Objektorientierte Programmiertechnik

Kapitel 1 – Grundbegriffe der objektorientierten Programmierung

Übersicht

1. Grundbegriffe der Programmierung	Inhalte
2. Einfache Beispielprogramme	✓ Grundbegriffe der OO-Programmierung
3. Datentypen und Variablen	
4. Ausdrücke und Operatoren	
5. Kontrollstrukturen	Objektorientierte Konzepte I
6. Blöcke und Methoden	
7. Klassen und Objekte	UML
8. Vererbung und Polymorphie	
9. Pakete	Objektorientierte Konzepte II
10. Ausnahmebehandlung	
11. Schnittstellen (Interfaces)	Einführung in der Programmiersprache Java
12. Geschachtelte Klassen	
13. Ein-/Ausgabe und Streams	
14. Applets / Oberflächenprogrammierung	

Das erste Programm

```
// Datei: HelloWorld.java

public class HelloWorld   // Klasse zur Ausgabe von Hello, world!
{
    public static void main (String[] args)   // Methode main() zur
    {                                         // Ausgabe der Zeichen-
        System.out.println ("Hello, world!"); // kette "Hello, world!"
    }
}
```

- Übersetzen (Kompilieren) des Programms mit ***javac***
- Der Interpreter ***java*** verlangt den Klassennamen, nicht den Dateinamen
- Java ist case sensitive
- In Java kann nur objektorientiert programmiert werden

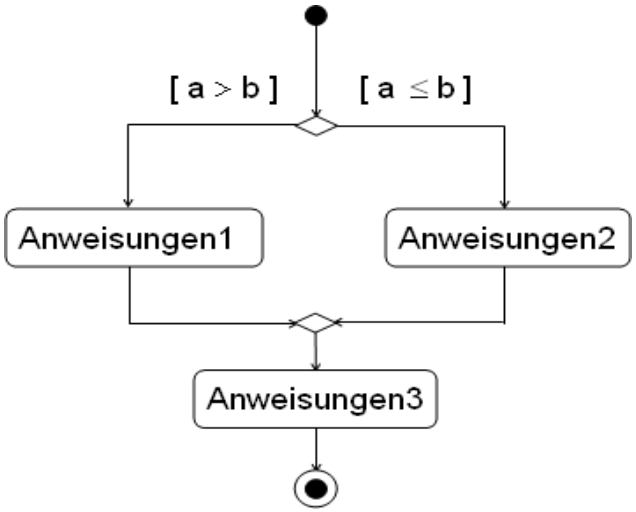
Vom Problem zum Programm I

- Bestandteile und Eigenschaften von Algorithmen
  - Eine Menge von Objekten
  - Eine Menge von Operationen
  - Ein definierter Anfangszustand
  - Ein gewünschter Endzustand
- Programm
  - Besteht aus einer Reihe einzelner Anweisungen, die sequenziell ausgeführt werden
  - Abarbeitungsreihenfolge wird auch als Kontrollfluss bezeichnet
  - Kontrollstrukturen steuern Anweisungen
  - Bei sequenziellen Abläufen gibt es Kontrollstrukturen für Selektion, Iteration und Sequenz

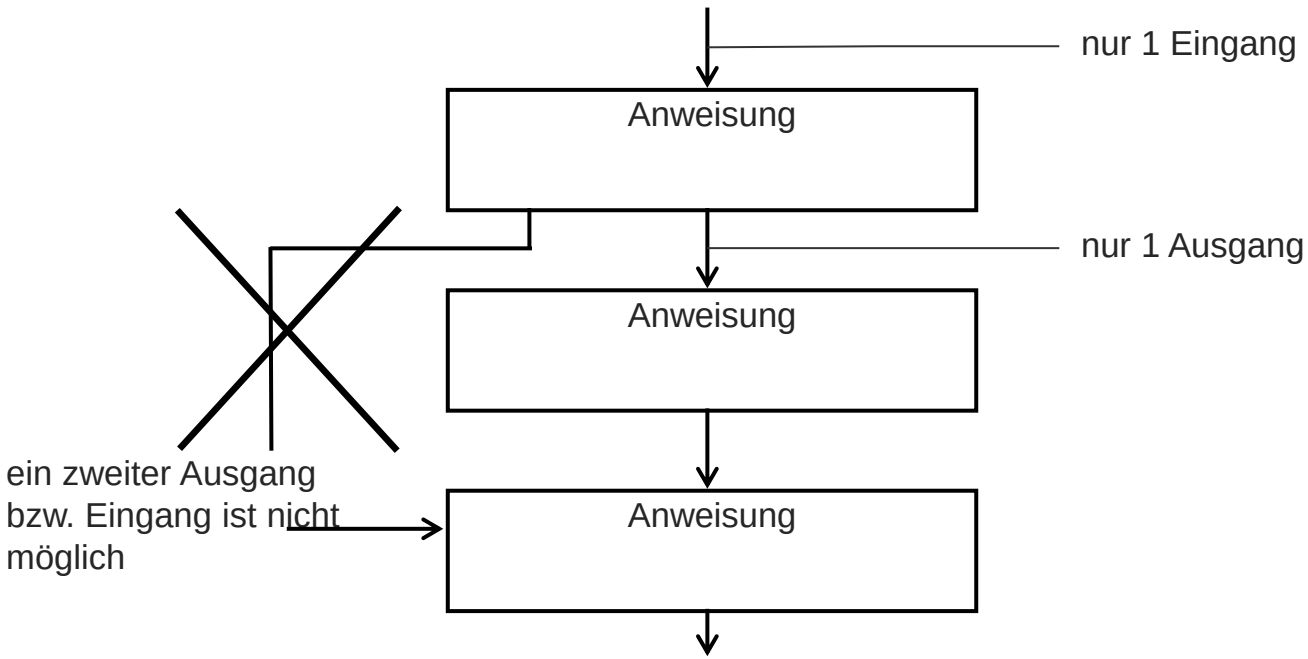
- Unter einer Kontrollstruktur versteht man eine Anweisung, welche die Abarbeitungsreihenfolge von Anweisungen beeinflusst

Vom Problem zum Programm II

- Grafische Darstellung einer Verzweigung



- Sequenz mit single entry und single exit



Nassi-Shneidermann-Diagramme I

- Natürliche Sprache vs. Formale Sprache
- Nassi-Shneiderman-Diagramm zur Visualisierung
- Strukturierte Programmierung:

  Zerlegen eines Gesamtproblems in Teilprobleme, z.B.:

  sin(0,1)*(34+sqrt(16))=>

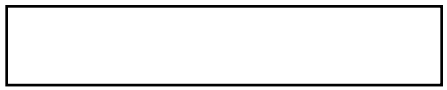
  0,0017*(34+sqrt(16))=>

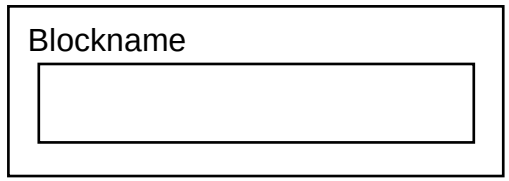
  0,0017*(34+4)=>

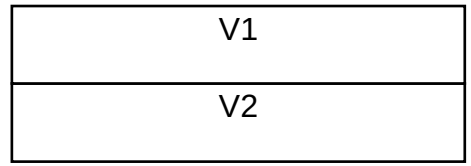
  0,0017*38=>

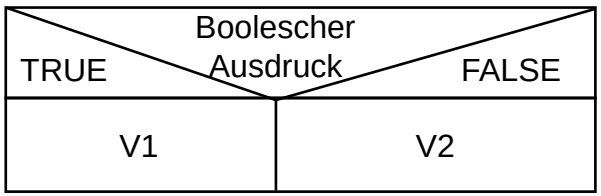
  0,066

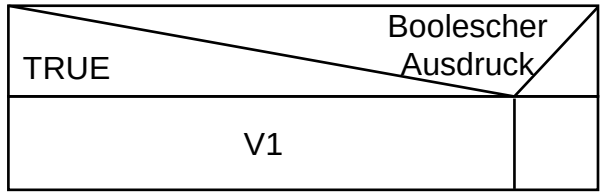
Nassi-Shneidermann-Diagramme II

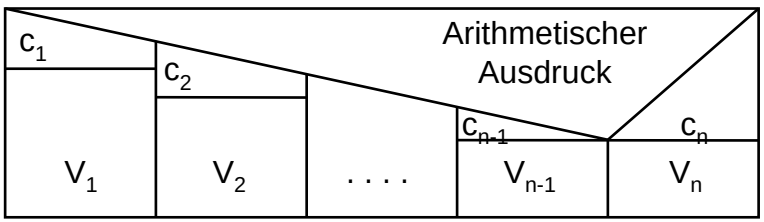




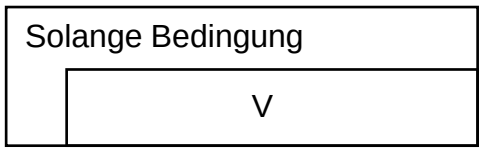


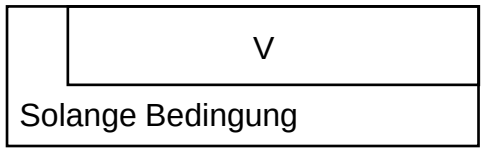


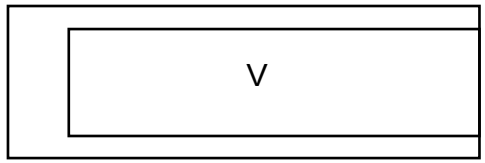


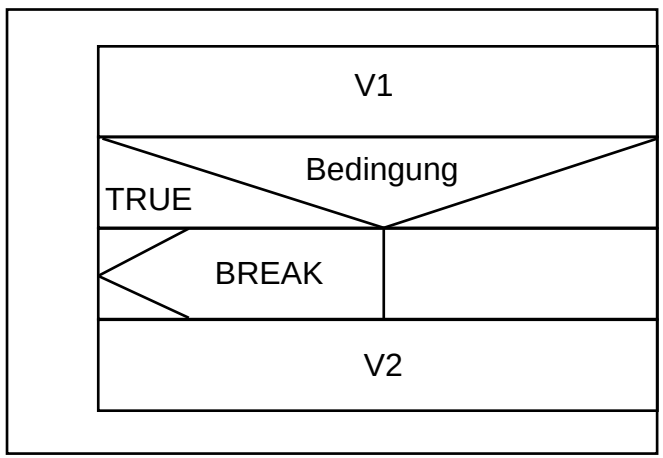


Nassi-Shneidermann-Diagramme III









Zeichen

- Ein Zeichen ist ein von anderen Zeichen unterscheidbares Objekt, welches in einem bestimmten Zusammenhang eine definierte Bedeutung trägt.

- Von einem Alphabet spricht man, wenn der Zeichenvorrat eine strenge Ordnung aufweist.

- Im Rechner werden Zeichen codiert dargestellt, z.B. als ASCII (American Standard Code for Information Interchange) Zeichen

Variable

- Eine Variable ist eine benannte Speicherstelle. Über den Variablennamen kann der Programmierer auf die entsprechende Speicherstelle zugreifen.

- Speicherstellen können nicht direkt physikalisch adressiert werden.

- Besitzt in Java drei Kennzeichen:
  - Name,
  - Datentyp/Typ
  - Wert/Inhalt

Bildhafter Vergleich: Schublade mit Beschriftung (Name), die nur bestimmte Dinge aufnehmen kann (Typ) und enthält irgendwas (Inhalt).

Datentypen I

- Höhere Programmiersprachen sind streng typisiert
- Einfache Datentypen (int) bedeutet, das sein Wert atomar ist
- Standardtypen werden vom Compiler zur Verfügung gestellt
- int: in Java 32 Bit, Wertebereich von -2^{31} bis $+2^{31}-1$
- float: reelle Zahlen, Darstellung als Exponentialzahl (Mantisse * Basis $^{\text{Exponent}}$). Mit höherer Rechengenauigkeit ist der Datentyp double.

Datentypen II

- Operationen auf einfachen Datentypen

Operator	Operanden	Ergebnis
Vorzeichenoperatoren +, - (unär)	int	➔ int
Binäre arithmetische Operatoren +, -, *, /, %	(int, int)	➔ int
Vergleichsoperatoren ==, <, <=, >, >=, !=	(int, int)	➔ boolean (Wahrheitswert)
Wertzuweisungsoperator =	int	➔ int

Datentypen III

- Datentypen können auch selbst definiert werden, z.B. zusammengesetzte Datentypen

```
struct Punkt
{
    int x;
    int y;
}
```

```
struct Punkt p;
p.x = 1;
p.x = 1;
```

- Erweiterung durch Hinzufügen von Methoden zu reinen Daten → Klasse

```
class Punkt
{
    int x;
    int y;

    int get_x()
    {
        return x;
    }
    int get_y()
    {
        return y;
    }
}
```

```
public static void main (String[] args)
{
    Punkt p = new Punkt();
    System.out.println(p.get_x());
    System.out.println(p.get_y());
}
```

Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

✓ Grundbegriffe der OO-Programmierung

✓ Objektorientierte Konzepte I

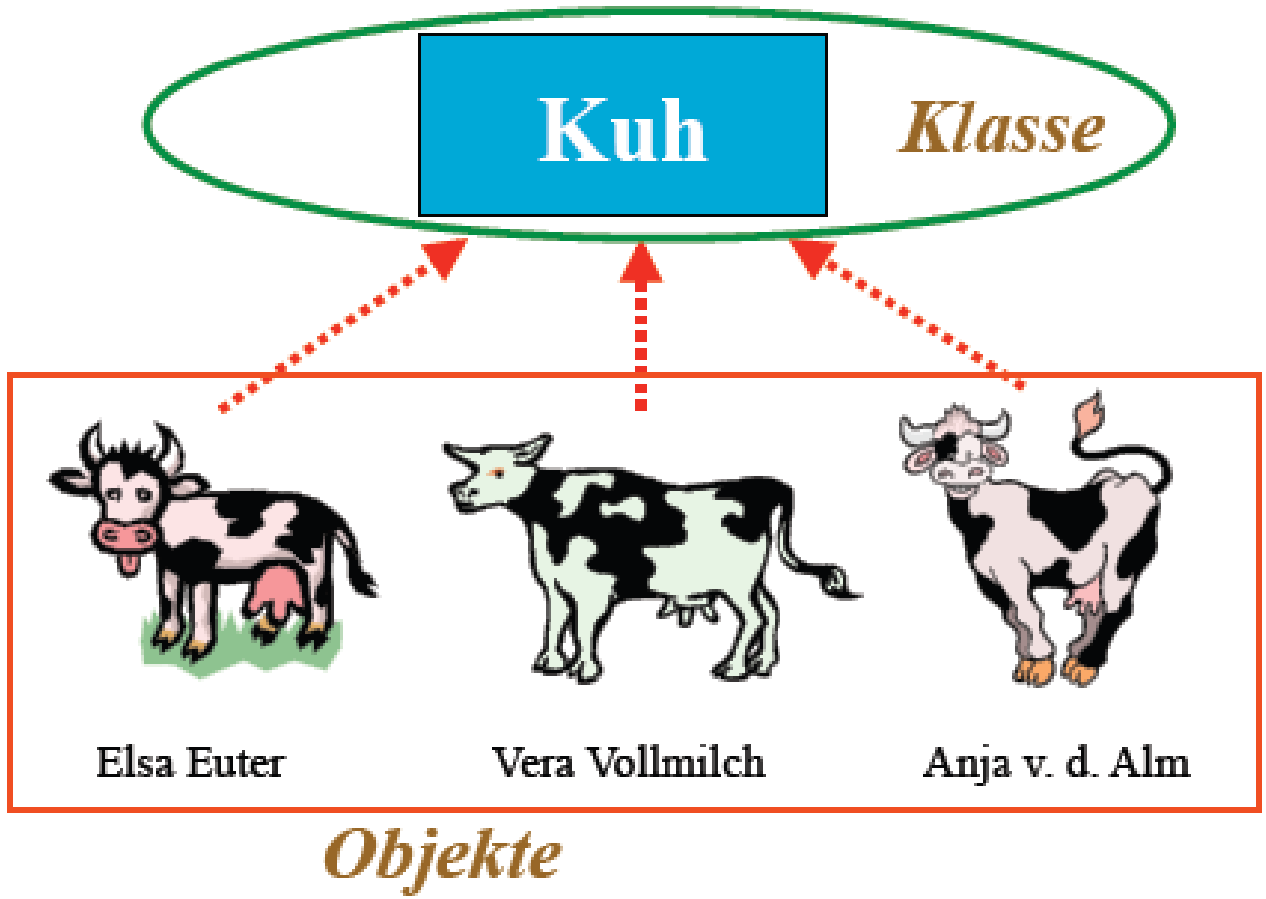
UML

Objektorientierte Konzepte II

Einführung in der Programmiersprache Java

Objekte I

- Schnelleinstieg



Quelle: Skript Prof. Dr. O. Lazar

Objekte II

- Schnelleinstieg

Wodurch kann man die Objekte voneinander unterscheiden?

==> Attribute: name, gewicht, geburtsjahr

Elsa Euter	Vera Vollmilch	Anja v. d. Alm
552	480	435
1990	1993	1955

Quelle: Skript Prof. Dr. O. Lazar

Objekte III

- Schnelleinstieg

Was kann man mit jedem einzelnen Objekt machen???

Das Alter abfragen
Das Gewicht abfragen
Heu fressen lassen
Alle Daten abfragen

==> Operationen (Methoden): gibMirDeinAlter, gibGewicht, fresseHeu, gibDaten

Quelle: Skript Prof. Dr. O. Lazar

Objekte IV

- Schnelleinstieg
  - Klasse
    - Eine Klasse beschreibt die Struktur und das Verhalten einer Menge gleichartiger Objekte. Sie ist eine Vorlage (Schablone, Template) für die Objekte.
  - Objekt
    - Ein Objekt ist ein Exemplar (Instanz, konkrete Ausprägung) einer Klasse, das sich entsprechend der Definition der Klasse verhält.

Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Grundbegriffe der OO-Programmierung
- ✓ Objektorientierte Konzepte I
- ✓ UML

Objektorientierte Konzepte II

Einführung in der Programmiersprache Java

Problem- und Lösungsbereich

- Zu Beginn eines Projektes befindet man sich im Problembereich

  (Anwendungsbereich) → also in der Begriffswelt des Kunden

  Vorteil: der Kunde versteht die Projektunterlagen

  Kunde und Entwickler sprechen dieselbe Sprache: Man versteht sich!

- Systemanalyse
  - Untersuchung der Geschäftsprozesse
  - Abbildung der Geschäftsprozesse auf Objekte

Systemanalyse

- Mittels Systemanalyse wird der Problembereich (die Begriffswelt des Kunden) zerlegt
- Eine Klasse entspricht einem Typ eines Gegenstandes der realen Welt
- Der Ansatz der Objektorientierung basiert darauf, Objekte der realen Welt mit Hilfe softwaretechnischer Mittel abzubilden
- Beim Systementwurf betritt der Entwickler den Lösungsbereich
- Klassen stellen Baupläne für Objekte dar. Klassen sind die Datentypen, die Objekte die Variablen (Instanzen) dieser Datentypen. Die Objekte werden gemäß den in den Klassen abgelegten Bauplänen erzeugt.

Unified Modelling Language

- Unified Modelling Language (UML)
  - ist eine graphische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Teilen von Software und anderen Systemen
  - Sie wird von der Object Management Group (OMG) entwickelt und ist sowohl von ihr als auch von der ISO standardisiert
  - UML ist heute eine der dominierenden Sprachen für die Modellierung von Softwaresystemen.
  - UML definiert 14 verschiedene Diagrammarten

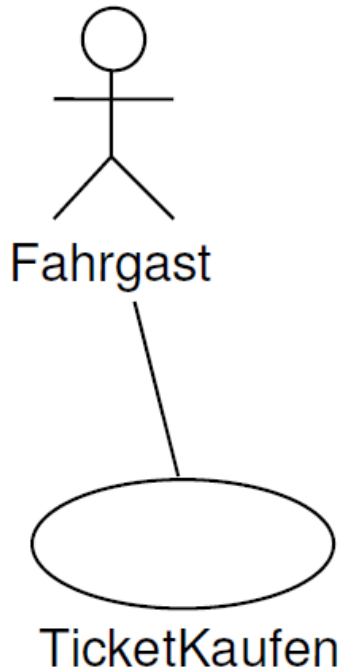
    Anwendungsfalldiagramm

    Interaktionsdiagramme (Sequenzdiagramm)

    Klassendiagramm
  - Zum Verständnis und zur Darstellung von Geschäftsprozessen eignen sich Anwendungsfalldiagramme

Use Case

- UML – Anwendungsfalldiagramm (use case diagram)
- Anwendungsfälle dienen bei der Anforderungsermittlung zur Repräsentation extern sichtbarer Funktionalität
- Anwendungsfallmodellierung ist eine rudimentäre und abgewandelte Form von Datenflussmodellierung, die in der objektorientierten Analyse (OOA) verwendet wird
- *Akteure* repräsentieren *Rollen*, in denen Benutzer und externe Systeme dem System gegenübertreten
- *Anwendungsfall (use case, Geschäftsprozess)* repräsentiert Folge von Interaktionen, die den beteiligten Akteuren einen gewissen Nutzen bringen
- Anwendungsfallmodell: Menge aller Anwendungsfälle. Beschreibt die Funktionalität des Systems aus Sicht des Benutzers auf sehr abstrakter Ebene vollständig

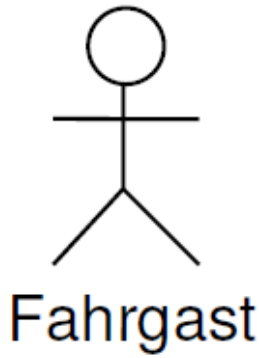


Akteure

- Akteure

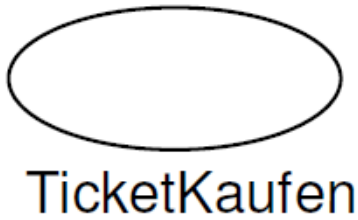
Akteure

- Akteur: Modelliert außerhalb des Systems stehende Entität, die mit dem System kommuniziert:
  - Benutzer
  - Externes System (z.B. externe Datenbank)
  - Physikalische Umgebung
  - Gegenbeispiel: Datenbank, die Teil des Systems ist, ist kein Akteur und wird deshalb im Anwendungsfallmodell nicht explizit modelliert!
- Akteur hat einen Namen;
  Bedeutung wird in einem Glossar beschrieben.
- Beispiele für Beschr. von Akteuren im Glossar:
  - Fahrgast: Eine Person, die einen Bus benutzen möchte oder benutzt.
  - GPS-Satellit: Liefert dem System die GPS-Koordinaten des aktuellen Standorts.

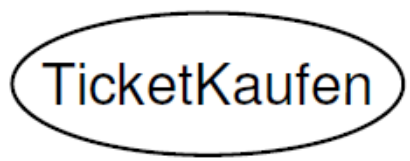


Anwendungsfall

- Anwendungsfall (AF)
- Ein Anwendungsfall repräsentiert eine Funktionalitätsklasse, die vom System erbracht wird.
- Für jeden Anwendungsfall wird eine textuelle Anwendungsfallbeschreibung erstellt (siehe nächste Folie)
- Zusätzlich wird ein UML- AF-Diagramm erstellt, um den Überblick zu erleichtern.
- Beachte: Die Anwendungsfallbeschreibungen sind wichtiger als das AF-Diagramm !

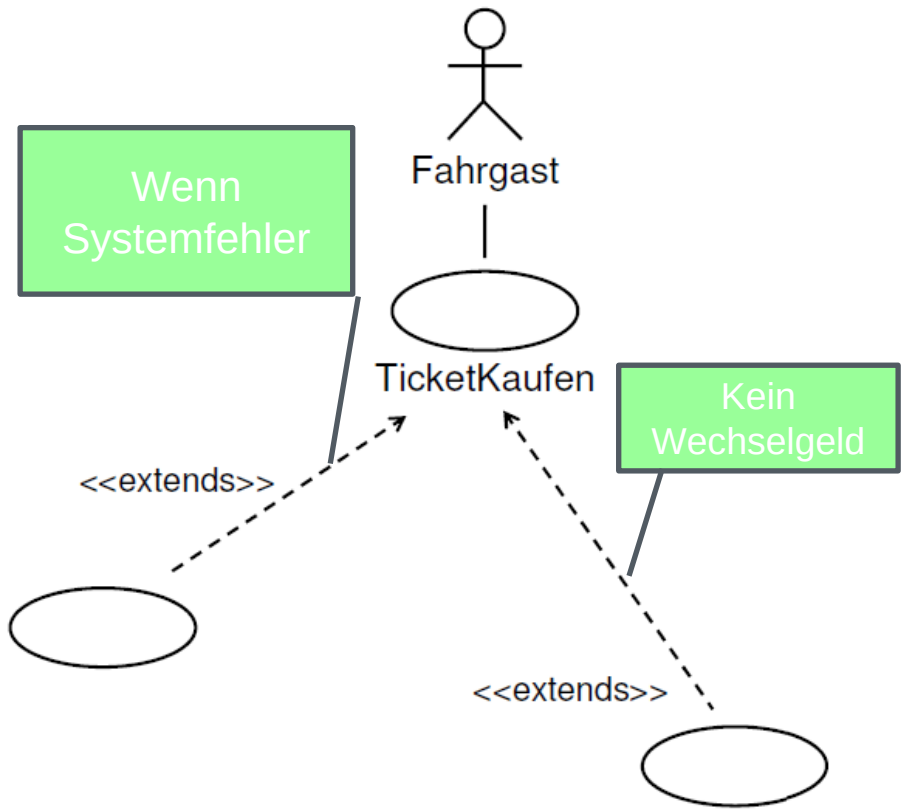


Alternative Darstellung:



Extends

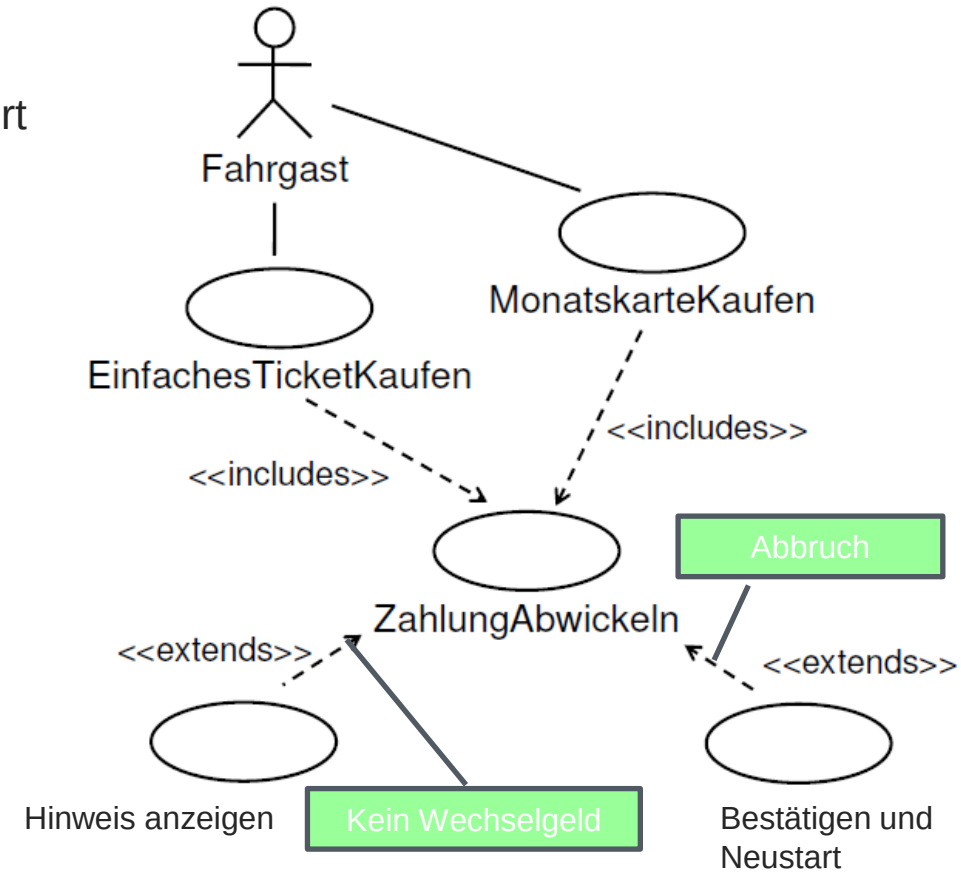
- **Die <<extends>>-Beziehung**
- <<extends>>-Beziehung dient der Repräsentation von Ausnahmefällen und selten aufgerufenen Sonderfällen.
- Diese werden als eigene AFs herausgezogen, um die Beschreibung klarer zu machen. Ihr Ereignisfluss ersetzt ggf. den Ereignisfluss des erweiterten AFs
- Erweiternde Anwendungsfälle können auch mehr als einen AF erweitern.
- Der <<extends>>-Pfeil zeigt vom erweiternden zum erweiterten AF
- Beziehung des Akteurs zum erweiternden AF wird nicht explizit eingezeichnet



Includes

- Die <<includes>>-Beziehung

  - Die <<includes>> -Beziehung repräsentiert Teilverhalten, das aus dem AF herausgezogen wird
  - Zweck: gleiche Beschreibung des Teilverhaltens bei anderen AFs wiederverwenden
  - Dient NICHT zur Beschreibung von Ausnahmefällen
  - Der <<includes>>-Pfeil zeigt vom benutzenden AF auf den Benutzten.

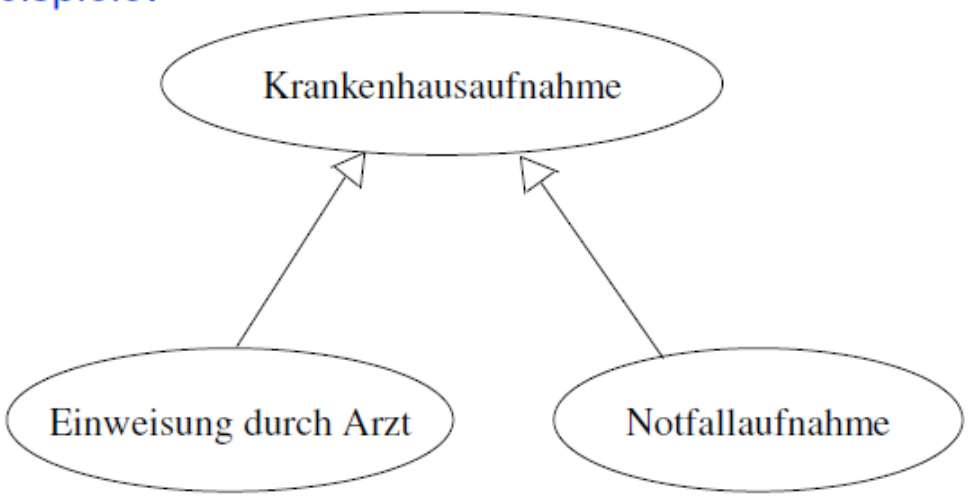


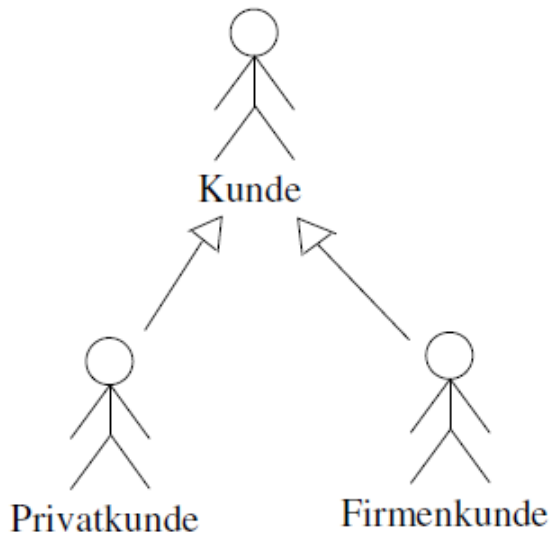
Generalisierung

Generalisierungsbeziehung

- Zwischen Anwendungsfällen und zwischen Akteuren können Generalisierungsbeziehungen spezifiziert werden
- Notation dafür durch Pfeil mit unausgefülltem Dreieck als Spitze vom spezielleren zum generelleren AF/Akteur (analog zu Vererbung in Klassendiagrammen ...)
- Speziellerer AF (bzw. Akteur) kann in der Rolle des generelleren AFs (bzw. Akteurs) an allen Beziehungen des generelleren AFs (bzw. Akteurs) teilnehmen, ohne, dass dies explizit eingezeichnet wird (Vermeiden von Redundanz).

Beispiele:

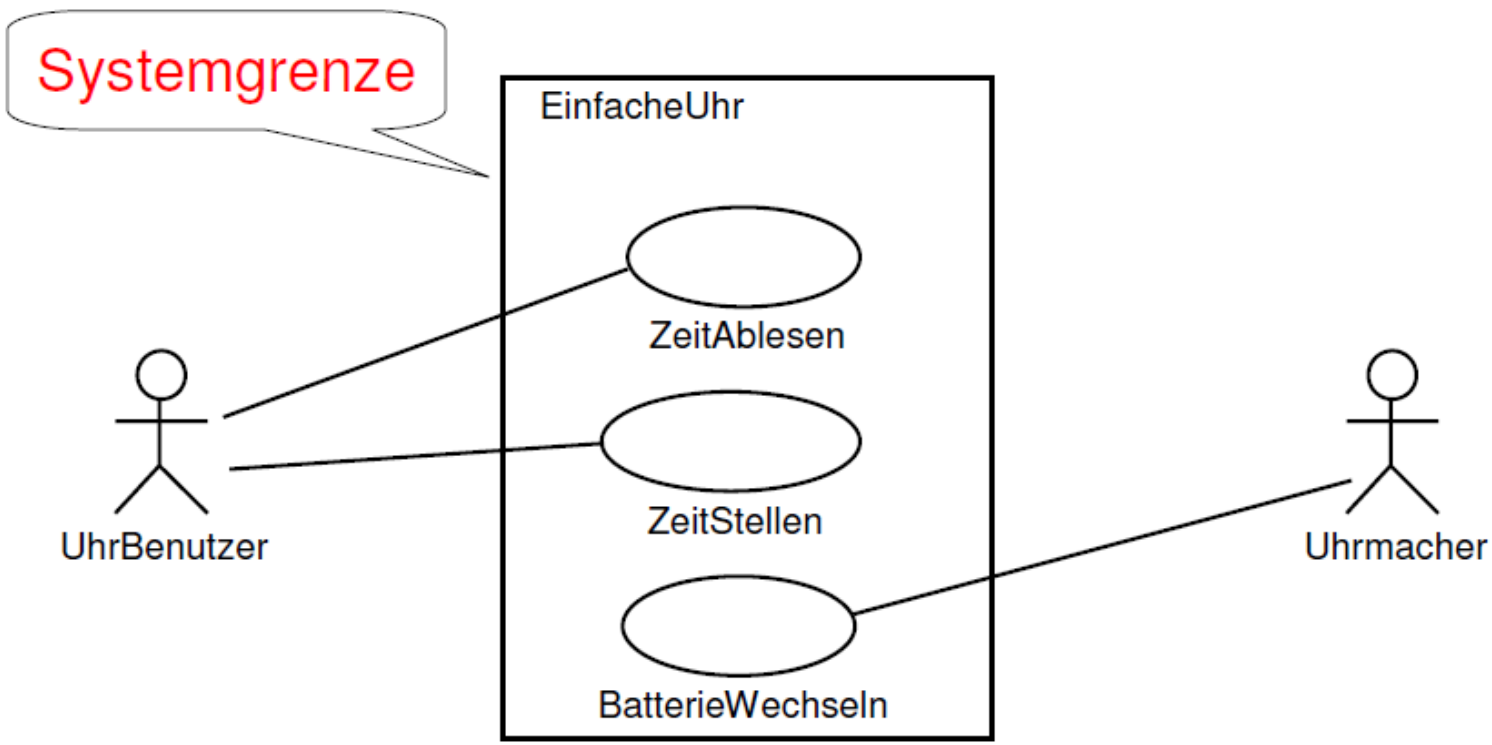




Systemgrenze

Systemgrenze

- AF-Modell dient insbesondere auch der Abgrenzung des Systems:
  Was tut das System und was nicht ?
- Systemgrenze wird als Kasten gezeigt.
- Akteure stehen immer außerhalb, AFs immer innerhalb der Systemgrenze.



UML Zusammenfassung

Anwendungsfalldiagramm: Zusammenfassung

- Zentrales Instrument beim Einstieg in die Anforderungsanalyse (Requirements Engineering)

- Graph mit folgenden Knoten:
  - Akteur: (actor)
    Rolle eines Benutzers im System (ggf. mehrere Rollen pro Benutzer)
    Ausgangspunkt für Bestimmung der Anwendungsfälle
    auch nicht-menschliche Akteure möglich (z.B. anderes System)
    dargestellt als Strichmännchen
  - Anwendungsfall: dargestellt durch Ellipse

- Kanten:
  - ungerichtete Kante zwischen Akteur und zugehörigem Anwendungsfall
  - gerichtete Kante (gestrichelt) und Beschriftung <<includes>> oder <<extends>> zwischen AFs
  - gerichtete Kanten (Pfeile) zur Erfassung von Generalisierungsbeziehungen

- Ein Kasten veranschaulicht die Systemgrenze

UML Werkzeug

ArgoUML

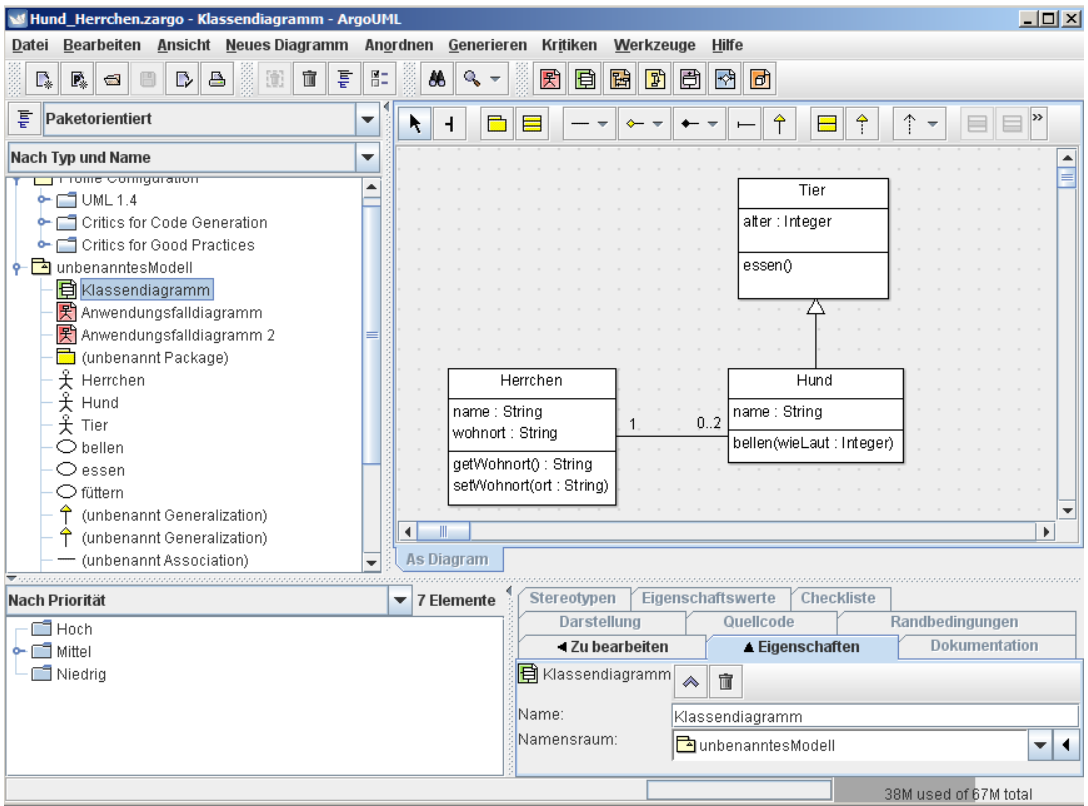
http://argouml.tigris.org

Welcome to ArgoUML

ArgoUML is the leading open source UML modeling tool and includes support for all standard UML 1.4 diagrams. It runs on any Java platform and is available in ten languages. See the feature list for more details.

ArgoUML 0.26 and 0.26.2 were downloaded over 80,000 times and are in use all over the world. See the download statistics.

ArgoUML is distributed under the Eclipse Public License (EPL) 1.0.



Aufgabe

Aufgabe 1.3 Anwendungsfalldiagramm

- Stellen Sie die nachfolgend textuell beschriebene Situation mittels eines Anwendungsfalldiagramms nach UML 2.0 grafisch korrekt dar:

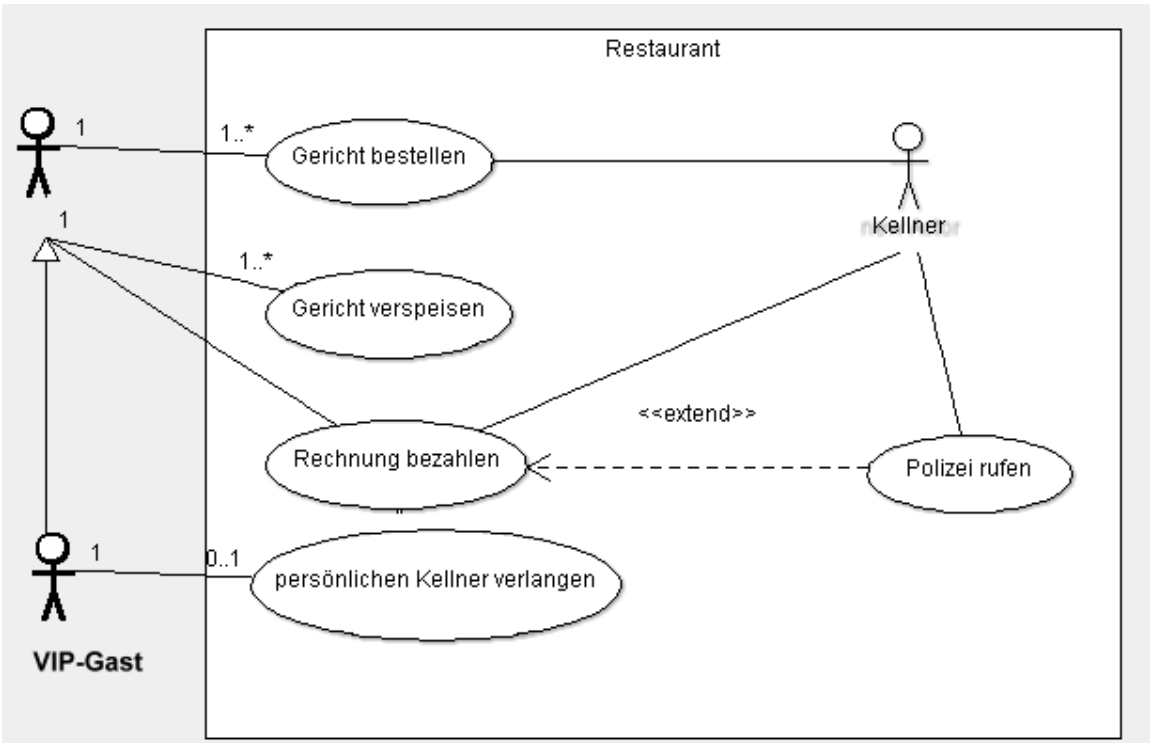
Ein Kunde möchte mit der Geldkarte am Automaten "Geld abheben". Der Akteur "Kunde" ist eine Generalisierung von "Kunde der eigenen Bank" und "Kunde fremder Banken". Die beiden spezialisierten Akteure kommunizieren über die Rolle Kunde mit dem Anwendungsfall „Geld abheben", der für beide Kundenarten gleichermaßen abläuft. Dieser Anwendungsfall enthält den Anwendungsfall "Konto- und Pin-Kontrolle", bei dem die Berechtigung des Kunden zur Kartennutzung überprüft wird. Wurde mehrfach eine falsche PIN eingegeben, wird die Karte eingezogen. Um dies zu modellieren wird der Anwendungsfall " Konto- und Pin-Kontrolle " mit dem Anwendungsfall "Karte einziehen" erweitert. Dieser wird nur unter der Bedingung abgearbeitet, dass der Kunde sich mehrfach nicht identifizieren konnte.

Quelle: Skript Prof. Dr. O. Lazar

Aufgabe

Aufgabe 1.4 Anwendungsfalldiagramm (5 Min.)

- Finden Sie die Fehler, die bezüglich der UML-Notation gemacht wurden? Notieren (Beschreiben) Sie die Fehler, die Sie finden kurz:



Quelle: Skript Prof. Dr. O. Lazar

Aufgabe 1.5 Anwendungsfalldiagramm

- Für die folgende Problembeschreibung ist der Use-Case mittels Template zu spezifizieren. Für eine Seminarverwaltung ist eine Anmeldung zu bearbeiten. Ist es ein neuer Kunde, dann sind die Daten zu erfassen. Existiert der Kunde bereits, dann ist zu prüfen, ob die Daten aktualisiert werden müssen. Weiterhin ist zu prüfen, ob das gewünschte Seminar angeboten wird und ob noch ein Platz frei ist. Wenn die Anmeldung durchgeführt werden kann, erhält der Kunde eine Anmeldebestätigung. Wenn kein Platz mehr frei ist, oder das angegebene Seminar nicht angeboten wird, dann muss beim Kunden nachgefragt werden, ob ein alternatives Seminar in Frage kommt. (15 Min.)

Use-Case-Bezeichnung:	**Anmeldung bearbeiten**
Ziel:	
Vorbedingung:	
Nachbedingung Erfolg:	
Nachbedingung Fehlschlag:	
Akteure:	
Auslösendes Ereignis:	
Beschreibung (Ablauf):	
Erweiterungen:	
Alternativen:	

Quelle: Skript Prof. Dr. O. Lazar

Aufgabe

Aufgabe 1.6 Anwendungsfalldiagramm (20 Min.)

- Für die Stadtbibliothek soll ein Softwaresystem entwickelt werden. Beschreiben Sie die typischen Arbeitsabläufe zur Ausleihe und Verwaltung von Büchern in einem Anwendungsfalldiagramm. Beteiligte Akteure sind Bibliothekaren, die Bibliotheksverwaltung und die Leser. Berücksichtigen Sie auch das Mahnwesen (durch die Verwaltung). In der Bibliothek können Leser mittels der Mitgliederkarte an Selbstbedienungs-Terminals selbst die eigene Adresse ändern, Bücher suchen und vorbestellen.

a. Stellen Sie die zuvor textuell beschriebene Situation mittels eines Anwendungsfalldiagramms nach UML 2.0 grafisch korrekt dar.

b. Beschreiben Sie den Prozess "Ausleihen eines Buches" mittels des vorgestellten Templates für Use Cases (aus Aufg. 1.5).

Quelle: Skript Prof. Dr. O. Lazar

Übersicht

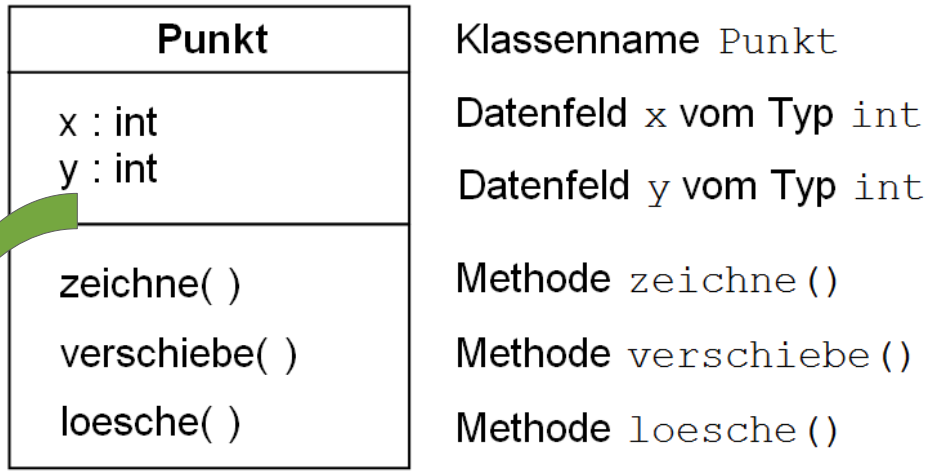
1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

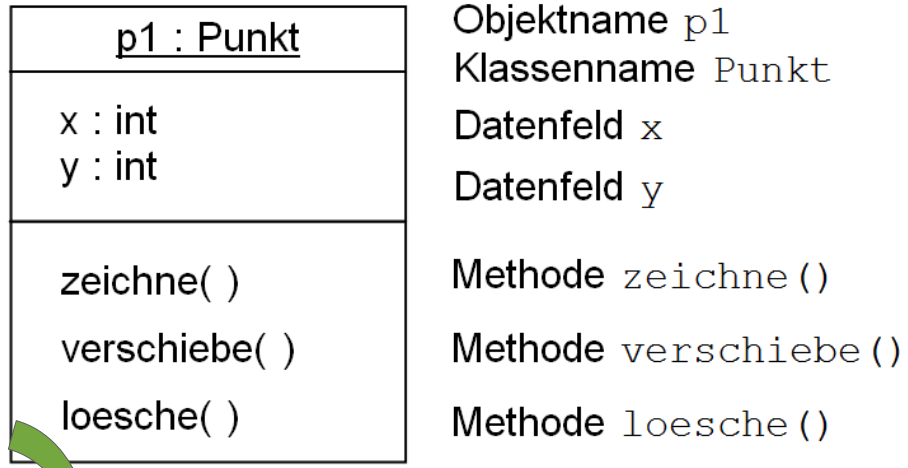
Inhalte

- ✓ Grundbegriffe der OO-Programmierung
- ✓ Objektorientierte Konzepte I
- ✓ UML
- ✓ Objektorientierte Konzepte II

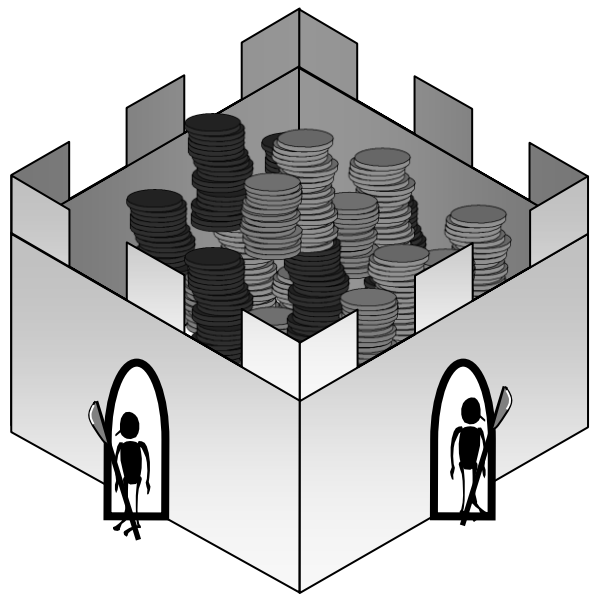
Einführung in der Programmiersprache Java

Modellierung mit Klassen und Objekten I



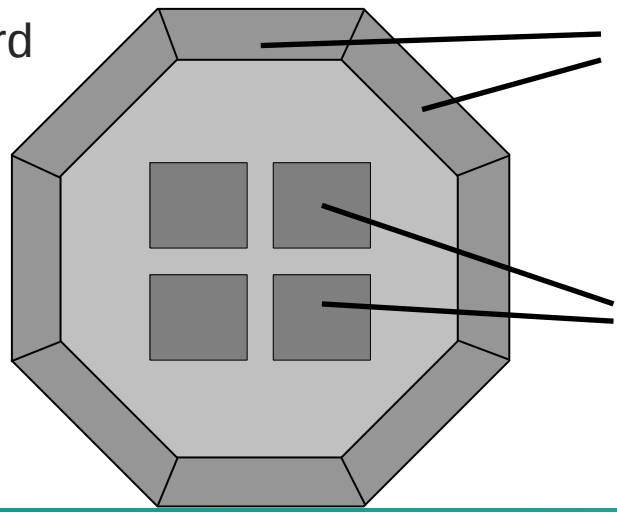


- Eine Klasse trägt einen Klassennamen und enthält Datenfelder und die Methoden, um auf diese Datenfelder zuzugreifen
- Objekt, also konkrete Instanz einer Klasse
- p1:Punkt wäre ein anonymes Objekt der Klasse Punkt



Modellierung mit Klassen und Objekten II

- Zugriff auf die Datenfelder erfolgt in der Regel über Zugriffsmethoden
- Methoden erfüllen die Aufgaben:
  - Werte der Datenfelder eines Objektes auszugeben
  - Datenfelder zu verändern
  - Neue Ergebnisse zu berechnen (mit Hilfe der gespeicherten Werte)
- Datenfelder definieren die Datenstruktur der Objekte, die Methoden bestimmen das Verhalten der Objekte
- Jede Kombination von Datenfeldern stellt einen Zustand dar, der als mikroskopischer Zustand eines Objektes bezeichnet wird
- Makroskopische Zustände haben eine Bedeutung für die Anwendung des Objektes.

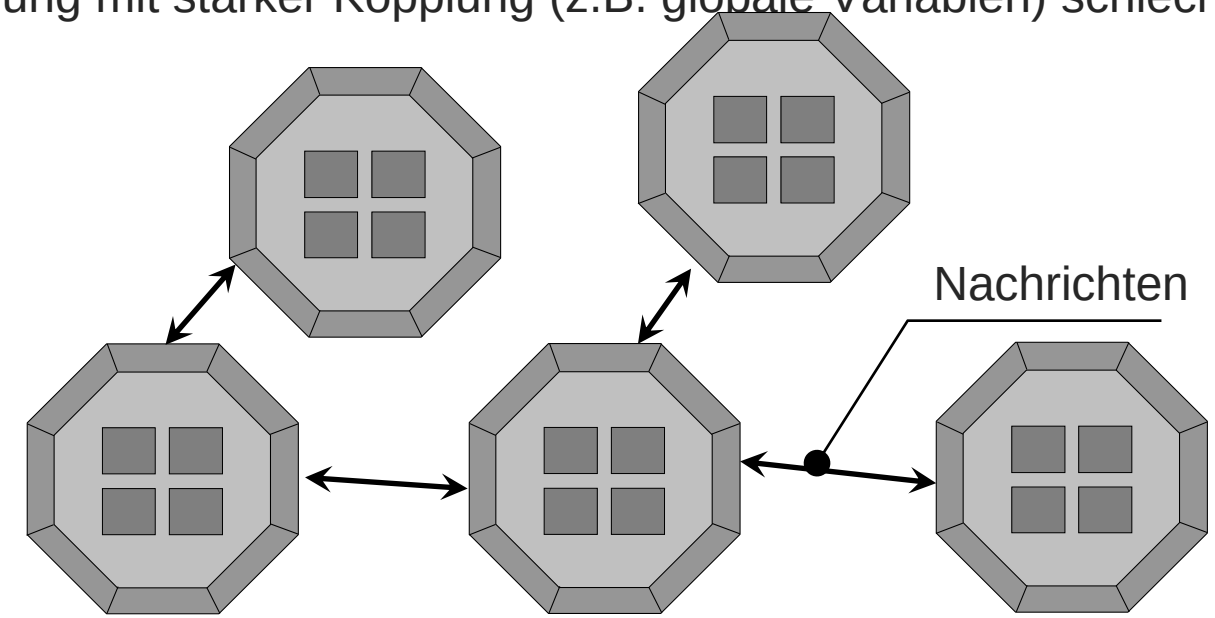


Methoden sind die Schnittstellen des Objektes.

Daten sind gekapselt. Sie können nur von den eigenen Methoden des Objektes manipuliert werden.

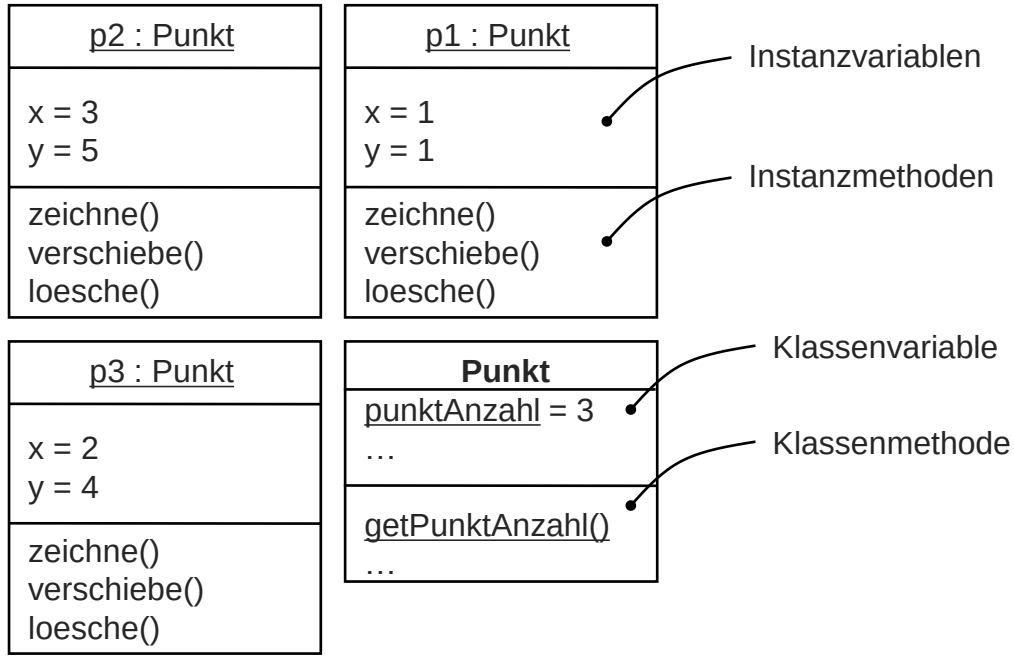
Modellierung mit Klassen und Objekten III

- Eine funktionale Leistung, die ein System zur Verfügung stellt, wird als Anwendungsfall (Use Case) bezeichnet. Ein Use Case stellt einen (Teil) eines Geschäftsprozesses dar, der durch ein EDV-System unterstützt wird.
- Kommunikation zwischen Objekten mittels Nachrichten (Aufruf von Methoden): Mit dem Aufruf einer Methode (schwache Kopplung) wird die Kontrolle an das Objekt, an das die Nachricht gerichtet ist, übergeben. Wie das Objekt handelt, ist Sache des Objekts.
- Prozedurale Programmierung mit starker Kopplung (z.B. globale Variablen) schlechter erweiterbar.



Modellierung mit Klassen und Objekten IV

- Instanzvariablen sind für die jeweilige Instanz gültig und können in jeder Instanz eine individuelle Ausprägung annehmen.
- Instanzmethoden greifen nur bei ihrer jeweiligen Instanz.
- Klassenvariablen stellen globale Variablen für alle Objekte einer Klasse dar.
- Instanzmethode kann auch auf Klassenvariablen zugreifen.

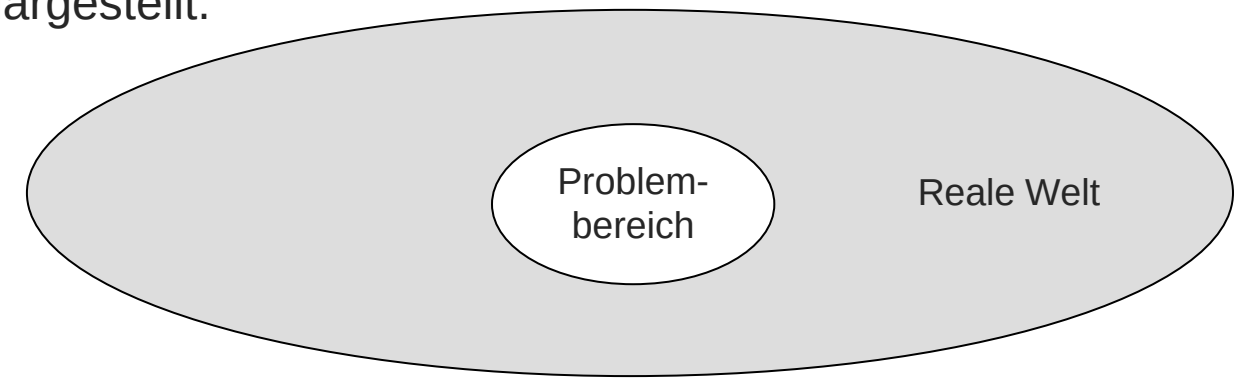


Das Konzept der Kapselung

- Information Hiding oder Geheimnisprinzip
  - Die inneren Eigenschaften sollen vor der Außenwelt verborgen sein
  - Nur die Aufrufschnittstelle der Schnittstellenmethoden soll exportiert werden
  - Dadurch können Teile des Systems modifiziert werden, ohne einen Einfluss auf andere Teile zu nehmen, solange die Schnittstellen konstant bleiben.
  - Kapselung: Nach außen sind nur die definierten Schnittstellen sichtbar. Eigenschaften die intern bleiben sollen, bleiben verborgen.

Abstraktion und Brechung der Komplexität I

- Komplexer Sachverhalt aus der realen Welt wird in einem Programm auf das Wesentliche konzentriert und damit vereinfacht dargestellt.
- Die Kunst der Abstraktion ist es
  - das Wesentliche zu erkennen
  - das Unwesentliche wegzulassen

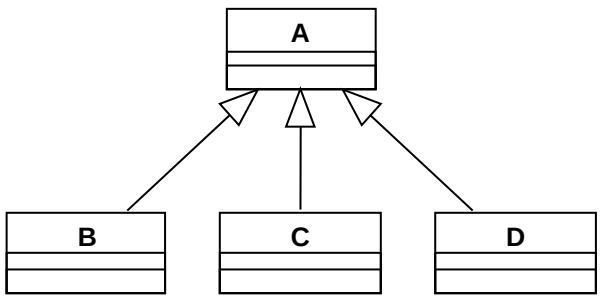


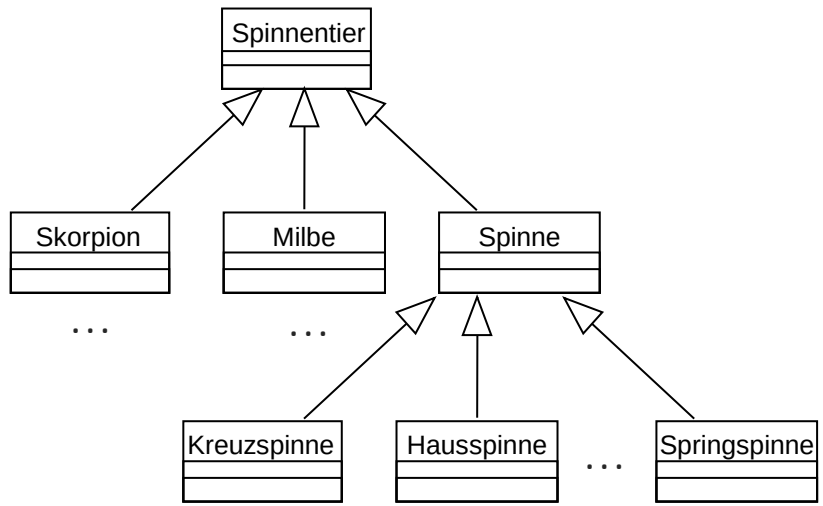
- Zum Finden der Objekte des Problembereichs
- Zum Erkennen der Datenfelder und Methoden eines Objekts
- Zur Festlegung der Schnittstellen eines Objektes
- Zur Bildung von Hierarchien

Abstraktion und Brechung der Komplexität II

- Vererbungshierarchie (auch „kind-of“ – Hierarchie)
- Klassen werden in Abstraktionsebenen angeordnet
  - **Generalisierung**: wenn man in der Hierarchie von unten nach oben geht
  - **Spezialisierung**: wenn man in der Hierarchie von oben nach unten geht

(die jeweils tiefer stehende Klasse ist eine Spezialisierung der Oberklasse)





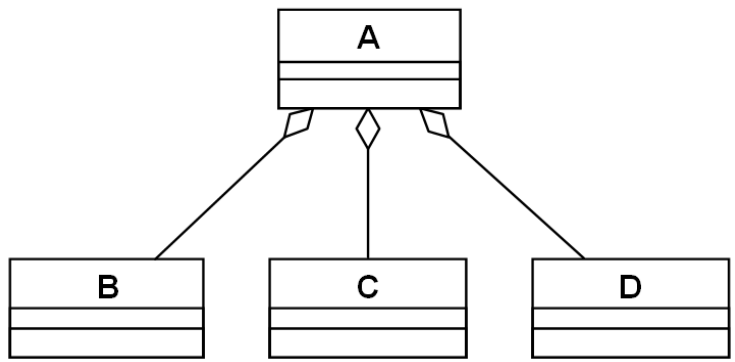
Abstraktion und Brechung der Komplexität III

- Zerlegungshierarchie (auch „part-of“ – Hierarchie)
- Ein Objekt kann als Datenfelder andere Objekte enthalten in Form von

  einer Komposition oder einer Aggregation
- **Komposition**: Objekte haben die gleiche Lebensdauer, sind „verschweißt“
- **Aggregation**: Lebensdauern von zusammengesetztem Objekt und den Komponenten ist entkoppelt.

  Beispiel: Auch wenn A zerstört wird,

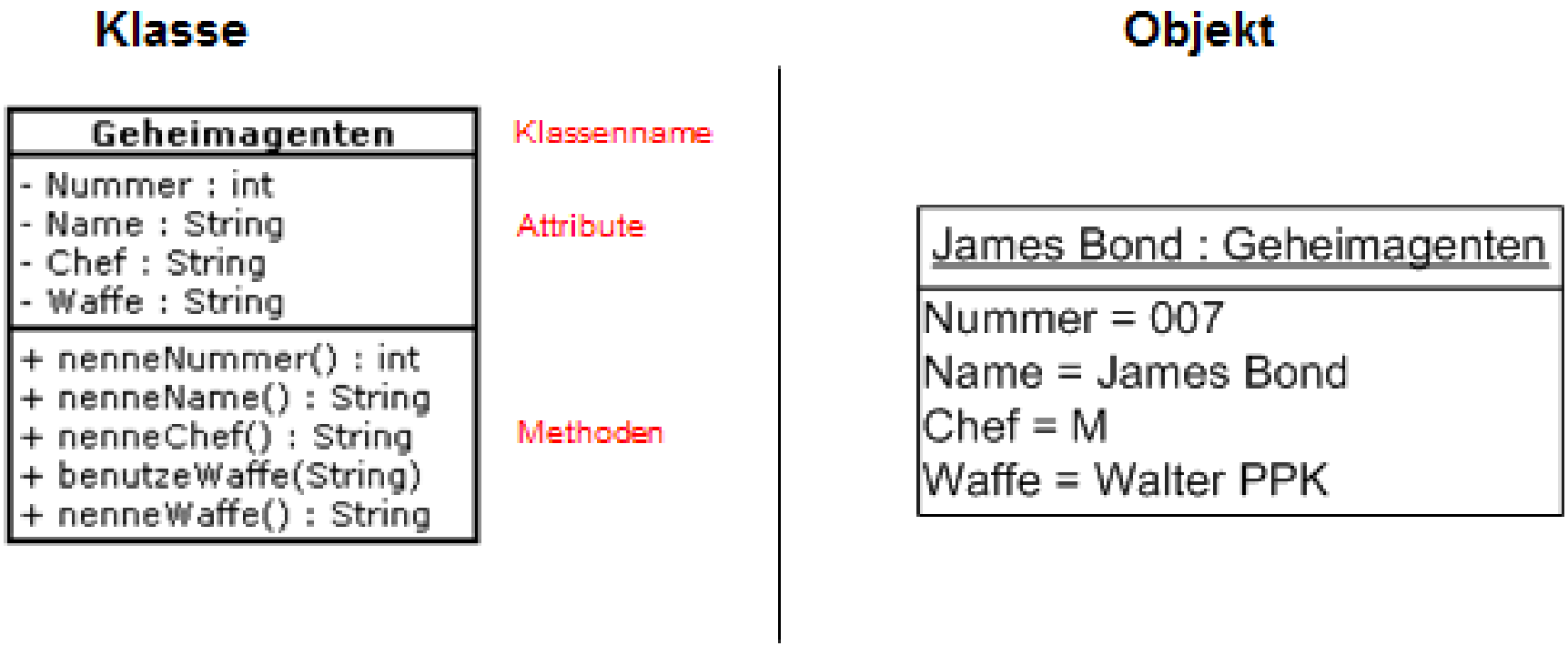
  existieren B, C und D weiter.



- Eine Referenz auf ein Objekt enthält als Wert die Adresse des Objektes, auf das die Referenz zeigt. Die Adresse gibt an, an welcher Stelle das Objekt im Arbeitsspeicher liegt.
- Software-Umsetzung mittels Referenzen, in Java nur Aggregation (C++ auch Komposition)

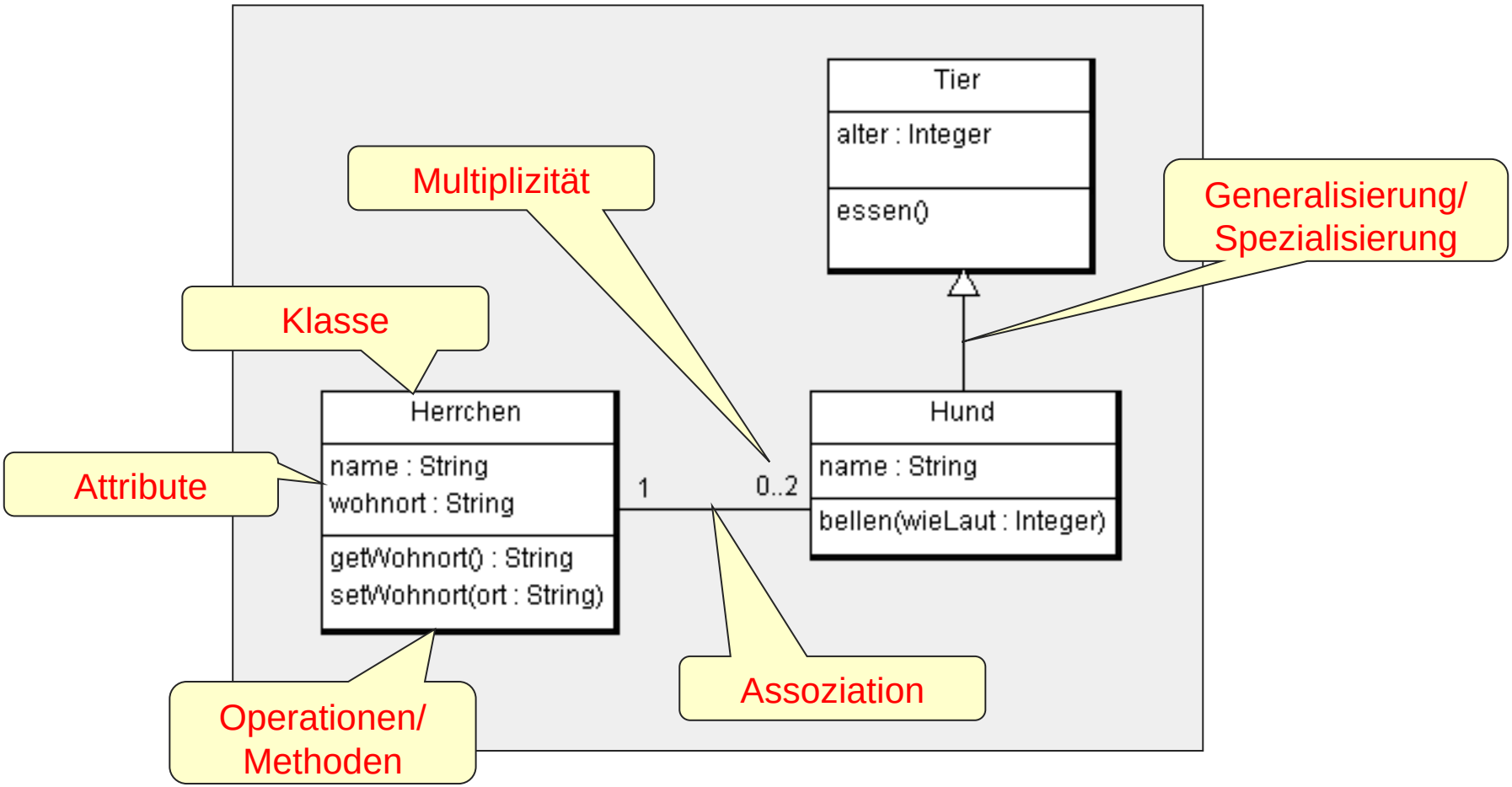
Klassen und Objekte I

- Klassen und Objekte
  - Klassen sind die Baupläne für Objekte
  - Klassen sind Datentypen, Objekte sind Instanzen dieser Datentypen
  - Objekte werden gemäß den in den Klassen abgelegten Bauplänen erzeugt



Klassen und Objekte II

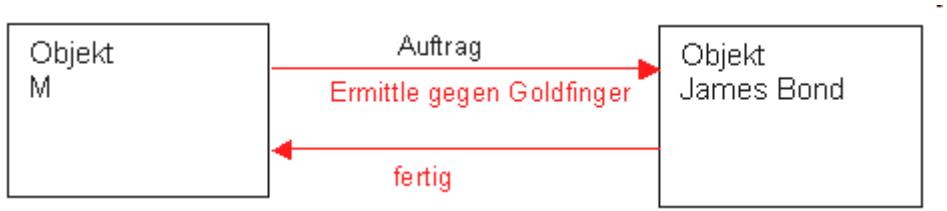
- Klassendiagramme repräsentieren die Struktur eines Systems



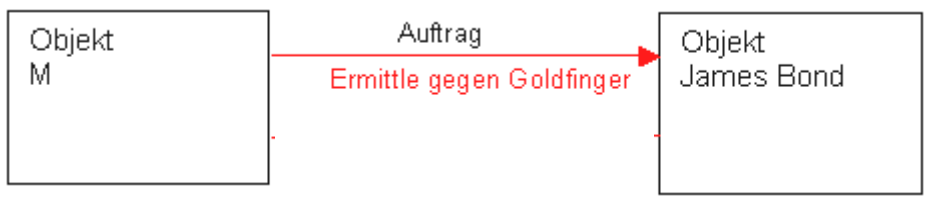
Klassen und Objekte III

- Zusammenarbeit von Objekten
  - Damit ein Objekt seine Umwelt und damit andere Objekte wahrnehmen kann, muss es auf Nachrichten reagieren können.
  - Diese Nachrichten führen beim Empfängerobjekt entweder dazu einen Auftrag selbst zu erfüllen oder eine Anfrage zu beantworten.

mit Rückgabe:
z.B. `boolean`



Ohne Rückgabe:
`void`



mit Rückgabe:
`String`

Aufgabe

Aufgabe 1.7 Klassendiagramm (20 Min.)

- Gegeben sei der folgende Sachverhalt:

> Jede Person hat einen Namen, eine Telefonnummer und E-Mail.
> Jede Wohnadresse wird von nur einer Person bewohnt. Es kann aber sein, dass einige Wohnadressen nicht bewohnt sind. Den Wohnadressen sind je eine Strasse, eine Stadt, eine PLZ und ein Land zugeteilt. Alle Wohnadressen können bestätigt werden und als Beschriftung (für Postversand) gedruckt werden.
> Es gibt zwei Sorten von Personen: Student, welcher sich für ein Modul einschreiben kann und Professor, welcher einen Lohn hat. Der Student besitzt eine Matrikelnummer und eine Durchschnittsnote.

- Modellieren Sie diesen Sachverhalt mit einem UML Klassendiagramm.

Quelle: Skript Prof. Dr. O. Lazar

Aufgabe

Aufgabe 1.8 Klassendiagramm (10 Min.)

- Gegeben sei der folgende Sachverhalt:

> In einem Grafiksystem bilden beliebig viele Grafikobjekte eine Gruppe. Eine Gruppe enthält Informationen über den Gruppennamen und die zugehörige Farbe zum Zeichnen seiner Grafikobjekte. Grafikobjekte sind Kreise und Rechtecke. Jedes Grafikobjekt hat eine bestimmte x- und y-Koordinate, Kreise haben einen Radius und Rechtecke eine Länge und Breite. Definieren Sie für alle Attribute Methoden zum Lesen und Setzen der Werte.

- Modellieren Sie diesen Sachverhalt mit einem UML Klassendiagramm.

Quelle: Skript Prof. Dr. O. Lazar

Erstes Programmbeispiel mit Objekten

- Klassenvariablen und Klassenmethoden erhalten bei Java das Schlüsselwort *static*
- Methoden sind in der Regel *public*
- Datenfelder sind in der Regel *private*
- Klassenbeschreibung ist das Schema zur Bildung von Objekten dieser Klasse und enthält
  - den Namen der Klasse
  - die Datenfelder dieser Klasse
  - die Methoden dieser Klasse

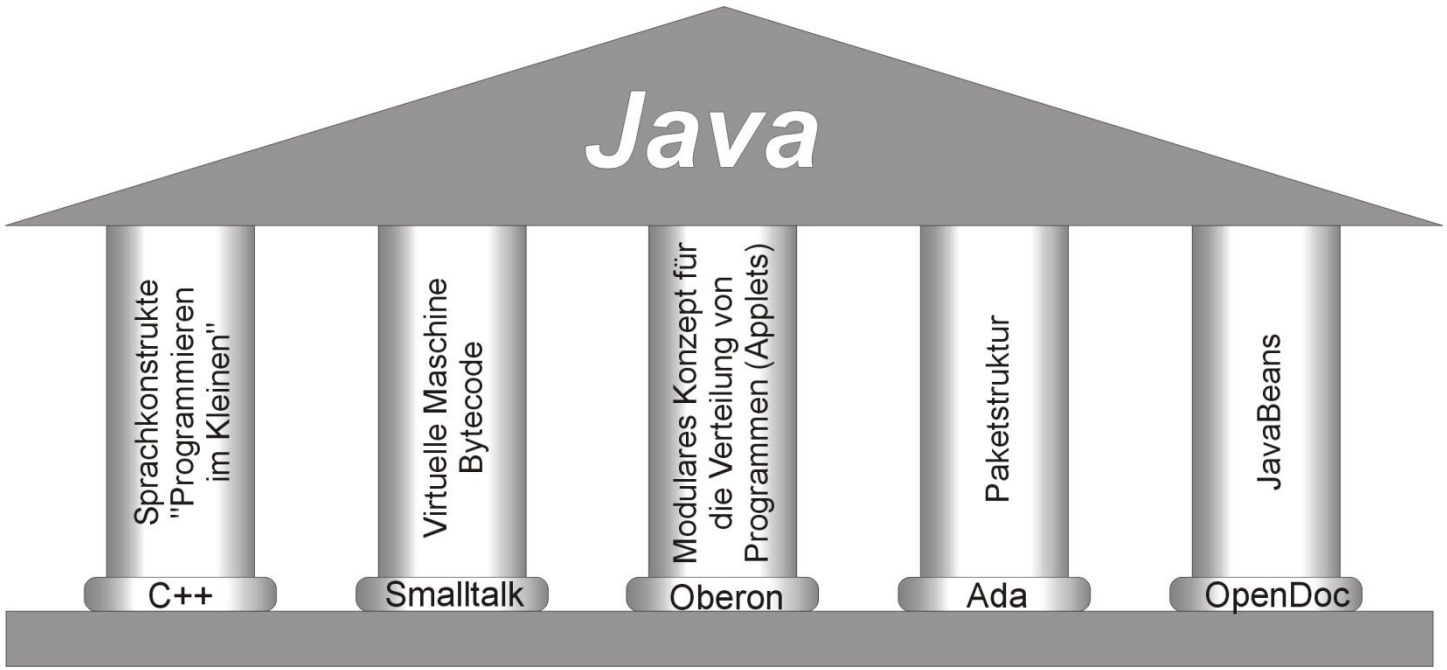
```
// Datei: Punkt.java
// Deklaration der Klasse Punkt. Dem Compiler wird gesagt, dass es
// eine Klasse Punkt gibt.
public class Punkt
{
   private int x;                    // Datenfelder für die x- und
   private int y;                    // y-Koordinate vom Typ int

   public int getX()                 // eine Methode, um den Wert
   {                                 // von x abzuholen
      return x;
   }
   public int getY()                 // eine Methode, um den Wert
   {                                 // von y abzuholen
      return y;
   }
   public void setX (int i)          // eine Methode, um den Wert
   {                                 // von x zu setzen
      x = i;
   }
   public void setY (int i)          // eine Methode, um den Wert
   {                                 // von y zu setzen
      y = i;
   }
   // Mit main() beginnt eine Java-Anwendung ihre Ausführung.
   public static void main (String[] args)
   {
      Punkt p = new Punkt();         // Punkt erzeugen
      p.setX (3);                    // Aufruf der Methode setX()
      p.setY (2);                    // Aufruf der Methode setY()

      System.out.println ("Die Koordinaten des Punktes p sind: ");
      System.out.println (p.getX()); // Wert von x wird ausgegeben
      System.out.println (p.getY()); // Wert von y wird ausgegeben
   }
}
```

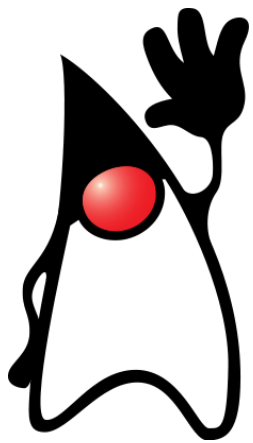
Übersicht

1. Grundbegriffe der Programmierung	Inhalte
2. Einfache Beispielprogramme	
3. Datentypen und Variablen	✓ Grundbegriffe der OO-Programmierung
4. Ausdrücke und Operatoren	
5. Kontrollstrukturen	✓ Objektorientierte Konzepte
6. Blöcke und Methoden	
7. Klassen und Objekte	✓ UML
8. Vererbung und Polymorphie	
9. Pakete	✓ Objektorientierte Konzepte
10. Ausnahmebehandlung	
11. Schnittstellen (Interfaces)	✓ Einführung in der Programmiersprache Java
12. Geschachtelte Klassen	
13. Ein-/Ausgabe und Streams	
14. Applets / Oberflächenprogrammierung	

Sprachkonzepte von Java

- Die Väter von Java





Die Urversion von Java wurde vom Frühjahr 1991 bis Sommer 1992 unter dem Namen *The Green Project* von 13 Entwicklern im Auftrag des US-amerikanischen Computerherstellers Sun Microsystems entwickelt. Ein Überbleibsel aus dem Green-Projekt ist der Duke von Joe Palrang, der zum bekannten Symbol bzw. Maskottchen geworden ist.

Eigenschaften von Java

- Einfachheit und Stabilität
  - gegenüber C/C++ wurden verschiedene Sprachkonstrukte (z.B. Zeigerarithmetik) weggelassen
- Objektorientiertheit
  - echte Objektorientierte Sprache
  - es ist nicht möglich Methoden/Attribute von den Klassen/Objekten zu trennen
- Verteilbarkeit (Client/Server, Applets)
  - optimal für Client/Server-Programmierung geeignet
  - z.B. Applets oder Java-Webstart
- Sicherheit
  - z.B. Verwendung von Zertifikaten, Sandbox
- Portierbarkeit (→ siehe Java Virtuelle Maschine)
  - Java ist plattformunabhängig (also unabhängig von Betriebssystem und Rechner-Hardware)

Die Java-Plattform I

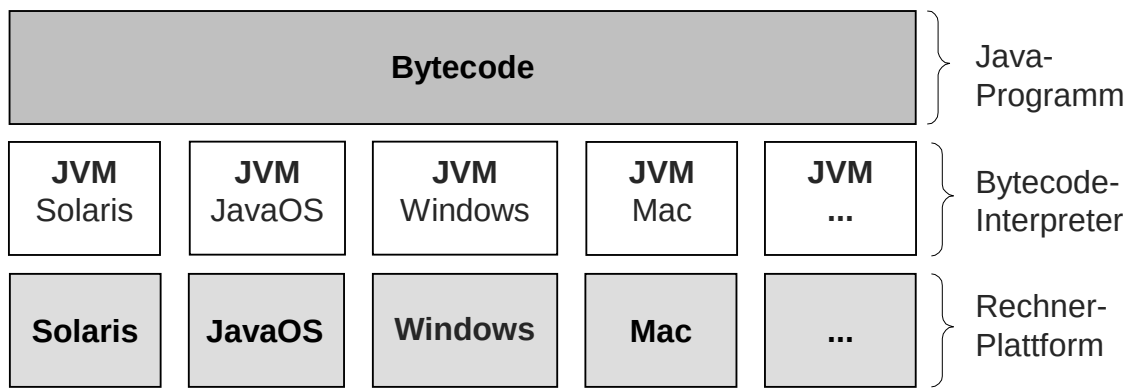
Zur Java-Plattform gehören:

- die Programmiersprache Java
- Werkzeuge/Tools (z.B. Java-Compiler → javac)
- die Java Virtuelle Maschine (JVM) → Bytecode-Interpreter
- umfassende Klassenbibliothek
- JavaFX (seit Java 7 Update 6)

- Es gibt drei Varianten der Java-Plattform:
  - Standard Edition (Java SE)
  - Enterprise Edition (Java EE) → Server Anwendungen
  - Micro Edition (Java ME) → mobile Endgeräte

Die Java-Plattform II

- Die Java Virtuelle Maschine (JVM)
  - Bei der Kompilierung von Java wird aus dem Quellcode nicht Maschinencode, sondern ein Zwischencode (**Bytecode**) erzeugt
  - dieser Bytecode ist nun für alle Plattformen gleich
  - Jede Rechner-Plattform besitzt eigenen Bytecode-Interpreter (JVM)
- …ist die Laufzeitumgebung und übernimmt verschiedene Aufgaben
  - Speicherverwaltung des Programms
  - Ein-/Ausgabeoperationen
  - setzt den Bytecode in Maschinencode des jeweiligen Prozessors um
  - Interaktionen mit dem Betriebssystem
- Das Laden der JVM erfolgt über die
  main()-Methode der Startklasse
  Aufruf: *java Klassenname*



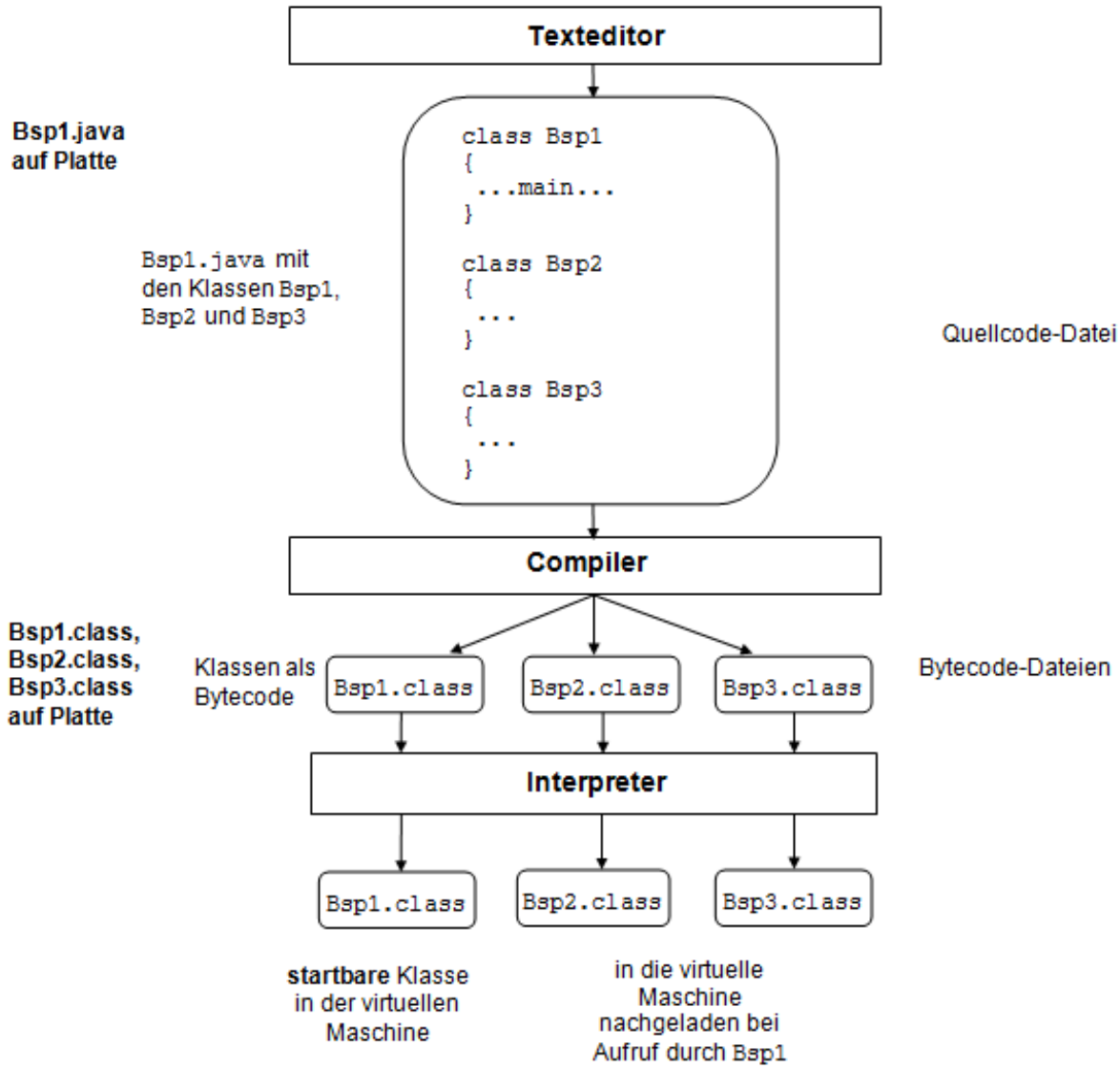
Die Java-Plattform III

- Java Klassenbibliothek
  - das Programmieren von Java-Anwendungen wird durch eine umfangreiche Klassenbibliothek vereinfacht
  - Java-API (Application Programming Interface)

- Unterteilung in die Bereiche
  - Java Base Libraries (grundlegende Funktionalitäten, z.B. string Verarbeitung)
  - Java Integration Libraries (systemübergreifende Funktionalität, z.B. RMI, DB)
  - Java User Interface Libraries (Schnittstellen z.B. für Drucker, Audio, GUI)

Programmerzeugung und -ausführung I

- In Java wird kein ausführbares Programm erzeugt
- Es wird Bytecode interpretiert, statt einer ausführbaren Datei gestartet.
- Klassen, die eine main()-Methode haben, können zum **Starten einer Java-Anwendung** verwendet werden



Programmerzeugung und -ausführung II

- **Vier Schritte des Kompilierens**
  - Lexikalische Analyse: Erkennung von Symbolen/Wörtern in der Zeichenkette des Programms, z.B. Schlüsselwörter, Operatoren usw.
  - Syntaxanalyse: Sind bei der lexikalischen Analyse erkannte Wörter formal zulässig geschrieben?
  - Semantikanalyse: Verwendung von Namen im Rahmen ihrer Gültigkeitsbereiche, Typverträglichkeit bei Ausdrücken
  - Codeerzeugung: Erzeugung des Bytecode aus dem analysierten Source Code

- Virtuelle Maschine stellt die Laufzeitumgebung für das auszuführende Programm:
  - Laden in die VM: java Startklasse, restlich benötigte Klassen werden nachgeladen
  - Ausführen des Bytecodes (Prozessor und OS abhängig)

Programmerzeugung und -ausführung III

- Eigenschaften des Bytecodes:
  - Besteht aus Instruktionen für die VM.
  - Instruktion ist je ein Byte lang.
  - Unabhängig von Hardware und Betriebssystem.
  - Ist maschinennah.
  - Wird mittels Bytecode-Interpreter ausgeführt.
  - Wegen Optimierung ist der Code sehr kompakt.
  - Portabilität.
  - Interpreter kann mit einer Sicherheitsschicht ausgestattet werden.

Das Java Development Kit

- Java Runtime Environment (JRE)
  - beinhaltet nur solche Bestandteile, die zum Ausführen von Java-Programmen benötigt werden
  - Bytecode-Interpreter für die jeweilige Plattform
  - Java Klassenbibliothek
- Java Development Kit (JDK)
  - Beinhaltet neben dem JRE auch Java-Entwicklungswerkzeuge
  - Java-Compiler (javac)
  - Java Dokumentationswerkzeug (javadoc)
  - Java Archiver (Erzeugung von JAR-Dateien)
  - Signieren von JAR-Dateien (jarsigner)
  - Visuelle Anzeige von Speicherverbrauch, Anzahl Threads, Java Heap-Space (jconsole)
  - …

Java-Anwendungen und Internet-Programmierung

- Nicht verwechseln:
  - Java-Anwendung/Applikation: Programme die auf einer lokalen Maschine laufen.
  - Java-Applets: Im Normalfall in eine Website eingebunden und aus dem Internet auf einen lokalen Rechner geladen um dort ausgeführt zu werden.
  - Java-Servlets: Erzeugen dynamisch html-Seiten für einen Web Browser, serverseitige Ausführung.
  - JavaServer Pages: Einbinden von Java in html-Seiten zur dynamischen Seitengenerierung.
  - JavaScript: Scriptsprache ohne direkten Zusammenhang zu Java als objektorientierte Sprache.