Kontrollfragen A

1. Welchen Hauptvorteil und welchen Hauptnachteil bringt uns das Konzept der Vererbung?

   Vorteil: Wiederverwendung (Vermeidung von Code-Duplikation)

   Nachteil: starke Kopplung

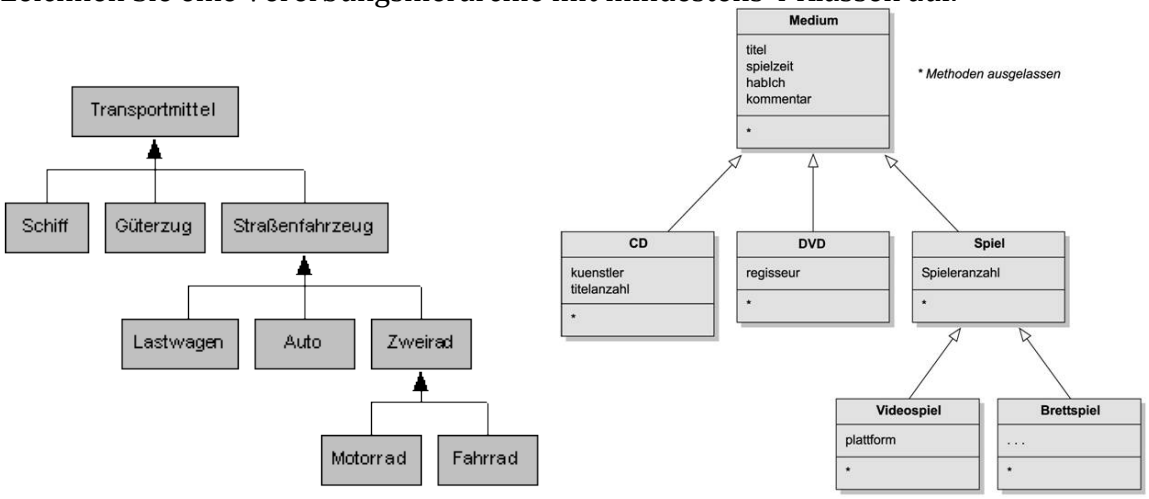
2. Java kennt die Basisklasse `Object`. Inwiefern macht dies Sinn?

   Alle Klassen erben automatisch von der Basisklasse Object. Dadurch kann man jedes beliebige Objekt in Java als Object ansprechen.

   Die Klasse Object definiert elementare Methoden, die für alle Objekt nützlich sind. (Diese Methoden müssen i.d.R. von abgeleiteten Klassen überschrieben werden.)

   Bsp.: .equals(), .toString()

3. Zeichnen Sie eine Vererbungshierarchie mit mindestens 4 Klassen auf.



4. Illustrieren Sie anhand Ihrer Klassenhierarchie
   - Substitution
   - Casting
   - Casting mit Compilerfehler
   - Casting mit Laufzeitfehler

   Schreiben Sie 4 entsprechende Code-Zeilen auf.

Substitution:	*Spiel meinSpiel = new Brettspiel();*
Casting:	*Brettspiel meinSchach = (Brettspiel) meinSpiel;*
Casting mit Compilerfehler:	*Videospiel meinVideoSpiel = (Videospiel) meinSchach;*
Casting mit Laufzeitfehler:	*Videospiel meinVideoSpiel = (Videospiel) meinSpiel;*

Kontrollfragen B

1. Welches Konzept steckt hinter der "schwachen Polymorphie"?
   Überladen / Overloading
   Mehrere Methoden mit gleichem Namen und unterschiedlicher Signatur in einer Klasse.

2. Welches Konzept steckt hinter der "starken Polymorphie"?
   Überschreiben / Overriding
   @Override

3. Was ist aus Ihrer Sicht der Hauptvorteil der Polymorphie?
   Polymorphie führt zu einfacherem und flexiblerem Programmcode. Sie ermöglicht die Handhabung von Objekten unterschiedlicher Klassen auf einer allgemeineren Ebene.

4. Das Überladen kann man bei Methoden anwenden. Wo noch?
   Konstruktoren

5. Wir haben "dynamisches Binden" kennen gelernt. Wieso spricht man von *dynamisch* und was wird gebunden?
   Dynamisch bedeutet auf Ebene Ausführung bzw. zur Laufzeit. (Statisch bedeutet auf Ebene Source-Code bzw. zur Compilationszeit.)

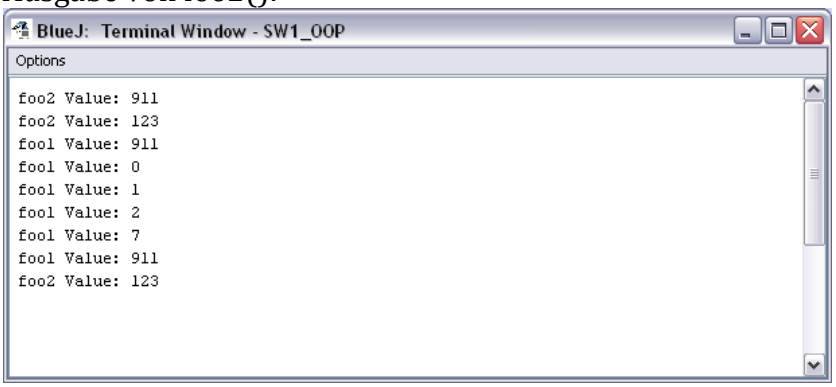
```
k = new Konto();
k = new Spar();
k = new Giro();
// k besitzt nacheinander die dynamischen Datentypen Konto, Spar und Giro.
```

6. Weshalb überschreibt man häufig `toString()`?
   Die Standard Ausgabe ist nicht immer sinnvoll.

7. Wozu alles dient das Schlüsselwort `super`?
   super() *Konstruktor-Aufruf der Oberklasse*
   super.method() *Methoden-Aufruf der Oberklasse*

Kontrollfragen C

1. Ausgabe von foo2():



Erklären Sie …

Die Variable a besitzt je nach Scope / Methode einen anderen Wert

Zugriff von: / Modifizierer:	eigener Klasse	Klasse in gleichem Package	Unterklasse in anderem Package	Nicht Unterklasse in anderem Package
private	ja	nein	nein	nein
- Standard bzw. Package Scope	ja	ja	nein	nein
protected	ja	ja	ja*	nein
public	ja	ja	ja*	ja*

(*) Die Klasse muss dann natürlich importiert sein.

```
1  public class Scope
2  {
3   // private int b = a;   // Würde nicht compiliert werden!
4      private int a = 911; // Für nachfolgende Dekl. sichtbar!
5
6      public void foo2()   // foo2() ist auch ausserhalb sichtbar!
7      {
8          System.out.println("foo2 Value: " + a);
9          int a = 123;
10         System.out.println("foo2 Value: " + a);
11         foo1();
12         System.out.println("foo2 Value: " + a);
13     }
14
15     private void foo1()  // foo1() ist in ganzer Klasse sichtbar!
16     {
17         System.out.println("foo1 Value: " + a);
18         int a = 7;
19         for (int b = 0; b < 3; b++) {
20             System.out.println("foo1 Value: " + b);
21         }
22         System.out.println("foo1 Value: " + a);
23         System.out.println("foo1 Value: " + this.a);
24      }
25 }
```

Value: 911 *Klassenvariable (Zeile 4)*
Value: 123 *Lokale Variable in foo2() (Zeile 9)*
Value: 911 *Klassenvariable (Zeile 4)*
Value: 0 *Lokale Variable in foo1() (Zeile 19)*
Value: 1 *Lokale Variable in foo1() (Zeile 19)*
Value: 2 *Lokale Variable in foo1() (Zeile 19)*
Value: 7 *Lokale Variable in foo1() (Zeile 18 – überschreiben der Variable von Zeile 4)*
Value: 911 *Klassenvariable (Zeile 4) – (da der Aufruf mittels this.a gemacht wird)*
Value: 123 *Lokale Variable in foo2() (Zeile 9) – diese wurde nicht durch foo1() geändert*