SoSe 2020

Prof. Dr. Margarita Esponda

Objektorientierte Programmierung

8. Übungsblatt

Lernziel: Polymorphie, generische Datentypen und Ausnahmebehandlung in Java..

1. Aufgabe (10 Punkte)

Antworten sie mit eigenen Worten folgende Fragen.

a) Wann ist es sinnvoll, Methoden als Klassenmethoden zu definieren? Welche Einschränkungen entstehen dabei?

b) Warum macht es keinen Sinn, lokale Variablen als **private** zu deklarieren? Können Klassenmethoden als **private** deklariert werden? Wenn ja: wann wäre das sinnvoll?

c) Was passiert, wenn in einer Klassendefinition kein Konstruktor definiert wird? Können Konstruktoren vererbt werden? Können Konstruktoren als **private** deklariert werden?

d) Wo sind Instanzvariablen sichtbar, wenn diese als **protected** deklariert werden? Können Konstruktoren als **protected** deklariert werden?

e) Was sind abstrakte Klassen? Wozu sind sie gut? Können Variablen einer abstrakten Klasse deklariert werden?

f) Können Objekte einer abstrakten Klasse erzeugt werden? Kann man Konstruktoren in abstrakten Klassen definieren?

g) Kann eine Variable vom Typ A einem Objekt der Klasse B zugewiesen werden, wenn A eine Unterklasse von B ist? Begründen Sie Ihre Antwort.

h) Können Variablen einer Klassen in Unterklassen überschrieben werden?

i) Was ist automatisches boxing/unboxing? Warum wurde das in Java eingeführt? Welche Probleme gibt es in Java damit?

j) Was ist eine generische Klasse? Wozu sind diese gut?

k) Was verstehen Sie unter Vererbungspolymorphie?

l) Was sind Ausnahmefehler? Was ist der Unterschied zwischen Runtime-Exceptions und allgemeinen Exceptions in Java?

2. Aufgabe (16 Punkte)

a) Programmieren Sie eine generische **Mug**-Klasse in Java, die Becher-Objekte modelliert. **Mug**-Objekte können verschiedene Arten von Flüssigkeiten beinhalten und dürfen beim Erzeugen nur mit einem konkreten Flüssigkeitsobjekt parametrisiert werden.

Die Arten von Flüssigkeiten, die in die **Mug**-Objekte gegossen werden können, sollen alle als Unterklassen folgender abstrakter **Liquid**-Klasse definiert werden.

b) Programmieren Sie mindestens zwei konkrete Unterklassen der **Liquid**-Klasse, um die **Mug**-Klasse testen zu können.

```
public abstract class Liquid {
        final protected String name;
        final protected java.awt.Color color;
        final protected boolean drinkable;
        protected int temperature = 18;
        protected Liquid( String name, Color color, boolean drinkable) {
                this.name = name;
                this.color = color;
                this.drinkable = drinkable;
        }
        public abstract String getName();
        public abstract Color getColor();
        public abstract boolean isDrinkable();
        public abstract void hitUp( int temperature);
        public abstract int getTemperature();
} // end of class Liquid
```

Der Inhalt eines **Mug**-Objekts darf nur mittels folgender Methoden verändert werden.

```
public void pour( int ml ) throws NotEnoughCapacityException;
/* hier wird Flüssigkeit in den Becher gegossen */
public void takeOut( int ml ) throws NotEnoughLiquidException;
/* der Becher wird geleert */
public void drink(int ml) throws UndrinkableException, NotEnoughLiquidException;
/* jemand trinkt ml Milliliter aus dem Becher */
public int empty();
/* der Becher wird geleert */
```

Mindestens folgende Hilfsmethode soll dazu noch programmiert werden:

```
public boolean isEmpty();
/* Frage, ob keine Flüssigkeit drin ist */
public boolean isHot();
/* Frage, ob die Flüssigkeit zu heiß ist */
```

c) Programmieren Sie in der **Exception**-Klassen entsprechende Konstruktoren, in denen aussagekräftige Information für die Ausnahmebehandlung gespeichert wird.

d) Schreiben Sie eine **TestMug**-Klasse, um alle Methoden der **Mug** Klasse ausführlich zu testen. Besonderes wichtig hier ist, Ihre Ausnahme-Klassen zu testen.

Wichtige Hinweise für die Java-Programmierung:

1) Verwenden Sie selbsterklärende Namen von Variablen und Methoden.
2) Für die Namen aller Bezeichner müssen Sie die **Java-Konventionen** verwenden.
3) Verwenden Sie vorgegebene Klassen- und Methodennamen.
4) Methoden sollten klein gehalten werden, sodass auf den ersten Blick ersichtlich ist, was diese Methode leistet.
5) Methoden sollten möglichst wenige Argumente haben.
6) Methoden sollten entweder den Zustand der Eingabeargumente ändern oder einen Rückgabewert liefern.
7) Verwenden Sie geeignete Hilfsvariablen und definieren Sie sinnvolle Hilfsmethoden in Ihren Klassendefinitionen.
8) Zahlen sollten durch Konstanten ersetzt werden.
9) Löschen Sie alle Programmzeilen und Variablen, die nicht verwendet werden.