Prof. Dr. M. Müller C. Aschermann, J. Hensel, J. Protze, P. Reble

Tutoraufgabe 1 (Ueberladen):

Betrachten Sie die folgenden Klassen:

Listing 1: X.java

```
1  public class X {
2      public int a = 23;
3      public X (int a) {                          // Signatur: X(I)
4          super();
5          this.a = a;
6      }
7
8      public X (float x) {                        // Signatur: X(F)
9          this((int) (x + 1));
10     }
11
12     public void f(int  i, X o) { }              // Signatur: X.f(ILX;)
13     public void f(long i, Y o) { }              // Signatur: X.f(JLY;)
14     public void f(long i, X o) { }              // Signatur: X.f(JLX;)
15 }
```

Listing 2: Y.java

```
1  public class Y extends X {
2      public float a = 42;
3      public Y (double a) {                       // Signatur: Y(D)
4          this((float) (a - 1));
5      }
6      public Y (float a) {                        // Signatur: Y(F)
7          super(a);
8          this.a = a;
9      }
10     public void f(int i, X o) { }               // Signatur: Y.f(ILX;)
11     public void f(int i, Y o) { }               // Signatur: Y.f(ILY;)
12     public void f(long i, X o) { }              // Signatur: Y.f(JLX;)
13 }
```

Listing 3: Z.java

```
1  public class Z {
2      public static void main (String [] args) {
3                                                  // a)
4          X xx1 = new X(42);                      // (1)
5          System.out.println("X.a: " + xx1.a);
6          X xx2 = new X(22.99f);                  // (2)
7          System.out.println("X.a: " + xx2.a);
8          X xy = new Y(7.5);                      // (3)
9          System.out.println("X.a: " + ((X) xy).a);
10         System.out.println("Y.a: " + ((Y) xy).a);
11         Y yy = new Y(7);                        // (4)
12         System.out.println("X.a: " + ((X) yy).a);
13         System.out.println("Y.a: " + ((Y) yy).a);
14                                                 // b)
15         int i = 1;
16         long l = 2;
17         xx1.f(i, xy);                           // (1)
18         xx1.f(l, xx1);                          // (2)
19         xx1.f(l, yy);                           // (3)
20         yy.f(i, yy);                            // (4)
21         yy.f(l, xy);                            // (5)
22         xy.f(i, xx1);                           // (6)
23         xy.f(l, xy);                            // (7)
24         xy.f(l, yy);                            // (8)
25     }
26 }
```

In dieser Aufgabe sollen Sie angeben, welche Methoden- und Konstruktoraufrufe stattfinden. Verwenden Sie hierzu keinen Computer, sondern nur die aus der Vorlesung bekannten Angaben zum Verhalten von Java. Verwenden Sie zur eindeutigen Bezeichnung die Funktionssignatur, die jeweils als Kommentar hinter jeder Funktionsdefinition steht. Begründen Sie Ihre Antwort kurz.

a) Geben Sie für die mit `(1)-(4)` markierten Konstruktoraufrufe in der Klasse `Z` jeweils an, welche Konstruktoren in welcher Reihenfolge von Java aufgerufen werden. Bedenken Sie, dass die Oberklasse von `X` die Klasse `Object` ist. Sie brauchen allerdings nur angeben, wann (und ob) ein Konstruktor von `Object` aufgerufen wird, ohne auf dadurch eventuell weiter ausgelöste Aufrufe einzugehen. Erklären Sie außerdem,

welche Attribute mit welchen Werten belegt werden und welche Werte durch die `println`-Anweisungen ausgegeben werden.

b) Geben Sie für die mit `(1)-(8)` markierten Aufrufe der Methode `f` in der Klasse `Z` jeweils an, welche Variante der Funktion von Java verwendet wird. Geben Sie hierzu die jeweilige Signatur an.

Lösung:

a)

a) `X(I)` durch expliziten Konstruktoraufruf, dann

`Object()` durch explizites `super()` in `X(I)`.

Das Attribut `a` wird durch den Konstruktor auf den übergegebenen Wert `42` gesetzt, daher wird `X.a: 42` ausgegeben.

b) `X(F)` durch expliziten Konstruktoraufruf, dann

`X(I)` durch expliziten Aufruf durch `this((int) ...)`, dann

`Object()` durch explizites `super()` in `X(I)`.

Das Attribut `a` wird auf den Wert `(int) (22.99f + 1) = (int) 23.99f = 23` gesetzt, daher wird `X.a: 23` ausgegeben.

c) `Y(D)` durch expliziten Konstruktoraufruf, dann

`Y(F)` durch expliziten Aufruf `this((float) ...)`, dann

`X(F)` durch expliziten Aufruf `super(a)`, dann

`X(I)` durch expliziten Aufruf durch `this((int) ...)`, dann

`Object()` durch explizites `super()` in `X(I)`.

Das Attribut `X.a` wird auf den Wert `(int) (((float) (7.5 - 1)) + 1) = (int) (((float) 6.5) + 1) = (int) (6.5 + 1) = (int) 7.5 = 7` gesetzt.

Das Attribut `Y.a` wird auf den Wert `(float) (7.5 - 1) = (float) (6.5) = 6.5` gesetzt.

Es wird daher zuerst `X.a: 7` und dann `Y.a = 6.5` ausgegeben.

d) `Y(F)` durch Aufruf von `Y(7)`, wobei `7` implizit von `int` nach `float` konvertiert wird, dann

`X(F)` durch expliziten Aufruf `super(a)`, dann

`X(I)` durch expliziten Aufruf durch `this((int) ...)`, dann

`Object()` durch explizites `super()` in `X(I)`.

Das Attribut `X.a` wird auf den Wert `(int) (7.0 + 1) = (int) 8.0 = 8` gesetzt.

Das Attribut `Y.a` wird auf den Wert `7.0` gesetzt. Es wird daher zuerst `X.a: 8` und dann `Y.a = 7.0` ausgegeben.

b)

a) `xx1.f(i, xy)`
Die Variable `xx1` hat den Typ `X` und die Variable `xy` ist als `X` deklariert. Daher ruft Java die (genau passende) Methode `X.f(ILX;)` auf.

b) `xx1.f(l, xx1)`
Die Variable `xx1` ist als `X` deklariert. Daher ruft Java die (genau passende) Methode `X.f(JLX;)` auf.

c) `xx1.f(l, yy)`
Die Variable `yy` ist als `Y` deklariert. Daher ruft Java die (genau passende) Methode `X.f(JLY;)` auf.

d) `yy.f(i, yy)`
Die Variable `yy` ist als `Y` deklariert. Daher ruft Java die (genau passende) Methode `Y.f(ILY;)` auf.

e) `yy.f(l, xy)`
Die Variable `xy` ist als `X` deklariert (dass hier eine Instanz vom Typ `Y` referenziert wird, spielt dabei keine Rolle!). Daher wird von Java die (genau passende) Methode `Y.f(JLX;)` aufgerufen.

f) `xy.f(i, xx1)`
Die Variablen `xy` und `xx1` sind als `X` deklariert. Daher wird zur Compile-Zeit festgelegt, dass die Methode mit der Signatur `(ILX;)` aufgerufen wird. Zur Laufzeit wird dann aber die Implementierung in `Y` verwendet, da `xy` ein Objekt vom Typ `Y` referenziert. Es wird also `Y.f(ILX;)` aufgerufen.

g) `xy.f(l, xy)`
Die Variable `xy` ist als `X` deklariert. Daher wird zur Compile-Zeit festgelegt, dass die Methode mit der Signatur `(JLX;)` aufgerufen wird (hierbei spielt keine Rolle, dass `xy` zur Laufzeit eine Instanz vom Typ `Y` referenziert). Zur Laufzeit wird dann aber die Implementierung in `Y` verwendet, da `xy` ein Objekt vom Typ `Y` referenziert. Es wird also `Y.f(JLX;)` aufgerufen.

h) `xy.f(l, yy)`
Die Variable `xy` ist als `X` deklariert, die Variable `yy` als `Y`. Daher wird zur Compile-Zeit festgelegt, dass die Methode mit der Signatur `(JLY;)` aufgerufen wird (hierbei spielt keine Rolle, dass `xy` zur Laufzeit eine Instanz vom Typ `Y` referenziert). Da diese in `Y` nicht überschrieben wird, wird also `X.f(JLY;)` aufgerufen.

Aufgabe 2 (Ueberladen): (12 Punkte)

Betrachten Sie die folgenden Klassen:

```
public class A {
    public static int y = 0;
    public int x = 23;
    public A () {  this(1); }                          // Signatur: A()
    public A (int x) {                                 // Signatur: A(I)
        super();
        y += x;
    }
    public void f(int  i, A o) { }                     // Signatur: A.f(ILA;)
    public void f(long i, B o) { }                     // Signatur: A.f(JLB;)
    public void f(int  i, B o) { }                     // Signatur: A.f(ILB;)
}

public class B extends A {
    int x = 17;
    public B () {                                      // Signatur: B()
      this(8);
      x++;
    }
    public B (int x) {                                 // Signatur: B(I)
        y = x;
        super.x = x;
    }
    public void f(int  i, A o) { }                     // Signatur: B.f(ILA;)
    public void f(long i, B o) { }                     // Signatur: B.f(JLB;)
    public void f(long i, A o) { }                     // Signatur: B.f(JLA;)
}

public class C {
    public static void main (String [] args) {         // a)
        A aa1 = new A();                               // (1)
        System.out.println (aa1.x);
        System.out.println (A.y);
        A aa2 = new A(42);                             // (2)
        System.out.println (aa2.y);
        B bb = new B();                                // (3)
```

```
        System.out.println (bb.x);
        System.out.println (((A) bb).x);
        System.out.println (bb.y);
        A ab = new B(3);                              // (4)
        System.out.println (ab.x);
        System.out.println (((B) ab).x);
        System.out.println (A.y);
        int i = 1; long l = 2;                        // b)
        aa1.f(i, ab);                                 // (1)
        aa1.f(l, bb);                                 // (2)
        aa1.f(i, bb);                                 // (3)
        bb.f(i, bb);                                  // (4)
        bb.f(l, ab);                                  // (5)
        ab.f(i, aa1);                                 // (6)
        ab.f(i, bb);                                  // (7)
        ab.f(l, bb);                                  // (8)
    }
}
```

In dieser Aufgabe sollen Sie angeben, welche Methoden- und Konstruktoraufrufe stattfinden. Verwenden Sie hierzu keinen Computer, sondern nur die aus der Vorlesung bekannten Angaben zum Verhalten von Java. Verwenden Sie zur eindeutigen Bezeichnung die Funktionssignatur, die jeweils als Kommentar hinter jeder Funktionsdefinition steht. Begründen Sie Ihre Antwort kurz.

a) Geben Sie für die mit `(1)-(4)` markierten Konstruktoraufrufe in der Klasse `C` jeweils an, welche Konstruktoren in welcher Reihenfolge von Java aufgerufen werden. Bedenken Sie, dass die Oberklasse von `A` die Klasse `Object` ist. Sie brauchen allerdings nur angeben, wann (und ob) ein Konstruktor von `Object` aufgerufen wird, ohne auf dadurch eventuell weiter ausgelöste Aufrufe einzugehen. Erklären Sie außerdem, welche Attribute mit welchen Werten belegt werden und welche Werte durch die `println`-Anweisungen ausgegeben werden.

b) Geben Sie für die mit `(1)-(8)` markierten Aufrufe der Methode `f` in der Klasse `C` jeweils an, welche Variante der Funktion von Java verwendet wird. Geben Sie hierzu die jeweilige Signatur an.

Lösung:

a)

a) `A()` durch expliziten Konstruktoraufruf, dann `A(I)` durch expliziten Aufruf von `this(1)`. Dann `Object()` durch explizites `super()` in `A(I)`. Das Attribut `x` wird durch den Konstruktor nicht gesetzt. Der Wert des statischen Attributs `y` wird um `1` erhöht. Somit wird der initiale Wert von `x`, also `23`, ausgegeben.

b) `A(I)` durch expliziten Konstruktoraufruf, dann `Object()` durch expliziten Aufruf des Elternkonstruktors. Im ersten Konstruktoraufruf `A(I)` wird der Wert des statischen Attributs `y` der Klasse `A` um `42` erhöht, ist also jetzt `43`. Der Wert des Attributs `x` wird nicht verändert. Somit wird `43` für `y` ausgegeben.

c) `B()` durch expliziten Konstruktoraufruf, dann `B(I)` durch expliziten Aufruf `this(8)`. Darin wird zuerst der Konstruktor `A()` durch impliziten Aufruf des Elternkonstruktors aufgerufen. Dieser ruft durch expliziten Aufruf `this(1)` den Konstruktor `A(I)` auf. In `A(I)` wird zuerst durch expliziten Aufruf der Elternkonstruktor `Object()` aufgerufen. Anschließend wird der Wert des statischen Attributs `y` um `1` auf `44` erhöht. Nun wird der Konstruktor `B(I)` weiter ausgewertet, welcher das statische Attribut `y` in der Klasse `A` auf `8` setzt. Außerdem wird der Wert des Attributs `x` in der Klasse `A` ebenfalls auf `8` gesetzt. Der Wert des Attributs `x` in der Klasse `B` ist davon nicht betroffen. Anschließend wird der verbleibende Code in `B()` ausgeführt, der den Wert des Attributs `x` der Klasse `B` von `17` auf `18` erhöht.

Da die Variable `bb` den Typ `B` hat, bezeichnet der Ausdruck `bb.x` das Feld `x` in der Klasse `B`. Somit gibt die erste `println`-Anweisung den Wert `18` aus. Die zweite Anweisung gibt `8` aus, was der Wert des Attributs `x` in der Klasse `A` ist. Die Ausgabe des Attributs `y` ergibt `8`.

d) `B(I)` durch expliziten Konstruktoraufruf, dann wird zuerst der Konstruktor `A()` durch impliziten Aufruf des Elternkonstruktors aufgerufen. Dieser ruft durch expliziten Aufruf `this(1)` den Konstruktor `A(I)` auf. In `A(I)` wird zuerst durch expliziten Aufruf der Elternkonstruktor `Object()` aufgerufen. Anschließend wird der Wert des statischen Attributs `y` um `1` auf `9` erhöht. Anschließend wird `B(I)` weiter ausgewertet. Hierbei wird das statische Attribut `y` auf `3` gesetzt, außerdem wird der Wert des Attributs `x` in der Klasse `A` ebenfalls auf `3` gesetzt.

Da das initialisierte Objekt `ab` den Typ `A` hat, bezeichnet der Ausdruck `ab.x` das Attribut `x` in der Klasse `A`. Dieses hat den Wert `3`, der auch ausgegeben wird. Die zweite Ausgabe ergibt `17`, da hiermit das Attribut `x` in der Klasse `B` gemeint ist und dieses nicht verändert wurde. Die Ausgabe des Attributs `y` ergibt ebenfalls `3`.

b) a) `aa1.f(i, ab)`
Die Variable `aa1` hat den Typ `A` und die Variable `ab` den Typ `A`. Somit ruft `Java` die Methode `A.f(ILA;)` auf.

b) `aa1.f(l, bb)`
Die Methode `A.f(JLB;)` passt genau zu dem Aufruf und wird daher ausgeführt.

c) `aa1.f(i, bb)`
Die Variable `bb` ist vom Typ `B`. Da `A` die Oberklasse von `B` ist, kann `bb` nur Objekte enthalten, die auch Instanzen von `A` sind. Die beiden Methoden `A.f(ILA;)` und `A.f(ILB;)` kommen deswegen in Frage. Letztere passt besser und wird aufgerufen.

d) `bb.f(i, bb)`
Die Variable `bb` ist vom Typ `B`. Die passendste Methode für diesen Aufruf hat die Signatur `f(ILB;)` und ist in der Klasse `A` deklariert. Aufgerufen wird daher `A.f(ILB;)`.

e) `bb.f(l, ab)`
Die Variable `ab` wurde mit Typ `A` deklariert. Somit passt die Methode `B.f(JLA;)` genau zu dem Aufruf und wird ausgeführt.

f) `ab.f(i, aa1)`
Die Variable `ab` ist vom Typ `A`, referenziert allerdings ein Objekt, das Instanz der Klasse `B` ist. Zur Compile-Zeit wird entsprechend des deklarierten Typs von `ab` die Methode `A.f(ILA;)` ausgewählt, da sie genau zum Aufruf passt. `ab` ist ein Objekt vom Typ `B`, und die Methode wird in `B` überschrieben. Somit wird `B.f(ILA;)` aufgerufen.

g) `ab.f(i, bb)`
Die Variable `ab` ist vom Typ `A`, enhält allerdings ein Objekt, das Instanz der Klasse `B` ist. Zur Compile-Zeit wird entsprechend des deklarierten Typs `A` von `ab` eine Methode in `A` ausgewählt. Hier passt `A.f(ILB;)` genau zum Aufruf. Zur Laufzeit wird nun beachtet, dass `ab` auf eine Instanz von `B` zeigt, wobei `B` die Methode `A.f(ILB;)` nicht überschreibt. Somit wird `A.f(ILB;)` aufgerufen.

h) `ab.f(l, bb)`
Die Variable `ab` ist vom Typ `A`, enhält allerdings ein Objekt, das Instanz der Klasse `B` ist. Zur Compile-Zeit wird entsprechend des deklarierten Typs `A` von `ab` eine Methode in `A` ausgewählt. Hier passt `A.f(JLB;)` genau zum Aufruf. Zur Laufzeit wird nun beachtet, dass `ab` auf eine Instanz von `B` zeigt, wobei `B` die Methode `A.f(JLB;)` überschreibt. Somit wird `B.f(JLB;)` aufgerufen.

Tutoraufgabe 3 (Verbrecher Jagd):

In einer kleinen Stadt kommt es gehäuft zu Diebstählen. Sperren Sie die Diebe weg! Verwenden Sie hierbei die Hilfsklasse `Zufall` (auf der Homepage), die zwei statische Methoden `int zahl(int)` und `String name()` enthält. Ein Aufruf `Zufall.zahl(i)` (für `i` größer 0) gibt eine zufällige Zahl zwischen 0 und $\texttt{i} - 1$ zurück. Ein Aufruf `Zufall.name()` gibt einen zufällig gewählten Namen zurück.

a) Schreiben Sie das Interface `Einwohner`, welches die Interaktion der Bürger der Stadt modelliert. Ein Einwohner hat einen Namen. Da sich der Name eines Einwohners nicht ändern soll, definieren Sie nur eine `get`-Methode, aber keine `set`-Methode. Jeder Einwohner besitzt eine bestimmte Menge Geld, gemessen in Euro. Definieren sie `setEigentum`- und `getEigentum`-Methoden für den Geldbetrag. Weiterhin hat

jeder Einwohner die Methode `hatDiebesgut`, welche angibt, ob ein Einwohner erfolgreiche Diebeszüge begangen hat (soll per Default `false` zurückgeben). Ein Einwohner kann eine Aktion ausführen. Dies geschieht durch die Methode `void aktion(Einwohner [] einwohner)`. Bei dem Aufruf dieser Methode soll per Default ausgegeben werden, dass der Einwohner spazieren geht. Das Argument `einwohner` wird dabei nicht benötigt, Sie können aber immer davon ausgehen, dass es keine `null`-Werte enthält.

b) Schreiben Sie die Klasse `Katze`, welche die verschiedenen Katzen der Stadt modelliert und das Interface `Einwohner` implementiert. Die Methode `aktion` muss dabei nicht verändert werden. Das Eigentum einer Katze ist immer 0 und der Name einer Katze ist immer "Kitty". Die Methode `setEigentum` einer Katze hat keinen Einfluss.

c) Schreiben Sie die Klasse `Buerger`, welche die "normalen" Bürger der Stadt modelliert und das Interface `Einwohner` implementiert. Die Methode `aktion` muss dabei nicht verändert werden. Das Eigentum und der Name des Bürgers sollten im Konstruktor zufällig gesetzt werden. Das Eigentum sollte dabei zwischen 0 und 1000 Euro liegen.

d) Ein Dieb ist ebenfalls ein Bürger und wird durch die Klasse `Dieb` repräsentiert. Ein Dieb hat ein Attribut `int diebesgut`, welches durch den Konstruktor mit 0 initialisiert wird. Die Methode `hatDiebesgut` soll zurückgeben, ob dieser Betrag größer als 0 ist. Bei einem Aufruf der Methode `void aktion(Einwohner [] einwohner)` hat der Dieb 5 Versuche, um Bürger zu bestehlen. In jedem Versuch soll ein Bürger aus dem Array `einwohner` zufällig ausgewählt werden. Ist es ein reicher Bürger, der mindestens 10 Euro besitzt, so klaut der Dieb ihm einen zufälligen Teil seines Eigentum (aber nicht seinen letzten Euro). Hierbei wird das Eigentum des Bürgers um den Betrag verringert, durch den sich das Attribut `diebesgut` des Diebes erhöht. Trifft der Dieb bei den Versuchen auf einen Polizisten, so bricht er die Aktion ab (die restlichen Versuche verfallen).

e) Ein Gefangener ist ein Dieb, der kein Diebesgut besitzt und im Gefängnis sitzt. Schreiben Sie die Klasse `Gefangener`. Aktionen von Gefangenen bestehen nur daraus, dass sich die Gefangenen im Gefängnis ärgern.

f) Ein Polizist ist ein Bürger, der Verbrecher jagt. Bei einem Aufruf der Methode `void aktion(Einwohner [] einwohner)` sucht der Polizist bei den Bürgern im Array `einwohner` nach Diebesgut. Hierzu durchläuft er das Array von vorne nach hinten und verwendet die Methode `hatDiebesgut` der Bürger. Findet der Polizist Diebesgut, so ersetzt er den Dieb an der Stelle im Array durch einen Gefangenen gleichen Namens.

g) Erstellen Sie die Klasse `Stadt`, welche das als `private` markierte Attribut `Einwohner[] einwohner` besitzt. Der Konstruktor dieser Klasse hat das Argument `int einwohnerzahl` und legt ein Array mit entsprechend vielen Bürgern an. Verwenden Sie die Methode `Zufall.zahl()` so, dass es jeweils etwa 25% Diebe, Polizisten, Katzen und normale Bürger gibt. In der statischen `main`-Methode der Klasse soll eine neue Stadt mit 10 Bürgern erstellt werden. Danach wird 10 mal ein zufälliger Bürger ausgewählt und seine Methode `aktion` mit allen Bürgern der Stadt aufgerufen.

Eine Lauf des Programms könnte beispielsweise die folgende Ausgabe erzeugen:

```
Polizist Luisa geht auf Verbrecherjagd
Dieb Christina sucht nach Diebesgut
Dieb Christina klaut Christina 51 Euro
Dieb Christina klaut Janna 16 Euro
Dieb Christina bricht den Beutezug ab
Polizist Clara geht auf Verbrecherjagd
Polizist Clara entlarvt Dieb Christina
Dieb Paul sucht nach Diebesgut
Dieb Paul klaut Christina 131 Euro
Dieb Paul klaut Frederike 79 Euro
Dieb Paul bricht den Beutezug ab
Gefangener Christina sitzt im Gefaengnis und aergert sich
Dieb Pia sucht nach Diebesgut
Dieb Pia klaut Paul 75 Euro
Dieb Pia bricht den Beutezug ab
```

```
Einwohner Frederike geht spazieren.
Gefangener Christina sizt im Gefaengnis und aergert sich
```

Lösung:

Listing 4: Einwohner.java

```
1  public interface Einwohner{
2    public String getName();
3    public int getEigentum();
4    public void setEigentum(int geld);
5    default public boolean hatDiebesgut(){
6      return false;
7    }
8    default public void aktion(Einwohner[] einwohner){
9      System.out.println("Einwohner "+getName()+" geht spazieren.");
10   }
11 }
```

Listing 5: Katze.java

```
1  public class Katze implements Einwohner{
2    public int getEigentum(){
3      return 0;
4    }
5    public void setEigentum(int e){}
6    public String getName(){
7      return "Kitty";
8    }
9  }
```

Listing 6: Buerger.java

```
1  public class Buerger implements Einwohner{
2    protected String name;
3    private int eigentum;
4
5    public void setEigentum(int arg){
6      eigentum = arg;
7    }
8    public int getEigentum(){
9      return eigentum;
10   }
11   public String getName(){
12     return name;
13   }
14   public Buerger(){
15     eigentum = Zufall.zahl(1000);
16     name = Zufall.name();
17   }
18 }
```

Listing 7: Dieb.java

```
1  public class Dieb extends Buerger {
2    private int diebesgut;
3    public Dieb(){
4      super();
5      diebesgut = 0;
6    }
7
8    public boolean hatDiebesgut(){
9      return diebesgut > 0;
10   }
11
12   public void aktion(Einwohner[] einwohner){
13     System.out.println("Dieb "+getName()+" sucht nach Diebesgut");
14     for(int i=0;i<5; i++){
15       int bestohlener_index = Zufall.zahl(einwohner.length);
16       Einwohner bestohlener_einwohner = einwohner[bestohlener_index];
17       if(bestohlener_einwohner instanceof Polizist){
18         System.out.println("Dieb "+getName()+" bricht den Beutezug ab");
19         return;
20       }
21       if(bestohlener_einwohner.getEigentum() >= 10){
22         int diebes_menge = Zufall.zahl(bestohlener_einwohner.getEigentum());
23         diebesgut += diebes_menge;
```

```
24          System.out.println("Dieb "+getName()+" klaut "+bestohlener_einwohner.getName()+ " "+ diebes_menge+" Euro");
25          bestohlener_einwohner.setEigentum(bestohlener_einwohner.getEigentum()-diebes_menge);
26        }
27      }
28    }
29  }
```

Listing 8: Gefangener.java

```
1  public class Gefangener extends Dieb{
2    public Gefangener(Einwohner frei){
3      this.name = frei.getName();
4    }
5    public void aktion(Einwohner[] einwohner){
6      System.out.println("Gefangener "+getName()+" sitzt im Gefaengnis und aergert sich");
7    }
8  }
```

Listing 9: Polizist.java

```
1  public class Polizist extends Buerger {
2    public void aktion(Einwohner[] einwohner){
3      System.out.println("Polizist "+getName()+" geht auf Verbrecherjagd");
4      for(int i=0; i < einwohner.length ; i++){
5        Einwohner kontrollierter_einwohner = einwohner[i];
6        if(kontrollierter_einwohner.hatDiebesgut()){
7          System.out.println("Polizist "+getName()+" entlarvt Dieb "+kontrollierter_einwohner.getName());
8          einwohner[i] = new Gefangener(kontrollierter_einwohner);
9        }
10     }
11   }
12 }
```

Listing 10: Stadt.java

```
1  public class Stadt {
2    private Einwohner[] buerger;
3    public Stadt(int einwohnerzahl){
4      buerger = new Einwohner[einwohnerzahl];
5      for(int i = 0; i<einwohnerzahl; i++){
6        int rnd = Zufall.zahl(100);
7        if(rnd < 25){
8          buerger[i] = new Buerger();
9          continue;
10       }
11       if(rnd < 50){
12         buerger[i] = new Katze();
13         continue;
14       }
15       if(rnd < 75){
16         buerger[i] = new Dieb();
17         continue;
18       }
19       buerger[i] = new Polizist();
20     }
21   }
22
23   public static void main(String[] args){
24     Stadt s = new Stadt(10);
25     for(int i = 0; i<10 ; i++){
26       int j = Zufall.zahl(10);
27       s.buerger[j].aktion(s.buerger);
28     }
29   }
30 }
```

Aufgabe 4 (Verkehrsimulation): (2 + 2 + 2 + 2 + 2 + 2 = 12 Punkte)

In einer kleinen Stadt kommt es gehäuft zu Unfällen. Simulieren Sie den Verkehr! Verwenden Sie hierbei die Hilfsklasse `Zufall` (auf der Homepage), die drei statischen Methoden `int zahl(int)`, `String name()` und `String modell()` enthält. Ein Aufruf `Zufall.zahl(i)` (für `i` größer 0) gibt eine zufällige Zahl zwischen 0 und `i` − 1 zurück. Ein Aufruf `Zufall.name()` gibt einen zufällig ausgewählten Namen zurück. Ein Aufruf `Zufall.modell()` gibt ein zufällig ausgewähltes Automodell zurück. Es werden an einigen Stellen Listen benötigt, Sie können entweder die bereitgestellte Liste Klasse verwenden, die ähnlich zur in der Vorlesung

behandlten Liste ist, oder die List-Collection verwenden. Die Werte der Liste sind von der Klasse Verkehrsteilnehmer. *Hinweis:* Beachten Sie die neu hinzugekommenen Funktionen `size()`, `add()`, `get(i)`, `remove(v)` und `clear()`.

a) Schreiben Sie das Interface `Verkehrsteilnehmer`, welches die Interaktion der Teilnehmer am Straßenverkehr modelliert. Jeder Teilnehmer hat eine eigene Höchstgeschwindigkeit `maxSpeed` und einen Namen `name`. Da sich Höchstgeschwindigkeit und Name nicht ändern sollen, definieren Sie jeweils nur die `get`-Methode, aber keine `set`-Methode. Definieren Sie ausserdem eine `getSpeed`-Methode, die als default den `getMaxSpeed` liefert. Jeder Verkehrsteilnehmer kann eine Aktion ausführen. Dies geschieht durch die Methode `void aktion(Liste teilnehmer)`. Außerdem können Verkehrsteilnehmer in einen Unfall verwickelt sein. Die Methode `Verkehrsteilnehmer unfall(Verkehrsteilnehmer gegner, Liste alle)` implementiert das Verhalten eines Verkehrsteilnehmers bei einem Unfall mit dem Unfallgegner `gegner`. Die Methode `boolean enter(Fussgaenger v);` implementiert das Einsteigen in ein Fahrzeug. Die default-Implementierung soll lediglich `false` zurückgeben.

b) Wir unterscheiden zwischen Personen und Fahrzeugen. Legen Sie die abstrakten Klassen `Person` und `Fahrzeug` an, die das Interface `Verkehrsteilnehmer` implementieren.

Die Geschwindigkeit von Personen soll beim Erzeugen gesetzt werden und nimmt zufällig Werte von 2 bis 10 an. Der Name wird auf einen zufälligen Namen initialisiert.

Ein Fahrzeug hat bei einer Aktion eine Chance von 50%, einen Unfall zu erleiden. In diesem Fall wird aus der Liste ein zufälliger Unfallgegner ausgewählt. Anschließend wird die Funktion `Verkehrsteilnehmer unfall(Verkehrsteilnehmer gegner, Liste alle)` für beide Unfallbeteiligten aufgerufen, wobei der jeweilige Gegner übergeben wird. Verwenden Sie den Rückgabewert, um den Verkehrsteilnehmer nach dem Unfall in der Liste zu ersetzen. Eine Person kann ein Fahrzeug mit der Funktion `bool enter(Fussgaenger v)` betreten. Falls das Fahrzeug bereits voll ist, wird der Zutritt verweigert. Andernfalls wir der Fußgänger zu den Passagieren hinzugefügt. Die Geschwindigkeit von Fahrzeugen soll beim Erzeugen gesetzt werden und nimmt Klassenspezifische Werte an. Der Name wird auf ein zufälliges Automodell initialisiert.

Legen Sie in beiden Klassen eine Implementierung von `Verkehrsteilnehmer unfall(Verkehrsteilnehmer gegner, Liste alle)` an, die `this` zurückgibt. Die Implementierungen werden später verfeinert.

c) Schreiben Sie die Klasse `Fußgänger`, welche die fußläufigen Personen der Simulation modelliert und die Klasse `Person` erweitert. Die Person versucht beim Aufruf von `aktion(...)` in ein zufälliges Fahrzeug einzusteigen – dabei wird ein zufälliger Verkehrsteilnehmer aus der Liste ausgewählt. Wenn der Versuch erfolgreich war muss der Fussgänger aus der Liste entfernt werden. Andernfalls geht der Fußgänger weiter spazieren. Die Aktion eines Fußgängers ist dann die Ausgabe von "[name] bewegt sich durch die Stadt".

Schreiben Sie die Klasse `Verletzter`, welche die verunfallten Personen der Simulation darstellt und ebenfalls die Klasse `Person` erweitert. Ein Verletzter behält das Wissen über seine Maximalgeschwindigkeit, gibt aber eine Geschwindigkeit von 0 an. Die Aktion von Verletzten besteht daraus, dass sie sich über das Krankenhaus ärgern. Sie benötigen eine Funktion, die einen Verletzten von einer Person ableitet.

d) Schreiben Sie die Klasse `Auto`, welche die normalen Fahrzeuge darstellt. Autos haben eine Maximalgeschwindigkeit von 140 bis 180. Bei Autos gibt es keinen expliziten Fahrer, nur Passagiere. Ein Auto ohne Passagiere bleibt stehen und gibt entsprechend die Geschwindigkeit 0 aus. Ein Auto kann maximal 5 Passagiere aufnehmen.

Schreiben Sie die Klasse `Bus`. Die Namen für die Busse sollen nach dem Schema "Bus i" vergeben werden, wobei i eine fortlaufende Nummerierung der Busse liefert. Busse haben eine Maximalgeschwindigkeit von 40 bis 100 und eine Kapazität von 30.
Hinweis: Es wird empfohlen für die Nummerierung der Busse eine statische Variable in der Bus-Klasse zu verwenden.

Schreiben Sie die Klasse `Taxi`, welche sich wie die normalen Fahrzeuge verhält.

Busse und Taxis erhalten bei der Initialisierung eine neue Person als Fahrer zugewiesen.

e) Implementierung von `unfall` Verkehrsteilnehmer können bei einem Unfall abhängig von den Geschwindigkeiten zu Schaden kommen:
Die Wahrscheinlichkeit für einen Schaden ist $\frac{speed_{Gegner}}{speed_{this}+speed_{Gegner}}$

Fußgänger werden im Schadensfall zu Verletzten.
Fahrzeuge sind im Schadensfall nicht mehr fahrtüchtig. Die Passagiere verlassen in diesem Fall unverletzt das Fahrzeug, nur ein Fahrer wird zum Verletzten.

f) Erstellen Sie die Klasse `Strassennetz`, welche das als `private` markierte Attribut `Liste verkehrsteilnehmer` besitzt. Der Konstruktor dieser Klasse hat das Argument `int teilnehmer` und legt eine Liste mit entsprechend vielen Verkehrsteilnehmern an. Es sollen zufällig etwa 60% Fussgänger, 20% Autos, 10% Busse und 10% Taxen erzeugt werden. Implementieren Sie eine öffentliche Methode `aktion`, die einen zufälligen Verkehrsteilnehmer aus der Liste auswählt und dessen `aktion`-Methode aufruft und dabei die Liste aller Verkehrteilnehmer übergibt. In der statischen `main`-Methode der Klasse soll ein neues Strassennetz mit 10 Verkehrsteilnehmern erstellt werden. Danach wird 25 mal die Methode `aktion` des neuen Strassennetzes aufgerufen.

Ein Lauf des Programms könnte beispielsweise die folgende Ausgabe erzeugen:

```
Der Bus 2 bewegt sich durch die Stadt.
Mattis steigt in Bus 3 ein.
Tabea steigt in Bus 3 ein.
Tobias steigt in Bus 3 ein.
Der Bus 4 bewegt sich durch die Stadt.
Der Bus 3 bewegt sich durch die Stadt.
Der Bus 1 hat einen Unfall mit Bus 2.
Der Bus 1 kommt beim Unfall zu Schaden.
Der Bus 2 bewegt sich durch die Stadt.
Tanja steigt in Bus 3 ein.
Der Bus 2 hat einen Unfall mit Bus 1.
Der Bus 3 hat einen Unfall mit Bus 3.
Der Bus 3 kommt beim Unfall zu Schaden.
Tanja bewegt sich durch die Stadt.
Der Bus 3 liegt defekt am Strassenrand.
Paula steigt in Bus 2 ein.
Tanja bewegt sich durch die Stadt.
Der Bus 2 bewegt sich durch die Stadt.
Tanja bewegt sich durch die Stadt.
Tanja steigt in Bus 2 ein.
Der Bus 3 liegt defekt am Strassenrand.
Mattis bewegt sich durch die Stadt.
Tobias steigt in Bus 4 ein.
Sascha bewegt sich durch die Stadt.
Mattis bewegt sich durch die Stadt.
Der Bus 3 liegt defekt am Strassenrand.
Der Bus 3 liegt defekt am Strassenrand.
```

Lösung:

a)

Listing 11: Verkehrsteilnehmer.java

```
1  public interface Verkehrsteilnehmer{
2    public String getName();
3    public int getMaxSpeed();
4    public void aktion(Liste v);
5    public Verkehrsteilnehmer unfall(Verkehrsteilnehmer v, Liste alle);
6    default public int getSpeed(){return getMaxSpeed();}
7    default public boolean enter(Fussgaenger v){return false;}
8  }
```

b)

Listing 12: Person.java

```
abstract class Person implements Verkehrsteilnehmer{
  //private Strassennetz sn;
  protected String name;
  protected int maxSpeed;
  public String getName(){return name;}
  public int getMaxSpeed(){return maxSpeed;}
  public int getSpeed(){return getMaxSpeed();}
  public boolean enter(Fussgaenger v){return false;};
/*  public void aktion(Liste v) {
    System.out.println(getName()+" bewegt sich durch die Stadt.");
  }*/
  public Verkehrsteilnehmer unfall(Verkehrsteilnehmer v, Liste alle)
  {
    Verkehrsteilnehmer ret = this;
    int chance = 100 * v.getSpeed() / (v.getSpeed()+getSpeed());
    if (Zufall.zahl(100)<chance) {
      System.out.println("Fussgaenger "+getName()+" kommt beim Unfall zu Schaden.");
      ret = new Verletzter(this);
    }
    return ret;
  }
  public Person() {
    this.name = Zufall.name();
    this.maxSpeed = Zufall.zahl(9)+2;
  }
}
```

Listing 13: Fahrzeug.java

```
abstract class Fahrzeug implements Verkehrsteilnehmer{
  protected Liste passagiere;
  protected String name;
  protected int maxSpeed;
  protected boolean fahrtuechtig;
  abstract protected int personenZahl();
  abstract protected int maxPersonenZahl();
  public String getName(){return name;}
  public int getMaxSpeed(){return maxSpeed;}
  public boolean enter(Fussgaenger v){
    if (personenZahl() < maxPersonenZahl() && fahrtuechtig){
        passagiere.add(v);
        System.out.println(v.getName()+" steigt in "+getName()+" ein.");
        return true;
    }
    return false;
  };
  public void aktion(Liste v) {
    if (!fahrtuechtig){
        System.out.println(this+" "+getName()+" liegt defekt am Strassenrand.");
    } else if (personenZahl()==0) {
        System.out.println(this+" "+getName()+" steht verlassen am Strassenrand.");
    } else {
        if (Zufall.zahl(100)<50)
            unfall(v);
        else
            System.out.println(this+" "+getName()+" bewegt sich durch die Stadt.");
    }
  }
  private void unfall(Liste v) {
    int gegnerId = Zufall.zahl(v.size());
    Element gegnerElement = v.get(gegnerId);
    Verkehrsteilnehmer gegner = gegnerElement.getWert();
    // check ob Verletzter
    if (!(gegnerElement.getWert() instanceof Verletzter)){
            System.out.println(this+" "+getName()+" hat einen Unfall mit "+gegner.getName()+".");
            unfall(gegner,v);
            gegnerElement.setWert(gegner.unfall(this,v));
    }
  }
  public Verkehrsteilnehmer unfall(Verkehrsteilnehmer v, Liste alle)
  {
    if (!fahrtuechtig)
      return this;
    int chance = 100;
    if (v!=this)
        chance = 100 * v.getSpeed() / (v.getSpeed()+getSpeed());
    if (Zufall.zahl(100)<chance) {
      System.out.println(this+" "+getName()+" kommt beim Unfall zu Schaden.");
      fahrtuechtig=false;
      alle.addAll(passagiere);
      passagiere.clear();
    }
```

```
54      return this;
55    }
56    public Fahrzeug() {
57      fahrtuechtig=true;
58      name = Zufall.modell();
59      passagiere = new Liste();
60    }
61  }
```

c)

Listing 14: Fussgaenger.java

```
1   public class Fussgaenger extends Person{
2     public void aktion(Liste v) {
3       int j = Zufall.zahl(v.size());
4       if (v.get(j).getWert().enter(this)){
5         v.remove(this);
6         return;
7       }
8       System.out.println(getName()+" bewegt sich durch die Stadt.");
9     }
10  }
```

Listing 15: Verletzter.java

```
1   public class Verletzter extends Person{
2     public Verletzter(Person p){
3       super();
4       name = p.getName();
5       maxSpeed = p.getMaxSpeed();
6     }
7     public void aktion(Liste v){
8       System.out.println(getName()+" liegt jammernd im Krankenhaus.");
9     }
10  }
```

d)

Listing 16: Auto.java

```
1   public class Auto extends Fahrzeug{
2     protected int personenZahl(){return passagiere.size();}
3     protected int maxPersonenZahl(){return 5;}
4     public String toString(){return "Auto";}
5     public Auto(){
6       super();
7       this.maxSpeed = Zufall.zahl(41)+140;
8       //art="Auto";
9     }
10    public int getSpeed() {
11          if (fahrtuechtig && personenZahl()>0)
12            return getMaxSpeed();
13          else
14            return 0;
15    }
16  }
```

Listing 17: Bus.java

```
1   public class Bus extends Fahrzeug{
2     protected int personenZahl(){return passagiere.size()+1;}
3     protected int maxPersonenZahl(){return 30;}
4     private Person fahrer;
5     static private int count=1;
6     public String toString(){return "Der";}
7     public Bus(){
8       super();
9       this.maxSpeed = Zufall.zahl(61)+40;
10      fahrer = new Fussgaenger();
11      name = "Bus " + count++;
12    }
13    public Verkehrsteilnehmer unfall(Verkehrsteilnehmer v, Liste alle){
14        if(fahrer != null) {
15            System.out.println("Fahrer "+fahrer.getName()+" kommt beim Unfall zu Schaden.");
16            alle.add(new Verletzter(fahrer));
17            fahrer = null;
18        }
19        return super.unfall(v,alle);
20    }
21  }
```

Listing 18: Taxi.java

```
1  public class Taxi extends Auto{
2    protected int personenZahl(){return passagiere.size()+1;}
3    private Person fahrer;
4    public Taxi(){
5      super();
6      fahrer = new Fussgaenger();
7    }
8    public String toString(){return "Taxi";}
9    public Verkehrsteilnehmer unfall(Verkehrsteilnehmer v, Liste alle){
10       if(fahrer != null) {
11           System.out.println("Fahrer "+fahrer.getName()+" kommt beim Unfall zu Schaden.");
12           alle.add(new Verletzter(fahrer));
13           fahrer = null;
14       }
15       return super.unfall(v,alle);
16   }
17 }
```

f)

Listing 19: Strassennetz.java

```
1  public class Strassennetz{
2    private Liste verkehrsteilnehmer;
3    public Strassennetz(int teilnehmer) {
4      verkehrsteilnehmer = new Liste();
5      for(int i = 0; i<teilnehmer; i++){
6        int rnd = Zufall.zahl(100);
7        if(rnd < 60){
8          verkehrsteilnehmer.add(new Fussgaenger());
9          continue;
10       }
11       if(rnd < 70){
12         verkehrsteilnehmer.add(new Auto());
13         continue;
14       }
15       if(rnd < 90){
16         verkehrsteilnehmer.add(new Bus());
17         continue;
18       }
19       verkehrsteilnehmer.add(new Taxi());
20     }
21   }
22
23   private void action()
24   {
25     int j = Zufall.zahl(verkehrsteilnehmer.size());
26     verkehrsteilnehmer.get(j).getWert().aktion(verkehrsteilnehmer);
27   }
28
29   public static void main(String[] args){
30     Strassennetz s = new Strassennetz(10);
31     for(int i = 0; i<25 ; i++){
32       s.action();
33     }
34   }
35
36 }
```