Aufgabenblatt 14

Lesen Sie Kapitel 13.5 (Interfaces), 13.7 (Weitere Themen der Objektorientierung, ab der 5. Auflage: 13.8) sowie das Kapitel 18 bis einschließlich 18.3. In der 5. Auflage können Sie das Kapitel 13.6 (Anonyme Klassen) überspringen. Lösen Sie anschließend die folgenden Aufgaben.

Für den Aufgabenteil b) müssen Sie die Klasse `Out.java` anpassen. Sie können diese nicht mit der Datei `Out.class` lösen. Gehen Sie hierzu wie folgt vor: Laden Sie sich die Datei `Out.java` von http://ssw.jku.at/JavaBuch/ herunter und speichern Sie diese in dem Ordner ab, in dem auch die Datei `Animal.java` liegt. Öffnen Sie dann die soeben heruntergeladene Datei und fügen sie vor der ersten Zeile die Zeile `package animal;` ein.

a) Alice, Bob und Claire haben die Aufgabe, Erweiterungen für eine Klasse `Person` zu schreiben. Die Instanzattribute und -methoden der Klasse `Person` darf das Entwicklerteam dabei nicht verändern.

(1) Die erste Aufgabe, die das kleine Entwicklerteam erfüllen muss, ist dass sie eine Klasse `Professor` erstellen sollen. Bei einem Professor ist der Professortitel Teil des Namens. Anstatt `"Bob Builder"` soll die Methode `getName()` den String `"Prof. Bob Builder"` zurückgeben. Die drei schreiben die nachfolgende Klasse, um diese Aufgabe zu lösen.

```
package person;

public class Professor extends Person {

    @Override
    public String getName() {
        return "Prof." + " " + getFirstName() + " " + getLastName();
    }

}
```

Das Programm compiliert jedoch nicht. Erklären Sie den Kompilierfehler und modifizieren Sie die Klasse so, dass sie compiliert. Das Programm sollte dann die nachfolgende Ausgabe liefern:

```
$> java person.Person
Name: Alice Wonder
Age: 23

Name: Prof. Bob Builder
Age: 42

$>
```

(4 Punkte)

```
package person;

public class Person {
    private String firstName;
    private String lastName;
    private int age;

    public Person(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public String getName() {
        return getFirstName() + " " + getLastName();
    }

    public int getAge() {
        return age;
    }

    public String getDetailedInformation() {
        return "Name: " + getName() + ", " + "Age: " + getAge();
    }

    public static void main(String[] args) {
        Person alice = new Person("Alice", "Wonder", 23);
        Person bob = new Professor("Bob", "Builder", 42);
//        Person claire = new Student("Claire", "Clear", 22, 987654321);

        System.out.println(alice.getDetailedInformation());
        System.out.println();
        System.out.println(bob.getDetailedInformation());
//        System.out.println();
//        System.out.println(claire.getDetailedInformation());
    }
}
```

(2) Unabhängig von dem Kompilierfehler hat das kleine Entwicklerteam einen schlechten Programmierstil. Benennen Sie die stilistischen Probleme, die ihnen bei der Implementierung der Methode `getName()` in der Klasse `Professor` auffallen und korrigieren sie diese Stilfehler. Kommentare im Quelltext genügen.

(4 Punkte)

b) Im nächsten Schritt soll das Entwicklerteam eine Klasse `Student` implementieren, die von der Klasse `Person` erbt und im selben Paket wie die Klasse `Person` liegt. Ein Student hat neben einem Vor- und Nachnamen auch eine Matrikel (engl. *register*). Die Matrikel kann durch einen `int`-Wert repräsentiert werden. Beim Aufruf von `getDetailedInformation()` soll die Matrikel des Studenten nach dem Alter in einer neuen Zeile ausgegeben werden. Helfen Sie dem Entwicklerteam aus, indem Sie die Klasse `Student` schreiben. Kommentieren Sie dann die mit `//` auskommentierten Zeilen in der `main`-Methode ein und testen Sie ihre Implementierung. Bei korrekter Implementierung sollte das Programm eine Ausgabe ähnlich der Folgenden liefern.

```
$> java person.Person
Name: Alice Wonder
Age: 23

Name: Prof. Bob Builder
Age: 42

Name: Claire Clear
Age: 22
Register: 987654321
$>
```

(6 Punkte)

c) `Animal.java`, `Dog.java`, `Duck.java`, `Crocodile.java`, `CanSwim.java`, `CanFly.java`: In dieser Aufgabe sollen verschiedene Tiere als Objekte dargestellt werden. Hierzu sind Gerüste für die Klassen `Animal`, `Dog`, `Duck` und `Crocodile` bereits vorgegeben. Sie dürfen die Methodenrümpfe modifizieren und weitere Felder und Methoden zu den Klassen hinzufügen. Die Methodenköpfe der bestehenden Methoden dürfen Sie nicht modifizieren.

(1) Die momentane Implementierung ist unvollständig. Wenn Sie das Programm ausführen, so erhalten Sie die folgende Ausgabe:

```
$> java animal.Animal
Doggo is a null.
Donald is a null.
Tick-Tock is a null.


$>
```

Implementieren Sie die Methoden `speak()` und `getSpecies()` sinnvoll. Achten Sie zusätzlich darauf, dass die Fellfarbe des Hundes bei Aufruf mit zurückgegeben wird. Vermeiden Sie Codewiederholung soweit möglich. Bei korrekter Implementierung sollte die folgende (oder eine ähnliche) Ausgabe erzeugt werden:

```
$> java animal.Animal
Doggo is a Dog. He has brown fur.
Donald is a Duck.
Tick-Tock is a Crocodile.

Dog Doggo barks.
Duck Donald quacks.
Crocodile Tick-Tock growls.
```

(6 Punkte)

(2) Schreiben Sie die Interfaces `CanSwim`, `CanFly` und `CanFetch`. Das Interface `CanSwim` enthält dabei nur eine Methode `void swim()`. Das Interface `CanFly` enthält ebenfalls nur eine Methode `void fly()` und das Interface `CanFetch` enthält eine Methode `void fetch(String objectToFetch)`. Die Interfaces sollen alle zu dem Paket `animal` gehören.

(6 Punkte)

(3) Den Tieren sollen jetzt noch zusätzliche Fähigkeiten gegeben werden:

- Jedes der hier aufgeführten Tiere kann schwimmen,
- Enten können fliegen und
- Hunde können Gegenstände apportieren.

Implementieren Sie in den entsprechenden Klassen die Interfaces. Die Aufrufe der Interface-Methoden sollen Konsolenausgaben erzeugen. Nachfolgend finden Sie Beispielausgaben für die Aufrufe der Interface-Methoden

Methode	Beispielausgabe
`dagobert.fly();`	`Duck Dagobert flies away.`
`tickTock.swim();`	`Crocodile Tick-Tock swims.`
`doggo.fetch("ball");`	`Dog Doggo fetches the ball.`

(6 Punkte)

(4) Sie finden am Ende `main`-Methode der Klasse `Animal` Pseudocode für eine Schleife. Realisieren Sie die Funktionalität dieses Pseudocodes in Java. Sie dürfen die Reihenfolge der `if`-Statements im Pseudocode ändern. Sie müssen mit einer `foreach`-Schleife über das Array `animals` iterieren und dürfen nicht auf die Variablen `doggo`, `donald` und `tickTock` zugreifen.

(8 Punkte)

Abgabetermin: Die Lösungen sind bis spätestens Donnerstag, den 1.2.2018 um 8:00 Uhr (strikt!) über das elektronische Abgabesystem einzureichen. Nachträglich eingereichte Lösungen zählen als nicht abgegeben.