Programmierkurs Java

UE 20 – Interfaces

Dr.-Ing. Dietrich Boles

Gliederung

- Definition
- Beispiel Graphik
- Polymorphie und dynamisches Binden
- Beispiel Fahrzeuge
- Erweiterte Interfaces
- Vergleich mit abstrakten Klassen
- Beispiel Nullstellen
- Beispiel Vergleichbar
- Beispiel GUIs
- Beispiel Gewicht
- Zusammenfassung

Definition (1)

Definition: **Interface**

- Ein Interface ist eine Art Klasse, die ausschließlich Konstanten und abstrakte Instanzmethoden deklariert (ab Java 8 mehr!)
- Java: Schlüsselwort **`interface`**

```
<Interface> ::= ["public"] "interface" <Ibezeichner> "{"
                    { <Konstante> |
                      <abstrakte Instanzmethode>
                    }
                "}"
<abstrakte Instanzmethode> ::=
            ["public"] ["abstract"] <typ>
                 <Mbezeichner> "(" <paramListe> ")" ";"
```

Definition (1) / Beispiel

```
interface Graphik {
  public abstract void draw();
}

interface Vergleichbar {
   public abstract boolean gleich(Vergleichbar obj);
   public abstract boolean kleiner(Vergleichbar obj);
   public abstract boolean groesser(Vergleichbar obj);
}
```

Definition (2)

- Klassen können ein oder mehrere Interfaces implementieren
- Java: Schlüsselwort **`implements`**

```
<class def> ::= "class" <Kbezeichner>
                [ "extends" <OKbezeichner> ]
                [ "implements" <Ibezeichner>
                  { "," <Ibezeichner> } ] "{" ... "}"
```

  - `Ibezeichner`: gültiges Interface
  - Die neue Klasse muss alle abstrakten Methoden der "implementierten Interfaces" definieren (identische Signaturen!) und implementieren!
  - implementierte Methoden von Interfaces sind immer **`public`**

Definition (2) / Beispiel

```
interface Graphik {
  public abstract void draw();
}
interface Vergleichbar {
  public abstract boolean gleich(Vergleichbar obj);
}

class Rechteck implements Graphik, Vergleichbar {
  public void draw() { ... }
  public boolean gleich(Vergleichbar obj) { ... }
}
class Quadrat extends Rechteck
  implements Graphik, Vergleichbar {
    public void draw() { IO.println(...); }
    // Methode "gleich" wird geerbt
}
```

Anmerkungen:

- Ein Interface gibt quasi eine "Vorschrift" vor.
- Ein Interface definiert einen neuen Typ `<Ibezeichner>`
- Es lassen sich keine Objekte vom Typ `<Ibezeichner>` erzeugen
- Es lassen sich wohl aber Objektvariablen vom Typ `<Ibezeichner>` definieren, **denen Objekte von das Interface implementierenden Klassen zugeordnet werden können**; damit kann Polymorphie/dynamisches Binden ausgenutzt werden!

```
Graphik obj1 = new Rechteck();  // Polymorphie
obj1.draw();  // dynamisches Binden

Vergleichbar obj2 = new Quadrat();
if (obj2.gleich(new Quadrat())) ...
```

Beispiel Fahrzeuge (1)

```
interface Landfahrzeug {
  public abstract void fahren();
}
interface Wasserfahrzeug {
  public abstract void schwimmen();
}
class Fahrzeug {
  Motor m;
  ...
}
class PKW
  extends Fahrzeug
  implements Landfahrzeug {
    public void fahren() { ... }
}
```

Beispiel Fahrzeuge (2)

```
class MotorBoot
  extends Fahrzeug
  implements Wasserfahrzeug {
    public void schwimmen() { ... }
}
class Amphibienfahrzeug
  extends Fahrzeug
  implements Landfahrzeug, Wasserfahrzeug {
    public void fahren() { ... }
    public void schwimmen() { ... }
}
...
<X> f = new Amphibienfahrzeug();
<X> kann sein (polymorph zu):
       Amphibienfahrzeug / Fahrzeug / Object /
       Landfahrzeug / Wasserfahrzeug
```

Erweiterte Interfaces

```
interface I {
  public abstract void f();
  public abstract void g();
}

interface J extends I {
  public abstract void h();
}

class A implements J {
  public void f() { /* Implementierung */ }
  public void g() { /* Implementierung */ }
  public void h() { /* Implementierung */ }
}

public class ExtInterfaceBeispiel {
  public static void main(String[] args) {
    I obj1 = new A();
    J obj2 = new A();
} }
```

Abstrakte Klassen - Interfaces / Vergleich

Abstrakte Klassen:

- abgeleiteten Klassen soll bereits ein bestimmtes Grundverhalten zur Verfügung gestellt werden ($\rightarrow$ Vererbung)
- (Einfach-)Polymorphie

Interfaces:

- ausschließliche Definition des Protokolls
- (Mehrfach-)Polymorphie

Regel:

- Wenn nichts vererbt werden soll, immer Interfaces verwenden!

Beispiel Nullstellen

```
interface Funktion { public 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 implements Funktion {
  public int f(int x) { return x;}
}
class QuadratFunktion implements Funktion {
  public int f(int x) { return x * x; }
}

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

Beispiel Vergleichbar (1)

```
interface Vergleichbar {
      public abstract boolean gleich(Vergleichbar obj);
      public abstract boolean kleiner(Vergleichbar obj);
      public abstract boolean groesser(Vergleichbar obj);
}

// Sortiert Mengen mit beliebigen Vergleichbar-Objekten!
class Sortieren {
  static void bubbleSort(Vergleichbar[] menge) {
    boolean veraendert = true;
    while (veraendert) {
      veraendert = false;
      for (int i=0; i<menge.length-1; i++) {
        if (menge[i].groesser(menge[i+1])) {
          Vergleichbar help = menge[i];
          menge[i] = menge[i+1];
          menge[i+1] = help;
          veraendert = true;
        }
} } } }
```

Beispiel Vergleichbar (2)

```
class Temperatur implements Vergleichbar {
   float grad;

   public Temperatur(float grad) {
     this.grad = grad;
   }
   public void aenderung(float grad) { this.grad += grad; }
   public String toString() { return this.grad + " Grad"; }

   public boolean gleich(Vergleichbar temp) {
     return this.grad == ((Temperatur)temp).grad;
   }

   public boolean kleiner(Vergleichbar temp) {
     return this.grad < ((Temperatur)temp).grad;
   }

   public boolean groesser(Vergleichbar temp) {
     return this.grad > ((Temperatur)temp).grad;
}  }
```

Beispiel Vergleichbar (3)

```
class Datum implements Vergleichbar {
   int jahr, monat, tag;

   public Datum(int jahr, int monat, int tag) {
     this.jahr = jahr;
     this.monat = monat;
     this.tag = tag;
   }

   public String toString() {
     return this.jahr + "/" + this.monat + "/" + this.tag;
   }

   // ...

   public boolean gleich(Vergleichbar dat) {
     return this.jahr == ((Datum)dat).jahr &&
            this.monat == ((Datum)dat).monat &&
            this.tag == ((Datum)dat).tag;
   }
```

Beispiel Vergleichbar (4)

```
    public boolean kleiner(Vergleichbar dat) {
      if (this.jahr < ((Datum)dat).jahr) return true;
      if (this.jahr > ((Datum)dat).jahr) return false;
      if (this.monat < ((Datum)dat).monat) return true;
      if (this.monat > ((Datum)dat).monat) return false;
      if (this.tag < ((Datum)dat).tag) return true;
      if (this.tag > ((Datum)dat).tag) return false;
      return false; // gleich
    }

    public boolean groesser(Vergleichbar dat) {
      if (this.jahr > ((Datum)dat).jahr) return true;
      if (this.jahr < ((Datum)dat).jahr) return false;
      if (this.monat > ((Datum)dat).monat) return true;
      if (this.monat < ((Datum)dat).monat) return false;
      if (this.tag > ((Datum)dat).tag) return true;
      if (this.tag < ((Datum)dat).tag) return false;
      return false; // gleich
    }
}
```

Beispiel Vergleichbar (5)

```
class VergleichbarTest {
   public static void main(String[] args) {
     Temperatur[] temperaturen = new Temperatur[5];
     for (int i=0; i<5; i++)
       temperaturen[i] =
           new Temperatur(IO.readFloat("Temperatur: "));

     Sortieren.bubbleSort(temperaturen);
     for (int i=0; i<5; i++)
       IO.println(temperaturen[i].toString());

     Datum[] dates = new Datum[3];
     for (int i=0; i<3; i++)
       dates[i] = new Datum(IO.readInt("Jahr: "),
                            IO.readInt("Monat: "),
                            IO.readInt("Tag: "));
     Sortieren.bubbleSort(dates);
     for (int i=0; i<3; i++)
       IO.println(dates[i].toString());
}  }
```

Beispiel GUIs (1)

```
import java.awt.*;
import java.awt.event.*;

/* public interface ActionListener {
     public void actionPerformed(ActionEvent e);
   }
   public class Button extends Component {
     public Button(String label) { ... }
     public void setLabel(String label) { ... }
     public String getLabel() { ... }
     public void addActionListener(ActionListener l)
     ...
   }
   public class Frame {
     public Frame(String title) { ... }
     public void add(Component comp) { ... }
     public void setBounds(int x, int y, int w, int h)
     public void setVisible(boolean vis) { ... }
   } */
```

Beispiel GUIs (2)

```
class ClickAction implements ActionListener {

  Button button;

  public ClickAction(Button b) {
    this.button = b;
  }

  public void actionPerformed(ActionEvent e) {
    if (this.button.getLabel().equals("Start"))
      this.button.setLabel("Stop");
    else
      this.button.setLabel("Start");
  }
}
```

Beispiel GUIs (3)

```
class GUIAnwendung {
  public static void main(String[] args) {
    Frame f = new Frame("Test");
    Button button = new Button("Start");
    button.addActionListener(
      new ClickAction(button));
    f.add(button);
    f.pack();
    f.setVisible(true);

// EventDispatcher-Thread ist aktiv:
// - registriert Events (Mouse, Tastatur, ...)
// - ordnet Events Component-Objekten zu
// - ruft registrierte Event-Listener-Methoden
//     auf (→ dynamisch gebunden)
  }
}
```

Zusammenfassung

- Interface: Ein Interface ist eine Art Klasse, die ausschließlich Konstanten und abstrakte Instanzmethoden deklariert
- Klassen können ein oder mehrere Interfaces implementieren; diese müssen alle (abstrakten) Methoden der "implementierten Interfaces" definieren (identische Signaturen!) und implementieren!
- Die Instantiierung von Interfaces ist nicht möglich; es lassen sich wohl aber Objektvariablen definieren, denen Objekte von das Interface implementierenden Klassen zugeordnet werden können; damit kann Polymorphie/dynamisches Binden ausgenutzt werden