HFT Stuttgart
Schellingstraße 24
70174 Stuttgart

Programmieren 1

8 Übung

8.1 Privater Zoo - Vererbungshierarchie und Interfaces

Ein Kind speichert seine Lieblingstiere in einem Array

```
Tier meineTiere[] = new Tier[] { new Hund("Bello"), new Hund("Hasso"), new Pinguin("Schnappi"),
        new Wellensittich("Fifi") };
```

Alle Haustiere sollen mit einer eindeutigen Marke (= laufende Nummer) gekennzeichnet werden. Die gewünschte Ausgabe ist wie folgt:

```
Hund [name=Bello, marke=1]
Hund [name=Hasso, marke=2]
Pinguin [name=Schnappi]
Wellensittich [name=Fifi, marke=3]
```

8.1.1 UML Diagramm

Erstellen Sie ein UML-Diagramm und verwenden Sie folgende Typen:

- Klasse für das Hauptprogramm: Anwendung
- Klassen: Tier, Vogel, Hund, Pinguin, Wellensittich
- Interface: Haustier mit der Methode void setMarke(int id)

Es sollen nur die Klassen Hund, Pinguin und Wellensittich konkret sein. Die Klassen Tier und Vogel sind `abstract`. Platzieren Sie das erforderliche Coding so in der Vererbungshierarchie, dass Redundanzen vermieden werden. Es genügt, mit Konstruktoren und `toString()` zu arbeiten.

Die Klassen Hund und Wellensittich implementieren das Interface Haustier. Nach der Objekterzeugung sollen die Haustiere über die Methode `setMarke()` eine fortlaufende Nummer, d.h. eine Erkennungsmarke erhalten.

8.1.2 Implementierung der Klassen

Implementieren Sie zunächst die Klassenhierarchie und verzichten Sie noch auf das Interface. In der gewünschten Ausgabe wird damit bei den Haustieren der Zusatz „marke = #“ fehlen.

Im Hauptprogramm geben Sie in einer Schleife alle Tiere des Arrays aus.

8.1.3 Implementierung des Interfaces

Implementieren Sie das Interface für die Klassen Hund und Wellensittich. Ändern Sie die Schleife so ab, dass im Schleifenrumpf bei Haustieren die Methode `setMarke()` des Interfaces gerufen wird. Geben Sie eine fortlaufende Nummer als Aktualparameter mit.

8.2 Studentenvergleich – Nutzung vorhandener Interfaces

Gegeben sei eine Klasse Student mit den Feldern nachname, vorname, studiengang und matNr. Im Hauptprogramm sollen die Studierenden wie folgt angelegt werden:

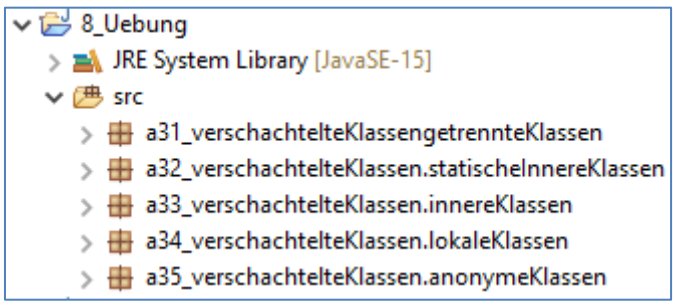
```
Student studenten[] = new Student[6];
studenten[0] = new Student("Baur", "Melanie", "INF", 42);
studenten[1] = new Student("Speiser", "Sebastian", "INF", 43);
studenten[2] = new Student("Rausch", "Alexander", "WINF", 44);
studenten[3] = new Student("Pado", "Ulrike", "WINF", 41);
studenten[4] = new Student("Knauth", "Stefan", "INF", 48);
studenten[5] = new Student("Mosler", "Christof", "WINF", 47);
```

Die Klasse Student soll das Interface `Comparable` implementieren. Das Sortierkritierum sei der Nachname. Anschließend ergänzen Sie das Hauptprogramm, indem Sie die statische Methode `Arrays.sort()` aufrufen, über das Array iterieren und die Einträge ausgeben.

8.3 Verschachtelte Klassen

8.3.1 Getrennte Klassen

Erstellen Sie in Eclipse folgende Projektstruktur:



Laden Sie aus Moodle das Coding für „a31…“ herunter und machen Sie sich mit den Klassen vertraut.

8.3.2 Statische innere Klasse

„Verschieben“ Sie das Coding der Klasse `Wuerfel` in die Klasse `Spieler` hinein. Die verschobene Klasse solle eine statische innere Klasse darstellen. Das Hauptprogramm bleibt unverändert.

8.3.3 Nichtstatische innere Klasse

„Verschieben“ Sie das Coding der Klasse `Wuerfel` in die Klasse Spieler hinein. Die verschobene Klasse solle eine innere Klasse darstellen. Das Hauptprogramm bleibt unverändert.

8.3.4 Lokale innere Klasse

„Verschieben“ Sie das Coding der Klasse `Wuerfel` in die Klasse Spieler hinein. Die verschobene Klasse solle eine lokale Klasse innerhalb der Methode `calculate()` darstellen. Das Hauptprogramm bleibt unverändert.

8.3.5 Anonyme innere Klasse

Kopieren Sie Ihr Coding aus der vorigen Teilaufgabe in das vorgesehene Paket. Erstellen Sie ein Interface `ZufallsZahl` mit der Methode `int random()`. In der Methode `calculate()` erstellen Sie eine anonyme innere Klasse, die das Interface Zufallszahl implementiert und die auf diese Weise gelieferte Implementierung der Methode `random()` aufruft.