IT 1 – WS 2019/20 – Übungsblatt 7

Informationen: Die Beispiele sind bis zur Übung am 21.11. vorzubereiten.

Beispiel 7.1:
Gegeben sind die folgenden Klassendefinitionen. Welche der angegebenen Anweisungen in `methode1()` führen zu Compiler-Fehlern? Warum?

```
public abstract class Tier {
      public Tier() {}
}

public class Vogel extends Tier {
      public Vogel() {}
}

public class Taube extends Tier {
      public Taube() {}

      private void methode1() {
            Tier tier1 = new Tier();
            Tier tier2 = new Taube();
            Vogel vogel1 = new Taube();
      }
}
```

Beispiel 7.2:
Schreiben Sie eine abstrakte Klasse *Steuerung* mit Konstruktor

```
public Steuerung(int minWert, int maxWert),
```

der eine Steuerung mit entsprechendem Minimal- bzw. Maximalwert anlegt.

Die Klasse *Steuerung* soll weiters über Methoden

```
public int getMin(),
public int getMax(),
```

verfügen, die den Minimal- bzw. den Maximalwert der Steuerung zurückgeben.

Die Methoden

```
public abstract int getWert(),
public abstract void rauf(),
public abstract void runter()
```

sollen abstrakte Methoden in *Steuerung* sein. (Für eine konkrete Klasse sollen diese Methoden den aktuell eingestellten Wert der Steuerung zurückgeben bzw. diesen erhöhen bzw. verringern, siehe nächstes Beispiel.)

Beispiel 7.3:
Schreiben Sie die Klasse *Heizungssteuerung* aus Beispiel 1.6 so um, dass diese von *Steuerung* aus Beispiel 7.2 erbt. Benennen Sie dafür die Methoden *getTemperatur()*, *waermer()* und *kuehler()* entsprechend um.

Beispiel 7.4:
Welche der folgenden Anweisungen geben aus welchem Grund eine Fehlermeldung?

```
ArrayList<String> a = new ArrayList<String>();
Set<Integer> b = new Set<Integer>();
Collection<Auto> c = new ArrayList<Auto>();
List<Cabrio> d = new Collection<Cabrio>();
```

Beispiel 7.5:
Schreiben Sie eine Klasse *Zentralsteuerung* mit Konstruktor

```
public Zentralsteuerung(List<Steuerung> dieSt),
```

der eine Liste von entsprechenden Steuerungen anlegt, wobei *Steuerung* die abstrakte Klasse aus Beispiel 7.2 ist.

Die Methode

```
public int getWert(int nr)
```

soll den aktuell eingestellten Wert der Steuerung mit der entsprechenden Nummer (in der Liste) zurückgeben.

Die Methoden

```
public void rauf(int nr),
public void runter(int nr)
```

sollen die entsprechende Methode *rauf* bzw. *runter* für die Steuerung mit der entsprechenden Nummer aufrufen. Testen Sie die Klasse *Zentralsteuerung*, indem Sie dem Konstruktor eine Liste von Heizungssteuerungen aus Beispiel 7.3 übergeben.

Beispiel 7.6:
Machen Sie aus der abstrakten Klasse *Steuerung* aus Beispiel 7.2 ein Interface. Welche Änderungen müssen Sie vornehmen? Welche Änderungen sind in der Klasse *Heizungssteuerung* aus Beispiel 7.3 und in der Klasse *Zentralsteuerung* in Beispiel 7.5 nötig?

Beispiel 7.7:
Gegeben seien die Klassen *Anlage* und *Produkt* sowie das Interface *Pruefabteilung*:

```
public interface Pruefabteilung
{
      /* prueft Produkt, und gibt true zurueck,
      wenn Pruefung erfolgreich, sonst false */
      public boolean pruefe(Produkt einProdukt)
}
```

Die Klasse *Anlage* verfügt über die Methode

```
public void bearbeite(Produkt einProdukt)
```

zur Bearbeitung des Produkts *einProdukt* auf der Anlage. In der Klasse *Produkt* sind die Methoden *equals(Object)* und *hashCode()* aus *Object* überschrieben.

Schreiben Sie eine Klasse *Produktion* mit einer Methode

```
public static Set<Produkt> bearbeite(Set<Produkt> dieProdukte,
                      Anlage dieAnlage, Pruefabteilung diePruefabteilung),
```

die alle Produkte in *dieProdukte* auf *dieAnlage* bearbeitet. Nach dem Bearbeiten eines Produktes soll dieses von der Prüfabteilung *diePruefabteilung* geprüft werden. Die Methode soll alle bearbeiteten Produkte aus *dieProdukte*, die erfolgreich geprüft wurden, in einem *Set* zurückgeben.

Beispiel 7.8:
Gegeben sei eine Interface *Funktion* mit einer Methode

```
public double getWertAnDerStelle(double x).
```

Für eine konkrete Klasse soll diese Methode den Funktionswert an der Stelle *x* zurückgeben.

Schreiben Sie eine Klasse *VerketteteFunktion*, die das Interface *Funktion* implementiert und über einen Konstruktor

```
public VerketteteFunktion(Funktion f, Funktion g)
```

verfügt, wobei *f* und *g* die zu verkettenden Funktionen sind. Die Methode *getWertAnDerStelle* soll für einen Parameter *x* die den Wert g*(f(x))* zurückgeben.

Beispiel 7.9:
Gegeben sei eine Klasse *Ort* mit einer Methode

```
public boolean istErreichbarVon(Ort einOrt),
```

die genau dann *true* zurückgibt, wenn der Ort von *einOrt* aus erreichbar ist. Weiters seien in *Ort* die Methoden *equals(Object)* und *hashCode()* aus *Object* überschrieben.

Schreiben Sie in einer Klasse Ihrer Wahl eine statische Methode

```
public static Set<Ort> getOrteErreichbarVon(Ort einOrt,
                                            Set<Ort> dieOrte),
```

die alle Orte in *dieOrte*, die von *einOrt* aus direkt oder über eine Folge von anderen erreichbaren Orten erreichbar sind, in einem Set zurück.

Hinweis: Ein mögliche Idee zur Umsetzung ist folgende: Geben Sie zunächst alle direkt erreichbaren Orte aus *dieOrte* in das Rückgabe-Set. Anschließend fügen Sie alle Orte aus *dieOrte*, die direkt von einem Ort im Rückgabe-Set aus erreichbar sind, zum Rückgabe-Set hinzu, und wiederholen das solange, bis sich das Rückgabe-Set nicht mehr ändert.