Programmierkurs Java

UE 19 – Abstrakte Klassen

Dr.-Ing. Dietrich Boles

Gliederung

- Motivation
- Definition
- Beispiel Graphik
- Beispiel Nullstellen
- Beispiel java.util.Calendar
- Zusammenfassung

Motivation

Prinzip der Vererbung:

- Aus existierenden Klassen können spezialisierte Klassen abgeleitet werden

Prinzip der abstrakten Klassen:

- Aus mehreren ähnlichen Klassen kann eine gemeinsame Oberklasse abstrahiert werden
- Sinn und Zweck: Teilvererbung und Ausnutzung der Polymorphie!

Schema:

```
class Rechteck   { void draw() {...} ...}
class Kreis      { void draw() {...} ...}
class Linie      { void draw() {...} ...}

"abstrakte" class Graphik {
  "abstrakte Methode" void draw();
}
```

Definition (1)

Definition: Abstrakte Klasse

- Eine abstrakte Klasse ist eine bewusst **unvollständige** Oberklasse, in der von einzelnen Methodenimplementierungen abstrahiert wird

  (**"abstrakte" Methoden**!)

```
<abstrakte Klasse> ::= „abstract“ „class“ <Kbezeichner> „{“
                       { <attribut> |
                         <methode> |
                         <konstruktor>
                         <abstrakte Methode>
                       } „}“
<abstrakte Methode> ::= „abstract“ <typ>
                        <Mbezeichner> „(“ <paramListe> „)“ „;“
```

Definition (2)

Beispiel:

```
abstract class Graphik {

    ...

    abstract void draw();

}
```

- Fehlende Methodenrümpfe werden erst in abgeleiteten Unterklassen implementiert.
- Die Instantiierung abstrakter Klassen ist nicht möglich, d.h. es lassen sich keine Objekte vom Typ `<Kbezeichner>` erzeugen
- Es lassen sich wohl aber Objektvariablen vom Typ `<Kbezeichner>` definieren; damit können Polymorphie und dynamisches Binden ausgenutzt werden!

Beispiel Graphik (1)

```
abstract class Graphik {
  String name;
  Graphik(String str) { this.name = str; }
  String getName() { return this.name; }
  abstract void draw();
}

class Rechteck extends Graphik {
  float width, height;
  Rechteck(String str, float w, float h) {
    super(str); this.width = w; this.height = h;
  }
  // geerbt: getName
  // weitere Methoden ...
  // implementiert:
  void draw() {
    IO.println("Rechteck:" + this.name);
} }
```

Beispiel Graphik (2)

```
class Kreis extends Graphik {
  float radius;
  Kreis(String str, float r) {
    super(str); this.radius = r;
  }
  // geerbt: getName
  // weitere Methoden ...
  // implementiert:
  void draw() {
    IO.println("Kreis:" + this.name);
} }
class Linie extends Graphik {
  ...
  // implementiert:
  void draw() {
    IO.println("Linie:" + this.name);
} }
```

Beispiel Graphik (3)

```
class Set {
  Graphik[] elems;
  int next;

  Set(int size) {
    this.elems = new Graphik[size];
    this.next = 0;
  }
  void add(Graphik obj) {
    if (this.next < this.elems.length)
      this.elems[this.next++] = obj;
  }
  Graphik[] get() {
    return this.elems;
} }
```

Beispiel Graphik (4)

```
class GraphikProbe {
  public static void main(String[] args) {
    Rechteck r1 = new Rechteck("Rechteck 1", 10, 20);
    Kreis k1 = new Kreis("Kreis 1", 50);
    Linie l1 = new Linie("Linie 1", 40);
    Rechteck r2 = new Rechteck("Rechteck 2", 15, 15);
    Graphik g1 = new Graphik("Graphik 1"); // Fehler!
    Set menge = new Set(6);
    menge.add(r1);     // Ausnutzung der Polymorphie!
    menge.add(r2);
    menge.add(k1);
    menge.add(l1);
    Graphik[] elements = menge.get();
    for (int i=0; i<elements.length; i++)
      if (elements[i] != null)
        elements[i].draw(); // Dynamisches Binden!
} } }
```

Beispiel Nullstellen

```
abstract class Funktion { abstract int f(int x); }

static boolean nullstelle(Funktion funk, int von, int bis) {
    for (int x = von; x <= bis; x++)
        if (funk.f(x) == 0) return true;
    return false;
}

class IdFunktion extends Funktion {
  int f(int x) { return x;}
}
class QuadratFunktion extends Funktion {
  int f(int x) { return x * x; }
}

System.out.println(nullstelle(new IdFunktion(), 2, 4));
System.out.println(nullstelle(new QuadratFunktion(), -3, 4));
```

Beispiel java.util.Calendar (1)

```
package java.util;

public abstract class Calendar {
    public static final int JANUARY;
    public static final int YEAR;
    public static final int MONTH;
    ...
    public void set(int field, int value)
    public int get(int field)
    ...
}

public class GregorianCalendar extends Calendar {
    public GregorianCalendar()   // aktuelle Zeit
    ...
}
```

Beispiel java.util.Calendar (2)

```
import java.util.Calendar;
import java.util.GregorianCalendar;


class Datum {
   public static void main(String[] args) {
        Calendar cal = new GregorianCalendar();
        IO.println("Jahr: " + cal.get(Calendar.YEAR));
        IO.println("Monat: " + cal.get(Calendar.MONTH));
        IO.println("Tag: " + cal.get(Calendar.DAY_OF_MONTH));
        cal.set(Calendar.YEAR, 2000);
        cal.set(Calendar.MONTH, Calendar.JANUARY);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        if ((new GregorianCalendar()).after(cal))
              IO.println("im 21. Jahrhundert");
   }
}
```

Zusammenfassung

- Abstrakte Klasse: eine bewusst unvollständige (Ober)Klasse, in der von einzelnen Methodenimplementierungen abstrahiert wird
- Fehlende Methodenrümpfe werden erst in abgeleiteten Unterklassen implementiert
- Die Instantiierung abstrakter Klassen ist nicht möglich; es lassen sich wohl aber Objektvariablen definieren, womit Polymorphie/dynamisches Binden ausgenutzt werden kann