gehen in einer Stadt spazieren und möchten eine Tasse Kaffee. Sie sehen ein Kaffee und gehen hinein, bestellen beim Kellner einen Kaffee und werden ihn auch innerhalb einer vernünftigen Zeit erhalten.

Dieses Szenario erscheint sicher realistisch. Es entspricht genau der objektorientierten Denkweise:

- Sie möchten eine Tasse Kaffee, haben aber selber keinen Kaffee
- Sie wissen, dass es Kaffee in Kaffees gibt, also gehen Sie dort hin
- Sie wissen, dass der Kellner eines Kaffees in der Lage sein müsste Ihnen eine Tasse Kaffee zu bringen, es ist ja sein Job.
- Also bestellen Sie beim Kellner Ihre Tasse Kaffee, vielleicht mit Zucker oder Creme oder einen Espresso oder einen ....
  Ihr Kaffee kann also auch noch spezielle Kennzeichen, Attribute besitzen.

Analog verhält es sich mit Objekten:

- Sie benötigen bei der Entwicklung Ihrer Arbeit ein bestimmtes Objekt
- Sie wissen, dass es entweder Standardpakete mit beliebig vielen Objekten (und Klassen) und weitere Konstrukte, wie etwa Objektbroker gibt, bei denen Sie nachfragen können, ob der Broker vielleicht ein Objekt kennt, welches Ihren Vorstellungen entspricht.

1.4.3.1. Vokabular der objektorientierten Programmierung

In der OO Programmierung verwendet man spezielle Begriffe an die man sich zum Teil erst gewöhnen muss. Hat man sich einmal an die Terminologie gewöhnt, erkennt man sehr schnell, dass die Terminologie sehr dicht an der realen Welt ist, dass man also vermutlich weniger lernen muss, wenn man direkt OO Programmierung lernt. Dadurch wird auch die Implementierung einfacher und die Kommunikation zwischen den Programmieren erleichtert.

Die wichtigsten OO Begriffe (für den Start) sind:

- Objekt
- Kapselung (Datenkapselung)
- Klassen
- Messages (Methoden)
- Vererbung (Erweiterung von Klassen)

Jedes dieser Konzepte werden wir im Folgenden genauer besprechen.

1.4.3.1.1. Ein Beispiel aus dem Alltag

Betrachten Sie ein Bankkonto. Sie können ein Bankkonto eröffnen, Geld einzahlen, abheben, den Kontostand abfragen oder das Konto auflösen. Sie sind interessiert an der Rendite, der Verzinsung, Überweisungsgebühren und Kontoführungsgebühren.

Zusätzliche Funktionen (Methoden in der OO Terminologie) könnten sein:
Ausdrucken des Jahresabschlusses, Kontobelege ausdrucken und ähnliches.

Jedes Bankkonto stellt ein Objekt dar. Die Zugriffe und Abfragen sind Methoden, die den Zugriff auf die Kontodaten gestatten. Das Konto hat eine Kontonummer und Ihre Personalien als Attribute.

1.4.3.1.2. Programmierung von Objekten

Der Programmcode in einer Datei enthält die Klassenbeschreibung, die Templates (Beschreibungen) mit deren Hilfe die Objekte gebaut werden. Zur Laufzeit werden im Programm die benötigten Objekte kreiert und miteinander in Kontakt gebracht (sie tauschen Meldungen aus, oder konservativer ausgedrückt: Sie programmieren Methodenaufrufe der Objekte).

Attribute (und Methoden) kann man auf der Stufe Klasse (Klassenvariablen) oder auf Stufe Objekt (Objektvariable) definieren. Klassenvariablen gelten für alle Objekte, Instanzen dieser Klasse. Objektvariablen beschreiben den Zustand eines Objekts.

1.4.3.2. Objekte

Definition : Ein *Objekt* ist ein Softwarebündel bestehend aus Variablen und dazugehörigen Methoden. Softwareobjekte entsprechen oft Objekten in der realen Welt.

Wie aus dem Namen ersichtlich spielen Objekte in der OO Technologie eine zentrale Rolle. Sie sehen um sich herum jede Menge Objekte: Stühle, Tische, Lampen, PCs, ...

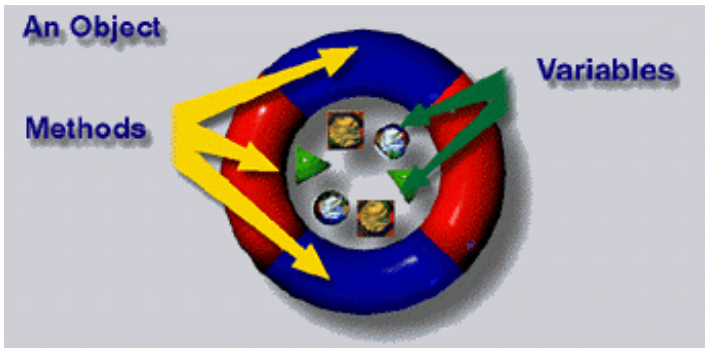
All diese Objekte haben eines gemeinsam:
1. sie besitzen einen ***Zustand*** und
2. zeigen ein bestimmtes ***Verhalten***

Der Zustand einer Person lässt sich festhalten, falls wir deren Namen, Adresse, Geschlecht, Haarfarbe, Auganfarbe, AHV Nummer, ... erfassen.

Der Zustand eines Fahrrades lässt sich festhalten, wenn wir dessen Attribute festlegen und Werte dafür bestimmen (Hersteller, Modell, Farbe, Rahmennummer, Übersetzung, Sattel, Achsen, Pedale, weiteres Spezialzubehör).

Software Objekte besitzen wie Objekte der realen Welt einen Zustand und zeigen ein bestimmtes Verhalten. Der Zustand eines Objekts wird mittels ***Variablen*** beschrieben; das Verhalten wird mittels ***Methoden*** festgehalten. Sie können also reale Objekte leicht auf Software Objekte abbilden. Der Übergang von der realen Welt in die Software Welt wird dadurch wesentlich leichter als im Rahmen des prozeduralen Vorgehens.

Software Objekte können aber auch abstrakte Konstrukte sein, wie beispielsweise ein Ereignis, ein Event, bei einem grafischen Benutzerinterface (GUI).

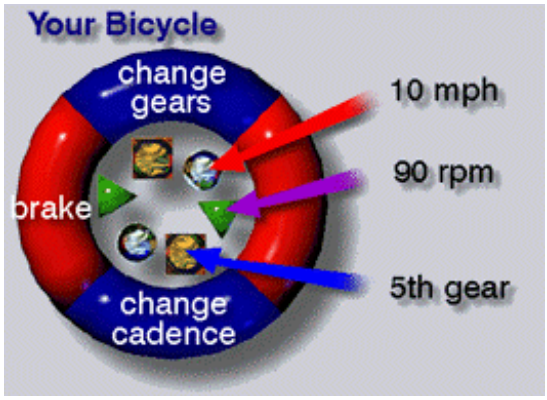


Ein Objekt kapselt seine Daten, die Variablen, so dass sie für andere Objekte nicht sichtbar sind.

Der Zugriff erfolgt ausschliesslich mit Hilfe von Zugriffsmethoden.

Die Notation entspricht nicht der heute gängigen UML (Unified Modelling Language), zeigt aber die Kapselung anschaulicher als die abstrakte UML Notation.

Schauen wir uns ein solches Objekt für ein Fahrrad an:



Das Software Objekt für ein Fahrrad besitzt beispielsweise Variablen wie Geschwindigkeit, Gang, Pedalkadenz (Attribute).

Die Attribute haben konkrete Werte, das Objekt befindet sich in einem bestimmten Zustand:
Geschwindigkeit : 10 mph;
Pedalkadenz : 90 rpm;
Gang: 5^{th}

Diese Variablen bezeichnet man auch als Instanzenvariablen, die Methoden eines Objekts als Instanzenmethoden; diese Terminologie unterstreicht den Unterschied zwischen Klassenvariablen und Objektvariablen.

Alles was zur Beschreibung eines Objekts nichts beiträgt wird weggelassen: beispielsweise macht es wenig Sinn Attribute zu definieren, welche nicht verändert werden, ausser es handelt sich um Konstanten. Die Objektbeschreibung und die dazugehörige Klassenbeschreibung sollte so schlank wie möglich sein, also keine unnötigen Attribute oder Methoden definieren.

1.4.3.3. Kapselung

Definition *Kapselung* verpackt die Objektdetails, Verhalten und Charakteristika, so dass ein Programmierer die Daten als Objekte zusammen mit Aktionen und Beziehungen zwischen den Objekten auf natürliche Art und Weise darstellen kann.

Die Objektvariablen bilden sozusagen den Kern des Objekts, die Methoden gruppieren sich darum und verstecken den Kern des Objekts vor anderen Objekten im Programm. Dieses Einpacken der Daten in Methoden wird als Kapselung bezeichnet.

Mit Hilfe der Kapselung kann man insbesondere die Details der Implementierung verbergen: Sie können die Methode eines Objekts verwenden, ohne wissen zu müssen, wie diese konkret implementiert wurde (Sie wissen ja auch nicht wie die trigonometrischen Funktionen implementiert sind).

Sie können also die Implementierung einer Methode ändern, ohne dass sich Programmänderungen ergeben.

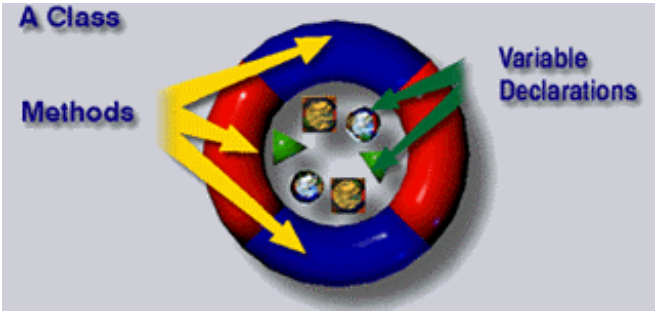
Kapselung hat auch noch folgende Vorteile:

- Modularität
  Der Programmcode für die Methoden kann unabhängig von der Implementation der verwendeten Klassen Methoden gewartet werden. Objekte sind weitestgehend unabhängig, entkoppelt.
  Sie können die Implementation der einzelnen Methoden des Bankkontos ändern, ohne dass dadurch das Buchungsprogramm, welches diese Kontobeschreibung (Objekte) verwendet, verändert werden müsste.

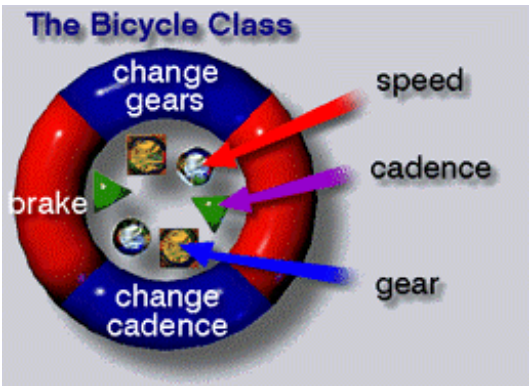
- Information Hiding
  Die Informationen über die Objekte sind nicht direkt sichtbar. Alles was Sie wissen müssen ist, welche Methoden Ihnen das Objekt zur Verfügung stellen kann.
  Bei der Implementation des Bankkontos können Sie lediglich mit Hilfe von speziellen Routinen, Methoden, auf die Datenfelder zugreifen.

1.4.3.4. Klassen

Definition Eine *Klasse* ist ein Blueprint oder Prototyp, welcher die Variablen und die Methoden definiert, welche allen Objekten, die zur Klasse gehören, gemeinsam sind.



In der realen Welt haben Sie oft die Situation, dass Sie viele ähnliche Objekte vor sich haben. Beispielsweise existieren neben Ihrem Fahrrad viele weitere Fahrräder, ähnliche oder gleiche. Ihr Fahrrad ist vermutlich nicht absolut einzigartig, selbst wenn Sie es aufgemotzt haben.



In der OO Terminologie sagt man: Ihr Fahrrad ist eine Instanz der Klasse der Objekte der Fahrräder. Alle Fahrräder besitzen einen Zustand (aktueller Gang, aktuelle Kadenz, zwei Räder) und ein Verhalten (Gang wechseln, bremsen).

Der Zustand wird durch Attribute festgehalten, das Verhalten in der Regel mit Hilfe von Verben, Tätigkeiten (Veränderungen) beschrieben.

Beim Bau eines Fahrrades weiss der Erbauer, was zu einem Fahrrad gehört - er verwendet den selben Blueprint wie alle andern. Es würde wenig Sinn machen, das Fahrrad jedesmal neu zu erfinden.

Im Sinnen der OO Technologie kann man viele Objekte finden, die immer wieder eingesetzt werden können - im 2D Bereich sind dies die üblichen geometrischen Figuren wie Rechtecke, Kreise, Flächen und ähnliches.

Wie der Fahrradhersteller wird der Programmierer kaum wieder neu einen Kreis erfinden! Java geht wesentlich weiter als andere Programmiersprachen und stellt Bausteine für sehr unterschiedliche Anwendungsbereiche zur Verfügung, von Klassen für 3D Welten, Multimedia, electronic mail, Client Server Computing, Workgroup Computing, aber auch klassische Klassen wie Listen und weitere Datenstrukturen.

1.4.3.4.1. Der Begriff Objekt

Ihnen ist sicher aufgefallen, dass Objekte und Klassen identisch dargestellt werden. Der Unterschied zwischen Klassen und Objekten ist oft unklar!

In der realen Welt ist es klar, dass Klassen selbst keine Objekte sind: die Beschreibung eines Fahrrades ist selber noch kein Fahrrad.

In der Software ist dieser Unterschied oft schwieriger zu erkennen. Das liegt daran, dass Software Objekte eben wie die Software Klassen auch nur Software sind und der Programmcode nicht materialisiert wird, wenn wir Objekte kreieren.

Allgemein gilt aber:

- Objekte sind konkret
- Klassen sind abstrakt, Beschreibungen

Sie müssen eine Instanz einer Klasse bilden, um ein Objekt zu erhalten. Beim Instanzieren werden die Variablen initialisiert, also mit Anfangswerten belegt. Damit erhält das Objekt einen definierten Zustand.

Wenn wir vom "senden einer Meldung an ein Objekt" sprechen, entspricht dies einem Aufruf einer Methode des Objekts (sei `A` die Klasse [in der Regel gross geschrieben], `a` sei eine Instanz der Klasse `A`, also ein Objekt, dann wird in Java die Methode `starte()` des Objekts a durch den Aufruf `a.starte()` ausgeführt).

1.4.3.4.2. Der Nutzen von Klassen

Objekte profitieren von der starken Modularisierung und dem Information Hiding.

Klassen profitieren von der Wiederverwendbarkeit.

Ein Hersteller von Fahrrädern kann den Blueprint für ein Fahrrad immer wieder verwenden. Er kann viele unterschiedliche Fahrräder nach dem selben Schema bauen.

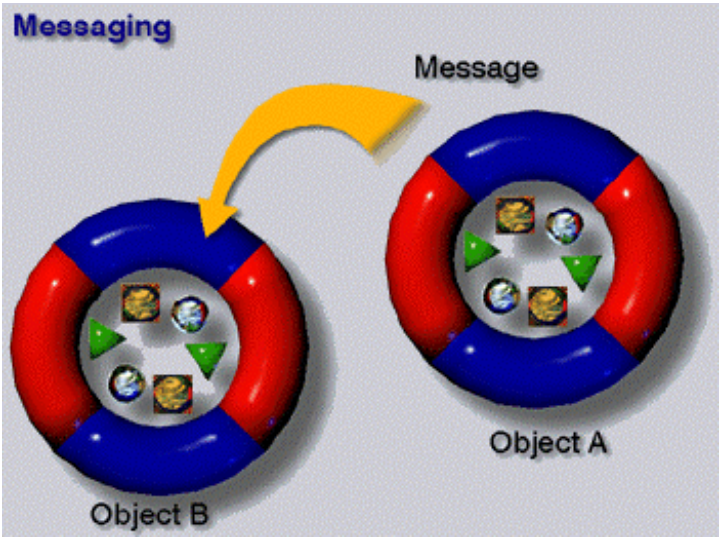
Software Entwickler verwenden Klassen immer wieder, und kreieren damit immer wieder neue Software Systeme, bestehend aus Objekten (Instanzen der Klassen).

1.4.3.5. Messages

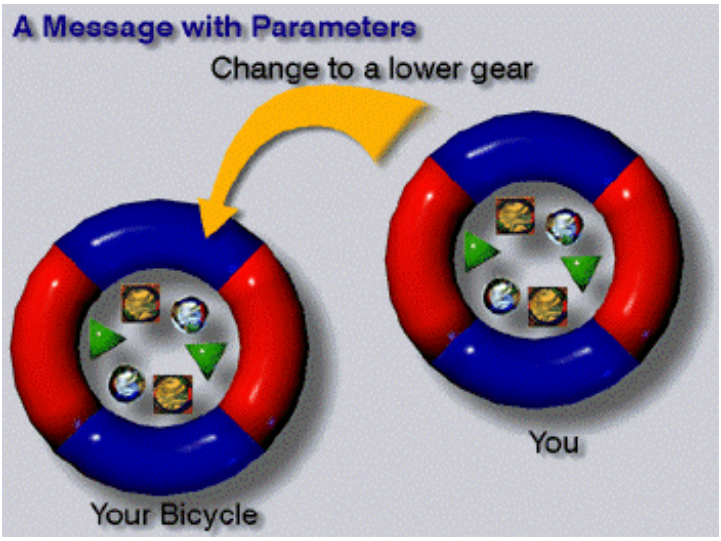
Definition *Messages* sind Methoden, mit deren Hilfe Software Objekte miteinander zusammenarbeiten und kommunizieren.

Ein einzelnes isoliertes Objekt ist nicht sehr sinnvoll einsetzbar. Es ist in der Regel Teil eines komplexen Software Systems, welches viele Objekte besitzt. Durch diese Wechselwirkung der Objekte wird die Funktionalität hergestellt.

Beim Fahrradhersteller finden Sie Spezialrohre aus aller Welt aus unterschiedlichem Material (Aluminium, Titan, Carbon,...). Der Hersteller lagert viele weitere Komponenten, wie beispielsweise Achsen, Räder, Schläuche, Reifen, Sattel, Übersetzungen und vieles mehr. Diese Komponenten isoliert betrachtet machen wenig Sinn; kombiniert als Fahrrad werden die Komponenten brauchbar.



Software Objekte kommunizieren untereinander durch das gegenseitige "senden" von Meldungen. Wenn das Objekt A will, dass das Objekt B eine seiner Methoden ausführt, sendet das Objekt A eine Meldung an das Objekt B, es beauftragt das Objekt B eine bestimmte Methode auszuführen. Falls das Objekt B Zusatzinformationen beim Ausführen seiner Methode vom Objekt A benötigt, muss A der Meldung an B Parameter mitgeben. Erst dann weiss B enau, was es für A zu tun hat.



Beim Fahrrad fahren können Sie nicht einfach eine Meldung an die Gangschaltung senden "Gang ändern"; Sie müssen konkreter werden und einen konkreten Gang als Parameter mitgeben: "lege Gang 3 ein" oder anders geschrieben: "Gang ändern(3)".

Analog verhält es sich, wenn wir einen höheren oder tieferen Gang einlegen wollen. Im einfachsten Fall haben wir in diesem Fall einen einfachen Methodenaufruf, ohne Parameter.

1.4.3.5.1. Eine Meldung besteht aus drei Komponenten

Aus den obigen Beispielen sehen Sie, dass eine Meldung aus drei (mindestens zwei, falls keine Parameter nötig sind) Komponenten besteht:

1. das Objekt, an das die Meldung gesandt wird
   (ein Fahrrad, eine Gangschaltung)

2. der Name der Methode, welche aufgerufen werden soll
   (Gang ändern, hoch schalten, runter schalten)

3. einige Parameter, falls die Methode diese benötigt
   (Gang 3 einlegen, Taste 3 drücken, linke Bremse aktivieren)

Diese drei Komponenten reichen dem empfangenden Objekt, um die verlangte Methode ausführen zu können. Es wird also keine weitere Information oder Kontext benötigt.

1.4.3.5.2. Zwei grosse Vorteile von Messages

Messages haben im Vergleich zu klassischen Routinen zwei klare Vorteile:

1. alles was ein Objekt erledigen kann, wird mit Hilfe seiner Methoden beschrieben. Message-passing beschreibt das gesamte Verhalten der Kommunikation zwischen Objekten

2. Objekte können Ihre Dienste (Methoden) überall anbieten. Kommunizierende Objekte können sich auf unterschiedlichen Maschinen befinden, sofern sie sich Messages zusenden können. Java hat gerade in dieser Richtung die Informatik wesentlich voran gebracht.

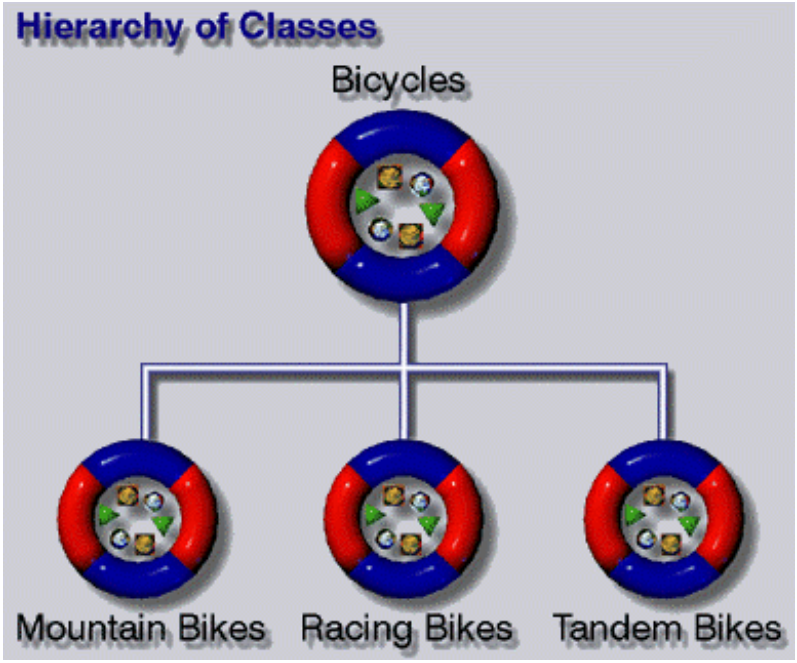
1.4.3.6. Vererbung

Definition Klassen erben Zustand und Verhalten von ihren Superklassen, Oberklassen. *Vererbung* stellt einen natürlichen und mächtigen Mechanismus dar, um Software Programme zu organisieren und zu strukturieren.

Allgemein gesprochen werden Objekte mit Hilfe von Klassen beschrieben. Falls Sie die Klasse kennen, wissen Sie bereits viel über das Objekt.

Falls Sie nicht wissen, was ein Mountain Bike ist, ich Ihnen aber sage, dass es sich dabei um ein robustes Fahrrad handelt, wissen Sie bereits, dass ein Mountainbike zwei Räder, einen Rahmen, Bremsen, eine Gangschaltung einen Sattel und einiges mehr besitzt.

In OO Systemen geht man einen Schritt weiter und gestattet Klassen mit Hilfe anderer Klassen zu definieren.



Mountain Bikes, Tandems, City Bikes, Rennräder sind spezielle Formen, Unterklassen der Klasse Fahrräder. Fahrrad ist also die Oberklasse (Superklasse) für diese Unterklassen.

Jede Subklasse, Unterklasse, erbt die Zustandsbeschreibung (die Variablendeklarationen) von der Oberklasse.

Mountain Bikes, Rennräder, Tandems haben eines gemeinsam: Kadenz, Geschwindigkeit, und so weiter.

Jede Subklasse, Unterklasse erbt Methoden (die Methodendefinition) von der Oberklasse.

Mountain Bikes, Rennräder, Tandems haben eines gemeinsam: bremsen, Gänge wechseln.

Allerdings beschränken sich die Unterklassen nicht auf Zustand und Methoden der Oberklasse. Das würde wenig Sinn machen! Unterklassen können weitere Variablen und Methoden definieren, also den Zustand und das Verhalten der Oberklasse erweitern.

Tandems verfügen über zwei Sitze, zwei Pedalsets, zwei Lenker und einiges mehr.

Unterklassen können auch Methoden der Oberklasse überschreiben und diese für die Bedürfnisse der Unterklasse anpassen.

Falls Ihr Fahrrad zwei Gangschaltungen besitzt, und viele Fahrräder haben dies, müssen Sie die Methode Gang wechseln erweitern, anpassen: Sie müssen angeben können, welchen Ganghebel Sie betätigen wollen.

Die Vererbung kann integriert werden: Unterklassen können selbst wieder Oberklassen von weiteren Unterklassen sein. je weiter unten eine Klasse angesiedelt ist, desto spezieller wird sie sein und desto präziser wird sie die Funktionalität eines konkreten Objekts beschreiben.

1.4.3.6.1. Der Nutzen der Vererbung

Vererbung bietet dem Programmierer folgende Vorteile:

- Unterklassen stellen Spezialisierungen dar, bezüglich des Verhaltens und der Zustandsbeschreibung, von allgemeineren Strukturen.
  Dank der Vererbung können wir den Programmcode der Oberklasse immer wieder verwenden.

- Programmierer können Oberklassen implementieren, welche das "generische" verhalten beschreiben (abstrakt, in Form abstrakter Klassen).
  Wesentliche Merkmale der Klasse werden definiert und partiell implementiert. Aber viele Details der Klasse bleiben undefiniert und werden nicht implementiert.
  Andere Programmierer ergänzen die Details in Form von spezialisierten Unterklassen.

1.4.4. Selbsttest

Zur Abwechslung einige Worträtsel, alle in Englisch: wir haben weiter vorne bereits einige charakteristische Attribute von OO Systemen kennen gelernt.

verstellte Begriffe:

alileebr	: zu Text 1
esaulb	: zu Text 2
cfieeiftn	: zu Text 3
tinmailbenaa	: zu Text 4
slcesas	: zu Text 5

1. Eine ..... Software wird bei einem Ausfall keinen grösseren Schaden anrichten.
2. ..... Software enthält eine saubere Dokumentation und kann vom Benutzer schnell produktiv eingesetzt werden
3. ..... Software verschwendet keine Systemressourcen
4. ..... Software kann den veränderlichen Anforderungen der Benutzer angepasst werden.
5. In der OO Programmierung enthalten ..... (Member) Daten und (Member) Methoden.

[9]Lösungen

[9] reliable, usable, efficient, maintainable, classes

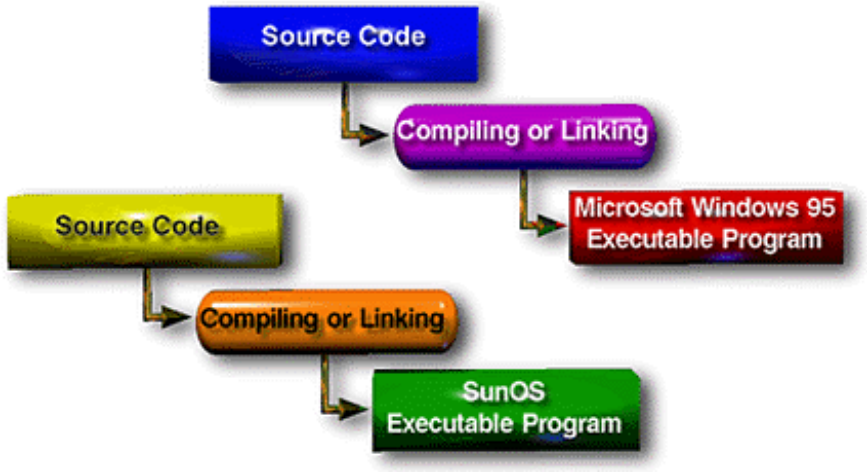
1.4.5. Der Konversionsprozess

Programmiersprachen assen sich nicht direkt auf einem Rechner ausführen. Das haben wir weiter vorne schon besprochen. Wir müssen also verstehen, wie ein Programm in Maschinencode übersetzt oder wie ein Programm interpretiert wird.

In der Welt der Programmiersprachen unterscheidet man ***interpretierte Sprachen*** und ***übersetzte Sprachen.***

Interpretierte Sprachen gestatten es dem Programmierer jeden Befehl der Reihe nach, so wie im Programm spezifiziert, auszuführen.
Der Vorteil interpretierter Programme liegt in der Entwicklungsphase, da der Programmierer das Resultat einer Änderung im Programmcode sofort sieht, ohne dass das Programm als Ganzes neu übersetzt werden muss.
Der Nachteil interpretierter Programme ist, dass das fertige Programm langsamer als ein übersetztes Programm ist, weil es vor der Ausführung interpretiert werden muss – jedes mal wenn wir das Programm ausführen. BASIC ist ein typischer Vertreter der interpretierten Sprachen.

Übersetzte Sprachen werden zuerst zusammengestellt, als eine Ansammlung von Befehlen, dann als Ganzes übersetzt und schliesslich ausgeführt. Die meisten modernen Sprachen sind übersetzbare Sprachen.
Der Vorteil übersetzter Sprachen ist, dass die Übersetzung in die Maschinensprache nur einmal nötig ist. C und PASCAL sind zwei typische Vertreter übersetzter Sprachen.
Der Nachteil übersetzter Sprachen ist, dass sie für eine spezifische Plattform, ein bestimmtes Betriebsystem oder eine CPU entwickelt werden müssen. Die Programme sind also Maschinen- spezifische oder ***Plattform- abhängige*** Programme.



Ein ***Interpreter*** übersetzt die Programmanweisungen in einen Zwischencode, welcher dann ausgeführt wird.
Ein ***Compiler*** übersetzt die Programmanweisungen mehr oder weniger direkt in die Maschinensprache.
Übersetzte Programme laufen in der Regel schneller als interpretierte Programme. Der Vorteil interpretierter Programme ist das Fehlen einer Übersetzung.

Einige Programmiersprachen, wie Java, sind Kombinationen beider Verfahren: Java Code wird übersetzt und interpretiert.

1.4.5.1.1. Übersetzen in der Java Programmiersprache

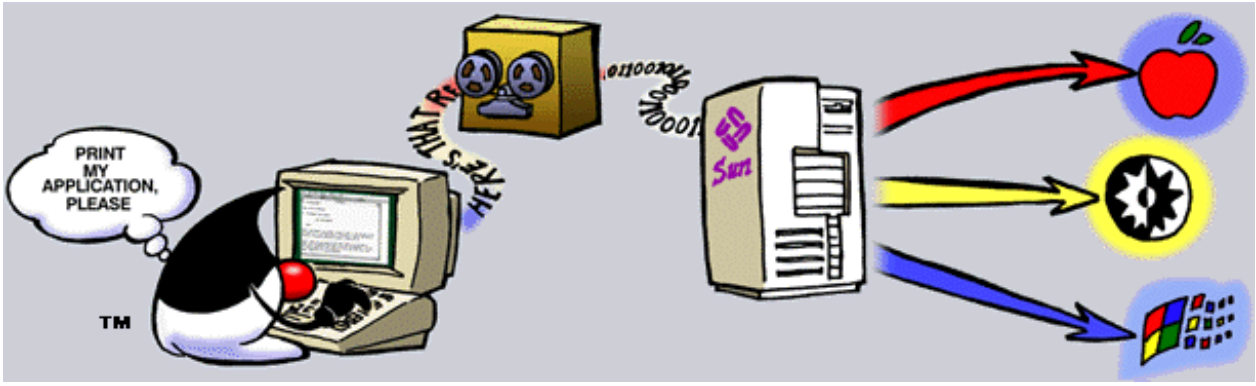
Java Programme sollten einmal übersetzt werden können und auf unterschiedlichen Plattformen lauffähig sein: Write ONCE, Run ANYWHERE™

Die übersetzten Programme werden in einer speziellen Form dargestellt, dem Bytecode. Dieser Bytecode kann auf allen Systemen ausgeführt werden auf denen die JVM installiert ist:

- die Java Virtual Machine (JVM) interpretiert und übersetzt den Bytecode in die Maschinensprache, auf der die JVM läuft. Damit ist Java ***platform-unabhängig.***

- die Java Virtual Machine ist für viele Systeme verfügbar: Solaris, Windows, Unix, MacIntosh, Lego Mindstorm, PalmPilots, ...

Diese Eigenschaft von Java ist ein spezieller Vorteil für den Java Programmierer - Portabilität und Kompatibilität.



1.4.6. Java Programmiersprache - Grundlagen

Die Spezifikation der Java Virtual Machine ***The Java Virtual Machine Specification*** definiert die Java Virtual Machine (JVM) als:

> ... eine imaginäre Maschine, welche in Software implementiert eine reale Maschine emuliert. Code für die Java Virtual Machine wird in `.class` Dateien gespeichert. Jede Class Datei enthält Code von höchstens einer öffentlich zugänglichen (public) Klasse.

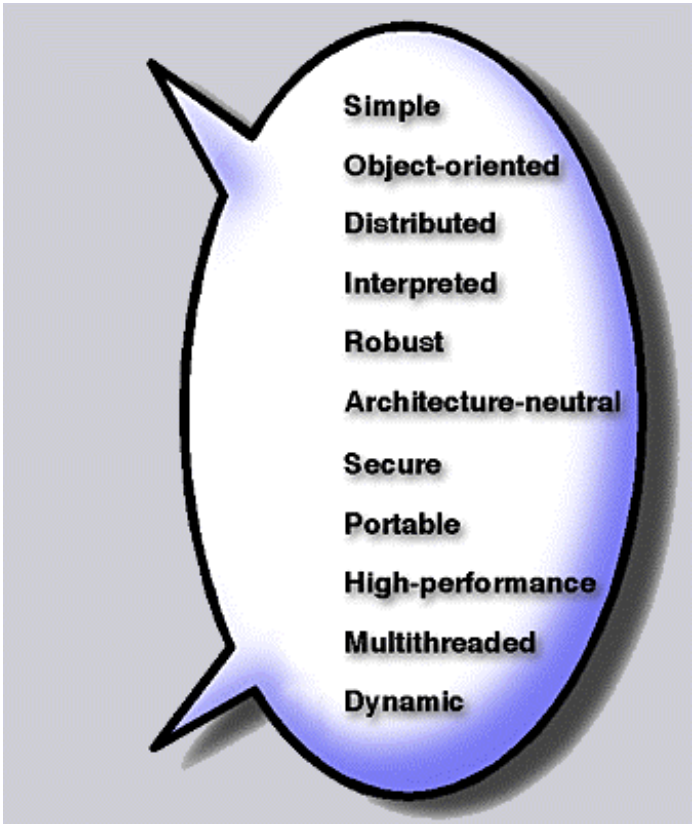
The Java Virtual Machine Specification stellt sozusagen die Hardwarespezifikation für alle Java Technologien dar. Java Programme sind Plattform- unabhängig weil die Übersetzung für eine generische Maschine gemacht wird. Die Verfügbarkeit der Java Sprachinterpreter für jede erdenkliche Hardware garantiert, dass die übersetzten Programme ausführbar sind.

Der Java Compiler übersetzt also den Programmcode in einen maschinenunabhängigen Zwischencode, den Bytecode.

Die Spezifikation des Class Dateiformat ist ebenfalls in der Spezifikation der JVM enthalten.

1.4.6.1. Spezifische Eigenschaften von Java

Die Java Programmiersprache zeichnet sich durch einige spezielle Eigenschaften aus, die in folgendem Bild zusammengefasst werden:



Schauen wir uns genauer an, was dies bedeutet.

1.4.6.1.1. Einfach

Die Java Programmiersprache ist eine einfache Sprache. Sie verzichtet auf viele Konzepte anderer objektorientierter Sprachen, wie Mehrfachvererbung, Pointer, die GOTO Anweisung und (Daten-)Strukturen.

Diese Konzepte führten in C und C++ oft zu Problemen.

Zusätzlich gibt es in Java nützliche Ergänzungen, wie den automatischen Garbage Collector, einem Programm, welches nicht mehr benötigte Objekte aus dem Speicher löscht.

1.4.6.1.2. Objektorientiert

Die Java Programmiersprache verwendet grundsätzliche Objekttechnologien mit einigen Ergänzungen und Weglassungen, mit dem Ziel, die Sprache einfach zu halten.

In Java ist alles ein Objekt. Java offeriert Schnittstellen zu Objekten und gestattet es auf einfache Art und Weise, Objekte mehrfach zu verwenden. Pakete fassen Klassen zu grösseren Einheiten zusammen. Pakete stehen auf dem Internet für sehr viele Standardaufgaben bereits fix fertig, dokumentiert und in der Regel getestet, zur Verfügung.

1.4.6.1.3. Distributed - verteilt

Java wurde speziell für den Einsatz in Netzwerken entwickelt und verfügt deswegen über einige Netzwerkfähigkeiten. Die Klassen in `java.net`, einem Package / Paket, erlauben beispielsweise den Zugriff auf URLs. Entfernte Dateien können problemlos gelesen oder geschrieben werden. Es ist auch nicht allzuschwierig Netzwerkprogramme, wie etwa einen Browser, Portscanner, WebCrawler und ähnliches zu "basteln", also aus den Grundbausteinen zusammenzubauen.

1.4.6.1.4. Interpretiert

Java ist eine interpretierte Programmiersprache. Der Java Compiler generiert Bytecode für die Java Virtual Machine (JVM), nicht Maschinencode. Um ein Java Programm auzuführen, muss der Bytecode vom Java Interpreter (JVM) interpretiert werden. Der Bytecode läuft auf allen Plattformen, auf denen die JVM verfügbar ist. Der Bytecode ist also plattformunabhängig.

1.4.6.1.5. Robust

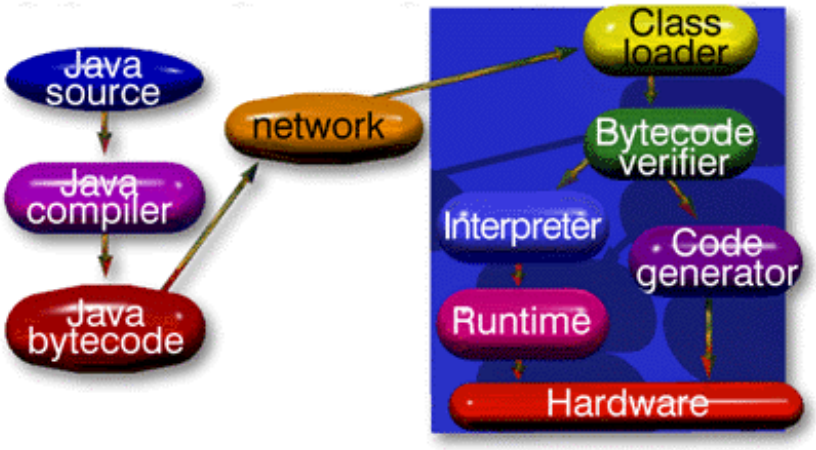
Ein robustes System ist in der Lage, auch nach einfachen Fehlern noch weiterzufahren. Java wurde bewusst entworfen, um robusten Code erzeigen zu können:

- Java ist eine "strongly-typed" Sprache: die Datentypen werden zur Übersetzungszeit geprüft und daher allfällige Fehler bereits erkannt
- Pointers und Pointer Arithmetik (direkter Zugriff auf Speicheradressen) werden nicht unterstützt. Diese Fähigkeit von C / C++ führte zu beliebig vielen Problemen, fehlen also in Java.
- Zugriffe auf Zeichenketten und Arrays / Datenfelder werden zur Laufzeit geprüft. Damit werden fehlerhafte Zugriffe abgefangen.
- automatische Garbage Collection räumt den Speicher im Hintergrund auf, ohne Zutun des Programmierers.

1.4.6.1.6. Architektur- neutral

Java Applikationen werden in architekturneutralen Bytecode übersetzt. Eine Java Applikation ist auf einem System lauffähig, sofern auf dem System die Java Virtual Machine (JVM, der Java Interpreter) vorhanden ist. Diese Fähigkeit ist besonders in heterogenen Netzwerken sehr wichtig. Spezielle JVMs erlauben Optimierungen (Just in Time Compiler, Hotspot Compiler) des Laufzeitverhaltens.

Ihr Java Programm



JVM

1.4.6.1.7. Sicher

Die JVM stellt bereits viele Sicherheitsmechanismen zur Verfügung, es ist also nicht nur ein einfacher Interpreter. Die Sprache selbst stellt zudem sicher, dass nicht auf gesperrte Speicherbereiche zugegriffen werden kann.

Beim Laden des Bytecodes wird dieser bereits geprüft und verifiziert. Der Java Code durchläuft mehrere Tests, bevor er auf einer Maschine ausgeführt werden darf.

Der Bytecode Verifier prüft die Codefragmente, überprüft das Format des Codes und stellt sicher, dass kein illegaler Programmcode ausgeführt wird. Insgesamt wird der Code vierfach geprüft bevor er von der JVM ausgeführt wird. Dies stellt weitestgehend sicher, dass der Code die Systemintegrität nicht verletzt.

1.4.6.1.8. Portabel

Portabilität heisst, dass ein Programm auf unterschiedlichen Plattformen ausgeführt werden kann, ohne dass der Programmcode angepasst werden muss. Da Java in Bytecode übersetzt wird und Bytecode portabel ist, gilt auch Java als portabel.

1.4.6.1.9. High-Performance

Mit der Einführung spezieller Compiler konnte der Nachteil der Interpretation des Programmcodes weitestgehend eliminiert werden. Der Just In Time Compiler optimiert den Bytecode für die Ausführung auf einer speziellen Zielplattform. Mehrere unabhängige Vergleiche haben gezeigt, dass damit Java in die Nähe der Performance von C gelangt, in einigen Punkten sogar schneller ist.

Sie können den Effekt der Optimierung zum Teil direkt beobachten: wenn Sie ein Programm mehrfach ausführen, wird es jedesmal schneller, speziell beim zweiten Ausführen ist in der Regel die Performancesteigerung frappant, zumindest bei Tests auf Linux mit der IBM JVM.

1.4.6.1.10. Multithreaded

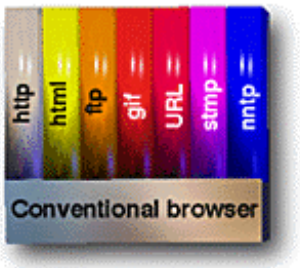
Threads werden oft als leichtgewichtige Prozesse angesehen. Es handelt sich um kleine Prozesse, die Teil eines grösseren Prozesses sind. Threads existieren also nicht unabhängig von einem sie umgebenden Prozess.

Threads sind in Java bereits von Anfang an eingebaut gewesen. Heute bieten auch C / C++ ähnliche Konzepte an. Da sie nicht nachträglich in Java eingebaut wurden, sind sie einfach ein Bestandteil der Sprache, also einfach zu benutzen und in der Regel stabiler als nachträglich eingebaute Threads in C/C++ . Die Performance des Multithreading Systems ist sicher besser, als bei einem aufgemotzten System.

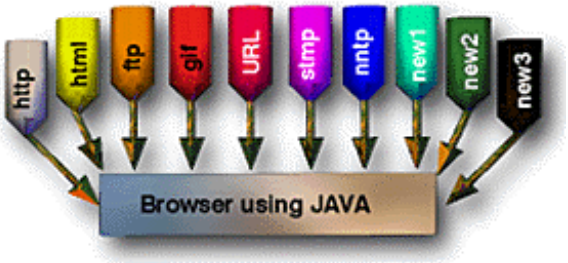
Threads werden für fast alles eingesetzt: zum Aufbau einer Grafik im Hintergrund; zum Herunterladen einer Webseite; für die Kommunikation mit einem Server.

1.4.6.1.11. Dynamisch

Java ist eine dynamische Sprache: Java Klassen können (dynamisch) zur Laufzeit in den Java Interpreter geladen und instanziert werden. Java verbindet die unterschiedlichen Klassen erst zur Laufzeit, falls nötig also, nicht einfach präventiv im Voraus.

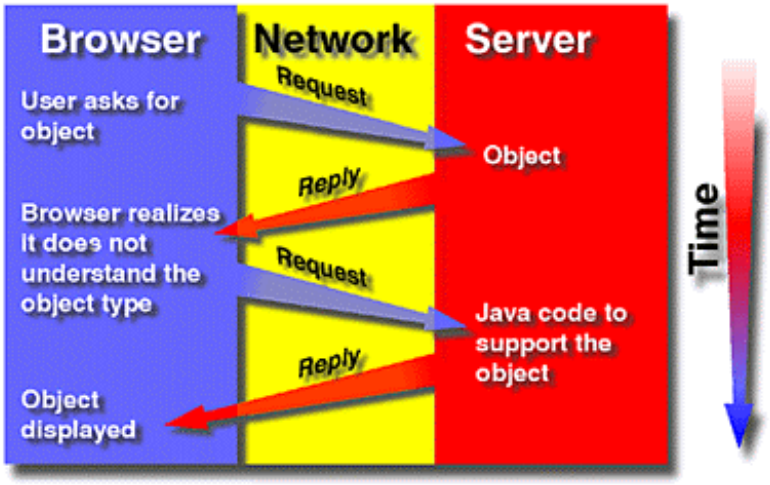


The monolith: each required peice of code is tightly bound into the browser code.



The federated system: the browser is a coordinator of pieces, and each piece is responsible for a function. Pieces can be added dynamically through the network.

Die Java Programmiersprache gestattet es auch neue Protokolle zu definieren. Falls Sie also eine neuartige Applikation planen, welche ein spezielles Protokoll benötigt, können Sie dieses selber implementieren (einen Protokollhandler und einen Contenthandler).



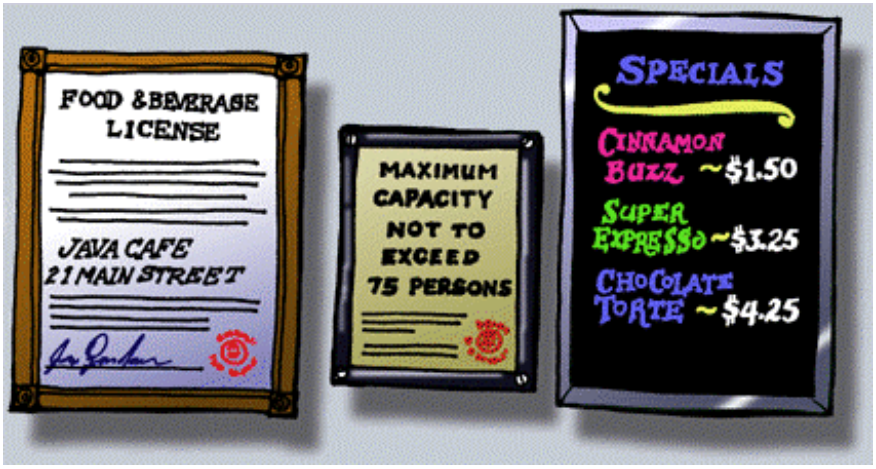
1.4.6.2. Programmkomponenten der Java Programmiersprache

Die primären Komponenten eines in Java programmierten Programms sind unter anderen, die wir später besprechen werden:

- Objekte
  - Attribute
  - Methoden

Wenn Sie eine Klasse entwerfen gehen Sie davon aus, das die Attribute korrekt definiert wurden und das Objekt korrekt modellieren.

Im Falle des Kaffees könnte die Klasse `Café` Attribute umfassen, welche die Anzahl Tische im Kaffee, einer Liste möglicher Getränke und eine Speisekarte beschreiben.
Unter Umständen werden auch Zusatzinformationen, wie Anzahl Fenster oder Fensterplätze, falls dies für das Kaffee nützlich ist.



Die Attribute hängen also von der konkreten Modellierung ab und können unterschiedlich ausfallen: für ein Objekt kann es unterschiedliche Modelle geben.

1.4.6.2.1. Objekte

Objekte sind Grössen, welche zur Laufzeit des Programms gebaut werden. Benutzen Sie Sie Objekte, um Daten zu speichern oder Probleme zu lösen. Daten werden in Variablen gespeichert, die sich im Objekt befinden und die mittels Methoden manipuliert werden.

1.4.6.2.2. Attribute

Attribute beschreiben und definieren ein Objekt und werden in Variablen innerhalb des Objekts abgespeichert. Jedesmal wenn ein Objekt als Instanz einer Klasse kreiert wird, werden die Variablen (Attribute) mit Werten belegt.

Falls Sie eine Klasse für ein Lebewesen definieren, werden Sie vermutlich ein Attribut für den Namen definieren. Die einzelnen Lebewesen, die Objekte also, werden sich durch mindestens diesen Namen unterscheiden.

1.4.6.2.3. Methoden

Methoden bestehen aus Anweisungen, die diskrete Aufgaben erfüllen oder beschreiben. Falls die Klasse ein Bankkonto beschreibt, wird sicher eine Methode abfragenKontostand definieren. mit deren Hilfe, wie der Name sagt, der Kontostand abgefragt werden kann.

1.4.6.3. Vorteile von Java als Programmiersprache

Java stellt einige Mechanismen zur Verfügung, die einem Programmierer das Programmieren erleichtern. Hier einige Beispiele:

- die Speicherverwaltung : Speicherzuordnung und Speicherfreigabe wird automatisiert
- Java stellt eine einfache, plattformunabhängige grafische Oberfläche zur Verfügung, mit der Sie Applikationen entwickeln können, welche auf allen Betriebsystem passend aussehen, zumindest gilt dies für Windows, MacIntosh und Motif. Im Falle von Windows verwendet man allerdings besser die von Microsoft entwickelten Klassen (Application Foundation Classes, AFCs), da diese vom Look und Feel her dichter bei Windows sind, als die Sun Implementierung.
- Java implementiert ein Eventmodell, welches es Ihnen gestattet, ereignisorientiert zu programmieren.
- Java stellt Ihnen Klassen zur Verfügung, mit deren Hilfe Sie Netzwerkprogramme realisieren können, unter anderem Zugriffe auf entfernte Dateien, Rechner und andere Netzwerkressourcen.
- Java stellt Klassen zur Verfügung, mit deren Hilfe Sie (übrigens länderspezifisch) Text ausgeben oder grafische Ausgaben gestalten können.
- Java gestattet es Ihnen Security Modelle zu definieren, Zugriffsmodelle zu spezifizieren und zu implementieren, die Ihnen eine detaillierte Zugriffskontrolle auf kritische Ressourcen gestatten.

Und wie immer: Java Programme sollten (fast) unverändert auf (fast) allen Betriebsystemen lauffähig sein, sofern die JVM implementiert wurde.

1.4.6.4. Prozedurale und objektorientierte Java Programmierung

Mit Java können Sie sowohl prozedurale als auch objektorientierte Programme schreiben.

Ein OO Programm verwendet Objekte, um Werte abzuspeichern und mit Methoden zu manipulieren. Die Daten werden im Objekt abgespeichert.

Im prozeduralen Programm werden Werte in Variablen abgespeichert, die lokal in Funktionen oder global definiert sind.

Die folgenden zwei Beispiele beschreiben einfachstmögliche Programme. Beide Applikationen schreiben den Text "... und so geht's..." auf den Bildschirm.

In diesem einfachen Fall ist die prozedurale Version kürzer. Das ändert sich jedoch bei komplexeren Programmen, bei denen die Wiederverwendung ein Thema wird.

1.4.6.4.1. Prozedurales Java Beispiel

Und hier ist das Beispiel:

```
1. public class MinProcApp {
2.    public static void main(String[ ] args) {
3.          System.out.println("...und so geht's...");
4.    }
5. }
```

Und so funktioniert das Ganze:

1. Definition einer Klasse : das Schlüsselwort `class` zeigt an, dass eine Klassendefinition folgt. Schlüsselworte werden immer klein geschrieben.
   Die Klasse heisst `MinProcApp`.
   Die Klammer `{` ist der Beginn des Blocks, in dem die Klasse definiert wird.

2. Diese Zeile ist der Starter des Programms. Es darf in normalen Applikationen nie fehlen und ist immer gleich. Falls man Applets programmiert muss diese Zeile durch eine andere ersetzt oder ergänzt werden (Sie können Programme schreiben, welche gleichzeitig als Applets und als Applikationen genutzt werden können).
   Die Methode `main`(...) wird als erstes ausgeführt, sie startet die Applikation.
   Im Falle eines Applets steht `init()`.

3. Diese Zeile produziert die gewünschte Ausgabe

4. Nun folgt die schliessende Klammer der Methode `main`.

5. und schliesslich die schliessende Klammer der Klasse.

1.4.6.4.2. Objektorientiertes Java Beispiel

Und hier ist das Beispiel:

```
1. public class MinOOpp {
2.          public static void main(String[ ] args) {
3.                Klasse obj = new Klasse();
4.                obj.undLosGehts();
5.    }
6. }
7. public class Klasse {
8.          public void undLosGehts() {
9.                System.out.println("...und so geht's...");
10.   }
11.}
```

Und so funktioniert das Ganze:

1. Definition einer Klasse : das Schlüsselwort `class` zeigt an, dass eine Klassendefinition folgt. Schlüsselworte werden immer klein geschrieben.
   Die Klasse heisst `MinOOApp`.
   Die Klammer `{` ist der Beginn des Blocks, in dem die Klasse definiert wird.

2. Diese Zeile ist der Starter des Programms. Es darf in normalen Applikationen nie fehlen und ist immer gleich. Falls man Applets programmiert muss diese Zeile durch eine andere ersetzt oder ergänzt werden (Sie können Programme schreiben, welche gleichzeitig als Applets und als Applikationen genutzt werden können).
   Die Methode `main`(...) wird als erstes ausgeführt, sie startet die Applikation.
   Im Falle eines Applets steht `init()`.

3. Im OO Paradigma delegieren wir die Arbeit an die Objekte:
   hier wird die Klasse `Klasse` instanziert, es wird ein Objekt `obj` erzeugt (`new`...).

4. Nun liegt es am Objekt die Arbeit zu erledigen. Unser Objekt tut dies, indem es die Methode `undLosGehts()` aufruft. Die Methode wird in der Klasse `Klasse` definiert, steht also dem Objekt `obj`, zur Verfügung.

5. Nun folgt die schliessende Klammer der Methode `main`.
6. und schliesslich die schliessende Klammer der Klasse.

7. Was noch fehlt, ist die Definition der Klasse `Klasse` und der dazugehörenden Methoden. Auch hier zeigt das Schlüsselwort `class` an, dass eine Klassendefinition folgt. Die Klasse nennen wir `Klasse`, etwas irritierend vielleicht.

8. Die Klasse enthält eine Methode, welche wir ab dieser Zeile definieren.

9. Die Methode enthält eine einzige Anweisung, die Ausgabe der bereits oben definierten Zeichenkette.

10. Nun folgt die schliessende Klammer der Methode `main`.
11. und schliesslich die schliessende Klammer der Klasse.

1.4.7. Selbsttest

Nun müssen Sie das vermittelte Wissen festigen.

Dazu haben Sie im Folgenden ein Puzzle, dann einen Selbsttest und schliesslich weitere Fragen / Antworten.

1.4.7.1. Ergänzen Sie folgende Aussagen

Die folgenden Sätze beschreiben OO und traditionelle Systeme. Die Beantwortung bzw. Ergänzung dieser Sätze bedingt, dass Sie die englischen Begriffe kennen und entsprechend einsetzen können.

Fragen	Antworten
Examples include procedural and object-oriented	PARADIGM
The output of a compiled Java program	BYTECODE
Files that are written to create a software program	SOURCE CODE
Abbreviation for Central Processing Unit - the primary chip in a computer	CPU
Functions that can be applied to data within an object	METHODS
Procedural programming is a _____ model of programming.	TRADITIONAL
The act of translating source code into machine-readable code	COMPILING
Common acronym used to describe the term "object-oriented"	OO
Symbols or names that stand for a value	VARIABLES
_____ (or verbs) express behavior and indicate object capability.	ACTION
The Java programming language is platform _____.	INDEPENDENT
Templates used to build objects; objects are instances of these	CLASSES
A particular operating system	PLATFORM
A series of steps to be performed	ALGORITHM
To design a class, you add attributes that ____ your application's objects.	MODEL
The Java programming language's special compiling provides this advantage	COMPATIBILITY
The process of combining elements to create a new entity	ENCAPSULATION
In this programming paradigm, a program is an algorithm	PROCEDURAL
Instructions that programmers create to control computers	SOFTWARE
Imaginary machine that is implemented in software on a real machine	VIRTUAL MACHINE
_____ system; interface between software and computer hardware	OPERATING
Represents things that are real or imaginary, simple or complex	OBJECT
Refers to components that run real time programs and interface directly with users	CLIENT
Abbreviation for Random Access Memory	RAM
Short for Arithmetic and Logic Unit	ALU
Short for component that coordinates activities of hardware components	OS

1.4.7.2. Quiz

Die folgenden Fragen sollen Ihnen helfen zu prüfen, ob Sie die Konzepte dieses Moduls verstanden haben.

Die Fragen müssen Sie mit wahr oder falsch beantworten, bzw. ankreuzen.

Die Lösungen finden Sie in der Fussnote.

Nach Abschluss des Quiz werden wir die wichtigsten Punkte dieses Moduls zusammenfassen und den Modul abschliessen.

1.4.7.2.1. Die Fragen

Und hier die Fragen[10]

1. Die zwei gängigsten Programmiermodelle, oder Paradigmas, sind das prozedurale Modell und das Java Modell. (wahr(falsch)

2. Die Übersetzung der Java Programme liefert eine Darstellung, welche spezifisch für eine bestimmte Plattform ist.(wahr/falsch)

3. Welches der folgenden Versionen ist die OO Version zum Anzeigen einer Meldung auf dem Bildschirm.
   a) Version A

```
public class AAAA {
      public static void main(String[ ] args) {
            XXXX obj = new Klasse();
            obj.undLosGehts();
       }
}
public class XXXX {
       public void undLosGehts() {
            System.out.println("...und so geht's...");
       }
}
```

   b) Version B

```
public class BBBB {
       public static void main(String[ ] args) {
            System.out.println("...und so geht's...");
       }
}
```

4. Die grundlegenden Konzepte einer OO Programmiersprache sind
   Objekte
   Methoden
   Attribute
   a) trifft zu
   b) trifft nicht zu

1.4.8. Zusammenfassung

In diesem Modul haben Sie gelernt, zu was eine Programmiersprache dient und wie sie arbeiten und welche Vorteile die Java Programmiersprachen- Architektur gegenüber traditionellen Sprachen hat.

ft zu;

Überlegen Sie sich kurz, welche Aspekte der Java Programmiersprache für Sie wichtig sein könnten:

- erläutern Sie, was ein Programmierer tut.
- identifizieren Sie die zwei am stärksten verbreiteten Programmierparadigmas oder Programmiermodelle.
- beschreiben Sie die fünf wichtigsten OO Konzepte.
- beschreiben Sie den Unterschied zwischen einem klassischen Konversionsvorgang, der Übersetzung eines Programmcodes in die Maschinensprache, und dem Übersetzungsvorgang in Java.
- welche Funktion hat die Java Virtual Machine und inwiefern ist die JVM speziell?
- welches sind die Hauptkomponenten eines OO Programms?

1.4.8.1. Fragen zur Vertiefung

Nachdem Sie die zwei einfachsten Programme in Java gesehen haben könnten Sie sich folgende Fragen durch den Kopf gehen lassen:

- Wie entwickelt und entwirft ein Programmierer seine Programme?
- Welche typischen Probleme werden vermutlich beim Entwickeln von Programmen auftreten?
- Welche Probleme lassen sich durch eine geschickte Wahl der Programmiersprache lösen?
- Wie funktioniert eigentlich eine Programmiersprache?
- Warum sind Programmierer so begeistert auf den Java Zug aufgestiegen?
- Inwiefern ist Java andern Programmiersprachen überlegen?

Nachdem Sie nun einen ersten Einblick in die Java Programmierung erhalten haben:

- was möchten Sie noch lernen?
- über welche Konzepte möchten Sie noch mehr lernen?
- welche Konzepte möchten Sie noch besser verstehen?

1.5. Modul 3 : Software Entwicklung

Komplexe Programme werden kaum oder selten ohne Planung erstellt. Wie bei anderen komplexe Aufgaben muss der Programmierer eine Analyse durchführen und anschliessend das Programm designen, bevor er überhaupt mit der Entwicklung anfängt.

1.5.1. Einleitung

In diesem Modul lernen Sie, im Sinne einer ersten Übersicht, Software- Entwicklungsmodelle kennen.

1.5.1.1. Lernziele

Nach dem Durcharbeiten diese Moduls sollten Sie in der Lage sein:

- einen generischen Software Entwicklungsprozess zu beschreiben
- vier Hauptkomponenten eines Softwareprogramms aufzulisten
- den Unterschied zwischen den Entwicklungsstrategien "Top Down", "Bottom Up" und "Meet in the Middle" erklären zu können
- zu erklären, welche Vorteile eine OO Methode im Vergleich zum traditionellen Software- Entwicklungsprozess hat
- die Unterschiede zwischen dem Wasserfall Modell und einem evolutionären Modell zur Entwicklung von Software aufzuzeigen

1.5.1.2. Machen Sie sich Ihre eigenen Gedanken

Wie immer vor einem neuen Modul sollten Sie sich Gedanken machen, im voraus, um was es im Modul gehen wird, was Sie bereits wissen, was Sie erwarten...:

- wie wird ein Softwaredesign erstellt?
- wie kann ein Entwickler den Entwicklungsprozess in Analyse und Design aufteilen?
- wie könnte ein Software- Entwicklungsprozess aus Ihrer Sicht aussehen?

1.5.2. Generelles zum Software Entwicklungsprozess

Wenn Sie Software entwickeln, müssen Sie sich darüber im Klaren sein, welche Anforderungen Ihr Kunde oder der Endbenutzer an das neu zu entwickelnde System hat.

Sie werden ein Programm immer für irgend jemanden entwickeln. Und falls Sie sich nicht darüber im Klaren sind, was diese Person von dem neuen System erwartet, wird er das neue System nicht akzeptieren.

Software Projekte kann man unterschiedlich entwickeln. Aber es ist immer wichtig, dass der Endbenutzer davon überzeugt ist, sein auf ihn zugeschnittenes System zu erhalten, mit dem er sich auskennt und das ihm hilft seine Probleme oder Aufgaben zu lösen.

Methoden zur Entwicklung von Software ähneln sich; aber es gibt auch signifikante Unterschiede bei der Vorgehensweise.

Folgender **Software Development Life Cycle** beschreibt die wesentlichen Schritte, welche den meisten Modellen in einer oder andern Form zugrunde liegen:

1. Konzeption und Analyse
   Identifikation der Anforderungen und gewünschten Ausstattung

2. System Design
   Design einer Lösung

3. Konstruktion und Benutzertest
   Entwickeln der Lösung mit Hilfe einer konkreten Programmiersprache

4. Kundentests und Benutzertests
   Freigeben einer vorläufigen Lösung (alpha / beta Version), um alle Komponenten der Software unter Einsatzbedingungen testen zu können.

5. Wartung
   Erweiterung, Fehlerbehebung und leichtere Modifikationen

Die Entwicklung (Konstruktion, Implementation und Testen) ist ein wesentlicher Bestandteil des Software Lebenszyklus; aber Wartung ist kritisch, weil diese sich über den gesamten Lebenslauf der Lösung erstreckt und allfällig neue Anforderungen berücksichtigen muss.

1.5.2.1. Die Hauptkomponenten eines Programms

Ein Softwareprogramm wird normalerweise vom Benutzer als eine Einheit, als eine Entität betrachtet. Aber aus Sicht des Entwicklers, also aus Design- und Entwicklungs- Sicht, besteht ein Softwareprogramm aus folgenden Komponenten:

- die **Präsentationskomponente** ist der Teil der Applikation, welcher für den Endbenutzer sichtbar ist, auf seinem Rechner oder Bildschirm.
  Hauptaufgabe dieser Komponente ist das Management der Benutzerinteraktionen und das zur Verfügung stellen einer Schnittstelle zwischen dem Benutzer und der Applikation.
- die **Funktionalitätskomponente** bildet die Applikation auf die Businesslogik ab. Dieser Teil führt alle Berechnungen und Datenmanipulationen aus, welche die eigentliche Applikation ausmachen. Diese Komponente ist also der eigentliche Kern der Applikation aus Sicht der Funktionalität.
- die **Datenkomponente** verwaltet die Daten der Applikation.
  Diese Daten werden von der Funktionalitätskomponente benutzt, um Berechnungen durchzuführen oder an die Präsentationskomponente weiterzuleiten, die sie dem Benutzer präsentiert. Die Daten können entweder aus Berechnungen oder vom Benutzer stammen. Die Datenkomponente muss eine solide Infrastruktur zur Verfügung stellen, welche sichere, verteilte Transaktionen gestattet.
- **Messages** sind der Mechanismus mit dem Daten vorwärts und rückwärts zwischen den drei Hauptkomponenten verschieben.

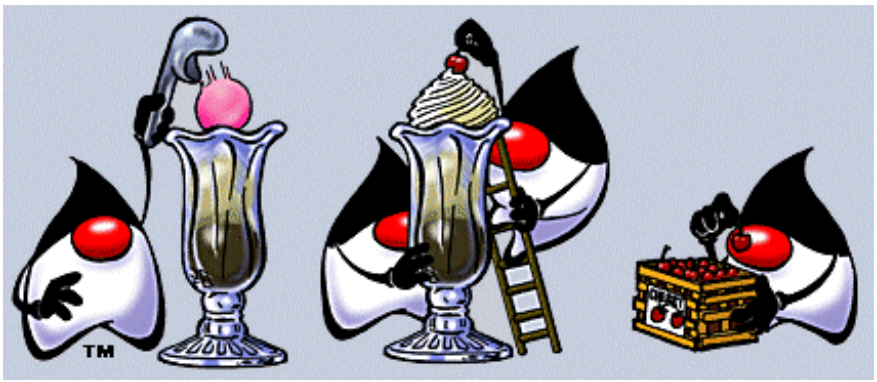
1.5.2.2. Designstrategien für die Software Entwicklung

Innerhalb der Softwareentwicklung ist es zwingend, dass man strukturierte und reproduzierbare Methoden verwendet. Viele dieser Designstrategien, welche über die letzten Jahre entwickelt wurden, sind nur innerhalb bestimmter Programmierparadigmas wertvoll. Aber zwei der einfachsten Techniken sind in allen Modellen einsetzbar:

- ***Top Down und***
- ***Bottom Up***

Diese Methoden sind nicht formal aber flexibel genug, um immer wieder eingesetzt werden zu können. Bei der Entwicklung von OO Software hat es sich gezeigt, dass eine Kombination der beidenTechniken noch effizienter ist. Diese Designstrategie bezeichnet man als

- ***Meet in the Middle***.



1.5.2.2.1. Bottom Up Design

Beim Bottom Up Prozess werden zuerst alle Komponenten zusammengesucht, die zur Lösung des Problems benötigt werden. Der Designer gruppiert alle Komponenten in grössere Komponenten, er setzt also schrittweise die Lösung aus kleinsten Bausteinen zusammen. Das Zusammensetzen führt man weiter bis man eine Lösung für das Problem hat.

1.5.2.2.2. Top Down Design

Beim Top Down Design macht man sich zuerst ein Bild vom System als Ganzes und versucht dieses dann in kleinere, überschaubarere Komponenten zu zerlegen. Diese Komponenten werden weiter genauso wie das Gesamtsystem zerlegt. Man erhält dadurch eine Hierarchie aller Komponenten des Systems.

1.5.2.3. Meet in theMiddle

Die Meet in the Middle Strategie ist Ideal für das objektorientierte (OO) Programmierung Paradigma, weil man dazu sinnvollerweise die drei OO Komponenten analysieren und designen muss:

1. **Klassen**
   diese bilden den höchsten Level der building Blocks für das Lösungsdesign.

2. **Attribute**
   Werte, welche zu Objekten zusammengefasst werden.

3. **Methoden**
   Funktionen, welche auf Attributen der Objekte operieren

Überlegen Sie sich , wie Sie eine Klasse zur Beschreibung von Autos definieren würden.
Top Down:
ein Auto besteht aus einer Liste von Bestandteilen (einer groben Stückliste)

- Fahrwerk
- Türen
- Räder
- Sitzen
- Motor
- Chassis
- ...

Bottom Up:
vorhanden sind Komponenten

- eine Liste vorhandener Räder
- unterschiedliche Sitze
- verschiedene Motoren
- verschiedene Bremsen
- ...

Top Down erhalten Sie eine Liste der *Objekte*, für die Sie Klassen definieren müssen. Offen ist noch, welche Attribute für Sie von Interesse sind.

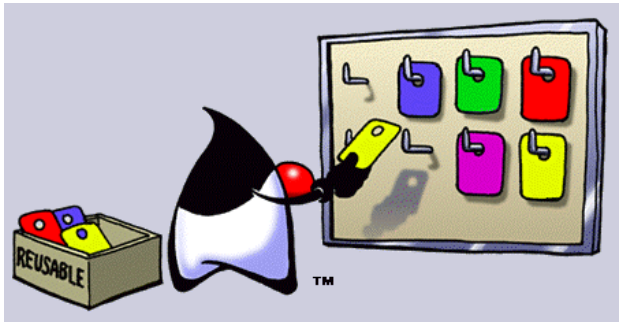
Bottom Up erhalten Sie eine Liste aller *Attribute*, die Sie zu Objekten zusammenbauen müssen.

Wenn Sie beide Verfahren kombinieren, erhalten Sie eine Liste der Objekte und eine Liste der Attribute und müssen einfach diese beiden Listen kombinieren. Sie *treffen sich in der Mitte*.

Falls Klassen auftauchen, für die Sie keine Attribute finden, oder Attribute, die zu keiner Klasse gehören, ist vermutlich ein Fehler vorhanden.

1.5.3. Der Objektorientierte Software Entwicklungsprozess

Wie wir an dem einfachen Java Beispiel gesehen haben, kann man die Vorteile der OO Methode nicht ausnutzen, wenn das Problem zu einfach und zu einzigartig ist, also keine Wiederverwendung und Vererbung möglich ist.



Die objektorientierte Vorgehensweise ist angepasst, falls:

- Sie komplexere Programme entwickeln und Sie dies in möglichst kurzer Zeit und günstiger tun wollen.
- Sie eine Entwicklungsabteilung haben, die als Ganzes produktiver werden soll (economiies of scale)
  Mit Hilfe der OO Methoden können Sie Komponenten entwickeln, welche von vielen Programmieren in vielen Projekten eingesetzt werden können. Dadurch kann die Entwicklungszeit reduziert werden, Projekte schneller abgeschlossen werden und neue Produkte preiswerter erstellt werden.
- Sie die Wartungskosten senken wollen.
  Existierende Applikationen zu warten verursacht in der Regel sehr hohe Kosten. Falls es Ihnen gelingt, mit Hilfe von OO Methoden die Software Funktionalität zu erweitern, ohne jedesmal einen sehr grossen Aufwand betreiben zu müssen, weil Sie Klassen erweitern können, und neue Algorithmen einbauen können, ohne alle Programme anpassen zu müssen (weil Objekte ein Information Hiding bewirken),sparen Sie insgesamt sehr viel von den üblichen Wartungskosten ein.

Die objektorientierte Vorgehensweise bietet viele Vorteile, unter anderem:

- **kognitive Kongruenz**
  Software Objekte lassen sich leicht auf reale Objekte abbilden. Dies erleichtert die Konstruktion und das Verständnis eines objektorientierten Programms im Vergleich zu prozeduralen Programmen, welche aus vielen individuellen Prozeduren bestehen.

- **Wiederverwendbarkeit**
  nachdem Sie einmal eine Objektklasse gebildet haben, können Sie diese Klasse in vielen unterschiedlichen Programmen anpassen.
  Falls Sie Varianten benötigen, lassen sie sich diese leicht durch Vererbung erstellen.

1.5.3.1. Objektorientierte Entwicklungsmethoden

Im Laufe der Jahre wurden viele unterschiedlichen Methoden zur Analyse, Design und Entwicklung von OO Software entwickelt.

- 1st Generation
  Von GE wurde die Object Modelling Technique (OMT) entwickelt.
  Grady Booch entwickelte eine eher auf die Programmierung ausgerichtete Methode.
  In Schweden wurde bei Ericsson eine Objectory genannte Methode entwickelt, welche sich stark auf die Berücksichtigung der Benutzeranforderungen konzentrierte (Use Cases, Anwendungsfälle der Software wurden als erstes beschrieben).

  All diese Methoden werden auch heute noch eingesetzt.

  Im Verlaufe der Entwicklung entstanden auch viele Hilfsprogramme und Werkzeuge zur Unterstützung dieser Methoden: Computer Aided Software Engineering (CASE) Tools.

- 2nd Generation
  Die unterschiedlichen Methoden wurden kombiniert.
  Es entstanden neue Methoden (Fusionen der bekannten Methoden).

- 3rd Generation
  Eine Standardnotation wurde eingeführt, die Unified Modelling Language (UML), aus dem Zusammenschluss der drei Hauptvertreter der 1st Generation Methoden (Rumbaugh von GE, Booch von Rational, Jacobson von Objectory).

  Zudem wurden Konzepte wie Nebenläufigkeit (Concurrency), komplexere Systeme, Echtzeitsysteme, ... mitberücksichtigt.

  Ideen aus neueren Programmiersprachen (Java, Eiffel) wurden aufgenommen und in die Methoden eingearbeitet.

1.5.3.1.1. Fusion

Fusion ist eine objektorientierte Software- Entwicklungsmethode. Sie nimmt für sich in Anspruch alle Aspekte der Software Entwicklung abzudecken: Analyse, Design und Implementation (alle Aspekte ?Spezifikation nur bedingt, Tests kaum, Wartung eigentlich auch nicht). Die Notation von Fusion gestattet eine systematische Darstellung und ein Auffinden von Objekten in einem System. Indem Fusion bestehende Ansätze und Methoden integriert, kann Fusion auch Verbindungen zu Codegeneratoren herstellen.

Fusion stellt somit einen (fast) durchgängigen Software- Lebenszyklus dar. Das gilt aber nur mit einigen Einschränkungen, insbesondere für Systeme mit Nebenläufigkeit, Concurrency.

1.5.3.2. Erweiterung des traditionellen Software Entwicklungslebenszyklus

Objektorientierte Programmierung wird in der Regel als der traditionellen Programmierung überlegen angesehen. Ein Lebenszyklusmodell für objektorientierte Software sieht typischerweise so aus wie auf dem folgenden Bild schematisch dargestellt wird:

Analyse Design Programmierung Testen Wartung

1.5.3.2.1. Analyse

Die objektorientierte Anforderungsphase und Analyse hat zum Ziel, dass der Entwickler :

- potentielle Objekte findet (Objektkandidaten), sowie die Charakterisierung dieser Kandidaten bestimmen kann
- eine Liste der Rollen, Attribute und des Verhaltens dieser Entitäten festlegen kann, welche sinnvollerweise als Objekte implementiert werden.

1.5.3.2.2. Design

Objektorientiertes Design beschreibt den Prozess, in dem der Designer

- die Struktur des Systems festlegt
- ein abstraktes Modell der Lösung beschreibt, wobei er möglichst eine direkte Verbindung zwischen abstrakten Objekten und realen Objekten beibehält.

1.5.3.2.3. Programmierung

In der opbjektorientierten Programmierung ist es Aufgabe des Entwicklers:

- die Strukturen der Objekte zu verwenden (Verhalten, Attribute, Beziehungen)
- bestehende Programmteile, Komponenten, möglichst weiter zu verwenden und gegebenenfalls (durch Refactoring) den Code zu verbessern.

1.5.3.2.4. Testen

Objektotientiertes Testen ist der Prozess bei dem der Entwickler oder Tester:

- Fehler isoliert und möglichst modulae Tests durchführt
- den Code, der aus den einzelnen Komponenten zusammengesetzt wurde, evaluiert und überprüft.

1.5.3.2.5. Wartung

Objektorientierte Wartung beschreibt die Aufgabe des Entwicklers:

- verbessernde Änderungen durchzuführen, inklusive kleeineren Designänderungen und Kopplung zwischen den Komponenten zu reduzieren und die Objekte möglichst dicht an die realen Objekte heranzubringen (verkleinern des semantischen Gaps)
- zusätzliche Funktion zum System hinzuzufügen, in der Regel durch Verfeinerung, Erweiterng bestehender Klassen.

1.5.4. Software Entwicklungsmodelle

Software Programme haben einen eigenen Lebenszyklus, von der ersten Idee, über Design, Entwicklung, Freigabe und Wartung. Falls das Programm sich als brauchbar erweist, wird es sicher weiterentwickelt, der Lebenszyklus startet also wieder bei einer Anforderung, einer Analyse oder in einer Programmänderung.

Die folgende Grafik listet einige dieser Entwicklungszyklen auf:

1.5.4.1.1. Staged Delivery Modell

In diesem Modell weiss der Entwickler im voraus was er wann abliefern wird. Das produkt wird in Phasen ausgeliefert (inkrementelle Imlementierung).

1.5.4.1.2. Design-To-Schedule Modell

Dieses Modell entspricht grob dem Staged Delivery Modell, allerdings mit einer wesentlichen Ausnahme: falls zum Stichtag für die Auslieferung eines Moduls dieser nicht bereits ist oder nicht vollständig ist, wird einfach ein System mit reduzierter Funktionalität (das was bereits vorliegt) ausgeliefert.

1.5.4.1.3. Das Spiralmodell

Das Spiralmodell gestattet es dem Programmierer den gesamten Entwicklungsprozess in eine Serie von Kleinstprojekten zu zerlegen. Häufig erstellt man dann einen Prototypen für eines der Miniprojekte, um sicherzustellen, dass das Projekt erfolgreich abgeschlossen werden kann.

1.5.4.1.4. Das Wasserfallmodell

In diesem Modell folgt eine Aktivität, eine Phase auf die andere, wie im lienaren Modell. Allerdings besteht die Möglichkeit zur vorherigen Phase zurück zu gehen (lineares Modell: Anforderungen, nalyse, Design, Implementierung / Programmierung, Testen, Wartung).

1.5.4.1.5. Evolutionäres Auslieferungsmodell

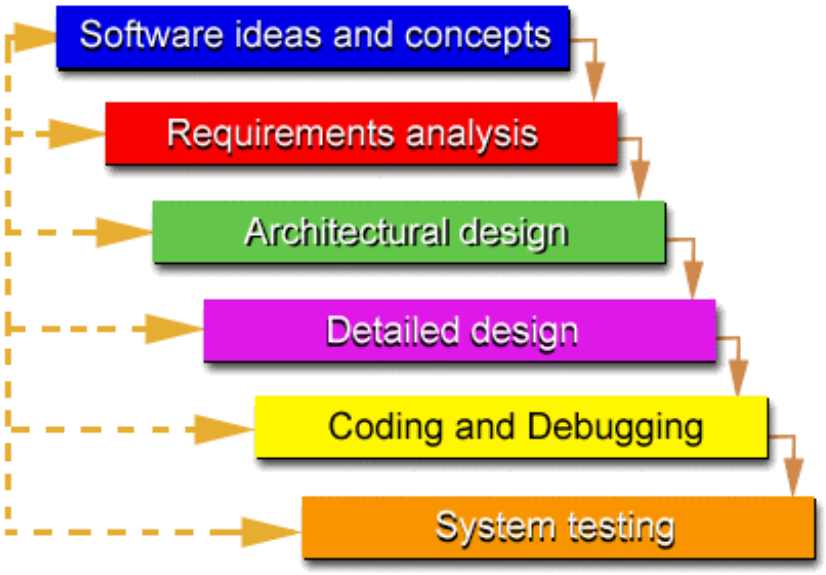
Der Entwickler gibt eine erste Version, eventuell ein Teilsystem mit kritischen Funktionen, frei und entwickelt weiter. In weiteren Versionen werden dann die Funktionalitäten erweitert, wobei Feedback von den Benutzern mitverarbeitet wird, bis schliesslich das gesamte Produkt vorliegt.

Schauen wir uns nun einige der Modelle etwas genauer an.

1.5.4.2. Das Wasserfall- Modell

Das Wasserfallmodell ist eines der ältesten Modell für die Softwareentwicklung. Anforderungsanalyse, Architekturdesign, detaillierter Design, Implementation und Testen, Systemtest werden der Reihe nach durchlaufen. Der Kunde oder Endbenutzer hat nur in der Anforderungsphase Gelegenheit sich zu äussern.

Dieses Modell ist nicht sehr flexibel, da der Entwickler lediglich vorwärts schaut, also nicht



zurück gehen kann, um allfällige offene Fragen noch beantworten zu können. Das Ergebnis kann, muss aber nicht, Software sein, die neben den Benutzerbedürfnissen entwickelt wird.

Was in den einzelnen Phasen geschieht, fassen wir in den folgenden Abschnitten kurz zusammen. Ich möchte betonen: diese Präsentation richtet sich nach den Ideen und dem Verständnis dieser Modelle aus Sun Microsystems Sicht! Eine partiell andere Sicht finden Sie beispielsweise im Skript "Software Engineering – klassisch und modern (Mit UML und Java)".

1.5.4.2.1. Software Ideen und Konzepte

Normalerweise werden die ersten Ideen von den Kunden entwickelt. Diese können mit dem Systemanalysten Kontakt aufnehmen, um allfällige Machbarkeitsfragen abkären zu können.

1.5.4.2.2. Anforderungsanalyse (Requirements Analysis)

Der Programmierer bestimmt möglichst genau, was von der Software erwartet wird.

1.5.4.2.3. Architekturdesign

Der Programmierer oder Systemarchitekt versucht das gesamte System zu verstehen und die Systemarchitektur, das Zusammenspiel der einzelnen Komponenten, zu definieren und die Komponenten selbst zu identifizieren.

1.5.4.2.4. Detailed Design

In dieser Phase sollten alle Details geklärt werden, sollten: das ist in der Regel kaum möglich.

1.5.4.2.5. Coding and Debugging

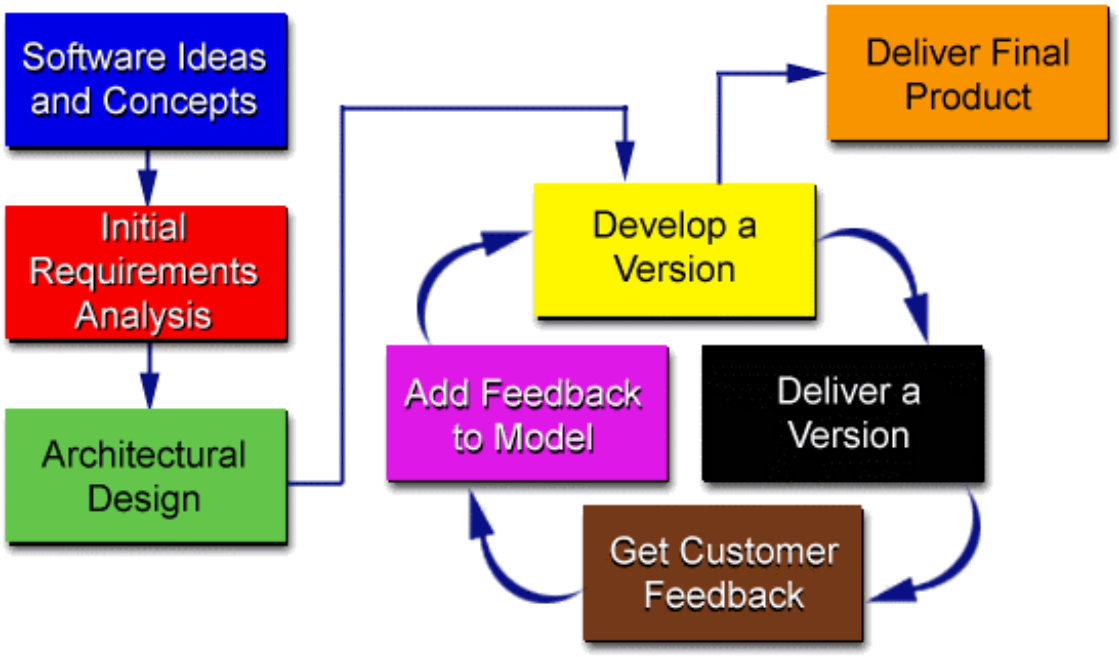
Der Programmierer realisiert das System in einer passenden Programmiersprache. Kodierung und Testen werden iteriert: es wird entwickelt, getestet, wieder entwickelt ... bis die endgültige Version zur Verfügung steht.

1.5.4.2.6. System Testing

Der Programmierer testet mit soviel Daten aus der realen Umgebung wie nur möglich. Falls eine Bankensoftware auf 100'000 Kunden ausgelegt ist, sollte der Test einen Belastungstest mit dieser Anzahl Benutzer oder Konten umfassen.

1.5.4.3. Evolutionäres Auslieferungsmodell

Das Evolutionary Delivery Modell ist ein neuer, moderneres Software Entwicklungsmodell, mit dem schneller und flexibler Software entwickelt werden kann. Die Methode berücksichtigt auch wesentlich stärker den Benutzer des Systems.



Das Modell besteht aus acht Phasen, welche zum Teil mehrfach durchlaufen werden, im Laufe der Entwicklung der Software.

1.5.4.3.1. Software Ideen und Konzepte

Normalerweise werden die ersten Ideen von den Kunden entwickelt. Häufig wird ein (Wegwerf-) Prototyp entwicklet, an dem der Kunde seine Wünsche präziser kommunizieren kann.

1.5.4.3.2. Anforderungsanalyse (Requirements Analysis)

Der Programmierer bestimmt möglichst genau, was von der Software erwartet wird.

1.5.4.3.3. Architekturdesign

Der Programmierer oder Systemarchitekt versucht das gesamte System zu verstehen und die Systemarchitektur, das Zusammenspiel der einzelnen Komponenten, zu definieren und die Komponenten selbst zu identifizieren. Ziel ist es, Software **Module** zu finden, aus denen das gesamte System aufgebaut werden kann. Nach einer weiteren Feedbackrunde mit dem Kunden können diese Module neu gruppiert oder einzelne Module verändert werden.

1.5.4.3.4. Entwicklung einer Vision

Der Entwickler implementiert mit der angepassten Programmiersprache die Module und baut damit einen Prototypen des Systems.

1.5.4.3.5. Customer Feedback

Der Entwickler sitzt mit dem Kunden zusammen und bespricht die Funktionalität und Look & Feel der Module und des Systems. Der Kunde stellt oft fest, dass eine Funktion nicht seinen Wünschen entspricht. In diesem Fall muss die Komponente einfach angepasst werden.

1.5.4.3.6. Add Feedback to the Model

Nachdem der Kunde den Prototypen geprüft hat und dem Entwickler ein Feedback geliefert hat, muss dieser die gewünschten Änderungen in das System, die Module, Komponenten oder die Architektur, das Zusammenspiel der Komponenten, einbauen, also das vorgeschlagene System den Kundenwünschen anpassen.
Die neu angepasst Version wird erneut vom Kunden geprüft, dieser liefert wieder ein Feedback und schliesslich ist der Kunde (hoffentlich) zufrieden.

1.5.4.3.7. Auslieferung des endgültigen Produkts

Mit der Ablieferung des endgültigen Produkts ist die Aufgabe des Entwicklers erledigt. Hoffentlich löst es die Probleme des Kunden!

1.5.5. Selbsttest

1.5.5.1. Zuordnung von Begriffen

Auch in diesem Test sollten Sie Ihre Englischkenntnisse auffrischen können: Der eigentliche Test besteht darin, zur linken Seite der Tabelle ein Wort auf der rechten Seite zu finden. Nett wie ich bin präsentiere ich gleich die Musterlösung.

Aussage	**Begriff**
Short for rapid application development	RAD
The highest level building block of an OO solution	Class
Function that completes an object	Method
When a whole class can be borrowed by another program	Reusability
The oldest model of the software life cycle	Waterfall Method
Two simple techniques that can be applied to any paradigm	Top Down / Bottom Up
The three components of a software program (application)	Presentation + Functionality + Data
More recent software life cycle model than Waterfall	Evolutionary Delivery Model
Envision overall problem, break into smaller pieces to solve	Top Down
Think of smallest components to solve problem, put them together	Bottom Up
Small values that together make up an object	Attributes
Three types of components in an OO system	Classes + Attributes + Methods
When objects map easily to their real-world counterparts	Cognitive Congruency
_____ system; interface between software and computer hardware	Operating
Short for Arithmetic and Logic Unit	ALU
Abbreviation for Random Access Memory	RAM
Common acronym used to describe object-oriented paradigm	OO
Used to translate source code in to machine-readable code	Virtual Machine
Files that are written to create a software program	Source Code
A particular operating system	Platform
A series of steps to be performed	Algorithm
The Java programming language is platform _____.	Independent
A symbol or name that stands for a value	Variable
The output of a compiled Java programming language program	Bytecode
Examples include procedural, object-oriented	Paradigm
In this programming paradigm, a program is an algorithm	Procedural

1.5.5.2. Multiple Choice Aufgaben

Die folgenden Tests dienen einer Kurzwiederholung des Stoffes dieser einführenden Einheit. Viele der vorgestellten Konzepte werden in den folgenden Studieneinheiten weiter vertieft!

Vieles wird Ihnen dann klarer werden. Ziel dieser Einheit war und ist es, Sie mit einigen Begriffen vertraut zu machen und etwas neugierig zu machen.

1. Zwei wichtige Nutzen eines OO Zugangs zur Softwareentwicklung sind die kognitive Kongruenz und die Wiederverwendung.
   Welche der folgenden Aussagen ist die Definition von kognitiver Kongruenz?

   a) Software Objekte sind leicht auf Objekte der realen Welt abbildbar. Dadurch ist es einfacher OO Programme zu verstehen als Programme, die aus dutzenden von individuellen Prozeduren bestehen.

   b) Wenn man ein Objekt oder eine Klasse für ein Programm erstellt hat, kann man diese Klasse immer wieder verwenden, also auch in anderen Programmen.
   [11]

2. Das Wasserfall und das Evolutionäre Entwicklungsmodell sind Software Lebenszyklusmodelle, welche aus der Praxis heraus entstanden sind.

   a) trifft zu?
   b) trifft nicht zu?
   [12]

3. Das Evolutionäre Softwareentwicklungsmodell besteht aus acht Phasen.
   Welche der folgenden Phasen gehört *nicht* zu diesem Modell?

   a) Systemtest
   b) Anforderungsphase (initial requirement analysis)
   [13]

[11] a
[12] a
[13] a : im Wasserfallmodel erstellt man ein System in einem Durchlauf; dort macht der Systemtest Sinn.

4. ..... ist der Prozess, bei dem man aus kleinen Modulen grössere Module zusammenbaut, um Probleme zu lösen. Der Designer gruppiert die Module zu Komponenten und schliesslich zur Systemlösung zusammen.

   a) Top Down Design
   b) Bottum Up Design
   c) OO Design
[14]

5. Im Sinne der präsentierten, vereinfachten Software- Architektur bildet die ..... Komponente auf die Applikation und die Businesslogik ab. Diese Schicht manipuliert oder berechnet Aufgaben, für welche die Applikation entworfen wurde, gemäss ihrer Designspezifikation.

   a) Präsentation
   b) Funktionalität
   c) Daten
[15]

[14] b
[15] b

1.5.6. Zusammenfassung

Na das wäre geschafft! Etwas umständlich macht es einem Sun schon!
Aber Sie haben auf jeden Fall die Kurseinheit ***Rechner und Grundlagen der Programmierung*** abgeschlossen.

In dieser Kurseinheit ging es darum einfache und typische Begriffe aus dem Java Umfeld kennen zu lernen.

Unabhängig davon, ob Sie Java Programmierer werden wollen oder nicht: Die Hoffnung ist, dass einige Begriffe aufgefrischt oder erläutert wurden, die in den folgenden Kurseinheiten benötigt werden.

Die folgenden Begriffe wurden erläutert:

- Standard Rechnerkomponneten
- Hardware und Software und deren Zusammenspiel
- grundlegende Programmierprinzipien
- Übersicht über die Programmierung
- Die Java Programmiersprache
- Software Entwicklung
- Objektorientierte Konzepte
- Software Entwicklungsmodelle

Zur Wiederholung:

a) Das Buch als Klasse
   Sie beschreiben generelle Kennzeichen eines Buches,
   seinen Autor,
   seinen Titel
   die ISBN Nummer und weitere generelle Kennzeichen (Attribute)
b) Das Buch als Objekt
   Sie haben ein konkretes Buch
   Der Autor heisst Franz Kafka
   Der Titel heisst Gesammelte Werke
   die ISBN Nummer ist 123-123-123

RECHNER- UND PROGRAMMIER- GRUNDLAGEN