Programmierung und Programmiersprachen

Organisation

Übungen

- Termin: **Dienstags**, **12:15 – 13:45** Uhr in **IC 04/628** und **IC 04/630**
- Beginn: 04.04.2023
- Online-Übung: **Freitags**, **14:15 – 15:45** Uhr via Web-Seminar (siehe Moodle)

Prüfung

- Präsenzklausur am **20.09.2023**

Programmierung und Programmiersprachen

Ziele der Lehrveranstaltung

Lösungsansätze zur Analyse und Entwicklung von Systemen, die wesentlich auf den Konzepten „Objekt“, „Klasse“ und „Vererbung“ beruhen

- Aufbau von Klassen und Interfaces
- Entwicklung von Vererbungshierarchien
- Anwendung von Entwurfsmustern
- Verwendung von vorhandenen Bibliotheken

RUHR-UNIVERSITÄT BOCHUM
Lehrstuhl für Informatik im Bauwesen

RUB

Programmierung und Programmiersprachen

Sommersemester 2023

Einführung in Java

Programmiersprache Java

Was ist Java?

- Java ist eine rein objektorientierte Programmiersprache
- Entstanden (im Wesentlichen) aus C++
- Überschaubare Anzahl von Sprachmitteln – im Gegensatz zu C++
- Plattformunabhängig (wegen Interpretation von Bytecode)
- Sauberes Modulkonzept
- Keine „richtigen“ Zeiger
- Automatische Speicherverwaltung

Programmiersprache Java

Ein erstes Beispiel

- Grundbaustein ist die Klasse
- Minimales Programm muss in eine Klasse eingebettet werden
- Datei muss wie die enthaltende `public` Klasse heißen
- Methode `main()` ist der Startpunkt für die Programmausführung
- Methode `println()` gibt in den Standardausgabe-Stream aus

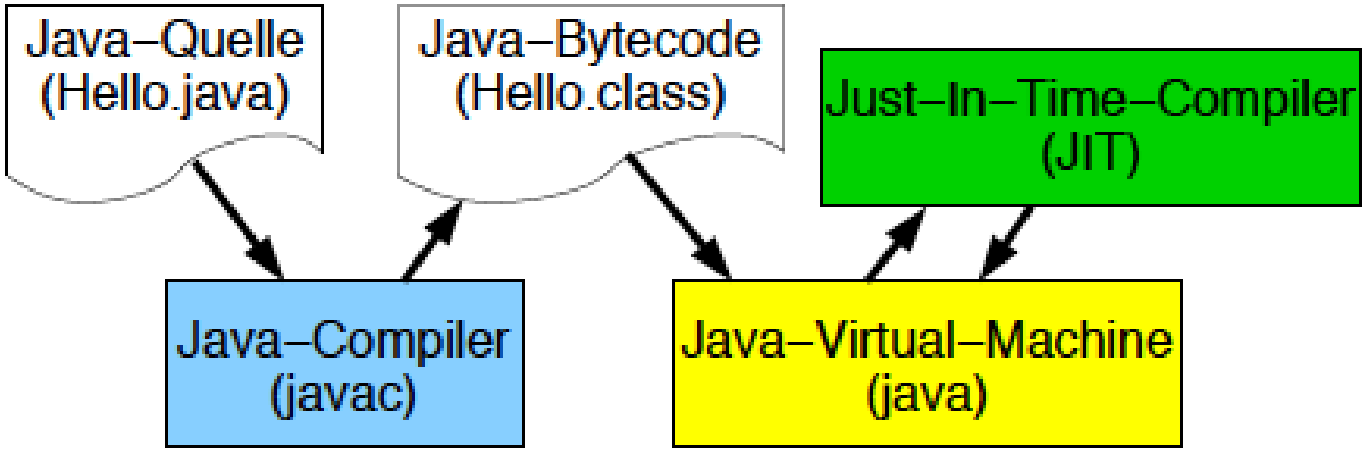
Hello.java

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Programmiersprache Java

Kompilierung und Ausführung

- Java-Quellcode wird vom Compiler in maschinenunabhängigen Bytecode übersetzt (Plattformunabhängigkeit)
- Java-Bytecode wird von virtueller Maschine (JVM) interpretiert
- Evtl. übersetzt ein – in die JVM integrierter – Just-In-Time-Compiler (JIT) Methoden vor ihrer ersten Ausführung



Programmiersprache Java

Primitive Datentypen

- Genau spezifizierte Größe aller primitiven Datentypen

Name des Typs	Art des Typs	Speicherplatzbedarf
byte	vorzeichenbehaftete ganze Zahl	1 Byte
short	vorzeichenbehaftete ganze Zahl	2 Bytes
int	vorzeichenbehaftete ganze Zahl	4 Bytes
long	vorzeichenbehaftete ganze Zahl	8 Bytes
float	reelle Zahl	4 Bytes
double	reelle Zahl	8 Bytes
boolean	boolescher Wert	1 Byte
char	Unicode-Zeichen	2 Bytes

Programmiersprache Java

Strings

- Kein primitiver Datentyp!
- Kann aber als Literal erstellt werden (ohne `new String("...");`)
- Konkatenieren mit +

```
String s1 = "Hallo";
String s2 = "Welt";
System.out.println(s1 + " " + s2);
```

Programmiersprache Java

Strings

Strings.java

```java
public class Strings {
    public static void main(String[] args) {
        // konstruiert ein neues String-Objekt
        String s = "Sag nie zweimal nie"; // NICHT: = new String("...");
        System.out.println("Laenge: " + s.length());
        System.out.println("3. Zeichen: " + s.charAt(2));
        System.out.println("Index von 'nie': " + s.indexOf("nie"));
        System.out.println("Index von 'nie': " + s.lastIndexOf("nie"));
        System.out.println("Teilstring: " + s.substring(4, 15));
        System.out.println("Ersetzt: " + s.replace('a', 'o'));
        System.out.println("Nur Grossbuchstaben: " + s.toUpperCase());
        if (s.equals("Sag niemals nie"))
            System.out.println("Strings sind gleich");
        else
            System.out.println("Strings sind ungleich");
    }
}
```

Programmiersprache Java

Arrays

- Array-Typen, wie z.B. `int[]` sind Klassen
- Die Länge eines Arrays wird bei der Instanziierung festgelegt und kann bestimmt werden

Vector.java

```
public class Vector {
    public static int[] add(int[] u, int[] v) {
        int[] w = new int[u.length]; // nicht-init. Array anlegen
        for (int i = 0; i < u.length; i++)
            w[i] = u[i] + v[i];      // ... und verwenden
        return w;
    }
    public static void main(String[] args) {
        int[] u = new int[] { 1, 3, 1 };    // init. Array anlegen
        int[] v = new int[] { 2, -1, 5 };
        int[] w = add(u, v);
    }
}
```

Programmiersprache Java

Kommandozeilen-Argumente

- Der Parameter `args` von `main()` enthält die Kommandozeilen-Argumente

Arguments.java

```
public class Arguments {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println("Argument " + i + ": "+ args[i]);
    }
}
```

- `String[]` ist ein Array-Typ mit String-Elementen
- `String` ist eine Standard-Klasse von Java
- Zahlen werden bei der Verkettung von Strings automatisch konvertiert
- Strings sind nicht veränderbar (keine Kopie notwendig)

Programmiersprache Java

Was ist eine Klasse?

- Eine Klasse beschreibt eine Sammlung von Eigenschaften (Attributen), Funktionen (Methoden) und Beziehungen zu anderen Klassen.
- Ein Objekt ist eine konkrete Instanz einer Klasse.

Programmiersprache Java

Klasse als Ansammlung von Attributen

Klasse

Person
+ name: String + alter: int + gewicht: double + groesse: double

Instanz (Objekt)

Max:Person
name = "Max Mustermann" alter = 31 gewicht = 81,4 kg groesse = 1,78 m

Maria:Person
name = „Maria Mustermann" alter = 45 gewicht = 73,8 kg groesse = 1,72 m

Programmiersprache Java

Klasse als Ansammlung von Attributen

- Attribute sind Variablen, die an eine Klasse gebunden sind
- z.T. auch als Felder bezeichnet
- Attribute haben immer eine Bezeichnung und einen Typ
  - Der Typ kann primitiv oder eine Klasse sein

```
public class Person {
    public String name;
    public int alter;
    public double gewicht;
    public double groesse;
}
```

Programmiersprache Java

Klasse als Ansammlung von Attributen

- In Java wird ein Objekt mit **new** instanziiert
- Primitive Attribute werden mit Standardwerten initialisiert

```
class Main {
    public static void main(String[] args) {
        Person mustermann = new Person();  // Erstelle ein neues Objekt
        mustermann.name = "Max Mustermann";
        mustermann.alter = 31;
        mustermann.gewicht = 80.4;
        mustermann.groesse = 1.78;

        Person musterfrau = new Person();  // Erstelle ein neues Objekt
        musterfrau.name = "Maria Musterfrau";
        musterfrau.alter = 45;
        musterfrau.gewicht = 73.8;
        musterfrau.groesse = 1.72;

        System.out.println("Max ist " + mustermann.alter " Jahre alt.");
        System.out.println("Maria ist " + musterfrau.alter " Jahre alt.");
    }
}
```

Programmiersprache Java

Klasse als Ansammlung von Funktionen

- Methoden sind Funktionen, die an eine Klasse gebunden sind
- Methoden haben Zugriff auf die Attribute eines Objektes

```
class Person {
    String name;
    int alter;
    double gewicht;
    double groesse;

    // Attribute können gelesen werden
    public double berechneBMI() {
        double bmi = gewicht / (groesse * groesse);
        return bmi;
    }

    // Attribute können verändert werden
    public void erhoeheAlter(int jahre) {
        alter = alter + jahre;
    }
}
```

Programmiersprache Java

Klasse als Ansammlung von Funktionen

- Methoden können nur auf instanziierten Objekten aufgerufen werden
- Der Zustand des Objektes kann dadurch verändert werden

```
class Main {
    public static void main(String[] args) {
        Person mustermann = new Person();  // Erstelle ein neues Objekt
        mustermann.name = "Max Mustermann";
        mustermann.alter = 31;
        mustermann.gewicht = 80.4;
        mustermann.groesse = 1.78;

        // Methoden können nur auf instanziierten Objekten aufgerufen werden!
        double musterBMI = mustermann.berechneBMI()
        System.out.println("BMI: " + musterBMI);

        mustermann.erhoeheAlter(10);
    }
}
```

Programmiersprache Java

Konstruktoren

- Konstruktoren kontrollieren die Instanziierung eines Objekts
- Müssen immer den Bezeichner der Klasse tragen

```
class Person {
    String name;
    int alter;
    double gewicht;
    double groesse;

    public Person(String name, int alter, double gewicht, double groesse) {
        this.name = name;
        this.alter = alter;
        this.gewicht = gewicht;
        this.groesse = groesse;
    }
}
```

Achtung! **name** ist der lokale Parameter **name** welcher das Attribut überdeckt. **this** ist eine Referenz auf das Objekt für das die Methode aufgerufen wird. Mit **this.**name kann man so auf das überdeckte Attribut zugreifen.

Programmiersprache Java

Statische Funktionen

- Auch Klassenmethoden genannt
- Haben kein Zugriff auf die Attribute eines Objekts
- Können ohne erstelltes Objekt aufgerufen werden

```
class Person {
    ...
    public static double berechneDurchschnittsalter(List<Person> personen) {
        double dAlter = 0;
        for(Person p: personen) {
            dAlter += p.getAlter();
        }
        return dAlter / p.size();
    }
}
class Main {
    public static void main(String[] args) {
        List<Person> personen = ...

        // Zum Aufruf ist kein Personenobjekt benötigt
        double durchschnittsalter = Person.berechneDurchschnittsalter(personen);
        System.out.println(durchschnittsalter);
    }
}
```

Programmiersprache Java

Referenzen

- Nicht-primitive Variablen oder Attribute sind Referenzen
- Referenzen sind ähnlich zu Zeigern in C++
  - keine Dereferenzierung notwendig und möglich
  - kein allgemeiner Zeigertyp wie etwa `void` in C++
  - keine Typumwandlungen von Referenzen erlaubt (nur Typecasting)
  - keine Zeigerarithmetik möglich
  - keine Referenzen auf primitive Datentypen (außer mit Wrapper-Klassen)
- Referenzen haben per Voreinstellung den Wert `null`
- Implizit erfolgt kein Aufruf eines Konstruktors

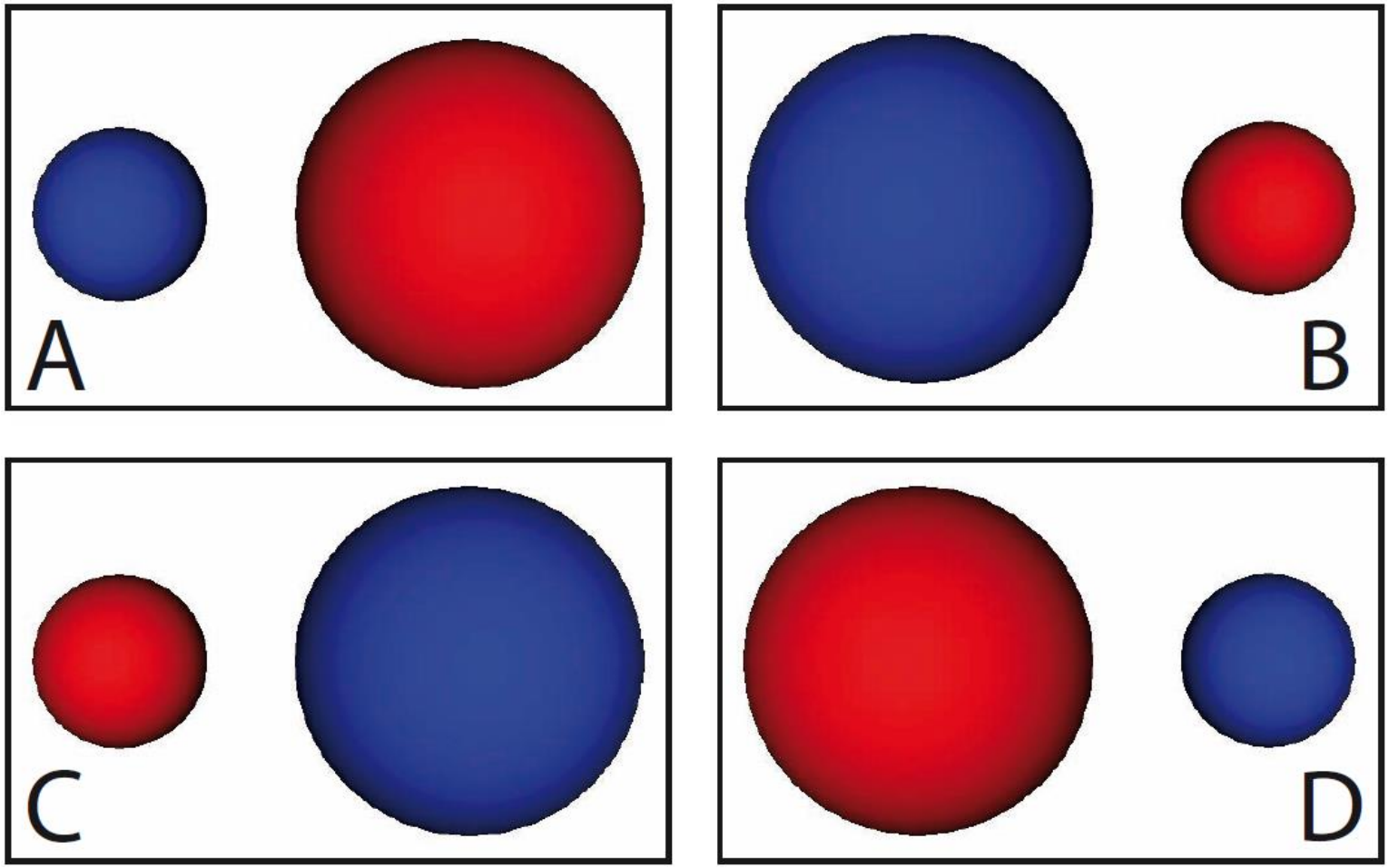
Programmiersprache Java

Beispiel: Arbeiten mit Objekten

```
import inf.v3d.obj.Sphere;

public class AssignmentOperatorProgram {

    public static void main(String[] args) {

        Sphere s;
        Sphere s1 = new Sphere(0, 0, 0); // x,y,z Mittelpunkt
        Sphere s2 = new Sphere(2, 0, 0);

        s = s2;
        s.setColor(Color.red);
        s2 = s1; s1 = s;
        s2.setRadius(0.5);
        s = s2;
        s.setColor(Color.blue);
    }
}
```

Programmiersprache Java

Was ist das Ergebnis?



Programmiersprache Java

Call by Value / Reference

- Nur „Call by Value“ in Java möglich
- Objektzustand kann sich aber ändern, die Referenz nicht
- Zuweisung (=) von Referenzen erzeugt noch keine Kopie
- Mit == werden bei Objekten immer die Referenzen verglichen
- Zum inhaltlichen Vergleich von Objekten kann die Methode `equals()` überschrieben werden

Programmiersprache Java

Call by Value / Reference

```
Sphere s1 = new Sphere(1,1,1);
Sphere s2 = new Sphere(1,1,1);

System.out.println(s1 == s2)      // False!
System.out.println(s1.equals(s2)) // True!

Sphere s3 = s1;

System.out.println(s1 == s3) // True!
System.out.println(s1.equals(s3)) // True!
```

Programmiersprache Java

Call by Value / Reference

```
class Sphere {
    private double l, w, h;

    public Sphere(double l, double w, double h) {
        this.l = l; this.w = w; this.h = h;
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof Sphere) {
            Sphere s = (Sphere) o;
            return (l == o.l && w == o.w && h == o.h);
        } else {
            return false;
        }
    }
}
```

Programmiersprache Java

Speicherverwaltung

- Es wird Speicher für ein Objekt reserviert, sobald der `new`-Operator aufgerufen wird
- Java hat eine automatische Speicherbereinigung (Garbage Collection)
  - Kein `delete`-Operator nötig
  - Speicher wird freigegeben wenn es keine Referenzen mehr auf ein Objekt gibt

Programmiersprache Java

Pakete

- Klassen und Schnittstellen können in Unterverzeichnisse in Form von Paketen angeordnet werden (Paketname kann mit `package` angegeben werden)
- Import von Klassen und Schnittstellen aus anderen Paketen erfolgt mittels dem Schlüsselwort `import`
- Paketnamen sollten möglichst eindeutig sein (Namenkollision vermeiden)
  - Üblich: Domainname rückwärts

```
de.rub.bi.inf.projekt.unterpaket
```

Programmiersprache Java

Zugriffsmodifikatoren

- Der Zugriff auf Klassen, Attribute und Methoden kann kontrolliert werden

Modifikator	Innerhalb einer Klasse	Innerhalb eines Pakets	Außerhalb des Pakets (Nur in Unterklasse)	Außerhalb des Pakets
`private`	Ja	Nein	Nein	Nein
`default`	Ja	Ja	Nein	Nein
`protected`	Ja	Ja	Ja	Nein
`public`	Ja	Ja	Ja	Ja

Programmiersprache Java

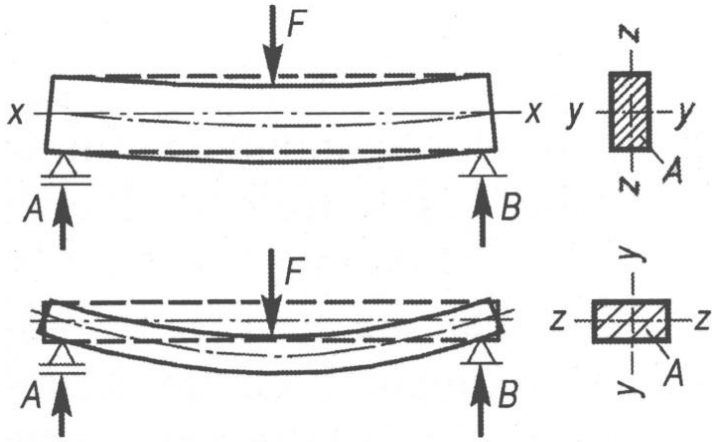
Vererbung

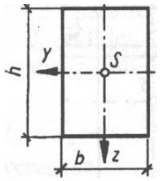
- Häufig ist es wünschenswert, dass bestimmte Attribute und Methoden in anderen Klassen wiederverwendet und ergänzt werden können
- Eine sehr effiziente Möglichkeit ist die Vererbung, dabei werden Attribute und Methoden einer Klasse (**Elternklasse**) an eine andere Klasse (**Kindklasse**) übertragen
  - Attribute und Methoden werden nicht doppelt definiert
  - Änderungen an der Elternklasse sind auch automatisch in den Kindklassen verfügbar
  - Es kann Funktionalität hinzugefügt werden ohne bestehende Klassen zu verändern

Programmiersprache Java

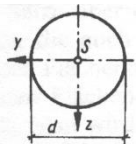
Beispiel Widerstandsmomente

- Maß für den Widerstand, den ein Balken einem Biegemoment entgegensetzt
- Widerstandsmomente ergeben sich aus der Geometrie der Querschnittsfläche
  - Flächenträgheitsmoment
  - Abstand der Randfaser zur neutralen Faser





$$W_y = \frac{bh^2}{6}, \quad W_z = \frac{hb^2}{6}$$

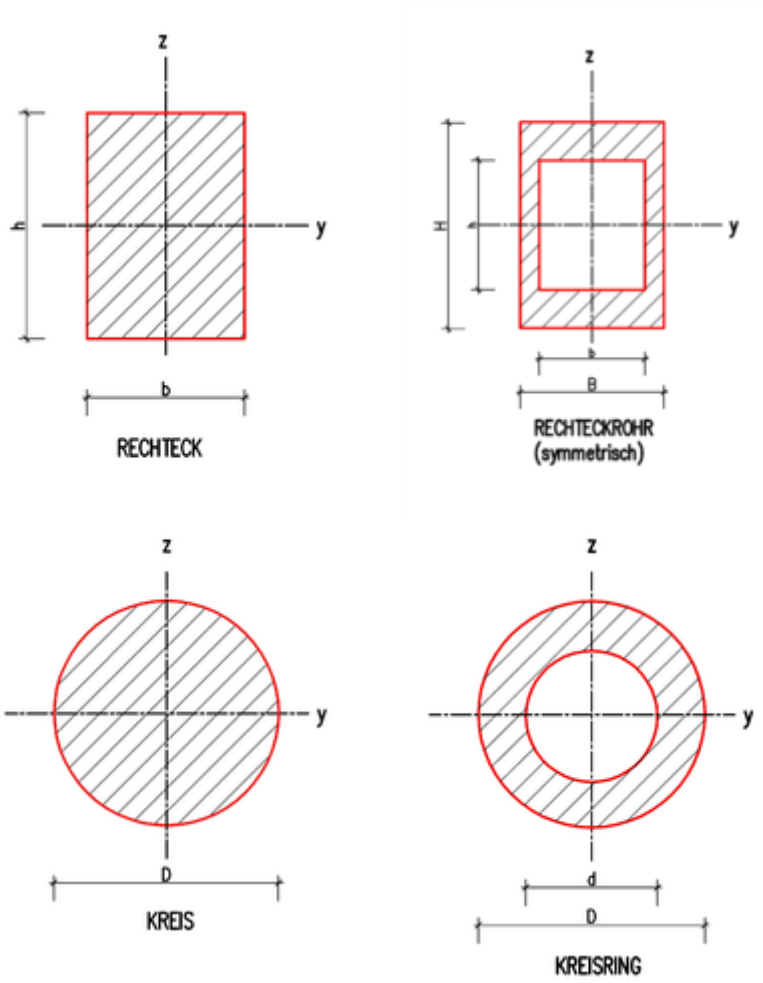


$$W_y = W_z = \frac{\pi r^3}{4}$$

Programmiersprache Java

Beispiel Widerstandsmomente

- Klassen für verschiedene Querschnittsformen
  - Rechteck
  - Hohlprofil
  - Kreis
  - Kreisring
- Mögliche Methoden
  - Berechnung von W_y
  - Berechnung von W_z



Programmiersprache Java

UML Diagramm

Rechteckprofil
- b: double - h: double
+ Rechteckprofil(b: double, h: double) + getWy(): double + getWz(): double

Kreisprofil
- d: double
+ Kreisprofil(d: double) + getWy(): double + getWz(): double

Hohlprofil
- b: double - h: double - B: double - H: double
+ Hohlprofil(b: double, h: double, B: double, H: double) + getWy(): double + getWz(): double

Kreisringprofil
- d: double - D: double
+ Kreisringprofil(d: double, d: double) + getWy(): double + getWz(): double

Programmiersprache Java

Implementierung

- Doppelte Attribute `b` und `h`

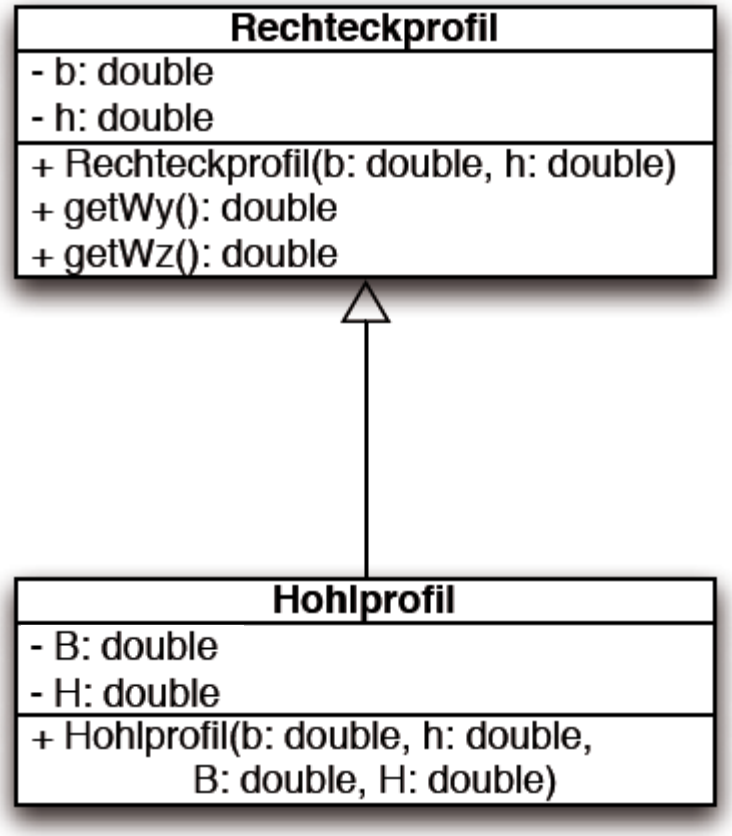
```
public class Rechteckprofil {

  // Attribute
  private double b, h;

  // Konstruktor
  public Rechteckprofil(double b,
     double h) {

     this.b = b;
     this.h = h;
  }

  // Widerstandsmoment Wy
  public double getWy() {
     return this.b*this.h*this.h/6.;
  }
}
```

```
public class Hohlprofil {

  // Attribute
  private double b, h;
  private double B, H;

  // Konstruktor
  public Hohlprofil(double b,
     double h, double B, double H) {

     this.b = b; this.h = h;
     this.B = B; this.H = H;
  }
  // Widerstandsmoment Wy
  public double getWy() {
     return (B*Math.pow(H,3)-
       b*Math.pow(h,3))/(6.0*H);
  }
}
```

Programmiersprache Java

Vererbung

- Die Klasse `Hohlprofil` soll alle Attribute und Methoden der Klasse `Rechteckprofil` erben
- Die vererbten Attribute und Methoden werden nicht noch einmal aufgenommen



Programmiersprache Java

Implementierung

- Elternklasse wird nach dem Schlüsselwort `extends` angegeben
- Zugriff auf Attribute und Methoden der Elternklasse erfolgt über das Schlüsselwort `super`

```
public class Hohlprofil extends Rechteckprofil {

    // Zusätzliche Attribute
    private double B, H;

    // Konstruktor
    public Hohlprofil(double b, double h,
                      double B, double H) {
        super(b, h);
        this.B = B;
        this.H = H;
    }
}
```

Programmiersprache Java

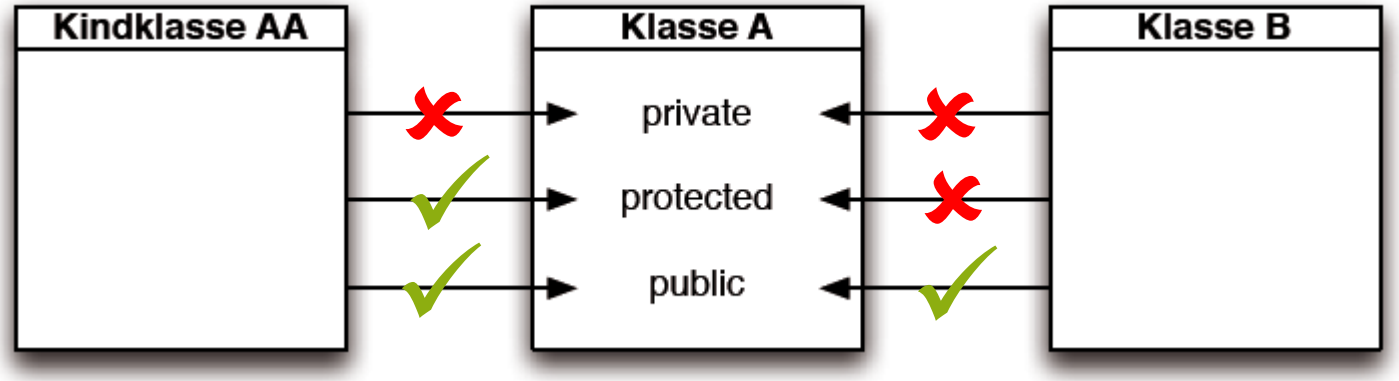
Überladen von Attributen und Methoden

- Die Klasse `Hohlprofil` soll jedoch eine andere Methode `getWy()` zur Berechnung verwenden
- Attribute und Methoden können in Kindklassen überladen werden
- Beim Aufruf wird dann das neue Attribut bzw. die neue Methode der Kindklasse verwendet
- Überladene Attribute bzw. Methoden der Elternklasse können über den Zugriff mittels `super` weiterhin verwendet werden

Programmiersprache Java

Zugriffsrechte

- Um den Zugriff auf Attribute und Methoden der Elternklasse zu ermöglichen, dürfen diese nicht als `private` gekennzeichnet sein
- Um einen eingeschränkten Zugriff nur für die Kindklassen zu ermöglichen, kann das Zugriffsrecht `protected` verwendet werden



Programmiersprache Java

Implementierung

```
public class Rechteckprofil {


  // Attribute
  protected double b, h;

  // Konstruktor
  public Rechteckprofil(double b,
     double h) {

     this.b = b;
     this.h = h;
  }

  // Widerstandsmoment Wy
  public double getWy() {
     return this.b*this.h*this.h/6.;
  }
}
```

```
public class Hohlprofil extends
     Rechteckprofil {

  // Attribute
  private double B, H;

  // Konstruktor
  public Hohlprofil(double b,
     double h, double B, double H) {

     super(b, h);
     this.B = B; this.H = H;
  }

  // Widerstandsmoment Wy
  public double getWy() {
     return (B*Math.pow(this.H,3)-
       super.b*Math.pow(super.h,3))
          /(6.0*H);
  }
}
```

Programmiersprache Java

Verwendung von Elternklassen (*Subtyping*)

Häufig gibt es die Anforderung Objekte verschiedener Kindklassen gemeinsam in einer Datenstruktur zu speichern

- z.B. Speicherung verschiedener Rechteckprofile und Hohlprofile in einer gemeinsamen Menge
- Bei der Angabe der Klasse der Elemente einer Datenstruktur bzw. einer Variablen, wird dann die Elternklasse angegeben

```
Rechteckprofil[] profile = new Rechteckprofil[3];

Rechteckprofil r1 = new Rechteckprofil(13.0, 4.0);
Rechteckprofil r2 = new Hohlprofil(13.0, 4.0, 10.0, 2.0);

profile[0] = r1;
profile[1] = r2;
profile[2] = new Hohlprofil(13.0, 4.0, 10.0, 2.0);
```

Programmiersprache Java

Vererbung

- Java 7 unterstützt nur *Single Inheritance* (Einfachvererbung)
  - Eine Kindklasse kann nur eine Elternklasse haben
- Seit Java 8 ist auch *Multiple Inheritance* (Mehrfachvererbung) durch Interfaces möglich (hier nicht betrachtet)
- Jede Klasse ist zumindest indirekt von der Oberklasse `Object` abgeleitet

Programmiersprache Java

Casting

- Mit dem `instanceof`-Operator kann geprüft werden, ob ein Objekt von einer Klasse abstammt
- Durch Casting kann man den Zugriff auf Unterklassenmethoden oder Attribute erlauben

```
Rechteckprofil r = getSomeProfil();
if(r instanceof Hohlprofil) {
    // r ist ein Hohlprofil, oder eine Unterklasse von Hohlprofil
    Hohlprofil h = (Hohlprofil) r;  // Casting
} else {
    // r ist ein Rechteckprofil, aber kein Hohlprofil
}
```

- Ist kein Casting möglich, so wird eine `ClassCastException` geworfen

Programmiersprache Java

Klasse Object

In der Programmiersprache Java gibt es eine spezielle Basisklasse `Object`, die automatisch als Elternklasse für alle anderen Klassen verwendet wird

Wichtige Methoden der Klasse `Object` sind

- Vergleich von zwei Objekten (Standard = Vergleich der Speicheradressen)

```
boolean equals(Object obj);
```

- Darstellung als Zeichenkette (Standard = Speicheradresse)

```
String toString();
```

Programmiersprache Java

Überladung von toString

Häufig wird die Methode `toString()` überladen, um eine eigene Darstellung, z.B. für die Ausgabe mittels `System.out.println`, zu ermöglichen

```
public class Rechteckprofil {


  // Attribute
  protected double b, h;
  //...

  // Zeichenkettendarstellung
  public String toString() {
     return "b="+this.b+
          "; h="+this.h;
  }
}
```

```
public class Hohlprofil extends
     Rechteckprofil {

  // Attribute
  private double B, H;
  //...

  // Zeichenkettendarstellung
  public String toString() {
     return super.toString()+
          "; B="+this.B+
          "; H="+this.H;
  }
}
```

Programmiersprache Java

Abstrakte Klassen und Schnittstellen

Wie kann sichergestellt werden, dass jedes Profil eine Methode zur Berechnung der Widerstandsmomente Wy und Wz besitzt?

1. **Verwendung einer (abstrakten) Basisklasse**
2. Verwendung einer Schnittstelle

```
public abstract class Profil {

  // Widerstandsmoment Wy
  public abstract double getWy();

  // Widerstandsmoment Wz
  public abstract double getWz();
}
```

Programmiersprache Java

Abstrakte Klassen

- Klasse von der **keine Objekte** erzeugt werden können
- Kann `abstract` Methoden definieren, welche nur die Methodensignatur vorgibt, ohne Implementierung
- Unterklassen müssen alle abstrakten Methoden implementieren
- Ausnahme: Ist die Unterklasse eine weitere abstrakte Klasse, so können die abstrakten Methoden weiterhin ohne Implementierung vererbt werden

Programmiersprache Java

Schnittstellen

Wie kann sichergestellt werden, dass jedes Profil eine Methode zur Berechnung der Widerstandsmomente Wy und Wz besitzt?

1. Verwendung einer (abstrakten) Basisklasse
2. **Verwendung einer Schnittstelle**

```
public interface Profil {

  // Widerstandsmoment Wy
  double getWy();

  // Widerstandsmoment Wz
  double getWz();
}
```

Programmiersprache Java

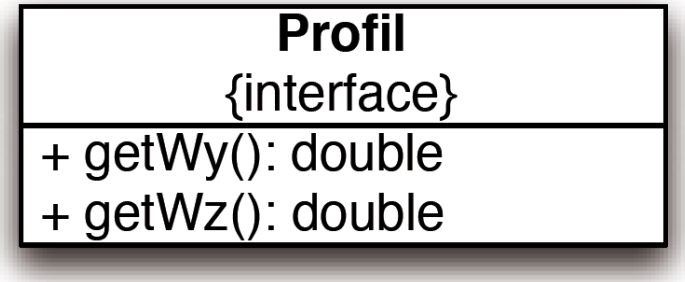
Schnittstellen

Eine Schnittstelle enthält keine Implementierungen, sondern deklariert nur den Kopf einer Methode

Die Deklaration einer Schnittstelle erinnert an eine abstrakte Klasse mit abstrakten Methoden, nur steht an Stelle von `class` das Schlüsselwort `interface`

Konstanten (`static final`-Variablen) sind in einer Schnittstelle erlaubt, statische Methoden jedoch nicht

Eine Schnittstelle darf keinen Konstruktor deklarieren



Programmiersprache Java

Schnittstellen und Vererbung

Möchte eine Klasse eine Schnittstelle verwenden, so folgt hinter dem Klassennamen das Schlüsselwort `implements` und dann der Name der Schnittstelle

Klassen werden vererbt und Schnittstellen implementiert

Implementiert eine Elternklasse eine bestimmte Schnittstelle, dann erfüllen auch alle Kindklassen diese Schnittstelle

Jede Klasse kann beliebig viele Schnittstellen implementieren (eine Art von Mehrfachvererbung)

Programmiersprache Java

Implementierung

```
public class Rechteckprofil
                  implements Profil {

  // Attribute
  protected double b, h;

  // Konstruktor
  public Rechteckprofil(double b,
     double h) {
     this.b = b; this.h = h;
  }

  // Widerstandsmoment Wy
  public double getWy() {
     return this.b*this.h*this.h/6.;
  }
  // Widerstandsmoment Wz
  public double getWz() {
     return this.h*this.b*this.b/6.;
  }
}
```

```
public class Kreisprofil
                  implements Profil {

  // Attribute
  protected double d;

  // Konstruktor
  public Kreisprofil (double d) {
     this.d = d;
  }

  // Widerstandsmoment Wy
  public double getWy() {
     return Math.pow(this.d,3)
                  *Math.PI/32.;
  }
  // Widerstandsmoment Wz
  public double getWz() {
     return this.getWy();
  }
}
```

Programmiersprache Java

Implementierung

```
public class Rechteckprofil
                  implements Profil {

  // Attribute
  protected double b, h;

  // Konstruktor
  public Rechteckprofil(double b,
     double h) {
     this.b = b; this.h = h;
  }

  // Widerstandsmoment Wy
  public double getWy() {
     return this.b*this.h*this.h/6.;
  }
  ...



}
```

```
public class Hohlprofil extends
     Rechteckprofil {

  // Attribute
  private double B, H;

  // Konstruktor
  public Hohlprofil(double b,
     double h, double B, double H) {

     super(b, h);
     this.B = B; this.H = H;
  }

  // Widerstandsmoment Wy
  public double getWy() {
     return (B*Math.pow(H,3)-
       super.b*Math.pow(super.h,3))
          /(6.0*H);
  }
  ...
}
```

Programmiersprache Java

Verwendung von Schnittstellen

Sollen verschiedene Querschnitttypen zusammenhängend gespeichert werden, kann auch eine Schnittstelle als Datentyp angegeben werden

```
// Menge von Querschnitten
Profil[] profile = new Profil[3];

// Verschiedene Querschitte erzeugen
profile[0] = new Rechteckprofil(4.0, 8.0);
profile[1] = new Kreisprofil(5.0);
profile[2] = new Hohlprofil(4.0, 8.0, 3.5, 3.5);

// Ausgabe alle Querschnitte
for (int i=0; i < profile.length; i++) {
    System.out.println(p[i]);
}
```

Abstrakte Klassen vs. Schnittstellen

- Schnittstellen sind ein „Versprechen“ eine Reihe an Methoden zu implementieren
  - Meist genutzt um Funktionalität für andere Klassen bereitzustellen (Comparable, Observalbe, etc.)
- Abstrakte Klassen fassen Klassen zusammen welche ähnliche Funktionalität haben
  - Häufig als Oberklasse für verschiedene Implementierungen genutzt (z.B. ArrayList und LinkedList erben beide von AbstractList)