University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Programmierung für Mathematik (HS13)

Übung 6

1 Aufgabe: Getter- und Setter-Methoden

1.1 Lernziele

1. Getter- und Setter-Methoden verstehen und anwenden.

1.2 Aufgabenstellung

a) Getter- und Setter-Methoden beschreiben

In der Vorlesung haben Sie Getter- und Setter-Methoden kennengelernt. Beschreiben Sie nun in einigen Sätzen, weshalb Getter- und Setter-Methoden nützlich sind und wann sie gebraucht werden.

b) Getter- und Setter-Methoden anwenden

Implementieren Sie eine Klasse `Trapezoid` so, dass Sie mittels Setter-Methoden die Seitenlängen a und c sowie die Höhe des Trapezes eingeben können. Ausserdem benötigen Sie Getter-Methoden, um die jeweiligen Seitenlängen und die Höhe einzeln auszugeben. Implementieren Sie zusätzlich eine Methode `getArea`, um die Fläche des Trapezes zu berechnen. Testen Sie die Funktionalität wie üblich mit einem `TestDriver`. Sie können davon aus gehen, dass die eingegebenen Seitenlängen und die Höhe positiv sind und wirklich ein Trapez ergeben.

2 Aufgabe: Roll the Dice

2.1 Lernziele

1. Arrays korrekt ausführen können.
2. Java API anwenden können.
3. `while`-Schleifen anwenden.

2.2 Aufgabenstellung

Anna spielt gerne mit ihrer Kollegin ein Würfelspiel. Dabei zählen sie die Anzahl der Würfe mit einem Würfel bis sie mindestens 30 Punkte gewürfelt haben. Danach ist das Spiel beendet. Anna möchte nun das Spiel am PC simulieren.

1. Erstellen Sie dazu eine Klasse `RollTheDice`, welche das Spiel simuliert. Erstellen Sie eine `rollDice()`-Methode, welche automatisch die Würfe ausführt und die Resultate speichert. Verwenden Sie dazu die bereits bekannte `Math.Random()`-Methode aus der Java API.
2. Erstellen Sie ausserdem eine `printResult()`-Methode, welche die Würfe sinnvoll auf der Kommandozeile ausgibt.
3. Erweitern Sie das Spiel so, dass Anna und ihre Kollegin das Spiel mehrmals nacheinander spielen können. Stellen Sie ausserdem eine `reset()`-Methode zur Verfügung, welche die gespeicherten Daten löscht.
4. Testen Sie die Funktionalität mit einem `TestDriver`.

3 Aufgabe: Repetition: Arrays

3.1 Lernziele

1. Eine bestehende Klasse ändern und dabei dieselben Schnittstellen beibehalten.
2. Arrays repetieren und vertiefen

3.2 Aufgabenstellung

In der letzten Übung haben Sie eine Klasse `MyList` geschrieben. Das Array in dieser Liste nutzt den Speicher optimal, in dem Sinn, dass nur genau soviel Speicher belegt wird wie auch tatsächlich gebraucht wird. Allerdings muss dafür die Grösse des Arrays immer geändert werden, wenn neue Werte hinzugefügt werden. Ziel dieser Übung ist es deshalb, die Klasse `MyList` umzugestalten, sodass es nicht mehr notwendig ist, das Array bei jeder Änderung anzupassen.

Implementieren Sie die Klasse `MyList` neu. Im Vergleich zu Ihrer ersten Implementierung, soll das Array nun `String`-Werte erlauben anstatt `int`-Werte. Beginnen Sie damit, das interne Array anzupassen und ändern Sie alle erforderlichen Methoden so ab, dass das Programm wieder ohne Fehler kompilliert. Überarbeiten Sie anschliessend die weiteren Methoden folgendermassen:

- `add`: Durchsuchen Sie das Array zunächst nach einer Stelle, an der noch kein Wert gespeichert ist. In einem `String[]`-Array ist der Wert an dieser Stelle `null`. Falls eine solche null-Position gefunden wird, fügen Sie den Wert dort ein. Ansonsten soll das Array anderthalb Mal um die aktuelle Grösse vergrössert werden und der neue Wert dann an der letzten Stelle eingefügt werden.
- `insert`: Passen Sie die Methode so an, dass der Wert immer an der gewünschten Stelle gespeichert wird. Ist das Array also zu kurz, verlängern Sie es um die benötigte Grösse und fügen Sie dann an dieser (gewünschten) Stelle den neuen Wert ein.
- `delete`: Erstellen Sie eine neue Methode, die eine Position als Parameter erwartet und den Wert an der gewünschten Stelle zurücksetzt (d.h. auf `null` setzt). Geben Sie eine entsprechende Fehlermeldung aus, falls der Wert der Position grösser ist als die Länge des Arrays.

Testen Sie das Verhalten Ihrer neuen `MyList`-Klasse mit dem folgenden TestDriver:

```java
public class TestDriver {

   public static void main(String[] args) {

      MyList l = new MyList();
      l.initializeArray(3);
      System.out.println("Size: " + l.size()); // Size: 3

      // Testing add
      l.add("a");
      l.add("b");
      l.add("c");
      l.add("d");
      l.add("e");
      l.print(); // a b c null e
      System.out.println("Size: " + l.size()); // Size: 6

      // Testing insert
      l.insert(0, "f");
      l.insert(1, "g");
      l.insert(2, "h");
      l.insert(8, "z");
      l.print(); // f g h d null e null z
      l.add("i");
      l.print(); // f g h d i e null z

      // Testing get
      l.get(3); // Array at position 3 is: g
      l.get(10); // The position is not within the range of the array!

      // Testing pull
      System.out.println("Size: " + l.size()); // Size: 8
      l.pull();
      l.pull();
      l.pull();
      System.out.println("Size: " + l.size()); // Size: 5

      // Testing delete
      l.delete(100); // The position is not within the range of the array!
      l.delete(2);
      l.print(); // f g null d i

      // Testing erase
      l.erase();
      System.out.println("Size: " + l.size()); // Size: 5
      l.print(); //null null null null null

```

```
48          // Testing SetArray
49          String[] newArray = new String[] { "aa", "bb", "cc", "dd", "ee", "ff
                ", "gg" };
50          l.setArray(newArray);
51          System.out.println("Size: " + l.size()); // Size: 7
52          l.print(); //aa bb cc dd ee ff gg
53      }
54  }
```

Listing 1: `TestDriver`-Klasse