Übungen zu

Objektorientierte Konzepte in Verteilten Systemen und Betriebssystemen

Wintersemester 2002/03

Teil I: Java

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

A Übersicht

A.1 Teil I: Java

- Die Sprache Java
- OO-Grundlagen in Java: Klassen, Objekte, Vererbung
- weitere OO-Konzepte in Java: Packages, Innere Klassen, Abstrakte Klassen, Final-Klassen und -Methoden
- automatische Tests mit JUnit
- Fehlerbehandlung, Ein-/Ausgabe, Threads

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

A Übersicht

A.2 Teil II: Verteilte Programmierung

- Sockets
- Serialization, Classloader
- RMI
- Message Passing, RPC, ORB
- ORB Erweiterungen: Sicherheit, Replikation, Persistenz

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

A Übersicht

A.3 Teil III: CORBA

- CORBA Programmieren in Java
- CORBA Services
- weiterführende Themen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

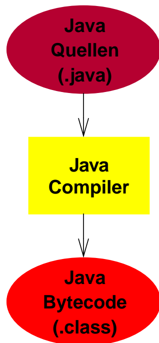
B Java - Überblick

- Was ist Java?
  - eine objektorientierte Programmiersprache mit folgenden Eigenschaften:
    - einfach
    - portabel
    - robust
  - ein sicheres Laufzeitsystem
    - beliebige Programme ohne Gefahr ausführen
    - Zugriffskontrolle auf lokale Ressourcen (z.B. Dateien) und das Netzwerk

© • Universität Erlangen-Nürnberg • Informatik 4, 2002
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

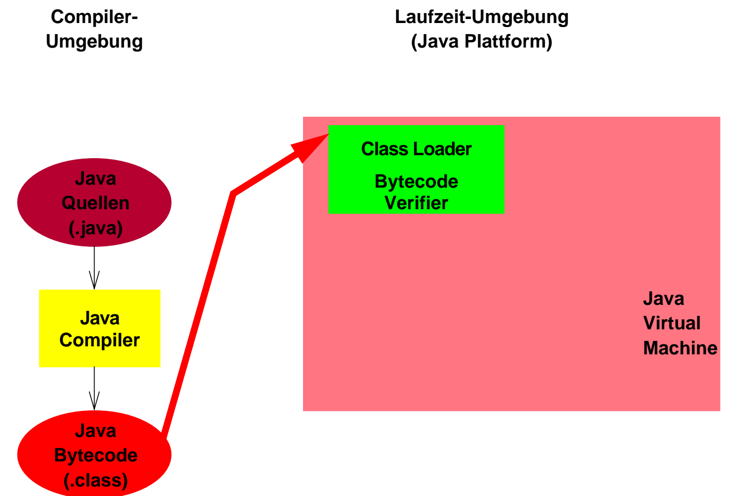
B.1 Das Java-System

Compiler-
Umgebung



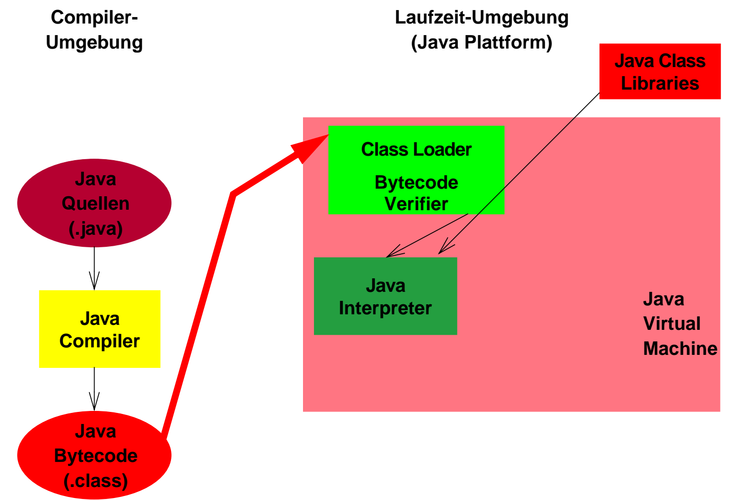
© • Universität Erlangen-Nürnberg • Informatik 4, 2002
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.1 Das Java-System (2)



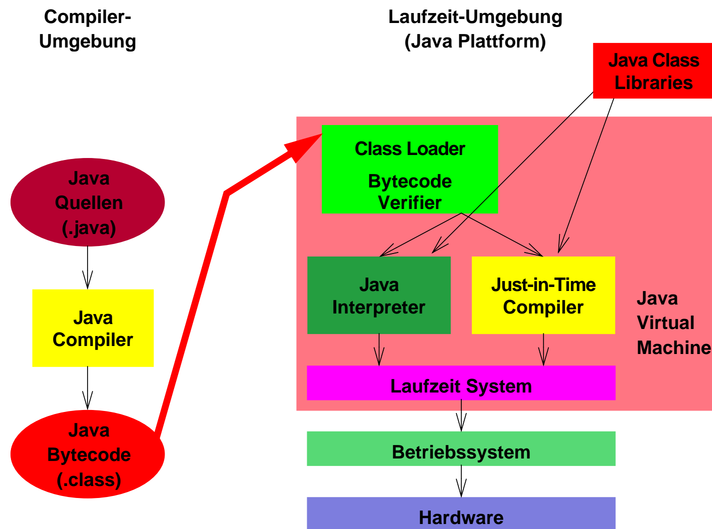
© • Universität Erlangen-Nürnberg • Informatik 4, 2002
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.1 Das Java-System (3)



© • Universität Erlangen-Nürnberg • Informatik 4, 2002
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.1 Das Java-System (4)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.2 Java vs. C/C++

- kein Präprozessor (#define)
- kein typedef, union, enum, struct
- keine Zeiger-Arithmetik
- keine Funktionen oder Prozeduren
- keine globalen Variablen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.3 Datentypen

- Primitive Datentypen:
  - byte, short, int, long (8/16/32/64 bit)
  - float, double (32/64 bit)
  - char (Unicode, 16 bit)
  - boolean (true, false)
  - “einfache” Operatoren: +,-,+=,<<,...
- Unterschiede zu C:
  - keine “unsigned” Datentypen
  - spezielle Datentypen “boolean” und Unicode char
  - kein `while (x--)`, stattdessen `while (x-- != 0)`
  - kein `if (ptr)`, stattdessen `if (ptr != null)`

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.4 Kommentare

- einzeilige Kommentare:

```
// Das ist ein Kommentar.
```

- mehrzeilige Kommentare:

```
/* Das ist
 * ein Kommentar.
 */
```

- Dokumentation (Erzeugt HTML-Dokumentation mit dem Tool `javadoc`):

```
/** Dieser Kommentar
 *  enthält javadoc-
 *  Dokumentation.
 */
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.5 Kontrollstrukturen

- Kontrollstrukturen wie in C:
  - for (... ; ... ; ...) { ... }
  - while (...) { ... } oder do { ... } while (...);
  - switch (...) { case .... }
  - if ....
  - **return**, **break**, **continue**
- Unterschiede zu C:
  - kein `goto`
  - `break` und `continue` mit Label:

```
outerloop: for ( .... ) {        // this is "outerloop"
    while ( .... ) {
        if ( ... ) break outerloop;// leave "outerloop"
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.6 Strings

- gute Unterstützung für Stringverarbeitung
- Datentyp `String`:

```
String hello = "Hallo";
```

- Zusammenfügen mit “+”:

```
String world = "Welt";
String greeting = hello + " " + world + "!";
System.out.println(greeting);
```

<u>Ausgabe</u>:

Hallo Welt!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.6 Strings (2)

- Zusammenfügen funktioniert auch mit `int`, `float`,...:

```
String test = "Result: " + (2 + 3);
```

Inhalt von test: "`Result: 5`"

- **Wichtig: Strings sind konstant.**

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.7 Arrays

- Deklaration mit ***type*** **name[] oder *type* [] name**
- Allokation mit `new`, keine Freigabe notwendig (garbage collector):

```
int numbers[] = new int[100];
```

- Elementzugriff wie in C: `name[index]`:

```
for (int n=0; n<100; n++) numbers[n] = n;
```

- automatische Bereichsüberprüfung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.7 Arrays (2)

- Deklaration und Initialisierung:

```
int [] firstPrimes = { 2, 3, 5, 7 };
```

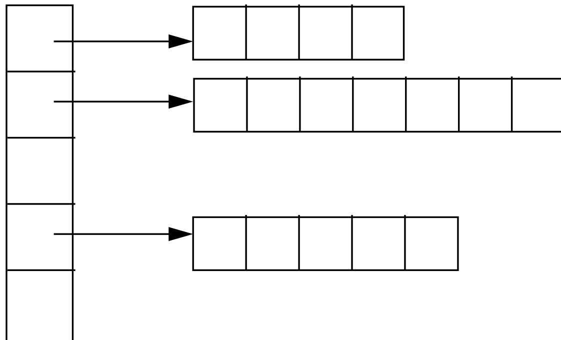
- Ermitteln der Länge eines Arrays mit `length`:

```
for(int i=0; i<firstPrimes.length; i++) {
    System.out.println(firstPrimes[i]);
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.7 Mehrdimensionale Arrays

- Array-Elemente können wiederum Arrays sein:



```
int [][] matrix = new int[5][4];
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C OO-Grundlagen mit Java

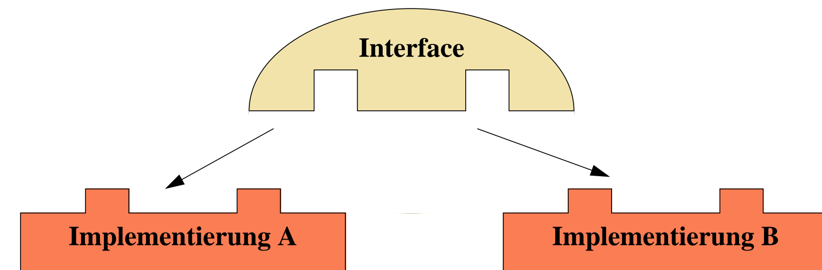
- Fundamentale Konzepte der objektorientierten Programmierung:

Abstraktion
und
Kapselung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.1 Abstraktion

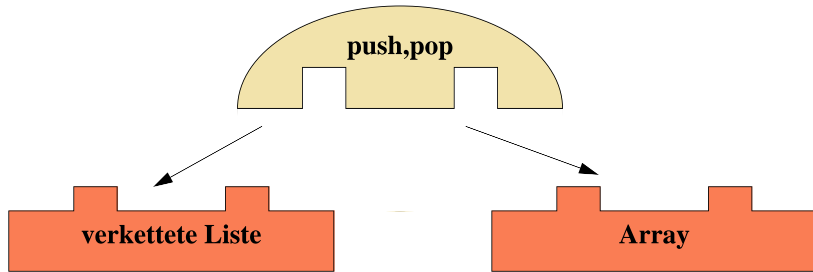
- Trennung von:
  - Schnittstelle (interface): Was kann getan werden?
  - Implementierung: Wie wird es gemacht?



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.1 Abstraktion

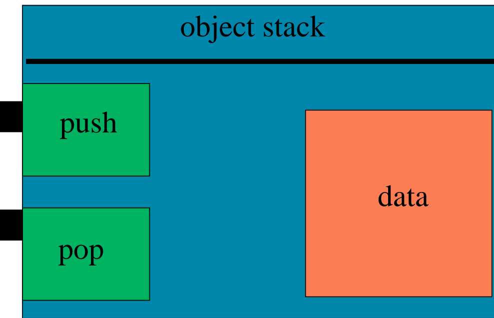
- Beispiel: stack
  - Interface: push, pop
  - Implementierung: verkettete Liste, Array



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.2 Kapselung

- Objekte: Gekapselte Datenstruktur, bestehend aus:
  - Daten (Instanzvariablen, Attribute)
  - Methoden (Operationen)

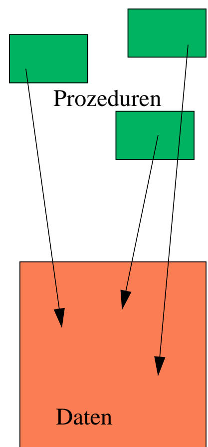


- Kapselung unterstützt die Bildung von Abstraktionen.

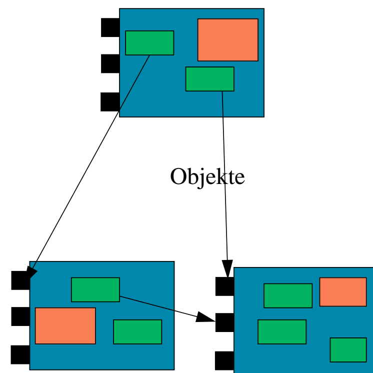
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.2 Kapselung

Prozedurale Programmierung



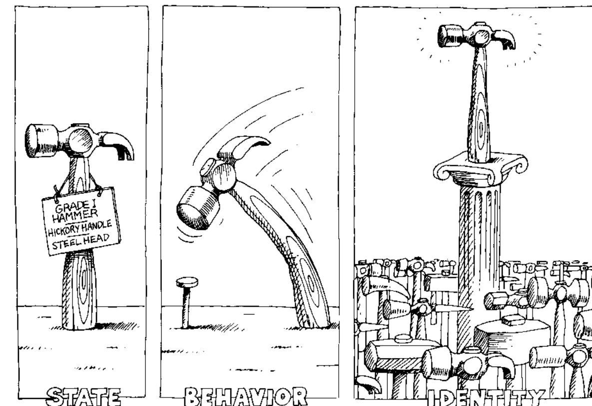
Objektorientierte Programmierung



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.3 Objekte

1 Eigenschaften von Objekten



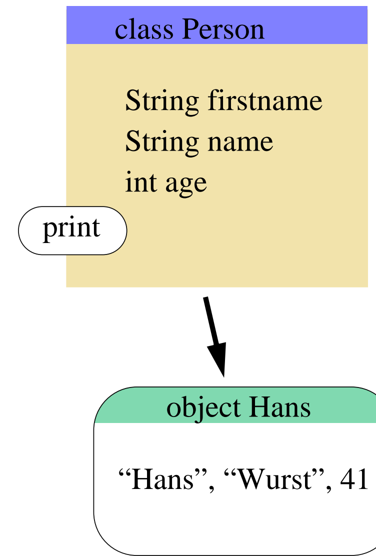
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Klassen

- Objekte sind *Instanzen* einer Klasse.
- Die Klasse bestimmt die interne Struktur und die Schnittstelle eines Objekts.

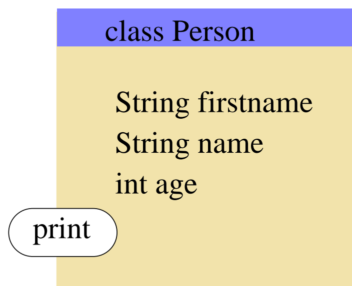
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Klassen - Beispiel



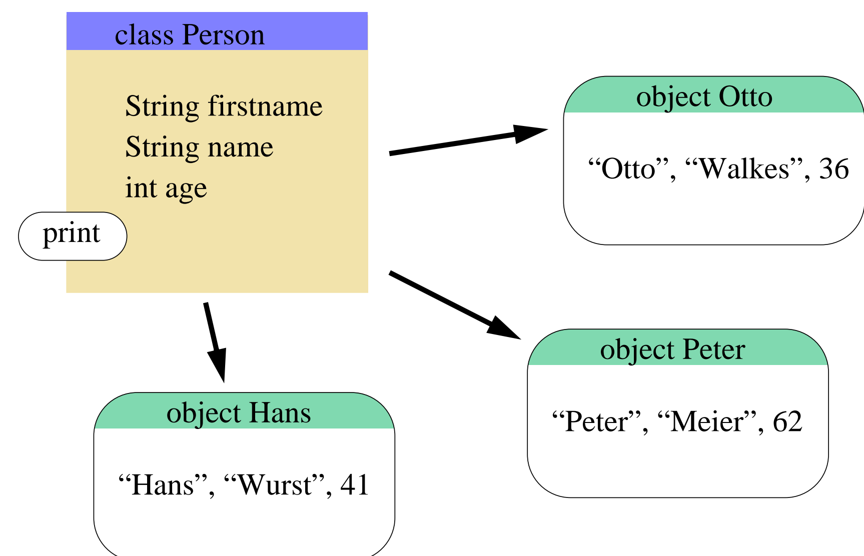
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Klassen - Beispiel



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Klassen - Beispiel

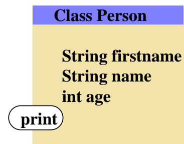


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Klassen - Beispiel

- Klassendefinition:

```
class Person {
    String firstname;
    String name;
    int age;
    void print() {
     System.out.println("Name: " + firstname + " " + name +
                        " Age: " + age);
    }
}
```

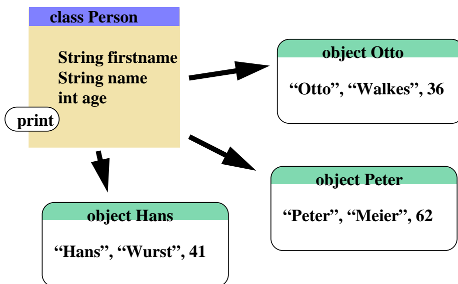


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Objekterzeugung

- Objekterzeugung (Instantiierung):

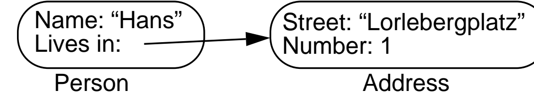
```
Person otto = new Person();
otto.firstname = "Otto";
otto.name = "Walkes";
otto.age = 36;
otto.print();
```



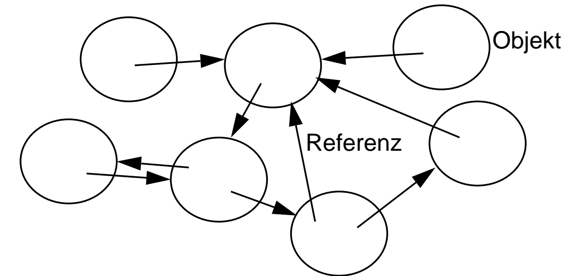
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Das Objekt-Netz

- Objekte können andere Objekte referenzieren.
  - Beispiel: eine Person kann eine Adresse referenzieren:



- Darstellung des Zustands eines objektorientierten Programms:

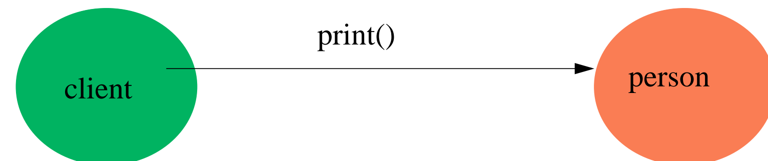


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Nachrichten / Methoden

- Objekte kommunizieren mit Hilfe von Nachrichten.
  - Jedes Objekt legt sein eigenes Verhalten selbst fest.
  - Die Objektsemantik ist nicht über das ganze Programm verteilt.
- Nachricht = Methodenaufruf an einem Objekt

```
person.print()
```



- objektorientiertes Programm: mehrere Objekte kommunizieren miteinander, um eine bestimmte Aufgabe zu erfüllen.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Methoden und Variablen

- Methoden können auf 3 verschiedene Arten von Variablen zugreifen:
  - Parameter (Argumente)
  - lokale Variablen
  - Instanzvariablen

```
class Point {
    int xPosition, yPosition;
    ...
    void move(int x, int y) {

        int i;
        ...
    }
}
```


© • Universität Erlangen-Nürnberg • Informatik 4, 2002
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Variablen

- Zugriff auf Instanzvariablen mit:
  - *instanceName.variableName*

```
class Person {
    String firstname;
    String name;
    int age;

    boolean sameName(Person otherPerson) {
        if (name == otherPerson.name) return true;
        return false;
    }
}
```


© • Universität Erlangen-Nürnberg • Informatik 4, 2002
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Variablen

- Parameter und lokale Variablen müssen unterschiedliche Namen haben.
- Parameter und lokale Variablen überdecken Instanzvariablen.

```
class Test {
    int a;
    void m(int a) {
        a = 5; // does not change instance variable a
    }
}
```


© • Universität Erlangen-Nürnberg • Informatik 4, 2002
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Der Parameter *this*

- Jede Methode hat einen *impliziten* Parameter **`this`**.
- **`this`**: Referenz auf die Instanz, an der die Methode aufgerufen wurde:

```
class Person {
    String name;
    ...
    void print() {
        System.out.println(this.name);
    }
}
```

- **`this`** kann bei Eindeutigkeit weggelassen werden.
- Beispiel für Mehrdeutigkeit:

```
class Person {
    ...
    boolean compare(String name) { return this.name == name; }
}
```


© • Universität Erlangen-Nürnberg • Informatik 4, 2002
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Überladen

- Methoden mit unterschiedlichen Parametern können den gleichen Namen haben.
- Beispiel:

```
class Date {
    ...
    void print(PrintStream stream) { stream.println(...); }
    void print() { print(System.out); }
}
```

- Hinweis: Überladen funktioniert nur mit Parametern nicht mit dem Typ des Rückgabewerts:

```
class Income {
    ...
    int computeIncome() { ... }
    float computeIncome() { ... } // Error !!
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 Konstruktoren

- Konstruktoren dienen der Initialisierung des Objekts.
- Name des Konstruktors = Name der Klasse.
- Der Konstruktor wird automatisch nach der Objekterzeugung aufgerufen.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Objekt-Initialisierung

- Erzeugen eines Objekts bedeutet Reservierung von Speicher.
- Dieser Speicher muss initialisiert werden.
- Eine Möglichkeit:
  - ◆ Explizites Aufrufen einer Initialisierungsmethode.
  - ◆ Nachteil: fehleranfällig.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 Konstruktoren (2)

- Initialisierung einer Person mit Name und Alter:

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    ...
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 Konstruktoren (3)

- Mehrere Konstrukturen sind möglich:

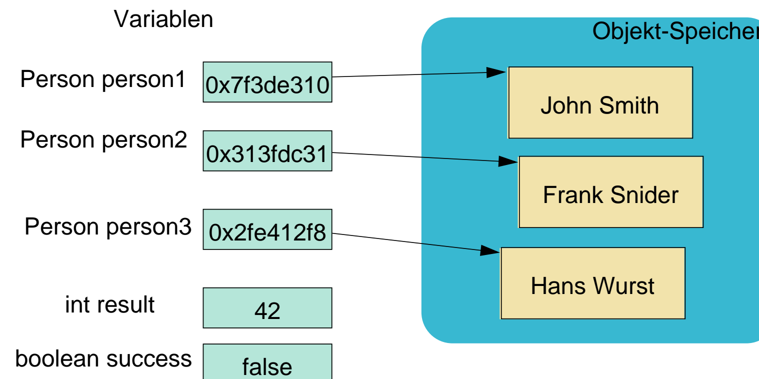
```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    Person(String name) {
        this.name = name;
        this.age = 18;
    }
    ...
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

11 Objekte und Referenzen

- Java-Variablen bezeichnen keine Objekte, sondern Referenzen.

```
Person p;       // Deklaration einer Referenz auf ein Objekt
                // der Klasse Person
p.print();      // Fehler: Methodenaufruf an einer null-
```



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 Konstruktoren (4)

- Aufruf eines anderen Konstruktors mit `this(...)`:

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    Person(String name) {
        this(name,18);
    }
    ...
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

12 Zuweisungen

- = weist einer Variable eine Referenz zu
- == vergleicht zwei Referenzen
- Einer Variable eines primitiven Datentypes kann keine Referenz zugewiesen werden.
- Einer Variable, welche eine Objektreferenz ist, kann niemals der Wert eines primitiven Datentypes zugewiesen werden.
- Beispiel:

```
Person p;       // Deklaration einer Referenz-Variablen
int i=42;    // Deklaration und Initialisierung einer
                // Variable eines primitiven Datentypes
p = i;       // Fehler: Zuweisung zwischen Referenz und
                // primitiven Datentyp
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

13 Aufrufsemantik von Methoden

- Objekt-Parameter werden als Referenz übergeben.
- Primitive Datentypen (int, float, etc.) werden als Wert übergeben.
- Beispiel:

```
void meth(int a, Person k) {
   a = 5;             // a: passed by value
   k.setAge(25);      // k: passed by reference
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

14 Gleichheit und Identität

- Unterschied zwischen *gleichen Objekten* und *identischen Objekten*:

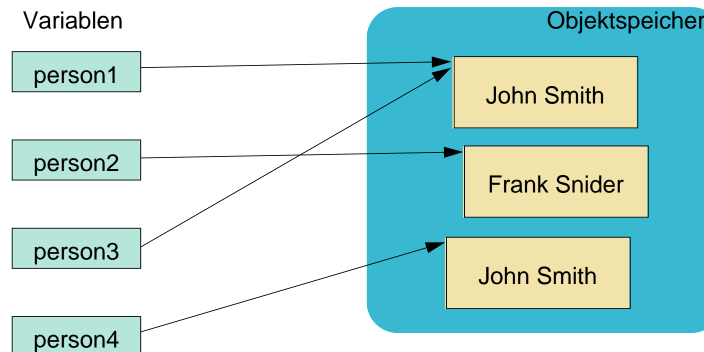
```
class Date {
  int day, month, year;
  Date(int day, int month, int year) {
    this.day = day; this.month = month; this.year = year;
  }
}
...
Date d = new Date(1,3,98);
Date d1 = new Date(1,3,98);
Date d2 = d;
```

◆ d und d1 sind gleich
◆ d und d2 sind identisch

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

15 Identität und Referenzen

- Identität: gleiche Referenz
- Gleichheit: Inhalt der referenzierten Objekte ist gleich



- Welche Personen sind identisch, welche gleich?

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

16 Testen von Gleichheit und Identität

- Identität kann mit dem Operator == getestet werden:

```
if (d == d1) { ... }
```

- Gleichheit kann mit der Methode **`equals`** getestet werden:

```
if (d.equals(d1)) { ... }
```

- Die Methode **`equals`** muss selbst implementiert werden:

```
class Date {
  ...
  public boolean equals(Object o) {
    if (! (o instanceof Date)) return false;
    Date d = (Date)o;
   return d.day == day && d.month == month && d.year == year;
  }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

16 Testen von Gleichheit und Identität (2)

- Wenn zwei Objekte gleich sind, dann müssen sie den gleichen Hash-Wert besitzen:

```
class Date {
   ...
  public int hashcode() {
    return day + month + year;
  }
}
```

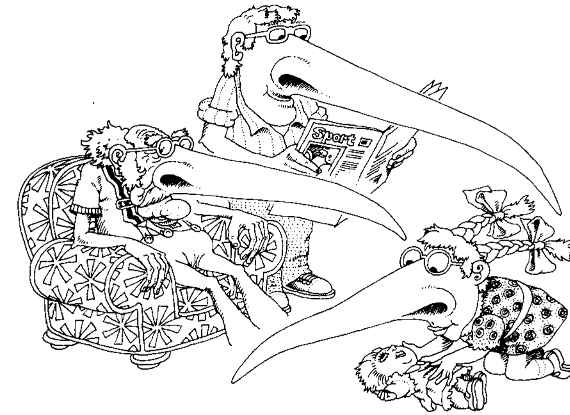

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

C.4 Vererbung

- Definition von Objekten durch Verweise auf andere Objekte.
  - ◆ Beispiel:
    - Definition eines Tieres: Ein Ding welches atmet und isst.
    - Definition einer Kuh: Ein Tier, dass “muuuu” macht und Milch gibt.
    - Kuh *erbt* die Eigenschaften “atmen” und “essen” von Tier.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Vererbung in der echten Welt



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Vererbung in Java

- Vererbung: Definition eines neuen Objekts auf der Basis einer spezialisierten Definition eines existierenden Objekts.
- Neue Klassen können von existierenden Klassen *abgeleitet* werden.
- Beispiel: Ein Kunde ist eine Person.

```
class Customer extends Person {
    int number;
    ...
}
```

- `Customer` ist eine *Unterklasse (subclass)* von `Person`
- `Person` ist die *Oberklasse (superclass)* von `Customer`.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Unterklassen

- Unterklassen erben
  - Zustand (Instanzvariablen) und
  - Verhalten (Methoden) von der Oberklasse.
- Unterklassen können:
  - neue Instanzvariablen einführen
  - neue Methoden einführen
  - geerbte Instanzvariablen verdecken (vorsicht!)
  - geerbte Methoden überschreiben

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

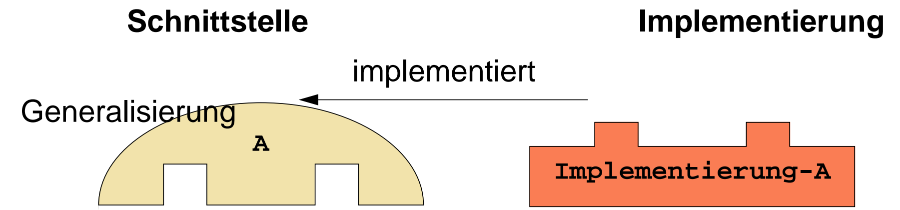
4 Das Ersetzungsprinzip

- Wenn ein Objekt der Oberklasse erwartet wird, kann immer auch ein Objekt einer Unterklasse verwendet werden.
  - wichtigster Typ des Polymorphismus
- Vererbung ist Spezialisierung ("ist ein" Relation)
- alles was für die Generalisierung gilt muss auch auf die Spezialisierung zutreffen.
  - Die Spezialisierung muss alle Anforderungen der Generalisierung erfüllen.

→ Abstraktion

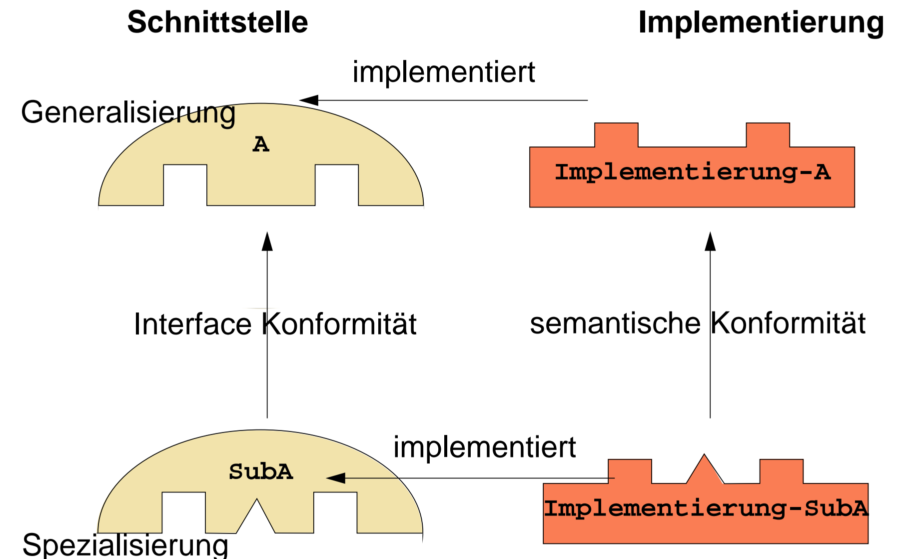
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Vererbung ist Spezialisierung



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

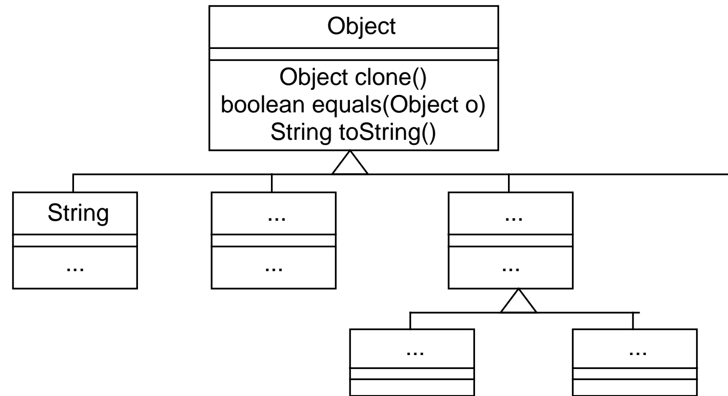
C.4 Vererbung ist Spezialisierung



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Vererbung in Java

- Klassen mit einfacher Vererbung
- Baum Hierarchie mit der Klasse `Object` als Basisklasse aller anderen Klassen



- primitive Typen (**int**,**float**,...) sind außerhalb des Klassenbaumes

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Die Klasse Object

- Die Klasse `Object`: Basisklasse aller anderen Klassen

```
class Person extends Object {... }
```

◆ **extends Object** kann wegfallen

```
class Person {... }
```

- Stellt Basisfunktionalität zur Verfügung, zum Beispiel:

◆ **boolean equals(Object o)** // Test auf Gleichheit
  - Standardimplementierung vergleicht die Referenzen
  - Jede Klasse sollte eine eigene Implementierung bereitstellen

◆ **String toString()** // Stringdarstellung eines Objekts
  - Standardimplementierung: Klassenname und Objekt ID
  - Jede Klasse sollte eine eigene Implementierung bereitstellen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Überschreiben

- Unterklassen können für geerbte Methoden eine neue Implementierung bereitstellen.
- Die neue Implementierung *überschreibt* die geerbte Implementierung:

```
class Person {
    String name;
    ...
    void print() {
        System.out.println("Person: " + name);
    }
}
class Customer extends Person {
    int number;
    ...
    void print() {
        System.out.println("Person: " + name);
        System.out.println("Customer number: " + number);
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Überschreiben (2)

- Die Implementierung der Oberklasse kann mit **super.*method*()** aufgerufen werden.
- Beispiel:

```
class Customer extends Person {
    int number;
    ...
    void print() {
        super.print();
        System.out.println("Customer number: " + number);
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Überladen vs. Überschreiben

- Um eine Methode zu überschreiben müssen die Typen der Parameter und des Rückwerts exakt übereinstimmen, ansonsten wird die Methode überladen.
- Um dem Ersetzungsprinzip gerecht zu werden würde es ausreichen, wenn:
  - die Typen der Parameter von einer Oberklasse der ursprünglichen Parameter sind
  - der Typ des Rückgabewertes von einer abgeleiteten Klasse des ursprünglichen Rückgabetyps ist.
- Java unterstützt das jedoch nicht!!!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

9 Überladen vs. Überschreiben (2)

- häufiger Fehler:

```
class Object {
    boolean equals(Object o) { ... }
    ...
}

class Customer {
    int number;
    boolean equals(Customer c) { return number == c.number;}
    ...
}
```

equals von Object wird nicht überschrieben

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

10 Dynamisches Binden

- Die Methoden werden erst bei einem Aufruf gebunden.
- *dynamischer Typ*: Typ/Klasse des referenzieren Objekts (die Klasse, die bei **new** verwendet wurde)
- *statischer Typ*: Typ der Referenz
- Durch den statischen Typ wird festgelegt, welche Methoden aufgerufen werden können.
- Der dynamische Typ legt fest, welche Methode verwendet wird:

```
Customer c = new Customer("Max", 1234);
Person p = c;      // dynamischer Typ von p ist Customer,
                   // statischer Typ ist Person

c.print();
p.print(); // obwohl die Referenz p den Typ Person hat, wird
           // die print() Methode von Customer verwendet
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

11 Sichtbarkeit und Vererbung

- Die Sichtbarkeit von Methoden darf in Unterklassen nicht eingeschränkt werden (Ersetzbarkeit!):.

```
class Person {
    public String getName() { ... }
}
class Customer extends Person {
    private String getName() { ... }
}
```

Error

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

12 Konstruktoren und Vererbung

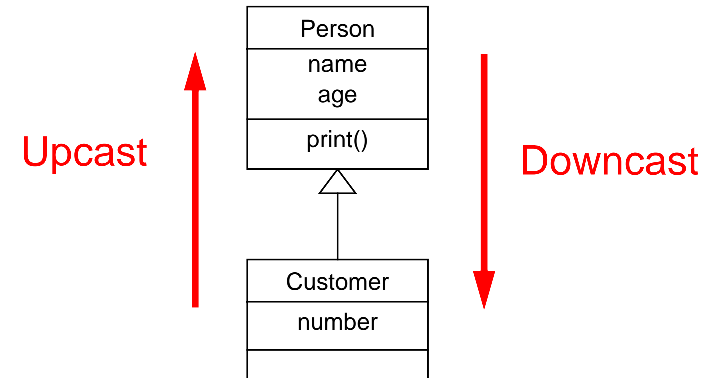
- Konstruktoren werden **nicht** vererbt
- Aufrufen eines Konstruktors der Oberklasse mittels `super(...)`
- `super(...)` muss die erste Anweisung in einem Konstruktor sein
- Falls die erste Anweisung nicht `super(...)` ist, so fügt der Compiler automatisch eine `super()` Anweisung ein.
- Standardkonstruktor:
  - wird vom Compiler erzeugt, falls *kein* Konstruktor definiert wird
  - enthält eine `super()` Anweisung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

13 Typenkonvertierung: Upcast

- Typenkonvertierung von einer Unterklassen zur Oberklasse (*upcast*) erfolgt automatisch:

```
Person p = new Customer(...);
```



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

12 Konstruktoren und Vererbung (2)

- Beispiel:

```
class Customer extends Person {
    int number;
    Customer(String name, int number) {
        super(name);
        this.number = number;
    }
    ...
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

13 Typenkonvertierung: Downcast

- Typenkonvertierung von der Oberklasse zu einer Unterklasse (*downcast*) explizit mittels Cast-Opertor:

```
Customer c = new Customer(...)
Person p = c;                // implizite Typenkonvertierung
Customer c2 = (Customer) p;  // explizite Typenkonvertierung
```

- Wenn das Objekt und die Variable nicht typkonform sind, wird eine `ClassCastException` generiert:

```
Customer c = new Customer(...)
Person p = c;                // implizite Typenkonvertierung
Employee e = (Employee) p;
           // erzeugt eine ClassCastException zur Laufzeit
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

14 Typ Ermittlung

- **`instanceof`** Operator:

```
Customer c = new Customer(...)
Person person = c;     // upcast
if (person instanceof Employee) {
    Employee employee = (Employee) person;
    ...
} else if (person instanceof Customer) {
    Customer customer = (Customer) person;
    ...
}
```

- **`instanceof`** mit der Oberklasse des dynamischen Typs ist ebenfalls **`true`**:

```
person instanceof Person
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

15 Die Klasse Class

- Die Klasse **`Class`**: Die Klasse aller Klassen.
- Ein **`Class`** Objekt repräsentiert eine Klasse oder ein Interface.
- Mit Hilfe eines **`Class`** Objekts können neue Instanzen erzeugt werden:

```
Customer c = new Customer();
...
Class aClass = c.getClass();
System.out.println("Class of c is:"+aClass);

Object o = aClass.newInstance();  // ein neues Customer Objekt

Person p = (Person) o;
```

- Ein **`Class`** Objekt kann aus dem Namen einer Klasse generiert werden:

```
Class aClass = Class.forName("Employee");
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

15 Die Klasse Class (2)

- seit Java 1.1: besser als **`Class.forName(...)`**

```
Class aClass = Employee.class;
...

Object o = aClass.newInstance();
Person p = (Person) o;
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D Aufgabe 1

- Anwendungen
- Zeichnen
- Interfaces in Java
- Explizites Laden von Klassen
- JUnit
- OOVS Homepage:
  http://www4.informatik.uni-erlangen.de/Lehre/WS02/V_OOVS/Uebung/

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.1 Anwendungen

- Eine Anwendung benötigt eine `main` Methode:

```
class MyApplication {
    public static void main(String[] args) {
        ...
    }
}
```

- Anwendungen werden durch den Java Interpreter gestartet.
  - Parameter: Hauptklasse (Klasse mit `main` Methode)

```
> set path=( /local/java-1.4/bin $path)
> setenv CLASSPATH /proj/i4oovs/felser/aufgabe1
> java MyApplication
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Zeichnen

1 Frame

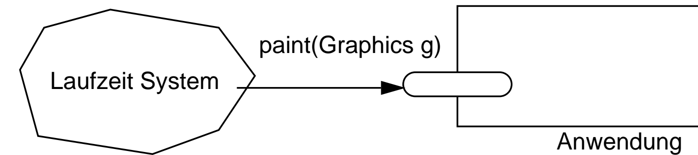
- Top-Level-Window zum anzeigen von grafischen Inhalten:

```
import java.awt.Frame;

public class MyApplication extends Frame {
    MyApplication(){
        super("MyApplication");
        setSize(400,300);
        setVisible(true);
    }
}
```

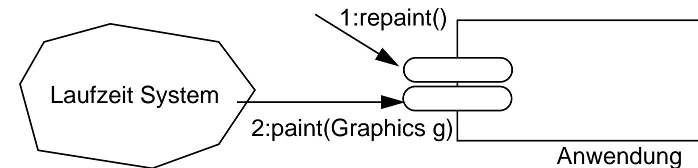

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Die paint Methode

- implizit: wenn ein Teil der Anwendung neu gezeichnet werden muss, so ruft das System die `paint()` Methode auf:



- explizit: `repaint()` veranlasst das System `paint()` aufzurufen:



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Das Graphics Objekt

- ...wird als Parameter an die `paint()` Methode übergeben:

```
import java.awt.Frame;
import java.awt.Graphics;

public class MyApplication extends Frame {
    ...
  public void paint(Graphics g) {
    ...
  }
}
```

- ... wird zum zeichnen verwendet:
  - drawRect(int x, int y, int w, int h)
  - drawOval(int x, int y, int w, int h)
  - ...

- Ursprung oben links!!

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.3 Schnittstellen in Java

- Schnittstellen werden in Java durch das Schlüsselwort **`interface`** definiert:

```
interface Stack{
    void push(Object o);
    Object pop();
}
```

- Zur Implementierung dient das Schlüsselwort **`implements`**:

```
class Stack_impl implements Stack{
    public void push(Object o){ .... }
    public Object pop(){ .... }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.3 Schnittstellen in Java (2)

- Verwendung von Interfaces:

```
public class Calculator {
    public void add(Stack store ) { ... }
}
....
Calculator calc = new Calculator();
Stack s1=new Stack_impl_array(...);
calc.add(s1);
Stack s2=new Stack_impl_list(...);
calc.add(s2);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.4 Explizites Laden von Klassen

- Klassenobjekte können dynamisch durch den Methodenaufruf **`Class.forName()`** geladen und instanziiert werden.
- Die Methode **`newInstance()`** liefert eine konkrete Instanz einer dynamisch geladenen Klasse.
- Beispiel:

```
try{
    Class c = Class.forName("TestClass");
    Object o = c.newInstance();
}catch(Exception e){
    System.err.println(e.toString());
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.5 Automatisches testen mit JUnit

- Unit-Test
  - testet die Methoden einer Klasse
  - ist jederzeit wiederholbar
  - verändert den Programmcode der Applikation nicht
  - gewährleistet definierte Funktion auch nach großen Modifikationen des Programmcodes
- Beispiel Money.java

```
public class Money {
    private double amount;

    public Money (double amount){ ....}
    public double getAmount(){ .... }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.5 Automatisches testen mit JUnit (2)

- Unit-Test für Money.java

```
public class MoneyTest extends TestCase {
    public MoneyTest (String name){
        super(name);
    }

    public void testAmount(){
        Money money = new Money(3.00);
        assertTrue(3.00 == money.getAmount());
    }

    public static void main(String[] args){
        junit.textui.TestRunner.run(MoneyTest.class);
    }
}
```

- Mehr Informationen unter http://www.junit.org oder nächste Woche in der Übung.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.6 Entwicklungsumgebung im CIP-Pool

- java - /local/java-1.4
- junit - /local/junit
- Entwickeln mit xemacs und jde (http://jdee.sunsite.dk/)
  - ◆ Anpassungen in `.emacs` oder `.xemacs/init.el`

```
(add-to-list 'load-path (expand-file-name
                         "/local/speedbar"))
(add-to-list 'load-path (expand-file-name
                         "/local/semantic"))
(add-to-list 'load-path (expand-file-name
                         "/local/eieio"))
(add-to-list 'load-path (expand-file-name
                         "/local/jde-2.2.8/lisp"))
(require 'jde)
```

  - ◆ `CLASSPATH` erweitern um `/local/jde-2.2.8/java/lib/jde.jar` und `/local/jde-2.2.8/java/lib/bsh.jar`

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.