Prof. Dr. J. Giesl
C. Aschermann, F. Frohn, J. Hensel, T. Ströder

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Mittwoch, den 10.12.2014 um 15:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.
- In einigen Aufgaben müssen Sie in Java programmieren und `.java`-Dateien anlegen. **Drucken** Sie diese aus **und** schicken Sie sie per **E-Mail** vor Mittwoch, dem 10.12.2014 um 15:00 Uhr an Ihre Tutorin/Ihren Tutor.
  Stellen Sie sicher, dass Ihr Programm von `javac` **akzeptiert** wird, ansonsten werden keine Punkte vergeben.

---

## Tutoraufgabe 1 (Überladung):

Betrachten Sie die folgenden Klassen:

Listing 1: X.java

```
public class X {
    public int a = 23;
    public X (int a) {                          // Signatur: X(I)
        super();
        this.a = a;
    }

    public X (float x) {                        // Signatur: X(F)
        this((int) (x + 1));
    }

    public void f(int  i, X o) { }              // Signatur: X.f(ILX;)
    public void f(long i, Y o) { }              // Signatur: X.f(JLY;)
    public void f(long i, X o) { }              // Signatur: X.f(JLX;)
}
```

Listing 2: Y.java

```
public class Y extends X {
    public float a = 42;
    public Y (double a) {                       // Signatur: Y(D)
        this((float) (a - 1));
    }
    public Y (float a) {                        // Signatur: Y(F)
        super(a);
        this.a = a;
    }
    public void f(int i, X o) { }               // Signatur: Y.f(ILX;)
    public void f(int i, Y o) { }               // Signatur: Y.f(ILY;)
    public void f(long i, X o) { }              // Signatur: Y.f(JLX;)
}
```

Listing 3: Z.java

```
public class Z {
    public static void main (String [] args) {
                                                // a)
        X xx1 = new X(42);                      // (1)
        System.out.println("X.a: " + xx1.a);
        X xx2 = new X(22.99f);                  // (2)
        System.out.println("X.a: " + xx2.a);
```

```
        X xy = new Y(7.5);                          // (3)
        System.out.println("X.a: " + ((X) xy).a);
        System.out.println("Y.a: " + ((Y) xy).a);
        Y yy = new Y(7);                            // (4)
        System.out.println("X.a: " + ((X) yy).a);
        System.out.println("Y.a: " + ((Y) yy).a);
                                                    // b)
        int i = 1;
        long l = 2;
        xx1.f(i, xy);                               // (1)
        xx1.f(l, xx1);                              // (2)
        xx1.f(l, yy);                               // (3)
        yy.f(i, yy);                                // (4)
        yy.f(l, xy);                                // (5)
        xy.f(i, xx1);                               // (6)
        xy.f(l, xy);                                // (7)
        xy.f(l, yy);                                // (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 `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.

## Aufgabe 2 (Überladung): (8 + 4 = 12 Punkte)

Betrachten Sie die folgenden Klassen:

Listing 4: A.java

```
public class A {
    public int y = 0;
    public static int x = 23;
    public A () {                                   // Signatur: A()
        this(x);
    }

    public A (int x) {                              // Signatur: A(I)
        super();
        y += x;
        x += 1;
        A.x += this.y;
    }

    public void f(int  i, A o) { }                  // Signatur: A.f(IA;)
    public void f(long i, B o) { }                  // Signatur: A.f(LB;)
    public void f(int  i, B o) { }                  // Signatur: A.f(IB;)
}
```

Listing 5: B.java

```
public class B extends A {
    int y = 17;
    public B () {                                   // Signatur: B()
      this(10);
      x++;
    }

    public B (int x) {                              // Signatur: B(I)
        y = x;
        super.y = x+5;
```

```
        super.x = x+2;
    }

    public void f(int  i, A o) {}                  // Signatur: B.f(IA;)
    public void f(long i, B o) {}                  // Signatur: B.f(LB;)
    public void f(long i, A o) {}                  // Signatur: B.f(LA;)
}
```

Listing 6: C.java

```
public class C {
    public static void main (String [] args) {
                                                   // a)
        A aa1 = new A();                           // (1)
        System.out.println (aa1.y);
        System.out.println (A.x);
        A aa2 = new A(42);                         // (2)
        System.out.println (aa2.x);
        B bb = new B();                            // (3)
        System.out.println (bb.y);
        System.out.println (((A) bb).y);
        System.out.println (bb.x);
        A ab = new B(4);                           // (4)
        System.out.println (ab.y);
        System.out.println (((B) ab).y);
        System.out.println (A.x);
                                                   // b)
        int i = 1;
        long o = 2;
        aa1.f(i, aa1);                             // (1)
        aa1.f(i, ab);                              // (2)
        aa1.f(o, bb);                              // (3)
        bb.f(o, bb);                               // (4)
        bb.f(o, ab);                               // (5)
        ab.f(i, aa1);                              // (6)
        ab.f(i, ab);                               // (7)
        ab.f(i, 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.

## Tutoraufgabe 3 (Klassenhierarchie):

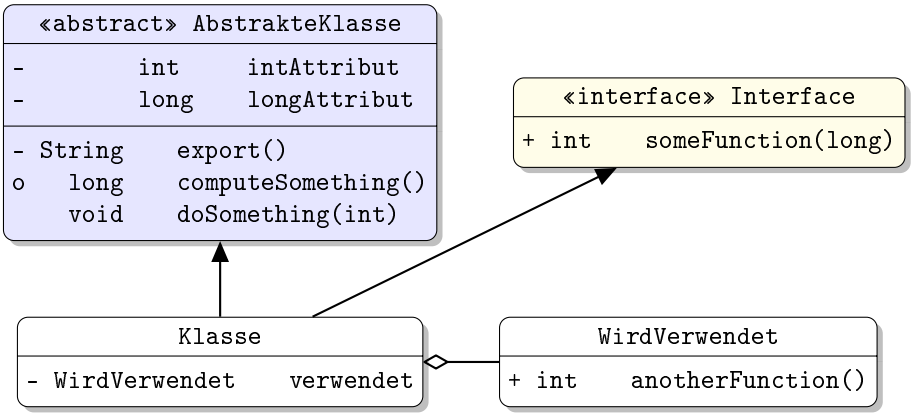
In dieser Aufgabe soll der Zusammenhang verschiedener Getränke zueinander in einer Klassenhierarchie modelliert werden. Dabei sollen folgende Fakten beachtet werden:

- Jedes Getränk hat ein bestimmtes Volumen.
- Wir wollen Apfelsaft und Kiwisaft betrachten. Apfelsaft kann klar oder trüb sein.
- Alle Saftarten können auch Fruchtfleisch enthalten.
- Wodka und Tequila sind zwei Spirituosen. Spirituosen haben einen bestimmten Alkoholgehalt.
- Wodka wird häufig aromatisiert hergestellt. Der Name dieses Aromas soll gespeichert werden können.

- Tequila gibt es als silbernen und als goldenen Tequila.
- Ein Mischgetränk ist ein Getränk, das aus verschiedenen anderen Getränken besteht.
- Mischgetränke und Säfte kann man schütteln, damit die Einzelteile (bzw. das Fruchtfleisch) sich gleichmäßig verteilen. Sie sollen daher eine Methode `schuetteln()` ohne Rückgabe zur Verfügung stellen.
- In unserer Modellierung gibt es keine weiteren Getränke.

**a)** Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die Getränke. Notieren Sie keine Konstruktoren, Getter und Setter. Sie müssen nicht markieren, ob Attribute `final` sein sollen. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen bzw. Interfaces zusammengefasst werden.

Verwenden Sie hierbei die folgende Notation:




Eine Klasse wird hier durch einen Kasten beschrieben, in dem der Name der Klasse sowie Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Weiterhin bedeutet der Pfeil $B \blacktriangleright A$, dass $A$ die Oberklasse von $B$ ist (also `class B extends A` bzw. `class B implements A`, falls $A$ ein Interface ist) und $A \diamond\!\!- B$, dass $A$ den Typ $B$ verwendet (z.B. als Typ eines Attributs oder in der Signatur einer Methode). Benutzen sie `+`, `-` und `o` um `public`, `private` und `protected` abzukürzen.

Tragen Sie keine vordefinierten Klassen (String, etc.) oder Pfeile dorthin in ihr Diagramm ein.

**b)** Implementieren Sie die Methode `enthaeltKiwi` in der Klasse `Mischgetraenk`, die genau dann `true` zurückgibt, wenn das Mischgetränk Kiwisaft enthält. Beachten Sie, dass Mischgetränke auch andere Mischgetränke enthalten können. Verwenden Sie Rekursion.

**Sie müssen die Funktion nicht kompilieren**, da es nicht nötig ist, die gesamte Klassenhierarchie zu implementieren. Schreiben Sie nur diese eine Funktion und kennzeichnen Sie die Funktion mit dem Schlüsselwort static, falls dies angebracht ist.

## Aufgabe 4 (Klassenhierarchie): (0 + 0 + 0 + 0 = 0 Punkte)

Wir wollen das Strategiespiel "Die Schlacht der Fünf Könige" betrachten.

- In dem Spiel existieren Einheiten und Gruppen, die der Spieler kontrollieren kann.
- Alle Einheiten und Gruppen haben eine Methode, welche die Gesundheit zurück gibt.
- Alle Einheiten und Gruppen können Schaden nehmen.
- Der Schaden, der einer Einheit oder Gruppe zugefügt wird, wird aus dem Schadenstyp (`String` z.B. "schwer", "magisch" etc.) und einer Schadensmenge (`double`) berechnet.
- Eine Gruppe enthält ein Array mit beliebig vielen Einheiten oder Untergruppen.

- Eine Gruppe kann ein Teil einer anderen Gruppe sein. Sie darf jedoch niemals ein Teil von sich selber oder von ihren eigenen Teilen sein.
- Wenn eine Gruppe Schaden nimmt, wird dieser gleichmäßig auf alle ihre Teile verteilt.
- Die Gesamtgesundheit einer Gruppe ist die Summe der Gesundheits-Werte ihrer Teile.
- Es gibt verschiedene Sorten Einheiten. Die größten und am gemeinsten aussehenden Einheiten sind Trolle. Bei Trollen ist vor allem die Größe entscheidend.
- Manche Trolle tragen zusätzlich eine Rüstung mit sich. Bei diesen Panzertrollen ist das Gewicht der Rüstung interessant.
- Einige Einheiten sind beritten, in diesem Fall ist die Geschwindigkeit des Reittiers in km/h interessant.
- Elben-Kürassiere sind berittene Einheiten. Bei ihnen ist die Anzahl Pistolen interessant, über die sie verfügen.
- Goblin-Husaren sind ebenfalls berittene Einheiten.
- Es gibt keine sonstigen Einheiten.

**a)** Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für Einheiten. Notieren Sie keine Konstruktoren, Getter und Setter. Sie müssen nicht markieren, ob Attribute final sein sollen. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen bzw. Interfaces zusammengefasst werden. Verwenden Sie hierbei die Notation aus der Tutoraufgabe 3. Welche Objekte realisieren Sie als Klasse und welche als Interface? Begründen Sie ihre Antwort.

**b)** Implementieren Sie die Funktionen `double gesundheit()` und `void schadenNehmen(String typ, double menge)`, die für die Berechnung der Gesundheit und die Schadensverteilung zuständig sind in allen relevanten Klassen / Interfaces.

Normale Einheiten geben einfach ihre Gesundheit zurück. Gruppen geben die Summe der Gesundheiten ihrer Teile zurück. Bei normalen Einheiten wird der Schaden einfach direkt von der Gesamtgesundheit abgezogen. Bei Gruppen wird der Schaden gleichmäßig auf alle Untereinheiten verteilt. Verwenden Sie eine Schleife, um alle Untereinheiten zu betrachten, und Rekursion um den Schaden zu verteilen.

Beispiel: Es sollen 6 Schadenspunkte auf eine Gruppe mit zwei Teilen verteilt werden. Beide Teile erhalten 3 Schadenspunkte. Der erste Teil ist eine Einheit, und entsprechend erleidet sie unmittelbar 3 Schadenspunkte. Der zweite Teil ist eine Untergruppe mit 2 Einheiten. Im nächsten Rekursionsschritt erhalten beide 1.5 Schadenspunkte.

Hinweise:

- Gehen Sie davon aus, dass die Teile einer Gruppe nie `null` sind.
- Diese Teilaufgabe gibt keine Punkte, da wir versehentlich die Musterösung veröffentlicht haben.

**c)** Bei Panzertrollen ist es wichtig, dass bei der Schadensberechnung der Schadenstyp eingeht. Wenn der Schadenstyp nicht "magisch" ist, wird von der Schadensmenge das Gewicht der Rüstung des Trolls abgezogen (Die Schadensmenge wird aber nie kleiner als `0`). Implementieren Sie dieses Verhalten an geeigneter Stelle.

Hinweise:

- Sie können die Methode `Math.max benutzen`.
- Diese Teilaufgabe gibt keine Punkte, da wir versehentlich die Musterösung veröffentlicht haben.

**d)** Stellen Sie Konstruktoren für alle Klassen bereit. Bedenken Sie dabei, dass jeweils auch der Super Konstruktor aufgerufen werden sollte. Nehmen Sie dafür an, das Trolle zu Beginn immer eine Gesundheit von 100 haben. Bei allen anderen Einheiten wird die Gesundheit im Konstruktor als Argument spezifiziert.

Hinweise:

- Diese Teilaufgabe gibt keine Punkte, da wir versehentlich die Musterösung veröffentlicht haben.

## Tutoraufgabe 5 (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 `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
```

## Aufgabe 6 (Interfaces): (2 + 2 + 4 = 8 Punkte)

Im Spiel von Aufgabe 4 betrachten wir den Technologiebaum der zu erlernende Fähigkeiten. Eine Rumpf-Implementierung der benötigten Klassen befindet sich auf der Webseite. Fähigkeiten unterteilen sich in verschiedene Kategorien. So gibt es z.B. magische Fähigkeiten, deren Ausführung Zauberpunkte kostet, und nur von magischen Einheiten benutzt werden können. Auch gibt es durch Fleiß erlernbare Fähigkeiten, die keine weiteren Kosten haben, allerdings Zeit zur Anwendung brauchen und prinzipiell von allen Einheiten ausgeführt werden können. Manche Fähigkeiten sind durch Fleiß lernbar und magisch gleichzeitig. Das Erlernen von Fähigkeiten verursacht Kosten. Außerdem kann jede Fähigkeit andere Fähigkeiten als Voraussetzung haben, die erlernt sein müssen, bevor die neue Fähigkeit erlernt werden kann. Beispielsweise könnte die Fähigkeit `KuerassiereAusbilden` die Voraussetzungen `RüstungSchmieden` und `PistolenHerstellen` haben, welche beide `Metallurgie` erfordern. Solche Voraussetzungen werden in dem Attribut `bedingungen` gespeichert.
Jede Fähigkeit hat noch einen booleschen Wert, der angibt, ob diese Fähigkeit bereits erlernt wurde. Alle durch Fleiß lernbaren oder magischen Fähigkeiten sollen die Möglichkeit haben, eine Beschreibung zu generieren, die die genauen Kosten bzw. die Ausführungsdauer auflistet. Diese sollen in den Interfaces `MagischeFaehigkeit` und `FleissFaehigkeit` als Default-Implementierung angelegt werden.

Listing 7: Faehigkeit.java

```
1  public abstract class Faehigkeit {
2    public boolean erlernt;
3    public Faehigkeit[] bedingungen;
4    public abstract int getLernKosten();
5
6    //4)
7  }
```

Listing 8: MagischeFaehigkeit.java

```
1  interface MagischeFaehigkeit {
2    // 2)
3    public int getZauberKosten();
4  }
```

Listing 9: FleissFaehigkeit.java

```
1  interface FleissFaehigkeit {
2   // 1)
3   public int getAusfuehrungsDauer();
4  }
```

Listing 10: FeuerballWerfen.java

```
1  class FeuerballWerfen extends Faehigkeit implements MagischeFaehigkeit {
2    public int getLernKosten(){
3      return 1000;
4    }
5    public int getZauberKosten(){
6      return 20;
7    }
8  }
```

Listing 11: SchattenWandeln.java

```
1  class SchattenWandeln extends Faehigkeit implements MagischeFaehigkeit , FleissFaehigkeit {
2    public int getLernKosten(){
3      return 100;
4    }
5    public int getAusfuehrungsDauer(){
6      return 10;
7    }
8    public int getZauberKosten(){
9      return 5;
10   }
11   // 3)
12 }
```

Listing 12: SchildwallErrichten.java

```
1  class SchildwallErrichten extends Faehigkeit implements FleissFaehigkeit {
2    public int getLernKosten(){
3      return 10;
4    }
5    public int getAusfuehrungsDauer(){
6      return 5;
7    }
8  }
```

**a)** Fügen Sie Default-Implementierungen für `public String beschreibung()` zu den beiden Interfaces (bei 1 und 2) hinzu. Sie sollen "`Diese Zauberfaehigkeit benötigt $zauberpunkte Zauberpunkte`" bzw. "`Diese Fleissfähigkeit braucht $zeit Sekunden`" zurückgeben. Ersetzen Sie jeweils $zeit und $zauberpunkte durch den Wert von `getAusfuehrungsDauer()` und `getZauberKosten()`.

**b)** Welche Probleme ergeben sich bei der Fähigkeit `SchattenWandeln`, und wie können Sie behoben werden? Verändern Sie die Klasse `SchattenWandeln` an der Stelle 3 so, dass BEIDE Default-Implementierungen aufgerufen und die Ergebnisse kombiniert werden.

**c)** Fügen Sie der Klasse `Faehigkeit` (bei 4) eine Methode hinzu, welche berechnet, wieviel es kosten würde, diese Fähigkeit und alle ihre noch nicht erlernten Voraussetzungen zu erlernen. Verwenden Sie hierfür Rekursion. Beachten Sie dabei, das es sich nicht wirklich um einen Baum handelt und mehrere Fähigkeiten dieselben Voraussetzungen haben können. Im obigen Beispiel dürfen die Kosten für `Metallurgie` nicht doppelt gezählt werden. Ein Beispiel für den erwarteten Output kann in `Main.java` gefunden werden.

Listing 13: Main.java

```
1
2  class Metallurgie extends Faehigkeit{
3    public int getLernKosten(){
4      return 1;
5    }
6  }
7
8  class RuestungSchmieden extends Faehigkeit{
9    public int getLernKosten(){
10     return 10;
11   }
12 }
13
14 class PistolenHerstellen extends Faehigkeit{
15   public int getLernKosten(){
16     return 100;
17   }
18 }
19
20 class KuerassiereAusbilden extends Faehigkeit{
21   public int getLernKosten(){
22     return 1000;
23   }
24 }
25
26 public class Main {
27
28   public static void main(String[] args){
29
30     SchattenWandeln s = new SchattenWandeln();
31     //sollte folgendes ausgeben:
32     //Diese  Zauberfaehigkeit benoetigt 5 Zauberpunkte
33     //Diese Fleissfaehigkeit braucht 10 Sekunden
34     System.out.println(s.beschreibung());
```

```
35
36      Metallurgie m = new Metallurgie();
37      RuestungSchmieden r = new RuestungSchmieden();
38      PistolenHerstellen p = new PistolenHerstellen();
39      KuerassiereAusbilden k = new KuerassiereAusbilden();
40
41      k.bedingungen = new Faehigkeit[]{r,p};
42      r.bedingungen = new Faehigkeit[]{m};
43      p.bedingungen = new Faehigkeit[]{m};
44      m.bedingungen = new Faehigkeit[]{};
45
46      //Sollte 1111 ausgeben
47      System.out.println(k.kostenFuerErforschung());
48    }
49  }
```