Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Programmierung 1
WiSe 25/26
Moritz Balg
Ausgabe: 08.12.2025
Bearbeiten bis: 14.12.2025

Übungsblatt 10

- Objektorientierte Programmierung 2 -

Aufgabe 1: Access Modifier

In der Vorlesung haben Sie die vier Zugriffsmodifikatoren (*engl. access modifier*) von Java kennengelernt. Access Modifier schränken den Zugriff von Funktionen, Variablen und Klassen ein. Füllen Sie folgende Tabelle weiter aus, indem Sie an den gültigen Stellen einen ✓-Symbol einfügen. In der Zelle public-Klasse ist ein ✓-Symbol, da bspw. Variablen mit diesem Access Modifier in der eigenen Klasse sichtbar sind.

Modifier	Klasse	Package	Subklasse	Global
`public`	✓			
`protected`				
`default`				
`private`				

Info

- In zukünftigen (Prüfungs-)Aufgaben wird häufig der Satz stehen: Alle ihre Attribute **müssen** `privat` sein.
- Wenn Sie in einer abgeleiteten Klasse auf Attribute der Oberklasse zugreifen wollen, liegt häugig ein Problem in der Softwaremodellierung vor. Die Logik zur Verarbeitung von Daten gehört dort hin, wo die Daten auch sind.
- Gehen Sie ihre alten Lösungen zu Aufgaben mit Klassen durch und ergänzen dort bei jeder Klasse, jedem Attribut und jeder Methode in sinnvoller Weise ein Zugriffsattribut (`private, public,...`) aus der Menge der jeweils möglichen, das an dieser Stelle dann Sinn macht.

Mögliche Lösung:

Modifier	Klasse	Package	Subklasse	Global
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	
`default`	✓	✓		
`private`	✓			

Aufgabe 2: Stack 👾

In der Vorlesung zu Abstrakte Datentypen haben wir einen Stapel (*engl. Stack*) kennengelernt. Entwickeln Sie einen allgemeinen Stack, der ein Behälter für Objekte eines beliebigen Referenztyps darstellt. Nutzen Sie dabei den Typ `Object` in geeigneter Weise.

Das Gerüst der Stack-Klasse ist wie folgt vorgegeben:

```
public class Stack {
  public Stack(int maximalGroesse) {...}
  public void push(Object o) {...}
```

```
  public Object pop() {...}
  public boolean isEmpty() {...}
}
```

Dem Konstruktur übergibt man die maximal Größe des Stacks. Die pop-Methode liefert das oberste Stack-Element und löscht es auch gleichzeitig vom Stack (im Gegensatz zur der pop-Funktion des abstrakten Datentyps damals). Füllen Sie sinnvoll diese Klasse aus. Sie können/müssen zusätzliche Instanzvariablen anlegen. Folgendes kleines Programm könen Sie für einen Test der Klasse verwenden:

```
public static void main(String[] args) {
  Stack st = new Stack(10);
  st.push("a");
  st.push("b");
  st.push("c");
  System.out.println(st.pop());
  System.out.println(st.pop());
  System.out.println(st.pop());
  System.out.println(st.isEmpty());
}
```

Als Anwendung des Stacks wurde in der Vorlesung vorgestellt, wie man mit Hilfe eines Stacks eine Folge von Werten spiegeln kann. Passen Sie das Verfahren auf die neu programmierte Stack-Klasse an. Nehmen Sie statt einer Folge einen String. Nutzen Sie statt der damals definierten Folge-Operationen (`isemptyfolge, first, rest,...`) in sinnvoller Weise String-Operationen als Basisoperationen:

- Über `str.length()` kann man ermitteln, ob ein String leer ist oder nicht.
- Über `str.charAt(0)` kann man das erste Element eines Strings bestimmen, das vom Typ ein char ist. Dies ist ein primitiver Typ und kein Referenztyp. Unser Stack arbeitet aber nur mit Referenztypen!
- Über `"" + c` kann man aus einem char-Wert (primitiver Typ) einen String-Wert machen (Referenztyp).
- Über `str.substring(...)` kann man einen Teil eines String bekommen, zum Beispiel den letzten Teil außer dem ersten Zeichen.

Die Methodensignatur in einer Klasse `Spiegeln` ist vorgegeben mit `public static String spiegeln(String s)`. Ein Testprogramm ist dann etwa:

```
public static void main(String[] args){
  String s = "hallo";
  System.out.println(s + "gespiegelt ist " + spiegeln(s));
}
```

Info

Laden Sie im Praktomat ihre beiden Dateien `Stack.java` und `Spiegeln.java` einzeln hoch, also jeweils pro Klasse eine Upload-Operation.

Aufgabe 3: Flugzeuge

Enwickeln Sie in dieser Aufgabe Klassen für Flugzeuge. Die Basisklasse soll die Klasse `Flugzeug` sein. Jedes Flugzeug hat eine Spannweite (reelle Zahl) und eine Sitzkapazität (natürliche Zahl).

Ein Segelflugzeug ist ein Flugzeug, das eine beliebige Spannweite hat, die man bei Erzeugung angeben muss, aber immer nur einen Sitzplatz für den Piloten (Schulungssegelflugzeuge mit 2 Sitzen betrachten wir hier nicht).

Ein Passagierflugzeug ist ein Flugzeug, hat aber (im Gegensatz zu einem Segelflugzeug) Motoren mit einem bestimmten Gesamtschub (in kN). Beim Erzeugen eines Passagierflugzeugs muss man Spannweite, Sitzplatzzahl und Schub eines solchen Flugzeugs angeben.

Eine A380 ist ein Passagierflugzeug, das 79,8 m Spannweite hat, 558 Sitzplätze und 4 Motoren mit jeweils 320 kN (mit dem Trent 972 Triebwerk).

Entwickeln Sie geeignete Klassen mit einer entsprechenden Klassenhierarchie. Sehen Sie in jeder Klasse einen Konstruktor vor, der der obigen Beschreibung angemessen ist. Sehen Sie in jeder Klasse eine `public String toString()`-Methode vor, die wesentliche Eigenschaften eines Objekts dieser Klasse als String liefert. Zählen Sie weiterhin (innerhalb der Klassen!), wieviele Flugzeuge von jeder Kategorie erzeugt wurden und sehen Sie entsprechende getter-Funktionen vor.

Nehmen Sie folgendes main in einer Klasse FlugzeugTest als Test:

```
public static void main(String[] args){
  //ein Segelflugzeug mit 17m Spannweite erzeugen
  Segelflugzeug sf = new Segelflugzeug(17.0);
  System.out.println(sf);
  //ein Passagierflugzeug mit 10,97 Spannweite, 4 Plaetzen und 1,3kN Schub(Cessna172P)
  Passagierflugzeug pg = new Passagierflugzeug(10.97, 4, 1.3);
  System.out.println(pg);
  //eine A380 erzeugen(Jede A380 sieht gleichaus.)
  A380 a380 = new A380();
  System.out.println(a380);
  //TODO hier die Anzahl jeder Kategorie jeweils ausgeben
}
```

Ergänzen Sie an der mit `TODO` markierten Stelle Code, so dass die Anzahl aller erzeugten Kategorien ausgegeben wird. Für das obige Beispiel müsste die Ausgabe etwa sein (eine A380 ist auch ein Passagierflugzeug):

```
Spannweite: 17.0, Sitze: 1 (Segelflugzeug)
Spannweite: 10.97, Sitze: 4, Schub: 1.3 (Passagierflugzeug)
Spannweite: 79.8, Sitze: 558, Schub: 1280.0 (Passagierflugzeug) A380
Anzahl Fluzgzeuge: 3
Anzahl Segelfluzgzeuge: 1
Anzahl Passagierfluzgzeuge: 2
Anzahl A380: 1
```

Mögliche Lösung:
Eine mögliche Lösung mit allen Klassen zu dieser Aufgabe finden Sie als flugzeuge.zip Datei in LEA.

Aufgabe 4: OOP Wissen

Welche der folgenden Aussagen sind wahr, welche falsch? Geben Sie jeweils eine kurze Begründung an.

1. Über eine Referenz kann keine Klassenmethode aufgerufen werden.
2. Jeder Konstruktor einer Klasse muss alle Instanzvariablen dieser Klasse initialisieren.
3. In einer Instanzmethode dürfen Klassenvariablen dieser Klasse mit einfachem Namen verwendet werden.
4. In einer Klassenmethode dürfen keine Instanzvariablen dieser Klasse mit einfachem Namen verwendet werden.
5. In jedem Konstruktor muss explizit ein Konstruktor der Oberklasse aufgerufen werden.
6. Für Instanzmethodenaufrufe wird bei Laufzeit die auszuführende Implementierung gewählt.
7. Über eine Referenz können Instanz- und Klassenmethoden aufgerufen werden.
8. Ein Konstruktor darf nicht auf Klassenvariablen zugreifen.

Mögliche Lösung:

1. falsch – für eine Referenz sind der Klassentyp und damit dessen Klassenmethoden bekannt
2. falsch – Instanzvariablen sind automatisch initialisiert und können, müssen aber nicht von Konstruktoren überschrieben werden
3. wahr – eine Instanz kennt ihre Klasse, also auch deren Klassenvariablen
4. wahr – eine Klassenmethode kennt keine Instanz, also auch keine Instanzvariablen

5. falsch – fehlt ein expliziter Aufruf, wird implizit super() aufgerufen
6. wahr – bei Laufzeit wird nach der speziellsten überschreibenden Variante gesucht
7. wahr – die Referenz liefert sowohl einen Klassentyp als auch eine Instanz
8. falsch – ein Konstruktor kennt seine Klasse und somit deren Klassenvariablen

Aufgabe 5: Fussball 👾

Beschreiben Sie in einer Klasse `Person` eine Person. Zu einer Person gehört ein Nachname. Über einen Konstruktor lässt sich eine Person erzeugen, wobei man den Namen als String übergeben muss. Sehen Sie eine Getter-Funktion vor, um den Namen auslesen zu können. Eine Methode `public String toString()` liefert (ebenfalls) den Nachnamen. Beispiel: `Meier`

Hinweis: Diese Klasse hat also mindestens diese Konstruktoren/Methoden:

```
Person(String name);
public String getName();
public String toString();
```

Ein Fußballspieler (Klasse `Fussballspieler`) ist eine Person, die ein jährliches Gehalt bekommt (ein ganzzahliger Wert), das man zusätzlich zu einem Namen bei der Konstruktion als zweites Argument angeben muss. Sehen Sie eine Getter- Methode für das Einkommen vor. Eine Methode public String toString() liefert den Namen und das Gehalt (als einfache Zahl, ohne Währungszeichen etc.) durch ein Leerzeichen getrennt als String. Beispiel: `Hummels 1000000`

Hinweis: Diese Klasse hat also mindestens diese Konstruktoren/Methoden:

```
Fussballspieler(String name, int einkommen);
public int getEinkommen();
public String toString();
```

Geben Sie eine Klasse `Mannschaft` an, die aus genau 5 Fussballspielern besteht (Hallenfußball). Es gibt einen Konstruktor, der aus genau 5 einzelnen Fußballspielern (Java-Objekte werden übergeben, nicht deren Namen!) eine Mannschaft erzeugt. Es werden also auch 5 Argumente an den Konstruktor übergeben. Sehen Sie eine Methode public int einkommen() vor, die das Gesamteinkommen dieser Mannschaft liefert (Summe aller Einzeleinkommen). Weiterhin soll es eine Methode public String toString() geben, die die Aufstellung der Mannschaft in der folgenden Form liefert (als String):

```
1. Horn
2. Kimmich
3. Hummels
4. Lewandowski
5. Modeste
```

Hinweis: `\n`

Hinweis: Diese Klasse hat also mindestens diese Konstruktoren/Methoden:

```
Mannschaft(Fussballspieler s1, Fussballspieler s2, Fussballspieler s3, Fussballspielers4,
Fussballspieler s5);
public int einkommen();
public String toString();
```

Geben Sie eine Klasse `Stadion` mit einem `main` an. Im main werden zwei Mannschaften erzeugt und 45.000 Zuschauer (dies sind einfache Personen). Als Namen der Zuschauer geben Sie `Zuschauer-i` an, wobei i der i-te Zuschauer ist, bei 1 beginnend. Bei den Fußballspielern geben Sie `Spieler-x-i` an, wobei x rot (1. Mannschaft) oder blau (2. Mannschaft) ist und mit i wiederum die Spieler durchgezählt werden (bei 1 beginnend).

Als Einkommen des i-ten Spielers geben Sie für die rote Mannschaft `10.000 * i` an, für die blaue Mannschaft `20.000 * i`. Lassen Sie anschließend auf dem Bildschirm die Namen der roten Mannschaft ausgeben (Format

siehe oben), eine Leerzeile, dann die Namen der blauen Mannschaft, eine Leerzeile und dann zu beiden Mannschaften jeweils das Gesamteinkommen. Geben Sie anschließend noch den Namen des ersten Zuschauers aus. Ihre Ausgabe müsste also für die oben gezeigten Beispieldaten wie folgt aussehen:

```
1. Spieler-rot-1
2. Spieler-rot-2
3. Spieler-rot-3
4. Spieler-rot-4
5. Spieler-rot-5

1. Spieler-blau-1
2. Spieler-blau-2
3. Spieler-blau-3
4. Spieler-blau-4
5. Spieler-blau-5

150000
300000
Zuschauer-1
```

Aufgabe 6: Codeanalyse (statische und dynamische Bindung)

Gegeben seien folgende Klassen:

```
class K1{
  public static int s;
  public int i;

  K1(){
    s++;
  }

  K1(int i){
    i=i;
  }

  public int f(int x){
    s+=x;
    return i++;
  }

  public static void g(){
    ++s;
  }

  public void h(){
    f(s);
    g();
  }

  public String toString(){
    return "s=" + s + ", i=" + i;
  }
}
```

```
class K2 extends K1{
  private static int s;
  private int i;

  K2(int i1){
    i=i1;
  }

  K2(int i1,int i2){
    super(i1);
    i=i1;
  }

  public int f(){
    i++;
    super.g();
    g();
    return i;
  }

  public static void g(){
    s++;
  }

  public void h(){
    s++;
  }

  public String toString(){
    return "s=" + s + ", i=" + i;
  }
}
```

```
public class TestK1K2{
  public static void main(String[] args){
    K1 k1 = new K1(1);
    K2 k2 = new K2(2);
    int i;
    k1.f(i=1);
    k2.f(i=1);
    System.out.println("k1:" + k1);
    System.out.println("k2:" + k2);
    K2 k3 = new K2(3,4);
    k3.f();
    k3.h();
    System.out.println("k1: " + k1);
    System.out.println("k2: " + k2);
    System.out.println("k3: " + k3);
    K1 k4 = new K2(5,6);
    System.out.println("k4: " + k4);
    k4.f(2);
    k4.h();
    System.out.println("k4: " + k4);
  }
}
```

Bei Ausführung des Programms kommt es zu folgender Ausgabe:

```
> java TestK1K2
k1:s=3, i=1
k2:s=0, i=2
k1:s=4, i=1
k2:s=2, i=2
k3:s=2, i=4
k4:s=2, i=5
k4:s=3, i=5
```

Erläutern Sie im Detail, wieso es zu genau dieser Ausgabe kommt. Oder anders ausgedrückt: Was passiert genau während des Programmablaufs?

Info

Bei Instanziierung eines Objekts werden zu Beginn alle Instanzvariablen mit dem entsprechenden Null-Wert initialisiert.

Mögliche Lösung:

In der Aufgabe kommen einige Gemeinheiten vor. Der parameterbehaftete Konstruktor der Klasse K1 hat beispielsweise keinen Effekt, da hier das Keyword `this` nicht eingesetzt wurde. Zusätzlich muss beachtet werden, dass bei der Instanziierung von Objekten implizit der Konstruktor der Oberklasse aufgerufen wird. Die vermutlich gemeinste Stelle im Code ist jedoch die versteckte Variabel `i` in der Klasse K2. Da die Variable mit dem gleichen Namen nochmal deklariert wurde, ist die vererbte Variabel `i` nicht Sichtbar. Demnach haben Objekte der Klasse K2 eigentlich zwei i-Variablen. Aufrufe von Funktionen der Oberklasse beziehen sich auf `super.i` (in der nachfolgenden Tabelle als i_s bezeichnet) und Aufrufe von Methoden der eigenen Klasse beziehen sich auf das `i` in der Subklasse.

Zeile	K1.s	K2.s	k1.i	k2.i	k2.i_s	k3.i	k3.i_s	k4.i	k4.i_s
`K1 k1 = new K1(1);`	0	0	0						
`K2 k2 = new K2(2);`	1			2					
`int i;`									
`k1.f(i=1);`	2		1						
`k2.f(i=1);`	3				1				
`System.out.println("k1:" + k1);`	`k1:s=3, i=1`								
`System.out.println("k2:" + k2);`	`k2:s=0, i=2`								
`K2 k3 = new K2(3,4);`						3			
`k3.f();`	4	1				4			
`k3.h();`		2							
`System.out.println("k1: " + k1);`	`k1:s=4, i=1`								
`System.out.println("k2: " + k2);`	`k2:s=2, i=2`								
`System.out.println("k3: " + k3);`	`k3:s=2, i=4`								
`K1 k4 = new K2(5,6);`								5	
`System.out.println("k4: " + k4);`	`k4:s=2, i=5`								
`k4.f(2);`	6								1
`k4.h();`		3							
`System.out.println("k4: " + k4);`	`k4:s=3, i=5`								

Aufgabe 7: Webshop 👾

In dieser Aufgabe soll ein Web Shop programmiert werden.

Ein Kunde (Klasse `Kunde`) wird beschrieben durch einen Vornamen und Nachnamen.

Gute und treue Kunden (Klasse `GuterKunde`) sind Kunden, die zusätzlich einen Rabatt auf Bestellungen bekommen (z.B. 0.05 für 5% Rabatt).

Ein Artikel (Klasse `Artikel`) im Web Shop wird beschrieben mit einem Namen, dem zugehörigen Preis in Euro (`double`) und dem aktuellen Bestand (wie viele Exemplare derzeit vorhanden sind; ganzzahlig).

In einem Web Shop (Klasse `WebShop`) können maximal 10 Kunden und 10 Artikel registriert sein. Ein neuer Artikel kann dem Web Shop hinzugefügt werden über eine Methode `void neuerArtikel(String name, double preis, int anzahl)`, wobei anzahl die Anzahl an Exemplaren ist, die zu diesem Artikel im Web Shop anschließend vorhanden sein soll. Ein neuer Kunde kann in einem Web Shop registriert werden über `Kunde neuerKunde(String vorname, String nachname)`, gute Kunden über die (dann überladene) Methode `Kunde neuerKunde(String vorname, String nachname, double nachlass)`. Beachten Sie hierbei, dass der Ergebnistyp dieser Methode Kunde ist, aber in der Methode ein GuterKunde erzeugt werden soll (siehe auch Hinweise unten).

In einem Web Shop wird die Bestellung eines Kunde bearbeitet über die Methode `String bestellen(Kunde kunde, String[] artikel)`. Der Parameter `kunde` steht für den bestellenden Kunden (was auch ein guter Kunde sein kann) und das String-Feld enthält die Namen (als `String`) der Artikel, die bestellt werden sollen (jeweils ein Exemplar davon pro Eintrag). Wenn ein Artikel ausgegeben wird, so reduziert sich der Bestand dieses Artikels um 1. Das Ergebnis der Methode ist ein String mit der Rechnung zu dieser Bestellung (siehe Beispiel unten).

Das nachfolgende Beispiel (in Klasse `WebShopTest`) erläutert den Aufbau der Rechnung, inklusive aller möglichen Fälle. Falls ein gewünschter Artikel nicht im Bestand vorhanden ist, soll eine entsprechende Ausgabe auf der Rechnung erfolgen (`nicht gefunden`). Und falls ein Artikel ausverkauft ist (der Bestand ist also 0), dann soll auch eine entsprechende Meldung auf der Rechnung ausgegeben werden (`nicht mehr vorhanden`). Gute Kunden bekommen ihren Rabatt bei jedem bestellten Artikel angerechnet. Alle diese Sonderfälle sind im Beispiel unten beim zweiten (guten) Kunden zu sehen und auch die vorgegebe Ausgabe für alle Fälle.

Für das folgende Beispielprogramm

```
public static void main(String[] args){
  //erzeuge Webshop
  WebShop w = new WebShop();

  //neuer Kunde
  Kunde kunde1 = w.neuerKunde("Helga","Herrlich");

  //neuer guter Kunde mit 5% Preisnachlass
  //Beachte: Ein GuterKunde-Objektwurde erzeugt, aber der Typ der Variablen ist Kunde!
  Kunde kunde2 = w.neuerKunde("Werner","Winzig",0.05);

  //neue Artikel (2x Leberwurst, 1x Nutella)
  w.neuerArtikel("Leberwurst", 1.95, 2);
  w.neuerArtikel("Nutella", 3.95, 1);

  //Kunde1 bestellt eine Leberwurst und ein Nutella
  String[] bestellung1 = {"Leberwurst", "Nutella"};
  String rechnung1 = w.bestellen(kunde1,bestellung1);

  //Rechnung Kunde1 ausgeben
  System.out.println(rechnung1);
```

```java
  //Kunde2 bestellt eine Leberwurst, ein Nutella und eine Butter
  String[] bestellung2 = {"Leberwurst", "Nutella", "Butter"};
  String rechnung2 = w.bestellen(kunde2, bestellung2);

  //Rechnung Kunde2 ausgeben
  System.out.println(rechnung2);
}
```

soll die Ausgabe (also die beiden erstellten Rechnungen hintereinander) folgendermaßen aussehen:

```
Rechnung fuer Helga Herrlich:
Leberwurst: 1.95
Nutella: 3.95
Gesamtpreis: 5.9

Rechnung fuer unseren guten Kunden Werner Winzig, Preisnachlass 5.0%:
Leberwurst: 1.8524999999999998
Nutella: nicht mehr vorhanden
Butter: nicht gefunden
Gesamtpreis: 1.8524999999999998
```

Info

- Polymorphismus
- instanceof
- Explizite Downcasts und implizite Upcasts
- Überladen und Überschreiben von Methoden
- Vergeben Sie in allen Klassen sinnvoll Zugriffsrechte für Instanz- und Klassenvariablen sowie Methoden.

Aufgabe 8: Uhren 👾

Wir modellieren Uhren in Software.

Eine Uhr (in der Klasse `Uhr` beschrieben) misst Stunden, Minuten und Sekunden. Über eine Methode `public void tick()` soll mit jedem Aufruf dieser Methode eine Sekunde auf dieser Uhr weitergezählt werden. Über eine Methode `public String toString()` soll die aktuelle Uhrzeit auf dieser Uhr als String geliefert werden im Format Stunde:Minute:Sekunde (jeweils ohne führende Nullen). Es soll zwei Möglichkeiten geben, wie man eine Uhr erzeugt: Ohne Angabe von Argumenten soll die Uhr auf 0:0:0 voreingestellt werden. Und als weitere Möglichkeit mit Angabe der Stunde, Minute und Sekunde (jeweils als int-Wert im erlaubten Bereich), auf die diese Uhr dann voreingestellt sein soll.

Überlegen Sie sich, was Instanzvariablen in ihrer Modellierung sind, ob und welche Konstruktoren Sie benötigen, welche Instanzmethoden vorgegeben sind / Sie benötigen und wie Sie die geforderte Programmlogik umsetzen.

Weiterhin gibt es Sportuhren (Klasse `Sportuhr`). Eine Sportuhr ist eine Uhr, die aber weiterhin die Herzfrequenz messen kann und eine Wechselanzeige hat, die wahlweise die Uhrzeit oder die Herzfrequenz anzeigt. Die Herzfrequenz wird gesetzt über eine Methode `public void setHerzfrequenz(int wert)` und der Wechsel in der Anzeige (bzw. was bei uns die Instanzmethode toString liefert) wird über die Methode `public void wechselAnzeige()` bewirkt. Mit jedem Aufruf dieser Methode wird die Anzeige zwischen Uhrzeit/Herzfrequenz gewechselt. Zu Beginn ist die Anzeige auf Uhrzeit eingestellt. Bei der Darstellung der Uhrzeit bzw. Herzfrequenz soll auch vorher im String angezeigt werden, welchen Wert die Anzeige gerade darstellt (Uhrzeit oder Herzfrequenz; siehe Beispielausgabe unten). Eine Sportuhr kann nur erzeugt mit der Startuhrzeit 0:0:0.

Überlegen Sie sich, wie Sie aufbauend auf der Uhr-Klasse Sportuhren modellieren können. Was sind die Unterschiede zu einer normalen Uhr? Welche Funktionalität benötigt eine Sportuhr-Klasse zusätzlich oder anders als eine Uhr-Klasse? Was lässt sich aus der Uhr-Klasse nutzen? Wie setzen Sie ihre Ideen in Java um? Nutzen Sie folgendes Testprogramm

```
/**
 * Test von Uhren
 * @author Rudolf Berrendorf
 * @version 1.0
 */
public class Test{
  public static void main(String[] args){
    Uhr u = new Uhr(9, 59, 59);
    System.out.println(u);
    u.tick();
    System.out.println(u);
    Sportuhr s = new Sportuhr();
    s.tick();
    System.out.println(s);
    s.setHerzfrequenz(91);
    s.wechselAnzeige();
    System.out.println(s);
  }
}
```

was folgende Ausgabe produzieren soll:

Beispiel

```
9:59:59
10:0:0
Uhrzeit: 0:0:1
Herzfrequenz: 91
```

10-40 80-300