Eingabe und Ausgabe

1. Buffered-Zeit

Schreiben Sie ein Programm welches den zeitlichen Unterschied der direkten (*FileWriter)* und der gepufferten (*BufferedWriter)* Zeichen-Ausgabe testet. Die Idee dabei ist mit beiden Methoden eine gleiche Anzahl an Wörtern in eine Datei zu schreiben und die jeweilige Zeit zu messen.

Schreiben Sie dazu eine Klasse *BufferedTester*:

- Dem Konstuktor wird die Anzahl an Wörtern, sowie das Wort welches in die Datei geschrieben wird, als Argument übergeben. (z.B. 1000, „Hallo“ → Es wird bei den Tests 1000 mal *Hallo* in die Datei geschrieben)
- Die Klasse besitzt eine Methode *testDirect()*, welches den Test für den direkten Schreibzugriff durchführt und die aufgewendete Zeit in Millisekunden zurück gibt.
- Die Klasse besitzt eine Methode *testBuffered()*, welches den Test für den gepufferten Schreibzugriff durchführt und die aufgewendete Zeit in Millisekunden zurück gibt.
- Die Klasse besitzt eine Methode *test()*, welches beide tests mit den oberen Methoden durchführt und beide Werte formatiert an die Konsole ausgibt.
- Nutzen Sie für die Zeitmessung folgenden Befehl:

```
System.currentTimeMillis()
```

Führen Sie Ihren Tester mit folgenden Werten aus und notieren Sie sich die Zeitwerte:

- 200x „Hello“
- 10‘000x „Hello“
- 1‘000‘000x „Hello“
- 500‘000x „HelloWorld“

2. Highscore Zahlenerraten

Schreiben Sie ein Programm, das eine zufällige Zahl zwischen 1 bis 3 generiert. Der Benutzer soll nun die Zahl erraten. Für jede erraten Zahl erhält der Benutzer einen Punkt. Wird eine falsche Zahl geraten, wird die Runde beendet und die Punktzahl in einer Text-Datei abgespeichert. Dabei soll zu dieser Punktzahl auch noch der Benutzername des Benutzers abgespeichert werden. Die Liste soll bei jedem Speicherzugriff zudem sortiert werden, so, dass immer die Person mit den meisten Punkten zuoberst steht. Sobald ein Highscore vorhanden ist, soll bei jeder Runde die Punktzahl der besten 10 Spielern dargestellt werden.

Programmablauf

1. Spiel wird gestartet. Falls vorhanden → Highscore wird angezeigt.
2. Spielername wird abgefragt.
3. Spielrunde startet und der Spieler wird nach der zu ratenden Zahl gefragt.
4. Runde wird solange wiederholt, bis der Spieler eine falsche Zahl angibt.
5. Die Punkte werden sortiert in einer Datei abgespeichert.
6. Spiel beendet.

Form

Das Programm kann als Konsolen-Programm oder GUI programmiert werden.

Hinweis

Beachten Sie, dass *try-catch*-Blöcke nur für Ausnahmefälle verwendet werden sollten. Sie werden aber im Spiel für den normalen Ablauf eine Überprüfung benötigen, um zu testen, ob die Highscore-Datei auch vorhanden ist. Diese Datei ist zum Beispiel beim ersten Spielstart noch nicht vorhanden. Daher stellt diese Situation keinen Ausnahmezustand dar und sollte entsprechend mit einer anderen Überprüfung behandelt werden. Sie können dazu die Klasse *File* aus der Java-Bibliothek verwenden. Dort finden Sie die Methode *exists()*, welche überprüft, ob die Datei vorhanden ist. Für das Laden der Datei wird dann weiterhin ein *try-catch*-Block benötigt (Die Datei kann immer noch durch einen Fehler nicht geladen werden. Dies stellt aber dann einen Ausnahmefall dar).

3. SerializedHighscore

In Java haben Sie die Möglichkeit Objekte zu „serialisieren“. Bei diesem Prozess wird die Struktur und der Zustand des Objektes in eine Reihe von Byte-Werten umgewandelt (Bytestrom). Diese Reihe lässt sich dann in eine Datei abspeichern. Dazu wird die Klasse des zu serialisierenden Objektes mit der Schnittstelle *Serializable* versehen:

```
public class Employee implements java.io.Serializable {
   public String name;
   public String address;
   public transient int SSN;
   public int number;

   public void mailCheck() {
      System.out.println("Mailing a check to " + name + " " + address);
   }
}
```

Dieses Objekt lässt sich nun in eine Datei abspeichern. Dazu wird für die Serialisierung die Klasse *ObjectOutputStream* verwendet. Um die umgewandelten Daten in eine Datei abzuspeichern wird die Klasse *FileOutputStream* verwendet:

```
import java.io.*;
public class SerializeDemo {

   public static void main(String [] args) {
      Employee e = new Employee();
      e.name = "Reyan Ali";
      e.address = "Phokka Kuan, Ambehta Peer";
      e.SSN = 11122333;
      e.number = 101;

      try {
         FileOutputStream fileOut =
         new FileOutputStream("/tmp/employee.ser");
         ObjectOutputStream out = new ObjectOutputStream(fileOut);
         out.writeObject(e);
         out.close();
         fileOut.close();
         System.out.printf("Serialized data is saved in /tmp/employee.ser");
      } catch (IOException i) {
         i.printStackTrace();
      }
   }
}
```

Um nun das Objekt wieder zu laden, werden entsprechend die Klassen *ObjectInputStream* und *FileInputStream* verwendet:

```
import java.io.*;
public class DeserializeDemo {

   public static void main(String [] args) {
      Employee e = null;
      try {
         FileInputStream fileIn = new FileInputStream("/tmp/employee.ser");
         ObjectInputStream in = new ObjectInputStream(fileIn);
         e = (Employee) in.readObject();
         in.close();
         fileIn.close();
      } catch (IOException i) {
         i.printStackTrace();
         return;
      } catch (ClassNotFoundException c) {
         System.out.println("Employee class not found");
         c.printStackTrace();
         return;
      }

      System.out.println("Deserialized Employee...");
      System.out.println("Name: " + e.name);
      System.out.println("Address: " + e.address);
      System.out.println("SSN: " + e.SSN);
      System.out.println("Number: " + e.number);
   }
}
```

Implementieren Sie diese Variante in Ihrem Highscore-Spiel, so, dass Objekt-Strukturen anstelle von Zeichen abgespeichert werden. Es kann sein, dass Sie dazu mehrere Klassen serialisieren müssen.