Programmierkurs Java

UE 13 - Vererbung

Dr.-Ing. Dietrich Boles

Gliederung

- Motivation
- Beispiel Stack
- Definitionen
- Beispiel Person
- Implementierung
- Besonderheiten in Java
- Klassen-Attribute
- Konstruktoren
- Schlüsselwort super
- Überschreiben von Methoden
- final
- Vorteile
- Zusammenfassung

Motivation

Motivation:

- gleichartige Objekte → Erstellung einer Klasse
- gleichartige Klassen → ?

Beispiele:

- Abstrahierung:

  n verschiedene Sortierungsalgorithmen (BubbleSort, QuickSort, ...)
- Erweiterung:

  Klasse Stack → Klasse SwapStack (Vertauschen der obersten Elemente)
- Spezialisierung:

  Klasse Rechteck → Klasse Quadrat

Idee:

- Man nehme eine existierende (Ober-)Klasse OK, leite eine neue (Unter-)Klasse UK davon ab (d.h. UK **erbt**/besitzt alle Instanzvariablen und Methoden von OK) und modifiziere die geerbte Implementierung bzw. erweitere das Attribute- und Methodenangebot
- Java: Schlüsselwort **`extends`**

Beispiel (Stack -> SwapStack) (1)

```
class Stack {
    int[] store;   // zum Speichern von Daten
    int current;   // aktueller Index

    Stack() {
      this.store = new int[10];
      this.current = -1;
    }
    boolean isFull() {
      return this.current == (this.store.length-1);
    }
    boolean isEmpty() {
      return this.current == -1;
    }
    void push(int value) {
      this.store[++this.current ] = value;
    }
    int pop() {
      int value = this.store[this.current];
      this.current--;
      return value;
}   }
```

Beispiel (Stack -> SwapStack) (2)

```
class SwapStack extends Stack {

  // alle Attribute und Methoden von Stack werden geerbt
  int pop() {  // Modifikation
    return this.store[this.current--];
  }

  void swap() { // Erweiterung
    if (this.current >= 1) {
      int speicher = this.store[this.current-1];
      this.store[this.current-1]=this.store[this.current];
      this.store[this.current] = speicher;
  } }

  public static void main(String[] args) {
    SwapStack stack = new SwapStack();
    stack.push(3); // geerbte Methode
    stack.push(5);
    stack.swap();  // neue Methode
    IO.println(stack.pop()); // modifiz. Meth.
} }
```

Definitionen (1)

Erben:

- Erben einer Klasse sind Klassen, die deren Spezifikation und Implementierung übernehmen und sie erweitern oder modifizieren (ohne Code-Duplizierung!)

Vererbung:

- Strukturierungsprinzip bei der Klassendefinition
- Neue Klassen (Unterklassen) können durch Erweiterung bzw. Modifikation bereits existierender Klassen (Oberklassen) definiert werden. Unterklassen erben dabei die Instanzvariablen und Methoden der Oberklassen.
- Die Vererbung führt zur hierarchischen Anordnung von Klassen und kann sich über mehrere Stufen erstrecken (**isA-Beziehungen**)

Ableitung:

- Prozess der Klassenbildung mit Hilfe des Vererbungsprinzips

Protokoll einer Klasse:

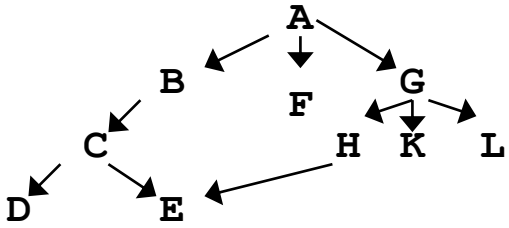
- Beschreibung der Schnittstelle einer Klasse, d.h. i. Allg. Menge der von außen zugreifbaren Methoden und Attribute der Klasse

Definitionen (2)

Oberklasse:
- Eine Klasse, von der eine andere Klasse abgeleitet worden ist

Unterklasse:
- Eine Klasse, die von einer anderen Klasse abgeleitet worden ist

Klassenhierarchie:
- Die mehrfach fortgesetzte Definition von Klassen mit Hilfe des Vererbungsprinzips führt zur Bildung von Klassenhierarchien



direkte Oberklasse/Unterklasse:
- Klasse, die in einer Klassenhierarchie direkt oberhalb/unterhalb einer bestimmten Klasse steht (d.h. von der direkt abgeleitet/geerbt wurde)

Mehrfachvererbung:
- Vererbungsbeziehung mit mehreren direkten Oberklassen

Beispiel (Person -> ...) (1)

```
class Person {

    String name;
    String vorname;

    Person(String n, String v) {
      this.name = n;
      this.vorname = v;
    }

    void println() {
      IO.println(this.vorname + " " + this.name);
    }

    void changeName(String n) {  // Hochzeit
      this.name = n;
    }
}
```

Beispiel (Person -> ...) (2)

```
class Student extends Person {

    // Attribute name und vorname werden geerbt
    int matrikelnummer;

    Student(String n, String v, int m) {
      this.name = n;
      this.vorname = v;
      this.matrikelnummer = m;
    }

    void println() { // Modifikation
      IO.println(this.vorname + " " + this.name + ": " +
                 matrikelnummer
                );
    }

    // Methode changeName wird geerbt

}
```

Beispiel (Person -> ...) (3)

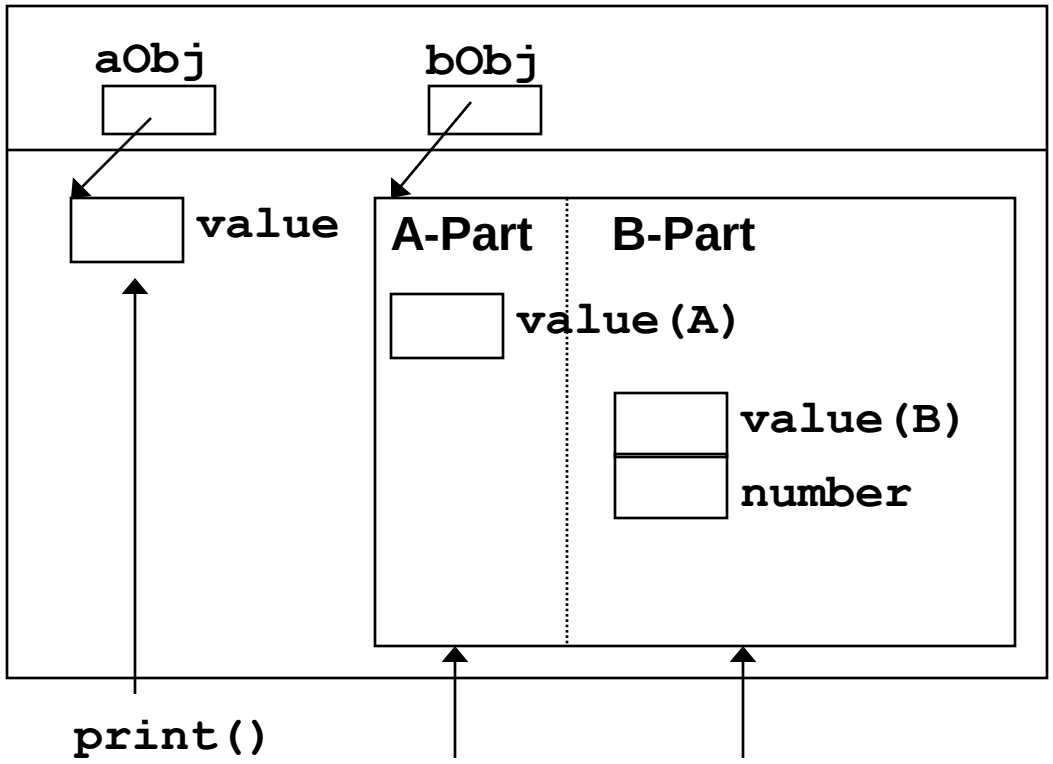
```
class Professor extends Person {

  // Attribute name und vorname werden geerbt
  String bueroNummer;

  Professor(String n, String v, String b) {
    this.name = n;
    this.vorname = v;
    this.bueroNummer = b;
  }
  void println() { // Modifikation
    IO.println("Prof. " + this.vorname + " " + this.name +
               ", Buero: " + this.bueroNummer
              );
  }

  // Methode changeName wird geerbt

  void changeBuero(String newB) { // Erweiterung
    this.bueroNummer = newB;
  }
}
```

Beispiel (Person -> ...) (4)

```
class PersonenTest {
  public static void main(String[] args) {
    Person dibo = new Person ("Boles", "Dietrich");
    Student heike = new Student("Meier", "Heike", 4711);
    Professor otto =
      new Professor("Schmidt", "Otto", "E 50");

    dibo.println();     // Dietrich Boles
    heike.println();    // Heike Meier: 4711
    otto.println();     // Prof. Otto Schmidt, Buero: E 50

    dibo.changeName("Mueller");
    heike.changeName("Schulz");
    otto.changeName("Schmitz");
    otto.changeBuero("E 49");

    dibo.println();     // Dietrich Mueller
    heike.println();    // Heike Schulz: 4711
    otto.println();     // Prof. Otto Schmitz, Buero: E 49
} }
```

Implementierung

```
class A {
  int value;
  void print();
}

class B extends A {
  int value;
  float number;
  void change();
}
```




```
class AB {
  public static void main(String[] args) {
    A aObj = new A();
    B bObj = new B();
} }
```

Besonderheiten in Java

- Keine Mehrfachvererbung (d.h. hinter **`extends`** kann immer nur ein Klassenname stehen!)
- es werden immer alle Instanz-Attribute und alle Methoden geerbt (keine Auswahl möglich)
- Klassen-Attribute werden nicht vererbt (dupliziert), sind jedoch in der Unterklasse zugreifbar (siehe Folie 14)
- Konstruktoren werden nicht vererbt (siehe Folien 15 ff)
- Jede Klasse wird implizit von einer Klasse **`Object`** (aus dem JDK) abgeleitet (siehe Folien 20f)

Vererbung und Klassen-Attribute

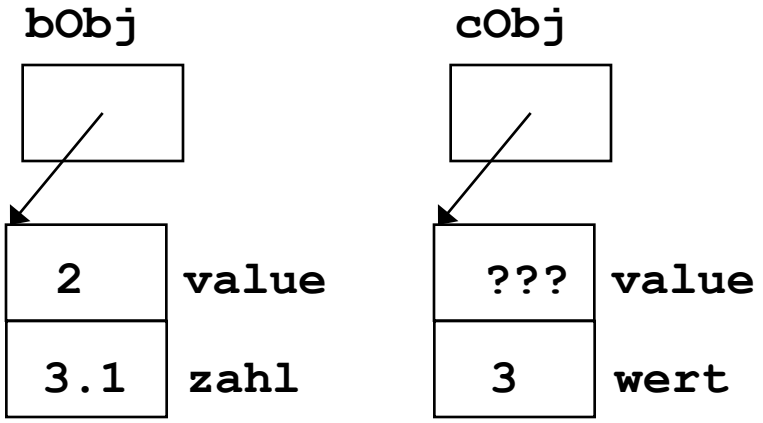
```
class A {
  static int wert = 0;
  static void incrWert() { A.wert++; }
  static int liefereWert() { return A.wert; }
}
class B extends A {
  static void decrWert() { B.wert--; }
}
class AB {
  public static void main(String[] args) {
    A.incrWert(); A.incrWert();
    B.incrWert();
    IO.println(A.liefereWert());  // 3
    IO.println(B.liefereWert());  // 3
    B.decrWert();
    IO.println(A.liefereWert());  // 2
    IO.println(B.liefereWert());  // 2
} }
```

Vererbung und Konstruktoren (1)

- Konstruktor: Initialisierung der Attribute eines neu erzeugten Objektes
- bez. der Vererbung ist zu beachten:
  - wird eine Klasse durch Vererbung definiert, dann muss die Unterklasse einen Konstruktor definieren, der einen Konstruktor der Oberklasse aufruft, und zwar mittels des **`super`**-Konstrukts als **erste** Anweisung der Konstruktorimplementierung
  - Ausnahme: die Oberklasse besitzt keinen oder einen parameterlosen Konstruktor; dann wird dieser automatisch aufgerufen

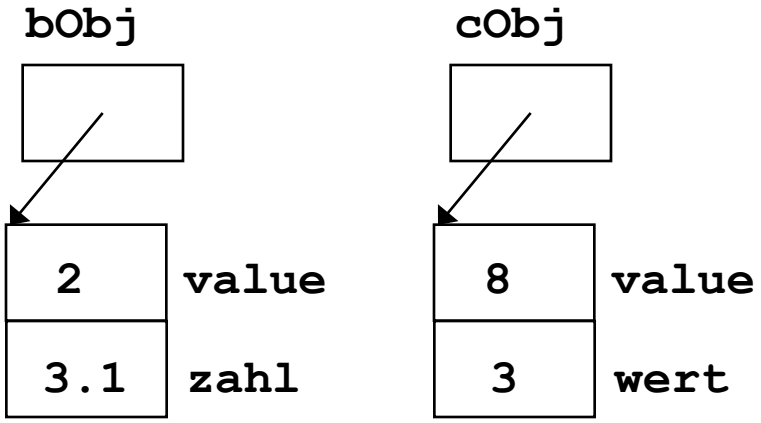
Vererbung und Konstruktoren (2)

```
class A {
  int value;
  A(int v) { this.value = v; }
}
class B extends A {
  double zahl;
  B(int v, double z) {
    super(v); // 1. Anweisung!
    this.zahl = z;
  }
}
class C extends A {
  int wert; // neue Variable!
  C() {
    this.wert = 3; // Fehler: super-Aufruf fehlt
} }

...
B bObj = new B(2, 3.1);
C cObj = new C();
```



Vererbung und Konstruktoren (3)

```
class A {
  int value;
  A() { this.value = 8; }
  A(int v) { this.value = v; }
}
class B extends A {
  double zahl;
  B(int v, double z) {
    super(v);
    this.zahl = z;
  }
}
class C extends A {
  int wert;
  C() {
    this.wert = 3; // ok: impliziter Aufruf von super()
} }

B bObj = new B(2, 3.1);
C cObj = new C();
```



Schlüsselwort super

- In allen Instanzmethoden einer Klasse verfügbar
- Referenz zum aktuellen Objekt als eine Instanz seiner Oberklasse
  - für Aufruf eines Konstruktors der Oberklasse
  - zum Zugriff auf geerbte (überschriebene) Methoden/Attribute
- Nutzung wie **`this`**

```
class That {
  int value = 2;
  String name() { return "That"; }
}

class More extends That {
  int value = 4;
  String name() { return "More"; }
  void printName() {
    IO.println(this.name());    // More
    IO.println(super.name());   // That
    IO.println(this.value);     // 4
    IO.println(super.value);    // 2
} }
```

Überschreiben von Methoden

Überladen:

- Methoden einer Klasse mit demselben Namen aber unterschiedlichen Parametern

Überschreiben:

- Ersetzung der Implementierung einer Methode einer Oberklasse durch eine neue Implementierung (Modifikation!). Die Methoden müssen identische Signaturen (Typ, Name, Parameter, …) haben!

```
class Stack {
  boolean isFull()
  boolean isEmpty()
  void push(int value)
  void push(float value)   // Überladen
  int pop()
}
class SwapStack extends Stack {
  void push(char value)    // Überladen
  void push(int value)     // Überschreiben
  int pop()                // Überschreiben
}
```

Klasse Object

- Jede Klasse in Java ist von der Klasse `Object` abgeleitet:

```
class Object {
  Object clone(); // Werte-Kopie
  boolean equals(Object obj); // Wertgleichheit?
  String toString(); // konvertiert in String
  ...
}

class A {}  <==>   class A extends Object {}
```

Schlüsselwort final

```
final class A { ... }
```

von A kann keine weitere Klasse abgeleitet werden

```
class B {
  final void f() { ... }
}
```

in einer Unterklasse von B kann die Methode f nicht überschrieben werden

```
class B {
  final static double PI = 3.1415;
  final int myValue = 12;
}
```

Definition von Konstanten

Vorteile der Vererbung

- Wiederverwendbarkeit
- Ableitung auch möglich ohne Quellcode der Oberklasse
- keine Quellcode-Duplizierung notwendig
- Fehlerkorrekturen bzw. Änderungen an einer Oberklasse wirken sich automatisch auch auf alle Unterklassen aus
- Grundlage der Polymorphie

Zusammenfassung

- Vererbung: Neue Klassen (Unterklassen) können durch Erweiterung bzw. Modifikation bereits existierender Klassen (Oberklassen) definiert werden; Unterklassen erben dabei die Instanzvariablen und Methoden der Oberklassen
- Ableitung: Prozess der Klassenbildung mit Hilfe des Vererbungsprinzips
- Überschreiben von Methoden: Ersetzung der Implementierung einer Methode einer Oberklasse durch eine neue Implementierung; die Methoden müssen identische Signaturen haben!