PROG1 Prüfung HS2016

Klasse	Name, Vorname

Zeit	90 Minuten
Höchstpunktzahl	60 Punkte
Hilfsmittel	Es sind keine Hilfsmittel zugelassen (keine Bücher, Ausdrucke, elektronische Geräte etc.)
Abgabe	• Schreiben Sie Ihre Prüfung nicht mit Bleistift oder roter Farbe. Ihre Lösung zu Aufgabe 1 dürfen Sie bei Bedarf mit Bleistift schreiben. • Es muss klar sein, welche Lösung wir bewerten sollen. Streichen Sie Lösungsversuche durch, die wir beim Korrigieren nicht beachten sollen. Mehrfachantworten werden nicht bewertet. • Schreiben Sie alle Lösungsblätter mit Ihrem Namen und Vornamen sowie der Aufgabennummer an. • Geben Sie alle Aufgaben- und Lösungsblätter ab.

	Max. Punktzahl	Erreichte Punktzahl
Aufgabe 1	**20**	
Aufgabe 2	**8**	
Aufgabe 3	**7**	
Aufgabe 4	**10**	
Aufgabe 5	**7**	
Aufgabe 6	**8**	

Total:	60	

Aufgabe 1 [20 Punkte]

In dieser Aufgabe entwickeln Sie Teile eines Programms zur Verwaltung von Daten aus der Telefonüberwachung. Lesen Sie bei jeder Teilaufgabe zunächst die ganze Spezifikation, bevor Sie die Teilaufgabe lösen.

a) [8 Punkte] Entwickeln Sie eine Klasse `Call` gemäss folgender Spezifikation:

- Es sollen unter anderem folgende Informationen pro Anruf gespeichert werden: die Rufnummer des Anrufenden und die Rufnummer des Angerufenen.
- Bei der Erzeugung eines `Call` muss man den Anrufenden und den Angerufenen angeben. Diese Angaben lassen sich nach der Erzeugung nicht mehr verändern.
- Man kann die gespeicherten Informationen einzeln abfragen. Schreiben Sie hier nur die Methode zur Abfrage der Rufnummer des Anrufenden.
- Die Rufnummern müssen mit einem "+" oder "00" beginnen. Ist dies nicht der Fall, soll eine `IllegalArgumentException` geworfen werden.
- Überschreiben Sie die `toString` Methode der Klasse Object so, dass ein String bestehend aus den pro Anruf gespeicherten Angaben zurückgegeben wird. Der String soll analog dem nachfolgenden Beispiel formatiert sein: **`00415550000000 => +415550000055`**
- Die Klasse soll über die Anzahl bisher erzeugter `Call` Objekte buchführen. Mittels der **statischen** Methode `getNumberOfCalls` kann diese Anzahl abgefragt werden.

Lösung Aufgabe a)

HINWEISE:

- Ihre Lösung muss die Clean-Code-Regeln einhalten. Einzige Ausnahme ist, dass Sie keine Javadoc hinschreiben müssen.

```
                Call
public class ____________________________________________
{









}
```

b) [4 Punkte] Damit Anrufe z.B. nach einer Analyse der Aufzeichnung eines Anrufes mit Stichworten (Tags) versehen werden können, soll eine zusätzliche Klasse `TaggableCall` entwickelt werden. Ihre Aufgabe ist es nun, eine Klasse `TaggableCall` zu implementieren, die folgende Anforderungen erfüllt:

- Sie bietet dieselbe Funktionalität wie die Klasse `Call`
- Objekte dieser Klasse sollen auch als Objekte vom Typ `Taggable` betrachtet werden können und deren Funktionalität bieten. Der Typ `Taggable` ist untenstehend definiert.

Falls Sie die Aufgabe a) nicht gelöst haben, nehmen Sie folgendes an: Es gibt eine (nicht final) Klasse `Call`, die Informationen über Anrufe speichert und deren einziger Konstruktor zwei Argumente vom Typ String entgegennimmt. Die beiden Argumente entsprechen den Telefonnummern des Anrufenden resp. des Angerufenen.

```
/**
  * Represents things to which one can add tags and where it can
  * be check whether they have a certain label/tag. A tag is a
  * single word, for example “blue”.
  */
public interface Taggable {
      /** Adds the specified tag. */
      public void addTag(String tag);

      /** Checks whether this object has the specified tag.*/
      public boolean hasTag(String tag);


}
```

Lösung Aufgabe b)

HINWEISE:

- Ihre Lösung muss die Clean-Code-Regeln einhalten. Einzige Ausnahme ist, dass Sie keine Javadoc hinschreiben müssen.

```
public class _____________________________________________
{









}
```

c) [4 Punkte] Schreiben Sie eine Klasse `CallHistory` wie folgt:

- Die Klasse kann eine beliebige Anzahl von Anrufen (`Call` Objekte) speichern. Eine neue `CallHistory` enthält zunächst keine Anrufe.
- Es können `Call` Objekte zu `CallHistory` hinzugefügt werden.
- Die `CallHistory` bietet eine Möglichkeit eine Liste aller Anrufe zu erhalten, die von einem bestimmten Anrufer stammen.

Falls Sie die Aufgabe a) nicht gelöst haben, nehmen Sie folgendes an: Es gibt eine (nicht final) Klasse `Call`, die Informationen über Anrufe speichert und die eine Methode zum Abfragen der Telefonnummer des Anrufers bietet.

Lösung:

- Ihre Lösung muss die Clean-Code-Regeln einhalten. Einzige Ausnahme ist, dass Sie keine Javadoc hinschreiben müssen.

```
public class _____________________________________________
{









}
```

d) [4 Punkte] Implementieren Sie eine Klasse `CustomCallHistory` gemäss folgender Spezifikation:

- Die Klasse hat mit Ausnahme einer zusätzlichen Methode `setFilter` die gleiche Schnittstelle wie die Klasse `CallHistory`.
- Die Klasse verhält sich bis auf eine kleine Ergänzung genau gleich wie die Klasse `CallHistory`. Die Ergänzung betrifft dass der Codes zum Hinzufügen von Anrufen der Klasse `CallHistory` nur dann ausgeführt werden soll, wenn der Anruf ein vom Benutzer vorgegebenen Filter passiert.
- Der vom Benutzer spezifizierbare Filter kann mittels folgender Methode der Klasse `CustomCallHistory` gesetzt werden:

```
/** Sets the specified filter. *
public void setFilter(CallFilter filter);
```

- Der Typ `CallFilter` verfügt über die folgende Methode:

```
/** Returns false, if the call passes the filter. *
public boolean filter(Call call);
```

Implementieren Sie die Klasse `CustomCallHistory`. Sie können davon ausgehen, dass es bereits verschiedene Filter vom Typ `CallFilter` gibt.

Lösung:

- Ihre Lösung muss die Clean-Code-Regeln einhalten. Einzige Ausnahme ist, dass Sie keine Javadoc hinschreiben müssen.

```
public class ____________________________________________
{









}
```

Aufgabe 2 [8 Punkte]

In der nachfolgenden Klasse fehlen Teile der Implementierung. Ergänzen Sie den Code an den bezeichneten Stellen so, dass er mit der jeweiligen Javadoc übereinstimmt. Sofern dies sinnvoll ist, verwenden Sie bestehende Funktionalität von anderen Methoden dieser Aufgabe. Sie können davon ausgehen, dass alle Parameter sinnvolle Werte enthalten (nicht `null` etc.).

```
public class Raum {
    private final int laenge;
    private final int breite;
    private final String raumArt;

    public Raum(int laenge, int breite, String raumArt) {
        this.laenge = laenge;
        this.breite = breite;
        this.raumArt = raumArt;
    }
    public int berechneFlaeche(){
        return laenge * breite;
    }
    public String getRaumArt() {
        return raumArt;
    }
}

/** Diese Klasse modelliert eine Wohnung. */
public class Wohnung {
      private final static String[] NOETIGE_RAUMARTEN =
                                        {"Badezimmer", "Kueche", "Schlafzimmer"};
      private final List<Raum> raeume;

      public Wohnung(List<Raum> raeume) {
            this.raeume = raeume;
      }

      /**
       * Berechnet die Summe der Fläche von allen Räumen in der Wohnung.
       * @return Summe der Fläche von allen Räumen.
       */
      public int gesammtFlaeche(){

                  //a) [2 Punkt]  HIER CODE EINFÜGEN









      }
```

```
        /**
         * Ermittelt ob eine Wohnung "vollständig" ist.
         * Eine Wohnung ist vollständig, wenn für jede Raumart aus der
         * Konstanten NOETIGE_RAUMARTEN mindestens ein Raum in der Wohnung
         * vorhanden ist.
         * @return true, wenn die Wohnung "vollständig" ist.
         */
        public boolean istWohnungVollstaendig(){

              // b) [3 Punkt] HIER CODE EINFÜGEN









        }

        /**
         * Bestimmt die Anzahl Räume für alle in der Wohnung vorhandenen Raumarten.
         * Das Ergebnis wird in Form einer Map zurückgeliefert.
         * @return Die Anzahl Räume pro Raumart
         */
        public _________________________________ anzahlRaeumeProRaumArt(){

              //c) [3 Punkte] ERGEBNISTYP der Methode und CODE einfügen









        }
}
```

Aufgabe 3 [7 Punkte]

a) [5 Punkte] Untenstehender Code einer Hausverwaltung kompiliert und funktioniert einwandfrei. Der Code enthält allerdings Designfehler und entspricht an verschiedenen Stellen nicht den Java Konventionen oder den Clean Code Regeln. Identifizieren Sie 5 unterschiedliche Unschönheiten und geben Sie die zugehörige(n) Zeilennummer(n) sowie eine kurze Problembeschreibung an. Machen Sie einen Vorschlag, wie man die Unschönheit beheben könnte. Ein Beispiel ist gegeben.

Zeile	Problembeschreibung	Lösungsvorschlag
44	*Unnötiger Kommentar*	*Kommentar entfernen*

```java
public class parkplatz {
   public parkplatz(int platzID) {
      this.platzID = platzID;
   }
   public int gibZustand() {
      return zustand;
   }
   @Override
   public int hashCode(){
      return platzID;
   }

   public int zustand;
   public int platzID;
}

public class Parkhaus {
   private Map<Integer, parkplatz> parkplaetze;

   public Parkhaus(int anzahlParkplaetze) {
      int platznummer = 1;
      parkplaetze = new HashMap<Integer, parkplatz>();

            for (int platz = 0; platz < anzahlParkplaetze; platz++) {
         parkplaetze.put(platznummer, new parkplatz(platznummer));
         platznummer++;
      }
   }
   public Parkhaus(int anzahlParkplaetze, int startnummer) {
      int platznummer = startnummer;
      parkplaetze = new HashMap<Integer, parkplatz>();

      for (int platz = 0; platz < anzahlParkplaetze; platz++) {
         parkplaetze.put(platznummer, new parkplatz(platznummer));
         platznummer++;
      }
   }
   public Map<Integer, parkplatz> getParkplaetze() {
      return parkplaetze;
   }
}

public class Ticketautomat {
   private Parkhaus parkhaus; //Variable für das Parkhaus

   // Hier steht weiterer, für die Aufgabe nicht relevanter Code

   private List<Integer> bestimmeFreieParkplaetze() {
      List<Integer> freieParkplaetze = new ArrayList<Integer>();

      for (Integer platznummer : parkhaus.getParkplaetze().keySet()) {
         if (parkhaus.getParkplaetze().get(platznummer).gibZustand() == 0) {
            freieParkplaetze.add(platznummer);
         }
      }
      return freieParkplaetze;
   }
}
```

b) [2 Punkte] Beantworten Sie folgenden Fragen mit Ja oder Nein. Eine falsche Antwort gibt -0.5 Punkte, eine korrekte Antwort 0.5 Punkte und keine Antwort 0 Punkte. Ein insgesamt negatives Punktesaldo wird auf 0 Punkte aufgerundet.

Ja	Nein	Aussage
		Eine hohe Kohäsion erleichtert die Wiederverwendung einer Klasse.
		„Entwurf nach Zuständigkeiten“ bedeutet, dass die Klassen vom zuständigen Entwickler modelliert werden.
		Hohe Kohäsion und lose Kopplung sind gleichzeitig anzustreben.
		Nutze eine Klasse A Objektmethoden einer Klasse B, dann kann die Kopplung durch Einführung eines Interfaces reduziert werden.

Aufgabe 4: Vererbung und Polymorphie [10 Punkte]

a) [2 Punkte] Kreuzen Sie in der folgenden Tabelle an, ob die Aussagen wahr oder falsch sind. Jede richtige Antwort gibt 0.5 Punkte; jede falsche Antwort gibt 0.5 Punkte Abzug; keine Antwort gibt keinen Punkt. Ein negatives Punktesaldo wird auf 0 Punkte aufgerundet.

Wahr	Falsch	Aussage
		Eine Klasse A hat einen Konstruktor `public A(double a)`. Eine Klasse B, die von A erbt, muss folglich ebenfalls einen Konstruktor `public B(double a)` haben.
		Klasse A und Klasse B seien jeweils abstrakt. Es kann eine Klasse C geben, die direkt von beiden Klassen A und B erbt und wie folgt deklariert wird: `public class C extends A, B.`
		Klasse B erweitert Klasse A, Klasse C erweitert Klasse B. Wenn ein Objekt der Klasse A erzeugt wird, dann wird je ein Konstruktor von A, B und C aufgerufen.
		Jede Java-Klasse implementiert auch die Klasse java.lang.Object.

b) [3 Punkte] Kreuzen Sie für jede Gruppe von Codezeilen die richtige Antwort an (die Gruppen haben keine Abhängigkeiten untereinander und sind jeweils einzeln zu betrachten):

OK: kompiliert und produziert keinen Laufzeitfehler
KN: kompiliert nicht
EX: kompiliert, führt aber zu einem Laufzeitfehler (`ClassCastException`)

Jede richtige Antwort gibt 0.5 Punkte; jede falsche Antwort gibt 0.5 Punkte Abzug; keine Antwort gibt keinen Punkt. Ein negatives Punktesaldo wird auf 0 Punkte aufgerundet.

```
public interface Landtier {}
public abstract class Tier {}
public abstract class Saeugetier extends Tier {}
public class Affe extends Saeugetier implements Landtier {}
public class Seehund extends Saeugetier {}
public class Hund extends Tier implements Landtier {}
public class Menschenaffe extends Affe {}
```

OK	KN	EX	
			`Tier tier = new Seehund();` `Object o = tier;`
			`Saeugetier saeugetier = new Affe();` `Landtier landtier = saeugetier;`
			`Affe affe = new Affe();` `Menschenaffe menschenaffe = (Menschenaffe) affe;`
			`Tier tier = new Affe();` `Saeugetier saeugetier = tier;`
			`Hund hund = new Landtier();`
			`Saeugetier saeugetier = new Seehund();` `Landtier landtier = (Landtier) saeugetier;`

c) [1Punkte] Gegeben sei folgende, korrekte Anweisung: Landtier `landtier = new Menschenaffe();`
Kreuzen Sie an, für welchen der nachfolgenden Datentypen der Ausdruck
`landtier instanceof` *`Datentyp`* wahr ist:

- O `Landtier`
- O `Tier`
- O `Saeugetier`
- O `Affe`
- O `Object`
- O `Seehund`
- O `Hund`
- O `Menschenaffe`

c) [4 Punkte] Die zwei nachfolgenden, stark vereinfachten Klassen beinhalten einige Fehler, die **ein korrektes Kompilieren** verhindern. Korrigieren Sie diese Fehler direkt im Code. **Ergänzen** Sie dazu falls nötig fehlenden Code oder **streichen Sie ganze Zeilen**. Die beiden Interfaces `Fahrplan` und `Kursfahrt` sind korrekt und sollen nicht verändert werden.

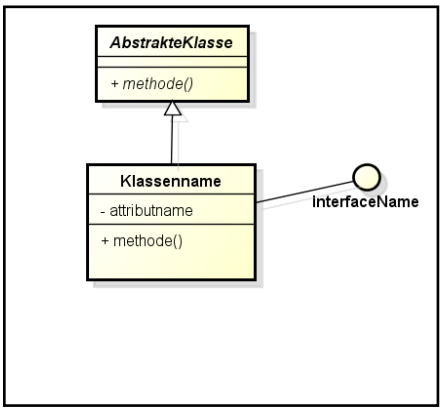
```
import java.util.Date;
import java.util.List;

interface Fahrplan {
  public List<String> getFahrplan(Date datum);
}


interface Kursfahrt {
  public String getKursnummer();
}


public abstract class PassagierFahrzeug {
  private final int maxAnzahlPassagiere;



  public PassagierFahrzeug(int maxAnzahlPassagiere) {


    this.maxAnzahlPassagiere = maxAnzahlPassagiere;


  }









} //End of PassagierFahrzeug
```

```
public class Schiff extends PassagierFahrzeug implements Kursfahrt {
  private int passagiereAnBord;
  private Fahrplan fahrplan;


  public Schiff(int kapazitaet, Fahrplan fahrplan) {

    this.kapazitaet = kapazitaet;



    if (fahrplan == null) {

      throw new IllegalArgumentException();

    }

    this.fahrplan = fahrplan;



  }

  public void einsteigen(int anzahlPassagiere) {

    if (anzahlPassagiere < 0 || anzahlPassagiere > getFreiePlaetze()){

      throw new IllegalArgumentException();

    }

    this.passagiereAnBord += anzahlPassagiere;



  }


  public int getFreiePlaetze() {

    return maxAnzahlPassagiere - passagiereAnBord;



  }
```

```
}//End of Schiff
```

Aufgabe 5: Softwareentwurf [7 Punkte]

Machen Sie einen Entwurf für die nachfolgend grob beschriebene Programmieraufgabe. Der Entwurf soll in Form eines Klassendiagramms **auf der nächsten Seite** erfolgen. Notieren Sie nur Methoden und Attribute, die explizit im Aufgabentext erwähnt sind. So können Sie z.B. alle Konstruktoren und Getter/Setter Methoden, die nicht explizit erwähnt sind, weglassen. Bezeichnen Sie abstrakte Klassen und Methoden mit dem Schlüsselwort *abstract*.



Legende:
+ public,
- private
protected
~ package

Zur Erinnerung:

- Eine Klasse listet nur Methoden welche diese deklariert/implementiert/überschreibt.

a) [6 Punkte] Entwerfen Sie eine Software für die Verwaltung von einfachen Sensoren und Aktoren der Firma Sense&Act.

HINWEIS: Lesen Sie zuerst **alle Punkte** einmal durch, bevor Sie mit dem Entwurf beginnen!

- Sensoren und Schalter sind Produkte und haben jeweils eine Seriennummer und ein Herstellungsdatum.
- Es gibt verschiedene Arten von Sensoren: Temperatur- und Feuchtigkeitssensoren.
- Jeder Sensor verfügt über die Methode `gibEinheit`. Bei Temperatursensoren gibt die Methode "°C" zurück und bei Feuchtigkeitssensoren "%".
- Sensoren verfügen über die Methode `messen`, wobei der Code für die Durchführung einer Messung für Temperatur- und Feuchtigkeitssensoren unterschiedlich ist.
- Es gibt verschiedene Arten von Schalter: Gewöhnliche Schalter und Messschalter.
- Alle Schalter verfügen über die Methode `schalten` die den Schalter ein- resp. ausschaltet.
- Sowohl Messschalter als auch Sensoren sind Messgeräte und können in einer Liste vom Typ `List<Messgeraet>` gespeichert werden.
- Messgeräte verfügen über eine `messen` und eine `gibEinheit` Methode.

b) [1 Punkt] Falls Sie in Ihrem Entwurf Interfaces verwenden, so schreiben Sie nachfolgen den dazugehörigen Java-Code nieder (ohne Javadoc):

Lösung Aufgabe a):

Aufgabe 6: Testen [8 Punkte]

In dieser Aufgabe entwickeln Sie die fehlenden Unit-Tests für die Methode `einsteigen()` einer Klasse `Schiff`. Die Schnittstelle der Klasse `Schiff` ist wie folgt definiert:

Konstruktor

```
public Schiff(int maxAnzahlPassagiere, Fahrplan fahrplan)
```

Erzeugt ein neues Schiff, das maximal die angegebene Anzahl Passagiere aufnehmen kann und eine Referenz auf einen `Fahrplan` besitzt.

Hinweis: Als Vereinfachung für diese Aufgabe gilt, dass `maxAnzahlPassagiere = 1000` ist und für alle Schiffe gleich ist, d.h. an Bord eines Schiffes können 0 bis 1000 Passagiere sein.

```
public int getFreiePlaetze()
```

Gibt die Anzahl der freien Plätze an Bord des Schiffes zurück.

Hinweis: Diese Methode muss in dieser Aufgabe nicht getestet werden, kann aber nützlich sein für Ihre Unit-Tests.

```
public void einsteigen(int anzahlPassagiere)
```

Diese Methode wird verwendet, um eine Anzahl von Passagieren an einer Schiffstation einsteigen zu lassen.
Es wird geprüft, ob es genügend freie Plätze an Bord des Schiffs hat:

- Falls genügend freie Plätze an Bord des Schiffes vorhanden sind, wird `anzahlPassagiere` zu `passagiereAnBord` addiert.
- Falls `anzahlPassagiere` negativ ist, wird eine `IllegalArgumentException` geworfen.
- Falls es nicht genügend freie Plätze an Bord des Schiffes hat, wird eine `IllegalArgumentException` geworfen.

Parameter:
`anzahlPassagiere` – Anzahl einsteigende Passagiere.

a) [2 Punkte] Notieren Sie die Äquivalenzklassen für Ihre Tests der Methode `einsteigen`(). Unterscheiden Sie dabei gültige und ungültige Äquivalenzklassen. Die Tabellen haben möglicherweise mehr Platz für Äquivalenzklassen als es Äquivalenzklassen gibt.

Gültige Äquivalenzklassen (gültige Eingabedaten):

Nr.	Beschreibung der zugehörigen Werte

Ungültige Äquivalenzklassen (ungültige Eingabedaten):

Nr.	Beschreibung der zugehörigen Werte

b) [3 Punkte] Notieren Sie 6 sinnvolle Testfälle. Beinhaltet ein Test den mehrmaligen Aufruf der Methode, dann listen Sie die Werte als Komma getrennte Liste in der Reihenfolge der Verwendung. Die Spalte „Erwartete Resultat“ gibt das Resultat nach dem letzten Aufruf an.

Nr.	Testwerte	Äq.Kl.	Erwartetes Resultat
1			
2			
3			
4			
5			
6			

c) [3 Punkte] Wählen Sie nun einen positiven und einen negativen Test aus Aufgabe b) aus und vervollständigen Sie nachfolgenden Code der Testklasse und des *Stub* `FahrplanStub`. Das *Stub* wird benötigt, weil zur Konstruktion eines `Schiff` Objekts ein Objekt vom Typ `Fahrplan` benötigt wird. Der Typ `Fahrplan` ist wie folgt gegeben:

```
interface Fahrplan {
  public List<String> getFahrplan(Date datum);
}
```

Lösung:

```
public class FahrplanStub ____________________________________________
```

```
}

public class SchiffTest {

  private static final int CAPACITY = 1000;
  private Schiff schiff;

  @Before
  public void setUp() {
    schiff = new Schiff(CAPACITY, new FahrplanStub());
  }
```

```
}
```