Dr. Markus Heckner 28. Februar 2013

Klausur Objektorientierte Programmierung

Sie haben zur Beantwortung der Fragen 60 Minuten Zeit. Es können 36 Punkte erreicht werden. Verwenden Sie sinnvolle Modifikatoren bei der Definition von Variablen, Konstanten und Methoden. Weglassen kann zu Punktabzug führen. Zur Lösung der Aufgaben ist die Programmiersprache Java zu verwenden. Lesen Sie immer die komplette Aufgabe, bevor Sie mit der Bearbeitung beginnen.

Aufgabe 1 Die StringQueue (10 Punkte)

Eine Queue in Java ist eine Klasse, die eine Liste von Elementen (hier: Strings) verwaltet. Neue Elemente werden der Liste immer am Ende hinzugefügt (`add`). Es ist möglich das erste Element aus der Liste abzufragen (`poll`). Bei Abfrage des ersten Elements wird dieses Element gelöscht und die Liste entsprechend verkleinert. Weiterhin lässt sich von einer Queue die Anzahl der enthaltenen Elemente abfragen (`size`). Das Verhalten einer Queue entspricht der Logik beim Drucken von Dokumenten: Das erste Dokument, das der Queue hinzugefügt wurde, wird auch als erstes gedruckt, weitere Dokumente werden am Ende der Queue hinzugefügt.

Das Verhalten einer Queue wird durch das folgende Interface spezifiziert:

```
1  /**
2   * This interface represents a collection of objects called a "queue" in which
3   * new Strings are added at the end of the queue and removed from the front.
4   * This represents a typical first-come/first-served waiting line.
5   */
6
7  public interface SimpleStringQueue {
8   /** Adds a new String to the end of the queue */
9   public void add(String str);
10
11  /** Removes and returns the first String (or null if queue is empty) */
12  public String poll();
13
14  /** Returns the number of entries in the queue. */
15  public int size();
16 }
```

Erstellen Sie die Klasse `StringQueue`, die das oben angegebene Interface implementiert, und die Strings mithilfe einer `ArrayList` als Instanzvariable verwaltet.

Ihre Implementierung der Klasse `StringQueue` sollte z.B. wie folgt funktionieren:

```
1  StringQueue queue = new StringQueue(); // creates an empty StringQueue
2  queue.add("Handout.docx"); // adds a String to the end of the queue
3  queue.add("Examples.docx"); // adds a String to the end of the queue
4  queue.add("Shopping_List.xlsx"); // adds a String to the end of the queue
5  queue.size(); // returns 3
6  queue.poll(); // returns Handout.docx
7  queue.size(); // returns 2
8  queue.poll(); // returns Examples.docx
9  queue.poll(); // returns Shopping_List.xlsx
10 queue.poll(); // returns null
```

Aufgabe 2 Der Freundevergleich auf Facebook (13 Punkte)

Jeder Facebook-Nutzer hat eine Liste von Freunden. Sie sollen ein Programm schreiben, das die Freundeslisten von zwei Nutzern vergleicht und ermittelt, wie viele gemeinsame Freunde diese beiden Nutzer haben.

Sie können in dieser Aufgabe auf die folgende statische Methode der Klasse `FacebookHelper` zurückgreifen, die den Namen eines Nutzers als Parameter erwartet und eine Liste der Freunde dieses Nutzers als String Array zurückgibt:

```
public static String[] getFacebookFriends(String userName)
```

Die Rückgabe der Methode für den Aufruf `getFacebookFriends("Bobby Baccalieri")` liefert beispielsweise ein Array mit den folgenden Einträgen:

Christopher Moltisanti
Adriana La Cerva
Richie Aprile
Jennifer Melfi
Salvatore Bonpensiero

Ein Aufruf wie `getFacebookFriends("Little Paulie Germani")` gibt eine andere Liste an Freunden zurück.

Gegeben ist der folgende Code:

```
import acm.program.ConsoleProgram;

public class FacebookFriendsStart extends ConsoleProgram {

 private static final String FB_USER_1 = "Bobby Baccalieri";
 private static final String FB_USER_2 = "Little Paulie Germani";

 public void run() {
  int numSharedFriends = calculateFacebookFriendMatches(FB_USER_1, FB_USER_2);
  println("Number of shared friends: " + numSharedFriends);
 }

 private int calculateFacebookFriendMatches(String userName1, String userName2) {
  //TODO: Implement this method, add additional private methods if necessary
  return 0;
 }
}
```

Ihre Aufgabe ist es, die Methode

```
public int calculateFacebookFriendMatches(String userName1, String userName2)
```

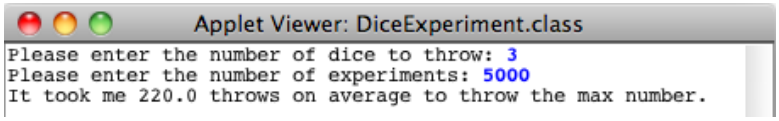
zu implementieren, die zurückgibt wie viele Freunde die beiden Nutzer gemeinsam haben. Achtung: Sie **müssen** die oben beschriebene Methode `getFacebookFriends` verwenden!

Aufgabe 3 Wahrscheinlichkeitsexperiment beim Würfeln (13 Punkte)

Versucht man mit einem Würfel hinreichend oft eine 6 zu würfeln, benötigt man im Durchschnitt 6 Versuche, um dieses Ergebnis zu erreichen. Um mit zwei Würfeln einen 6er-Pasch zu würfeln benötigt man durchschnittlich 36 Versuche. Bei drei Würfeln benötigt man 216 Versuche, usw.

In dieser Aufgabe sollen Sie ein Programm erstellen, das versucht diese Ergebnisse zu reproduzieren. Dazu werden vom Nutzer die Anzahl der Würfel (`numDice`) und die Anzahl der Experimente (`numExperiments`) eingelesen. Ein Experiment besteht daraus, so lange mit der in `numDice` spezifizierten Würfelanzahl zu würfeln, bis das Würfelergebnis nur aus 6ern besteht. Für die Anzahl der Versuche soll ein Durchschnitt über alle Experimente berechnet werden (`averageThrows`) und wieder an den Nutzer ausgegeben werden. Dabei ist es in diesem Beispiel unerheblich, ob Sie gleichzeitig mit allen Würfeln würfeln, oder einzeln hintereinander.

Die folgende Abbildung zeigt einen Beispiellauf des Programms mit drei Würfeln und 5000 Experimenten:



Gegeben ist der folgende Code:

```
import acm.program.ConsoleProgram;
import de.ur.mi.util.RandomGenerator;

public class DiceExperimentStart extends ConsoleProgram {
 private RandomGenerator rGen = RandomGenerator.getInstance();

 public void run() {
   int numDice = readInt("Please enter the number of dice to throw: ");
   int numExperiments = readInt("Please enter the number of times the experiment should be repeated: ");

   double averageThrows = calculateAverageThrowsUntilMax(numDice, numExperiments);
   println("It took me " + averageThrows + " throws on average to throw the max number.");
 }

 /**
  * Calculate the average number of throws needed to reach the maximal dice roll result with numDice
  * in numExperiments
  * @param numDice
  * @param numExperiments
  * @return average number of throws needed to reach the max dice roll result
  */
 private double calculateAverageThrowsUntilMax(int numDice, int numExperiments) {
  //TODO: Implement this method, add additional private methods if necessary
  return 0;
 }

 /**
  * Throw one die
  * @return result of die throw
  */
 private int throwOneDie() {
  return rGen.nextInt(1, 6);
 }
}
```

Ihre Aufgabe ist es, die durchschnittliche Anzahl an Würfen bis zum Maximalergebnis (d.h. nur aus 6ern bestehend) bei `numExperiments` Experimenten zu berechnen. Ergänzen sie die Methode `calculateAverageThrowsUntilMax` und fügen ggf. eigene Methoden hinzu. Tipp: Verwenden Sie die Methode `throwOneDie`, um einmal zu Würfeln.

Viel Erfolg!