6. Globalübung (zu Übungsblatt 8)

Inhalt:

- Klassenhierarchien
  - Verdecken von Attributen
  - Überschreiben von Methoden
- Codeanalyse
  - Analyse von JAVA-Programmen

Semestralklausur

- Klausurtermin: Mittwoch 11.01.2006
  8:15 – 9:45 im Audimax (nicht in Aula1)

- Anmeldung: Mittwoch (14.12.05) und
  Donnerstag (15.12.05)
  in den jeweiligen Tutorien

- Vorbereitung: Rechnen alter Klausuren
  (s. Webseiten der Vorlesung)

Anmerkung:

- Bei Vergleichen von Referenztypen möglichst immer die Methode

`equals`

und nicht

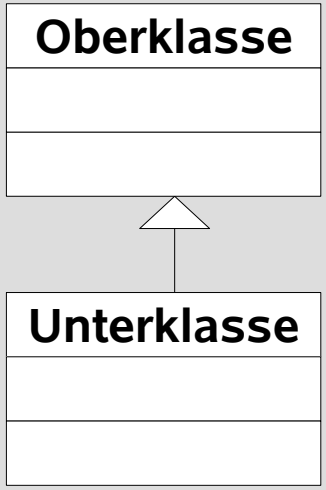
`== Operator`

verwenden!

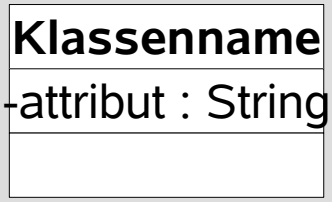
Themen der Vorlesung

- was haben wir bis jetzt kennengelernt:
  - Vererbung mit `extends`
  - implizite und explizite Datentypanpassung
  - Objekte in Klassenhierarchien
  - Konstruktoren in Klassenhierarchien
  - Verdecken von Attributen
  - Überschreiben von Methoden
  - finale Methoden

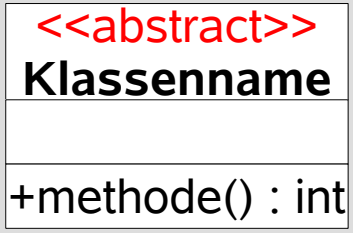
einfache UML Diagramme



Unterklasse erbt
von Oberklasse



Klasse
mit Attributen



abstrakte Klasse
mit Methode

dabei bedeutet:
- \- ein ***private*** Attribut oder eine ***private*** Methode
- \+ ein ***public*** Attribut oder eine ***public*** Methode
- \# ein ***protected*** Attribut oder eine ***protected*** Methode

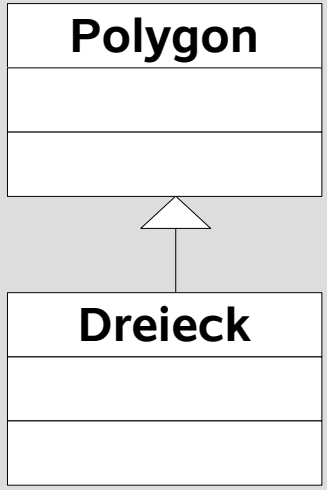
Vererbung mit `extends`

- Unterklasse `extends` `Oberklasse`

Beispiel:

```
public class Dreieck extends Polygon
```

in UML:



implizite und explizite Datentypanpassung (Casting)

- implizit:
  Variable der Oberklasse =
  Referenz auf ein Objekt der Unterklasse

  Beispiel: `Dreieck d, Polygon p`
  Erlaubt: `p = d;`
- explizit:
  Beispiel: Verboten: `d = p;`
  Erlaubt: `d = (Dreieck)p;`

Objekte in Klassenhierarchien

- Beispiel: `A` Oberklasse, `B` Unterklasse

```
B b = new B();      (1)
A a = b;            (2)
```

- Resultat:
  - (1) Objekt von Typ `B` wird erzeugt
    dieses enthält ein Objekt vom Typ `A`
  - (2) `a` ist Referenz auf den Teil des Objekts, der Objekt der Klasse `A` ist

Konstruktoren in Klassenhierarchien

- Aufruf eines Konstruktors der Oberklasse
  mit `super(...);`
  - wobei "..." die Parameter des Konstruktors der Oberklasse spezifiziert
  - `super(...)` muss immer als erstes in einem Konstruktor aufgerufen werden
  - wird `super(...)` nicht explizit aufgerufen, so wird implizit der Standardkonstruktor aufgerufen

Konstruktoren in Klassenhierarchien

- Aufruf eines Konstruktors der eigenen Klasse
  mit `this(...);`
  - auch hier spezifiziert "..." die Parameter des Konstruktors, der aufgerufen werden soll
  - `this(...)` muss immer als erstes in einem Konstruktor aufgerufen werden

Verdecken von Attributen

- gleichnamige Attribute werden überdeckt
  (die der Oberklasse von denen der Unterklasse)

daher sollte man nie gleiche Namen in Unter- und Oberklasse verwenden
(die gleichnamigen Attribute sind sonst in der Unterklasse nicht mehr sichtbar)

Beispiel: Verdeckung von Attributen

```
public class A {
   int x = 1;
}

public class B extends A {
   int x = 2;
}
```

Vorsicht:
überdeckt Variable `x` aus Klasse `A`

Überschreiben von Methoden

- Methoden von Oberklassen können in ihren Unterklassen
  - neu definiert
  - erweitert werden

oder auch

Beispiel: Überschreiben von Methoden

Redefinition einer Funktion

```
public class A {
   int f (){
      return 1;
   }
}

public class B extends A {
   int f (){
      return 2;
   }
}
```

Definiert die Funktion `f` neu

Beispiel: Überschreiben von Methoden

Erweiterung einer Funktion

```
public class A {
   int f (){
      return 1;
   }
}

public class B extends A {
   int f (){
      int i = super.f();
      return i+2;
   }
}
```

Finale Methoden

- werden eingeleitet durch das Schlüsselwort

```
final
```

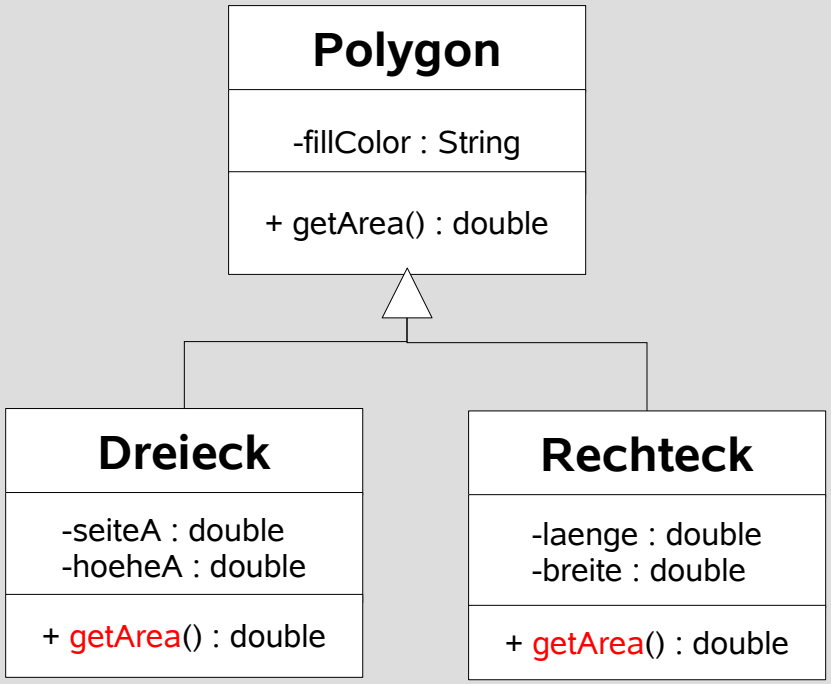
- Wirkung: Methode darf nicht mehr überschrieben werden

- Beispiel: `public final int f()`

Klassenhierarchie (siehe Übungsaufgabe)

- Spezifikation
  - Ein Polygon hat eine Füllfarbe und eine Methode `public double getArea()`
  - Ein Dreieck hat eine Seite und eine Höhe (auf dieser Seite). Außerdem kann man die Fläche einfach berechnen ➡ Redefinition von `getArea`
  - Ein Rechteck hat eine Breite und eine Länge. Auch hier kann man die Fläche einfach berechnen ➡ Redefinition von `getArea`

UML-Beispielhierarchie für geometrische Objekte

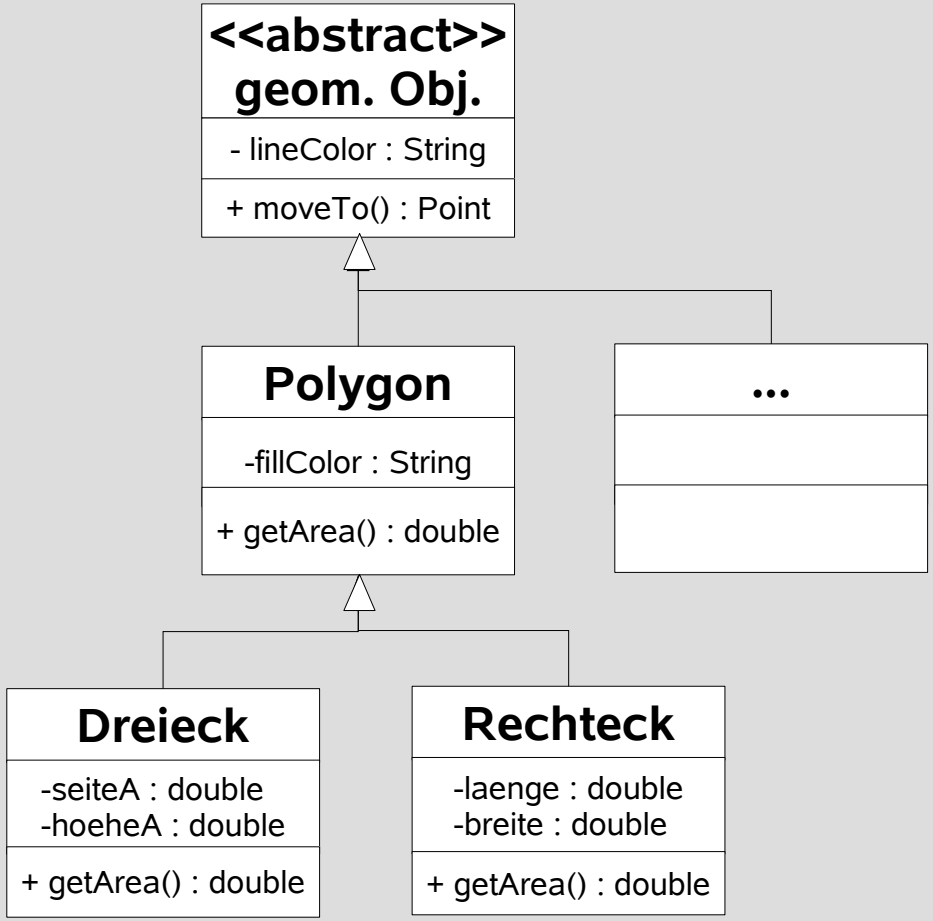


- Überschreiben von Methoden:
  - `getArea()` wird in den Klassen Dreieck und Rechteck neu definiert

Implementierung von `getArea()`

```
public class Dreieck extends Polygon {
  double grundseite;
  double hoehe;
  ...
  public double getArea() {
    return 1.0/2.0 * grundseite * hoehe;
  }
}

public class Rechteck extends Polygon {
  double laenge;
  double breite;
  ...
  public double getArea() {
    return laenge * breite;
  }
}
```

Beispiel mit abstrakter Oberklasse



Analyse eines JAVA-Programms (alte Klausuraufgabe)

- Analyse von
  - Klassenvariablen
  - Objektvariablen
  - Funktionsaufrufen
- Auffinden von Compilerfehlern

Beispiel 1: (alte Klausuraufgabe)

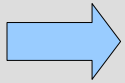
```
public class A {
   public int wert = 3;

   public static int eq (A x1, A x2)
   {
      if (x1 == x2)
        return 1;
      else
        return 2;
   }

   public int eq (A x) {
      if (this == x)
        return 3;
      else
        return 4;
   }
}
```

```
public class B extends A {

   public B (int wert) {
      this.wert = this.wert + wert;
   }

   public int eq (A x) {
      if (wert == x.wert)
        return 5;
      else
        return 6;
   }

   public int eq (B y) {
      if (this == y)
        return 7;
      else
        return 8;
   }
}
```

Beispiel 2

(mit Speicherrepräsentation der entstehenden Objekte)

```
public class A {
  public int elem = 1;
  public B toB() {
    B b = new B();
    b.elem = this.elem;
    return b;
  }
}

public class B extends A {
  public A setElem (int value) {
    this.elem = value;
    return this;
  }
  public A toA() {
    return this;
  }
}
```



Oft ist es hilfreich, sich die Speicherrepräsentation der entstehenden Objekte anzuschauen

Semestralklausur

- Klausurtermin: Mittwoch 11.01.2006
  8:15 – 9:45 im Audimax (nicht in Aula1)

- Anmeldung: Mittwoch (14.12.05) und
  Donnerstag (15.12.05)
  in den jeweiligen Tutorien

- Vorbereitung: Rechnen alter Klausuren
  (s. Webseiten der Vorlesung)