Praktikum Grundlagen der Programmierung

Prof. Dr. Harald Räcke, R. Palenta, A. Reuss, S. Schulze Frielinghaus WS 2016/17

Übungsblatt 10 Abgabefrist: keine Abgabe erforderlich

Σ Punkte: 0 | Σ Bonuspunkte: 0

Aufgabe 10.1 (P) Wir sind doch alle gleich

In der Vorlesung haben wir den Gleichheitsoperator == kennen gelernt, der auf Identität testet. Oft wollen wir aber wissen, ob ein Objekt „ähnlich“ zu einem anderen Objekt ist und nicht, ob es sich um genau das gleiche Objekt handelt. Solch eine Ähnlichkeitsbeziehung zwischen Objekten soll via der Methode `boolean equals(Object o)` hergestellt werden. Betrachten Sie folgendes Beispiel:

```
public static void main(String[] args) {
    String a = new String("Hello Student!");
    String b = new String("Hello Student!");

    if (a == b) {
        System.out.println("Sind gleich!");
    } else {
        System.out.println("Sind nicht gleich!");
    }
}
```

Das Program gibt auf der Konsole `"Sind nicht gleich!"` aus. Warum? Der ==-Operator prüft auf Identität. D.h. in diesem Fall, dass `a` und `b` unterschiedliche Objekte im Speicher sind, die zwar den gleichen String repräsentieren, aber unterschiedliche Referenzen haben. Somit sind sie eben nicht gleich und der Identitätscheck schlägt fehl! Oft wollen wir aber nicht auf Identität testen, sondern ob zwei Objekte Eigenschaften teilen also zum Beispiel ob sie den gleichen String repräsentieren. Für diese Fälle gibt es in Java die Methode `equals`:

```
public static void main(String[] args) {
    String a = new String("Hello Student!");
    String b = new String("Hello Student!");

    if (a.equals(b)) {
        System.out.println("Sind gleich!");
    } else {
        System.out.println("Sind nicht gleich!");
    }
}
```

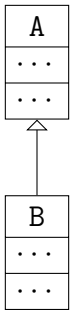
In letzterem Beispiel wird auf der Konsole `"Sind gleich!"` ausgegeben, obwohl die Referenzen `a` und `b` auf andere Objekte im Speicher zeigen.

Die Methode `equals` ist in der Klasse `Object` definiert und kann somit in allen abgeleiteten Klassen überschrieben werden. Der Java-Standard bzw. in der API-Dokumentation[1] finden wir folgende Eigenschaften, die jede `equals`-Methode erfüllen muss:

[1] `https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-`

- It is reflexive: for any non-null reference value x, x.equals(x) should return true.
- It is symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
- It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
- It is consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.
- For any non-null reference value x, x.equals(null) should return false.

Bevor wir weiter machen, schauen wir uns den **instanceof**-Operator in Java an. Betrachten Sie folgende Klassenhierarchie:



Der **instanceof**-Operator gibt **true** zurück wenn ein Objekt von einer angegebenen (Sub-)Klasse ist und andernfalls **false**. Schauen wir uns folgendes Beispiel an:

```
A a = new A();
B b = new B();
A b_as_A = b;

System.out.println(b instanceof A);       // true
System.out.println(b_as_A instanceof B);  // true

System.out.println(a instanceof A);       // true
System.out.println(a instanceof B);       // false
```

Wie erwartet wird nur in der letzten Zeile **false** ausgegeben, da ein Objekt vom Typen `A` nunmal keine Instanz von der (Sub-)Klasse `B` sein kann.

Mit Hilfe des **instanceof**-Operators wollen wir nun die `equals`-Methoden für die Klassen `A` und `B` implementieren. Um das Beispiel realistischer zu Gestalten haben wir der Klasse `A` eine Objektvariable `foo` spendiert. Objekte sollen gleich sein, sofern die Variable `foo` den gleichen Wert hat.

```
1  public class A {
2
3      int foo;
4
```

```
5       @Override
6       public boolean equals(Object o) {
7           if (this == o) return true;
8           if (o == null) return false;
9           if (!(o instanceof A)) return false;
10          final A other = (A) o;
11          return this.foo == other.foo;
12      }
13  }
14
15  public class B extends A {
16
17      @Override
18      public boolean equals(Object o) {
19          if (this == o) return true;
20          if (o == null) return false;
21          if (!(o instanceof B)) return false;
22          final B other = (B) o;
23          return this.foo == other.foo;
24      }
25  }
26
27  public class Main {
28
29      public static void main(String[] args) {
30          A a = new A();
31          B b = new B();
32          System.out.println("a.equals(b) = " + a.equals(b)
33                  + "\nb.equals(a) = " + b.equals(a));
34      }
35  }
```

Das Program gibt auf der Konsole `"a.equals(b) = true"` und `"b.equals(a) = false"` aus. Wir beobachten, dass die Implementierung der `equals`-Methode nicht symmetrisch ist. Dies liegt an dem Verhalten des `instanceof`-Operators.

Tauschen Sie nun die Zeilen 9 und 21 gegen

```
if (getClass() != o.getClass()) return false;
```

aus. Das Program gibt nun auf der Konsole `"a.equals(b) = false"` und `"b.equals(a) = false"` aus.

Die Methode `getClass()` liefert ein Objekt zurück, welches die zugehörige Klasse des jeweiligen Objektes repräsentiert. D.h. testen wir auf (Un)Gleichheit zwischen den zu repräsentierenden Klassenobjekten, dann überprüfen wir, ob beide Objekte von dem *exakt gleichen* Typ sind. Wohingegen ein Überprüfung wie `x instanceof Y` nur überprüft ob das Objekt `x` eine Instanz der *Klasse* oder *Subklasse* `Y` ist. D.h. hier testen wir nicht auf den exakt gleichen Typ.

Mehr dazu findet Ihr u.A. auch hier `https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op2.html` und hier `https://docs.oracle.com/javase/tutorial/reflect/class/classNew.html`.

Zusammengefasst, um die geforderten Eigenschaften der `equals`-Methode zu erfüllen, müssen wir bei der Implementierung ganz genau hinschauen ob wir `instanceof` oder `getClass()` verwenden wollen. Letzteres ist meist die sicherere Wahl. Ist die Klasse `final`, d.h. es kann keine Klasse von dieser erben, dann können wir auch `instanceof` verwenden.

Hinweis: Bisher wissen wir noch nicht was ein Hashcode ist. Wir sollten uns an dieser Stelle aber schon einmal merken, dass wann immer wir die Methode `public boolean equals(Object o)` überschreiben, wir auch immer die Methode `public int hashCode()` überschreiben sollten und vice versa. Wieso? Weil wenn `x.equals(y)` gilt dann auch `x.hashCode() == y.hashCode()` gelten muss (beachten Sie, dass es sich hierbei nur um eine Implikation und nicht eine Bi-Implikation handelt)!

Aufgabe 10.2 (P) Generische Queue

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Methoden (siehe Anhang).

Wir wollen die bekannte Datenstruktur Queue als generischen Datenstruktur implementieren. Schreiben Sie in einer Datei `Queue.java` die Klasse `Queue<T>`.

- Implementieren Sie in der Klasse `Queue<T>` eine innere Klasse `Entry<T>`, die lediglich die beiden privaten Objektvariablen `Entry<T> next` und `T elem;` und einen Konstruktor `Entry(T elem)`, der die Objektvariablen entsprechend initalisiert.
- Überlegen Sie sich welche privaten Objektvariablen Sie für Ihre Implementierung in der Klasse `Queue<T>` benötigen. Verwenden Sie nur private Membervariablen.
- Schreiben Sie in der Klasse `Queue<T>` eine Methode `public boolean isEmpty()`, die `true` zurückliefert, falls die Queue leer ist und ansonsten `false`.
- Schreiben Sie in der Klasse `Queue<T>` eine Methode `public void enqueue(T newElement)`, die der Queue ein neues Element mit dem Wert `newElement` hinzufügt.
- Schreiben Sie in der Klasse `Queue<T>` eine Methode `public T dequeue()`, die ein Element entsprechend dem FIFO-Prinzip entfernt und den Wert dieses Elements zurückliefert. Ist die Queue leer, soll `null` zurückgeliefert werden.
- Schreiben Sie in der Klasse `Queue<T>` eine Methode `public String toString()`, die eine String-Repräsentation der Elemente der Queue zurückliefert, wobei das Element (von den noch in der Queue vorhanden), das als erstes hinzugefügt wurde, an erster Position steht, usw.
- Testen Sie Ihre Implementierung.

Aufgabe 10.3 (P) abstrakte geometrische Körper

Wir betrachten erneut die Klassenhierarchie zur Modellierungsaufgabe geometrischer Körper. Diskutieren Sie, warum die Klasse `Grundflaeche` besser als abstrakte Klasse definiert werden sollte und insbesondere welche Methoden in der Klasse `Grundflaeche` direkt implementiert und welche lediglich deklariert werden sollten.

Passen Sie die Implementierung der Klasse `Grundflaeche` entsprechend an.

Erlaubte Java-Methoden

MiniJava:

```
public static int readInt()
public static int readInt(String s)
public static int read()
public static int read(String s)
public static String readString()
public static String readString(String s)
public static void write(String output)
public static void write(int output)
public static int drawCard()
public static int dice()
```

returns an integer from the interval $[2, 11]$

returns an integer from the interval $[1, 6]$

String:

```
public char charAt(int index)
```

Returns the char value at the specified index. An index ranges from `0` to `length() - 1`. The first char value of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

Example: `String s = "Hello Students"; char c = s.charAt(7);` saves the character `'t'` in the variable `c`.

```
public boolean isEmpty()
```

Returns `true` if, and only if, `length()` is `0`.

```
public int length()
```

Returns the length of this string. The length is equal to the number of Unicode code units in the string.

Example: `String s = "Hello Students"; int l = s.length();` saves the value `14` in the variable `l`.

```
public boolean equals(Object obj)
```

Compares this string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as this object.
Example: `String s = "Hello Students"; boolean v = s.equals("Hello Students");` here `v` is evaluated to `true`.

System:

`System.out.print(x)` prints the object `x` to the console

`System.out.println(x)` prints the object `x` to the console and terminates the line

Selbstgeschriebene Methoden, die selber nur erlaubte Methoden verwenden, sind erlaubt, sofern dies nicht explizit in der Aufgabenstellung verboten wurde. Methoden, die im gegebenen Programmgerüst definiert wurden, dürfen verwendet werden.