Musterprüfung 1

Name, Vorname

Zeit	90 Minuten
Höchstpunktzahl	60 Punkte
Hilfsmittel	Es sind keine Hilfsmittel zugelassen (keine Bücher, Ausdrucke, elektronische Geräte etc.)
Abgabe	• 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	**21**	
Aufgabe 2	**7**	
Aufgabe 3	**6**	
Aufgabe 4	**11**	
Aufgabe 5	**6**	
Aufgabe 6	**9**	
Total:	**60**	

Aufgabe 1 [21 Punkte]

In dieser Aufgabe sollen Sie Code zur Verwaltung der Parkplätze in einem Parkhaus schreiben. Verwenden Sie geeignete Klassen und Unterklassen für die Modellierung. Javadoc Kommentare dürfen Sie weglassen.

a) [10 Punkte] Entwickeln Sie ein Grundsystem gemäss nachfolgender Spezifikation. Funktionalität die nicht explizit gefordert ist muss NICHT implementiert werden. Vorgegeben ist bereits die folgende Klasse `Fahrzeug` (ohne Javadoc):

```
public class Fahrzeug {
  private String nummernschild;

  public Fahrzeug(String nummernschild) {
    this.nummernschild = nummernschild;
  }
  public String getNummernschild() {
    return nummernschild;
  }
}
```

Spezifikation:

- Ein Parkplatz hat eine eindeutige Nummer. Die Nummer ist eine positive ganze Zahl (muss nicht geprüft werden) und kann nach deren Initialisierung nicht mehr verändert werden.
- Ein Parkplatz kann ein `Fahrzeug` aufnehmen. Ist der Parkplatz bereits belegt, wird eine Exception vom Typ `NichtParkierbarException` geworfen.
- Ein Parkhaus hat eine fixe Anzahl Parkplätze. Diese Anzahl wird beim Erstellen des Parkhauses festgelegt und kann nachträglich nicht mehr geändert werden.
- Beim Erstellen des Parkhauses werden alle zugehörigen Parkplätze erzeugt. Die Nummern beginnen mit 1 und sind fortlaufend.
- Um ein `Fahrzeug` im Parkhaus zu parkieren gibt man die Nummer des Parkplatzes an. Falls die Nummer nicht existiert oder der Parkplatz bereits besetzt ist, wird eine Exception vom Typ `NichtParkierbarException` geworfen. Falls der Parkplatz frei ist wird das `Fahrzeug` auf diesem Parkplatz parkiert.

b) [2 Punkte] Schreiben Sie eine Applikation, die ein Parkhaus mit 100 Parkplätzen erzeugt und ein Fahrzeug mit Nummernschild ZH 123 auf Parkplatz Nummer 10 parkiert.

c) [4 Punkte] Erweitern Sie Ihren Code um Funktionalität, welche eine Liste der Nummern der freien Parkplätze zurückgibt. Falls es keinen freien Parkplatz gibt, soll die Liste leer sein.

d) [5 Punkte] Schreiben Sie eine Klasse `AutomatisiertesParkhaus`, die das nachfolgende Interface implementiert. Das automatisierte Parkhaus soll ansonsten die gleiche Funktionalität wie ein normales Parkhaus aufweisen.

```
public interface Parkierbar {

  /**Parkiert das Fahrzeug auf einem
   * beliebigen freien Platz.
   * @param fahrzeug Das zu parkierende Fahrzeug.
   * @return Wahr, falls das Fahrzeug parkiert werden konnte.
   */
  public boolean parkieren(Fahrzeug fahrzeug);
}
```

Aufgabe 2 – Sammlungen / Bibliotheksklassen [7 Punkte]

1. [3 Punkte] Sie sollen eine wie folgt deklarierte Sammlung verarbeiten:

   ```
   HashMap <Integer, String> alphabet;
   ```

   Die Sammlung ist bereits mit den Buchstaben des Alphabets befüllt worden. Die Position des Buchstabens innerhalb des Alphabets dient dabei als Schlüssel (Key). Beispiel: 1 (Key) => A (Value)

   Schreiben sie eine Schleife, die **jeden dritten** Buchstaben aus der Sammlung entfernt, wobei mit dem Buchstaben **B** angefangen wird.

2. [4 Punkte] Sie müssen aus einer Liste mit Suchanfragen diejenigen Suchanfragen entfernen, die verbotene Wörter enthalten. Folgende Sachverhalte sind gegeben:
   a. Eine Suchanfrage besteht aus durch Komma getrennten Wörtern wobei es zwischen Komma und Wort jeweils noch Leerzeichen haben kann. (z.B. “Taxi, Zürich“ oder “Taxi,Zürich“)

   b. Die Suchanfragen befinden sich in einer wie folgt deklarierten Sammlung:
      ```
      ArrayList<String> suchanfragen;
      ```

   c. Die verbotenen Wörter liegen allesamt in einer wie folgt deklarierten Sammlung vor:
      ```
      HashSet<String> verboteneWoerter
      ```

   d. Bei der Speicherung der verbotenen Wörter wurden nur Kleinbuchstaben verwendet.

Aufgabe 3: Klassenentwurf [6 Punkte]

a) [2 Punkte] Kreuzen Sie die korrekten Aussagen an. Korrekt gesetzte Kreuze geben 0.5 Punkte. Falsch gesetzte Kreuze geben -0.5 Punkte. Ein negatives Punktesaldo wird auf 0 Punkte aufgerundet.

Wahr	Falsch	
☒	☐	Code-Duplizierung ist ein Hinweis für einen schlechten Entwurf.
☒	☐	Code-Duplizierung sollte nur bei abstrakten Klassen vermieden werden.
☐	☒	Vererbung kann helfen, Code-Duplizierung zu vermindern.
☒	☐	Code-Duplizierung beeinflusst die Wartbarkeit des Codes negativ.

b) [2 Punkte] Kreuzen Sie die korrekten Aussagen an. Korrekt gesetzte Kreuze geben 0.5 Punkte. Falsch gesetzte Kreuze geben -0.5 Punkte. Ein negatives Punktesaldo wird auf 0 Punkte aufgerundet.

Wahr	Falsch	
☒	☐	Klassen mit hoher Kohäsion lassen sich eher wiederverwenden als solche mit tiefer Kohäsion.
☐	☐	Hohe Kohäsion führt zu lose gekoppelten Klassen.
☐	☐	Hohe Kohäsion macht die Wiederverwendbarkeit von Methoden umständlich, da sie nur für eine einzige Aufgabe zuständig sind.
☐	☐	Es kann nur entweder die Klasse oder deren Methoden eine hohe Kohäsion aufweisen, jedoch nie beide zusammen.

c) [1 Punkt] Geben sie den Vorteil einer losen Kopplung an:

d) [1 Punkt] Was wird unter "Entwurf nach Zuständigkeiten" verstanden?

Aufgabe 4: Testen [11 Punkte]

Sie sollen die Methode `addiereMinuten` der Klasse Uhr testen. Die Klasse hat folgende Schnittstelle:

Class Uhr

Diese Klasse modelliert eine einfache Uhr mit Stunden und Minutenangabe im 24 Stunden Format. Sie speichert folglich Uhrzeiten von 00:00 bis 23:59. Die Uhr kann mittels hinzuzählen von Minuten manipuliert werden. Neu erzeugte Uhren speichern die Zeit 00:00.

Method Detail

addiereMinuten

```
public void addiereMinuten(int minuten)
```

Rechnet zur Uhrzeit die gegeben Anzahl Minuten hinzu. Rechnet man z.B. beim Stand von 23:50 15 Minuten hinzu, beträgt die Uhrzeit anschliessend 00:05. Die Anzahl Minuten muss mindestens 1 und darf höchstens 1440 (=24*60) sein.

Parameters:

`minuten` - Anzahl Minuten

Throws:

`java.lang.IllegalArgumentException` - Bei Minutenwerten ausserhalb des zulässigen Bereichs

getMinuten

```
public int getMinuten()
```

Returns:

Gibt den Wert der Minutenanzeige zurueck.

getStunden

```
public int getStunden()
```

Returns:

Gibt den Wert der Stundenanzeige zurueck.

a) [3 Punkte] Notieren Sie die Äquivalenzklassen (gültige und ungültige) für Ihre Tests:

gültig:
1: 1 - 1440
ungültige
2: <1
3: >1440

b) [5 Punkte] Notieren Sie in die untenstehende Tabelle Ihre Testfälle für diese Klasse. Als Ausgangslage für den Test soll jeweils eine neu erzeugte Uhr dienen (=>Initial auf: 00:00).

Testfall	Testwerte	Äquivalenzklasse	Erwartetes Resultat

c) [3 Punkte] Fügen Sie zu der untenstehenden Testklasse *eine* Testmethode für einen negativen Test der Methode `addiereMinuten` hinzu:

```
public class UhrTest {

  private Uhr uhr;

  @Before
  public void setUp(){
    uhr = new Uhr();
  }

  @Test(expected=IllegalArgumentException.class)
  public void testeErzeugeNull() {
    new Uhr(null);
  }

  //Hier kommt Ihre Testmethode









}
```

Aufgabe 5: Vererbung [6 Punkte]

Gegeben sind folgende drei Klassen:

```
public class Backwaren {
  public Backwaren() { System.out.println("Backwaren"); }
  public Backwaren(String name) { System.out.println("Backwaren: " + name); }
  public Backwaren(boolean vollkorn) {
    if (!vollkorn) {
      System.out.println("Backware ist ohne Vollkorn");
    } else {
      System.out.println("Backware ist aus Vollkorn");
    }
  }
}
public class Brot extends Backwaren {
  public Brot() {
    super("ein Brot");
    System.out.println("Brot");
  }
  public Brot(int gramm) { System.out.println(gramm + " Gramm Brot"); }
  public Brot(boolean bio, String name) {
    super(bio);
    if (bio) { System.out.println("Brot ist Bio"); }
  }
}
public class Gipfeli extends Brot {
  public Gipfeli() { System.out.println("Gipfeli"); }
  public Gipfeli(int gramm) {
    super(gramm);
    System.out.println(gramm + " Gramm Gipfeli");
  }
  public Gipfeli(String name, boolean bio) {
    super(bio, name);
    System.out.println("Gipfeli: " + name);
  }
}
```

In einer anderen Klasse wird folgender Code ausgeführt:

```
Gipfeli g = new Gipfeli();
Gipfeli g2 = new Gipfeli(40);
Gipfeli g3 = new Gipfeli("Laugengipfel", true);
```

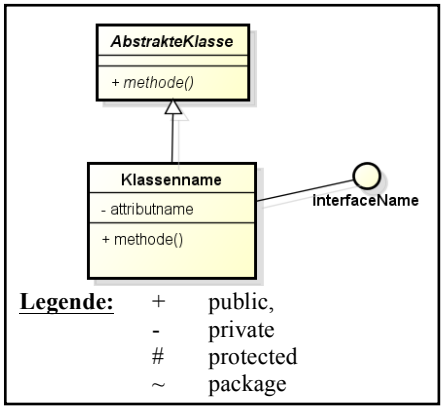
Was wird auf der Konsole ausgegeben?

Aufgabe 6: Abstrakte Klassen und Interfaces [9 Punkte]

a) [6 Punkte] Machen Sie einen Entwurf für das nachfolgend grob beschriebene Rollenspiel. Der Entwurf soll in hier in Form eines Klassendiagramms erfolgen. Notieren Sie in den jeweiligen Klassen alle dort vorhandenen explizit genannten Attribute sowie alle Methoden, die eine der folgenden speziellen Rollen einnehmen:

- Abstrakte Methoden
- Methoden die durch andere Klassen überschrieben werden
- Methoden die eine Methode überschreiben.

Bezeichnen Sie abstrakte Klassen oder Methoden zudem mit dem Schlüsselwort *abstract*.



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

- Es gibt drei Arten von Spielfiguren: Elfen, Orks und Menschen. Alle haben dieselben Attribute: `Stärke`, `Gesundheit`, `Geschicklichkeit`, und `Intelligenz`.
- Bei einem Stufenaufstieg einer Spielfigur wird deren Methode `stufenaufstieg()` aufgerufen. Elfen, Orks und Menschen verhalten sich dabei unterschiedlich (Orks gewinnen z.B. viel an Stärke und Elfen an Intelligenz)
- Neben Spielfiguren gibt es vom Computer gesteuerte Monster: Totbeisser und BlutVogel. Die Monster haben viele Gemeinsamkeiten (Attribute: `Stärke`, `Agressivitaet` und `Gesundheit`), weisen aber auch einige Besonderheiten auf.
- Monster und Spielfiguren sind Lebewesen
- Weiter gibt es noch Geschütztürme.
- Geschütztürme, Monster und Spielfiguren können gegeneinander kämpfen. Lebewesen hingegen verfügen nicht über diese Fähigkeit.
- Der für die Steuerung der Kämpfe zuständige Teil des Spiels ruft zum Starten eines Kampfes die Methode `kaempfe(Kampffaehig gegner)` beim Angreifer auf.

Lebewesen, Spielfigur, Monster sind abstracte Klassen

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 wirft keine Exception zur Laufzeit
KN: kompiliert nicht
EX: kompiliert, wirft aber eine Exception zur Laufzeit (ClassCastException)

Jede richtige Antwort gibt 0.5 Punkte. Jede falsche Antwort gibt 0.5 Punkte Abzug
Keine Antwort gibt keinen Punkt; Min: 0 Punkte; Max: 3 Punkte)

```
public abstract class Fahrzeug {}
public abstract class Auto extends Fahrzeug {}
public interface Turbo {}
public class Rennauto extends Auto implements Turbo {}
public class Oldtimer extends Auto {}
public class Lastwagen extends Fahrzeug implements Turbo {}
```

OK	KN	EX	
X			`Auto auto = new Oldtimer();`
	X		`Fahrzeug fahrzeug = new Auto();` Auto ist abstrakte Klasse
X			`Auto auto = new Rennauto();` `Fahrzeug fahrzeug = auto;`
	X		`Turbo turbo = new Rennauto();` `Rennauto rennauto = turbo;` (CAST) fehlt
		X	`Turbo turbo = new Lastwagen();` `Auto auto = (Auto) turbo;` Lastwagen doesn't extend Auto
		X	`Turbo turbo = (Turbo) new Oldtimer();` Oldtimer doesn't implent Turbo