**TU Berlin**
**Software and Business Engineering**
**Prof. Dr. Ingo Weber**
**Dr. Tobias Heindel**
**Richard Hobeck**

**WiSe 2021/22**
**Aufgabe 6**
**Block 2**
**29. November 2021**

# Aufgabenblatt 6 *Programmieren I*

**Hinweise**

- Die Abgabe dieser Übungsaufgaben muss bis spätestens Sonntag, den 5. Dezember 2021 um 23:59 Uhr im ISIS-Kurs erfolgt sein. Es gelten die Ihnen bekannten Übungsbedingungen.
- Lösungen zu diesen Aufgaben sind als gezippter Projektordner abzugeben. Eine Anleitung zum Zippen von Projekten finden Sie auf der Seite des ISIS-Kurses. *Bitte benutzen Sie einen Dateinamen der Form* `VornameNachname.zip`.
- Bitte beachten Sie, dass Abgaben im Rahmen der Übungsleistung für die Zulassung zur Klausur relevant sind. Durch Plagiieren verwirken Sie sich die Möglichkeit zur Zulassung zur Klausur in diesem Semester.

Zwei verschiedene Instanzen einer Klasse können unter Umständen das gleiche lebensweltliche Objekt bezeichnen. Um dieser Tatsache gerecht zu werden können wir die `equals`-Methode von Objekten überschreiben. Die folgende Aufgabe hat dieses zum Thema.

## Aufgabe 6.1 Überschreiben von Methoden (aka method overriding) **(2 Punkte)**

- Fügen Sie der Klasse `Person` vom (vierten Aufgabenblatt) weitere Felder hinzu, sodass alle Felder zusammen eine einzige Person beschreiben.
- Überschreiben Sie die `equals`-Methode entsprechend.
- Überschreiben Sie auch die `toString`-Methode, um die relevanten Daten einer Person auszugeben.

Schließlich, fügen Sie der `Person`-Klasse eine `main`-Methode hinzu, in der Sie zwei verschiedene Instanzen der Klasse `Person` erzeugen, die die gleiche Person repräsentieren. Geben Sie eine der beiden Instanzen mittels `System.out.println` aus. Die Ausgabe könnte ungefähr wie folgt aussehen.

```
Person:
Name 'Muster' Vorname 'Io'
  ⋮
höchste wissenschaftliche Qualifikation: keine
Hochschulabschlüsse: Bachelor Mathematik (TUM)
Schulabschlüsse: Abitur


Process finished with exit code 0
```

Die drei Punkte ⋮ sollen Sie durch eine geeignete Ausgabe ergänzen.

Ein Studenplan hat einen Eintrag für jeden Wochentag (der Arbeitswoche). Generell stellt sich die Frage, warum man Arrays nicht auch mit den Konstanten von `enum`-Klassen identifizieren könnte. Die nächste Aufgabe folgt dieser Idee, für den Fall von `Integer`-Arrays. Das folgende Beispiel illustriert die benutzte Funktionalität von `java.lang.Enum`—der impliziten Superklasse aller `enum`-Klassen.

```
public enum EnumConstantsFTW {
    A,B,C;
    static Object[] test(Class c){
        return c.getEnumConstants();
    }
    public static void main(String[] args) {
        System.out.println(A.getClass() + " hat als Superklasse die Klasse "
                + A.getClass().getSuperclass());

        System.out.print(A.getDeclaringClass() + " hat folgende Konstanten: ");
        for(Object e : test(EnumConstantsFTW.class)){
            System.out.print(e +", ");
        }
        System.out.println("\b\b.");
        EnumConstantsFTW.class.getEnumConstants();
    }
}
```

**Aufgabe 6.2** Wie `Integer`-Arrays aber mit `enum`-indices **(2 Punkte)**

Erstellen Sie eine Klasse `IntegerByEnum` mit

- einem `Integer`-Array als Instanz-Variable,
- einem Konstruktor, der ein `Class`-Objekt als Eingabe bekommt und "innerhalb" des Objekts ein geeignetes `Integer`-Array erstellt (siehe `getEnumConstants`),
- einer Methode `put` ohne Rückgabe mit einem `Enum`-Formalparameter und einem `Integer`-Formalparameter, der den Wert in das `Integer`-Array schreibt,
- einer Methode `get` mit `Integer`-Rückgabewert und mit einem `Enum`-Formalparameter, die den entsprechenden Wert aus dem `Integer`-Array liest.

```
IntegerByEnum x = new IntegerByEnum(BeliebigeEnum.class);

for (Enum e : BeliebigeEnum.class.getEnumConstants()){
    x.put(e,e.ordinal());
}

for (Enum e : BeliebigeEnum.class.getEnumConstants()){
    System.out.println(e + ":" + x.get(e));
}
```

Abbildung 1: Testen von `IntegerByEnum`

Zum Lösen der Aufgabe können Sie insbesondere

- die `ordinal`-Methode benutzen, um die Indizes im Array zu erlangen, und
- die Methode `getEnumConstants` benutzen, um alle Konstanten zu bekommen (ggf. in Kobination mit und `getDeclaringClass`).

Am Ende, Testen Sie Ihr Programm mit einer `enum`-Klasse Ihrer Wahl—im Zweifelsfall diejenige mit drei Konstanten `HAUSTIER`, `SCHUHGROESSE`, `LIEBLINGSSPORT` (siehe auch Abbildung 1).

Zusatzaufgaben

**Aufgabe 6.3** (3 Zusatzpunkte)

Betrachten Sie folgende Klasse.

```
public class Basic {
    public Integer a;

    @Override
    public String toString() {
        return "Basic{" +
                "a=" + a +
                '}';
    }
```

(a) Schreiben Sie zwei Interfaces `Multiplicable` and `Summable` mit Methoden `multiplyWith` und `sumWith`, die beide

- einen `Integer` als Formalparameter bekommen und
- keinen Rückgabewert liefern.

(b) Implementieren Sie eine Klasse `MultiplyAndSum`, die von `Basic` erbt und beide Interfaces implementiert, und wenn Sie die `main`-Methode von Abbildung 2 hinzufügen und ausführen, dann

```
public static void main(String[] args) {
    MultiplyAndSum x = new MultiplyAndSum();
    x.a = 1;
    x.multiplyWith(2);
    MultiplyAndSum y = new MultiplyAndSum();
    y.a = 1;
    y.sumyWith(3);
    System.out.println(x);
    System.out.println(y);
}
```

Abbildung 2: Hauptmethode von `MultiplyAndSum`

soll folgende Ausgabe erfolgen:

```
Basic{a=2}
Basic{a=4}

Process finished with exit code 0
```

(c) Erweitern Sie das Interface `Multiplicable` zu `MultiplicableByDefault` mit Getter- und Setter-Methoden `getA` und `setA`, und fügen Sie eine `default`-Implementierung für `defaultMultiplyWith` hinzu, in Analogie zur `MultiplyWith`-Methode.

Schließlich, testen Sie Ihr Programm, indem Sie Klasse `TestIt` von Abbildung 3 hinzufügen und die `main`-Methode ausführen.[1]

[1] Hint: roadrunner.

```
class TestIt extends MultiplyAndSum implements MultiplicableByDefault{
    @Override
    public void setA(Integer a) {
        this.a = a;
        System.out.print("Meep ");
    }

    @Override
    public Integer getA() {
        return this.a;
    }



    public static void main(String[] args) {
        TestIt z = (new TestIt());
        z.setA(100);
        z.defaultMultiplyWith(10);
    }
}
```

Abbildung 3: The final test