Vererbung und Polymorphie

Aufgabe 1: Vererbungsbaum entwerfen

a) Finden Sie sinnvolle Beziehungen. Füllen Sie die letzten Spalten aus.

Hinweis: Es kann nicht alles mit etwas anderem verbunden werden.

Klasse	**Ist-Ein (Is-a)**	**Hat-Ein (Has-a)**	**Oberklasse**	**Unterklassen**
Musiker				
Rock-Star				
Fan				
Gitarre				
Pianist				
E-Gitarre				
Konzertpianist				

b) Zeichnen Sie die Vererbungsklassendiagramme. Gibt es ggf. auch Hat-Ein-Beziehungen, welche Sie einzeichnen können?

Diese Seite ist leer, Aufgabe 2 siehe nachfolgende Seite.

Aufgabe 2: Überschreiben und Überladen von Methoden

```
class A {

     void f() { System.out.println("A.f()"); }

     void f(int i) { System.out.println("A.f(int)"); }

     void g() { System.out.println("A.g()"); }

}


class B extends A {

     void f() { super.f();
                System.out.println("B.f()");
     }

     void g(int i) { System.out.println("B.g(int)"); }

     void h() { System.out.println("B.h()");
                this.g();
     }
}
```

a) `A` ist Oberklasse der Klasse `B`. Welche Methoden besitzt die Klasse `B` (ohne die von `Object` geerbten Methoden)?
Entscheiden sie für die Methoden der Klasse `B`, ob sie jeweils geerbt, überschrieben oder zusätzlich hinzugekommen sind. Welche Methoden werden überladen?

Methode	Geerbt?	Überschrieben?	Zusätzlich?	Überladen?	Begründung

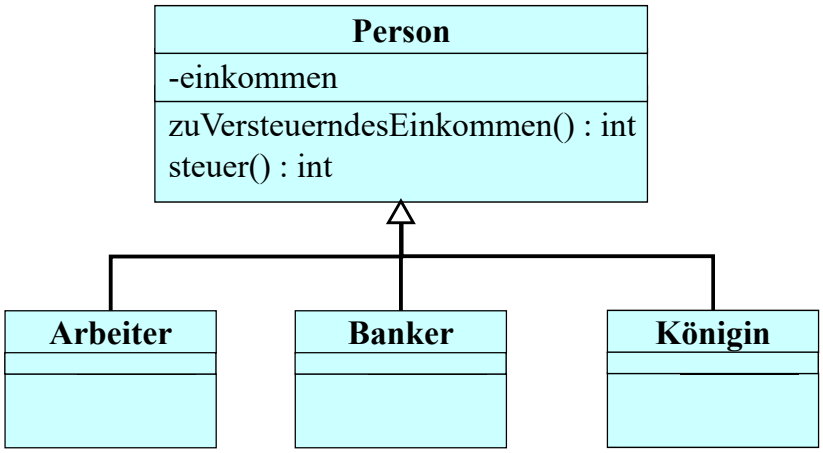
b) Welche der folgenden Anweisungen sind korrekt? Was wird – falls korrekt – jeweils ausgegeben (und welche Methode wird dabei ausgeführt)?

Anweisung	**korrekt?**	**Ausgabe**	**Begründung**
`A o = new A();` `o.f();`			
`B o = new B();` `o.f();`			
`A o = new B();` `o.f();`			
`A o = (A) new B();` `o.f();`			
`B o = new A();` `o.f();`			
`B o = (B) new A();` `o.f();`			
`A o = new A();` `o.g(5);`			
`B o = new B();` `o.g(5);`			
`A o = new B();` `o.h();`			
`B o = new B();` `o.h();`			
`B o = new B();` `o.toString();`			

c) Üb ersetzen Sie die Klassen mit **`javac`**. Betrachten Sie mit **`javap`** den Inhalt der erzeugten Klassendateien A.class sowie B.class (Hinweis: Aufruf mit `javap A` bzw. `javap B`).
Achten Sie insbesondere auf die Vererbungshierarchie und die Liste der Methoden – Fällte Ihnen dabei etwas auf?

Aufgabe 3: Polymorphie

In modernen Ländern basiert das Finanz- und Steuersystem auf IT-Systemen. Die verschiedenen Personen eines Landes seien durch folgende Klassenhierarchie modelliert:

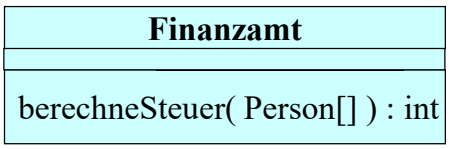


Das Attribut ***einkommen*** enthält das tatsächliche Jahreseinkommen einer Person (in Pfund £). Die Methoden ***int zuVersteuerndesEinkommen()*** und ***int steuer()*** sollen jeweils die korrekten Werte entsprechend der nachfolgenden (vereinfachten und nicht sehr realistischen ☺) Regeln berechnen:

1. Jede **Person** muss Steuern auf sein gesamtes Einkommen zahlen, sofern diese Regeln nichts anderes besagen. Folglich entspricht das zu versteuernde Einkommen normalerweise dem tatsächlichen Einkommen einer Person.
2. Jede Person muss 25% seines (zu versteuernden) Einkommens als Steuer bezahlen. Die Steuer kann dabei auf ganze Pfund abgerundet werden.
3. **Arbeiter** erhalten einen Steuerfreibetrag von 2.400 £. Dieser ist für die Kosten des Weges zur Arbeit und dergleichen gedacht.
4. Solange die Banken in der gegenwärtigen Krise von der Regierung unterstützt werden, muss jeder **Banker** 1.000 £ zusätzlich an Steuern bezahlen.
5. Die **Königin** braucht gar *keine* Steuern zu bezahlen.
6. Der zu zahlende Steuerbetrag ist niemals höher als das Einkommen einer Person, und niemals negativ.

Da mit jährlichen Änderungen des Steuersatzes zu rechnen ist, soll Ihre Implementierung entsprechend änderungsfreundlich sein. Die Steuerberechnung gemäß Regel 2 soll daher nur an einer Stelle in der Klassenhierarchie erfolgen.

Das Finanzamt kann die gesamten Steuereinnahmen des Landes mit Hilfe der Methode ***int berechneSteuer(Person[])*** berechnen:



a) Implementieren Sie die Klassen der ***Person***-Hierarchie. Welche Methoden sollten in der Klasse ***Person*** implementiert werden und welche Methoden sollten in ihren Unterklassen überschrieben werden?

b) Implementieren Sie das Finanzamt (Klasse ***Finanzamt***). Vergessen Sie nicht ihr Programm entsprechend zu testen.

c) Wie viel Steuern kann das Finanzamt von der folgenden Personen-Gruppe eintreiben?

Person	**Einkommen**
Person (“Joe Unemployed”)	6.400 £
Arbeiter (“Suzi Hard-working”)	36.000 £
Banker (“Fred Moneymaker”)	4.000.000 £
Königin (“Elisabeth”)	1.000.000 £

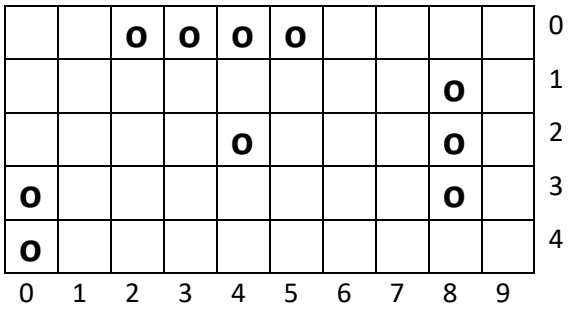
d) Implementieren Sie zum Testen ihrer Funktionen eine geeignete ***toString()***-Methode, um zu den Personen folgende Daten auszugeben:
- das (Brutto-)Einkommen (vor Steuern)
- das zu versteuernde Einkommen
- die zu zahlende Steuer
- das übrigbleibende Netto-Einkommen (Brutto-Einkommen abzüglich der Steuer)

e) (Optional) Erweitern Sie zur besseren Lesbarkeit die Klasse ***Person*** um einen Namen.

Zusatz-Aufgabe 4: Schiffe-Versenken

Realisieren Sie das Spiel Schiffe-Versenken mit einem 2-dimensionalen Spielfeld.

Erweitern Sie dazu Ihre Klasse **SchiffeVersenken1D** aus Übung 1, indem Sie ein 2-dimensionales Array von **Schiff**-Objekten (siehe Aufgabe 4 in Übung 2) anlegen. Dabei sollen mehrere Schiffe (idealerweise unterschiedlich groß und mit verschiedener Ausrichtung horizontal bzw. vertikal) platziert werden.



Hinweis:

- Bei der initialen Platzierung der Schiffe können Sie die Methode **overlapps** verwenden, um zu testen, ob ein zu platzierendes Schiff mit einem anderen kollidieren würde.
- Frage: Wie können Sie überprüfen, ob ein zufällig erzeugtes Schiffs-Objekt (d.h. mit zufällig gewählter Position, Größe & Ausrichtung in x- bzw. y-Richtung) über den Spielfeldrand ragt?

Zusatz-Aufgabe 5 (für Java-Experten): Java-Rätsel: instanceof und Cast

Was machen jeweils die folgenden Java-Programme?

Hinweis:

- Versuchen Sie zunächst, die Lösung „im Kopf“ zu finden, und überprüfen Sie erst danach Ihre Vermutung am Rechner. Versuchen Sie dann herauszufinden, warum Ihre gedachte Lösung ggf. falsch war.
- Bei Fragen wenden Sie sich direkt an den Dozenten.

```
public class Type1 {
   public static void main(String[] args) {
     String s = null;
     System.out.println(s instanceof String);
   }
}

// ---------------------------------------------------

public class Type2 {
   public static void main(String[] args) {
     System.out.println(new Type2() instanceof String);
   }
}

// ---------------------------------------------------

public class Type3 {
   public static void main(String[] args) {
     Type3 t3 = (Type3) new Object();
   }
}
```