Programmieren II, SS12

Tutorium: Übungen

Musterlösungen findest du jeweils eine Woche nach dem Tutorium auf der Tutoriumsseite.

Aufgabe 1: Multidimensionale Arrays

Wir wollen uns ein kleines, rudimentäres „Schiffe versenken“ programmieren. Dabei soll das Spielfeld durch ein zweidimensionales Array dargestellt werden. Auf dem Feld platzieren wir ein einzelnes Schiff mit den Maßen 1x1 – es soll also nur eine einzige Koordinate geben, welche gefunden werden muss. Diese muss der Spieler erraten, um zu gewinnen.

a) Lade dir eine der beiden Dateien `Battleship.java` oder `Battleship_easy.java` von der Tutoriumsseite herunter (letztere beinhaltet einige Hilfestellungen). Die Positionen, an denen später der Code eingefügt werden soll, sind mit Kommentaren markiert.

b) Zunächst die Vorbereitungen beim Spielstart („preparations“): Erstelle das Spielfeld (hier bieten sich entweder ein int-Array oder ein boolesches Array an). Platziere nun das Schiff auf den von der `Math.random()`-Methode zufällig errechneten Koordinaten.

Tipp: Ändere an der entsprechenden Stelle im Array einfach den Wert, sodass du sie später mit einer einfachen if-Abfrage wiederfinden kannst!

c) Vervollständige nun die Spieleroutine („game routine“): Lass den Spieler zunächst die x-Koordinate eingeben und speichere sie in eine Variable. Dann lass ihn die y-Koordinate eingeben und verfahre ebenso. Wenn du beide Koordinaten hast, überprüfe, ob an der angegebenen Stelle das Schiff platziert wurde!

Hat der Spieler einen Treffer gelandet, beende das Spiel mit einer Siegesmeldung. Ging der Schuss ins Wasser, lass den Spieler eine neue Koordinate eingeben. Die Ausgabe könnte beispielsweise so aussehen:

```
Bitte X-Koordinate angeben: 2
Bitte Y-Koordinate angeben: 3
Feuere auf 2, 3: Daneben...
Bitte X-Koordinate angeben: 5
Bitte Y-Koordinate angeben: 4
Feuere auf 5, 4: Treffer!
```

Aufgabe 2: String-Methoden

Sich ein neues Passwort für einen neuen PC-Account oder ein soziales Netzwerk auszudenken, ist meistens eine knifflige Aufgabe. Viele Menschen wählen ihre Passwörter nach den Anfangsbuchstaben eines Satzes oder einer Zeile aus ihrem Lieblingslied, sodass sie es sich leicht merken können. „And she's buying a Stairway to Heaven“ wird so beispielsweise zu „AsbaStH“.

Wir wollen uns eine Methode `getPassword(String satz)` schreiben, das uns derartige Passwörter generiert, wenn wir einen entsprechenden Satz eingeben. Generiert werden soll dabei allerdings zu einem gegebenen Satz ein Passwort, welches aus den *letzten* Buchstaben bzw. Zeichen der Wörter des Satzes gebildet wird. Dabei soll bei Buchstaben pro Zeichen *zwischen Groß- und Kleinschreibung gewechselt* werden.

Als Wörter nehmen wir der Einfachheit halber alle durch ein oder mehrere Leerzeichen getrennten Folgen von Zeichen an. "And she's buying a Stairway to Heaven." besteht somit aus den Teilwörtern "And" "she's" "buying" "a" "Stairway" "to" "Heaven.". Die Rückgabe zum Aufruf:

```
getPassword("And she's buying a Stairway to Heaven.");
```

wäre also:

```
DsGaYo.
```

a) Schreibe das Gerüst für die Klasse `PasswordGenerator` und die Methode `getPassword(String)`.

b) Nutze String-Methoden, um den übergebenen Satz in seine Wörter zu zerlegen. Welche String-Methode benötigst du dafür und welchen Datentyp bekommst du zurück?

Erstelle dir außerdem einen leeren Passwort-String. Diesen füllst du später Stück für Stück mit den gesuchten Buchstaben auf.

c) Iteriere nun über die Wörter des Satzes. Welche String-Methoden und Datentypen benötigst du, um an ihre letzten Buchstaben zu kommen? Füge diesen zum Passwort-String hinzu.

d) BONUS: Baue eine Abfrage ein, welche vor dem Hinzufügen des Buchstabens dafür sorgt, dass dieser abwechselnd als Groß- bzw. Kleinbuchstaben hinzugefügt wird!

Aufgabe 3: Dateien einlesen und bearbeiten

Heute geht es darum, Dateien einzulesen und Text mit regulären Ausdrücken zu bearbeiten. Zu diesem Zweck findest du die Textdatei `poe.txt` auf der Tutoriumsseite. Diese wollen wir mit einer Klasse `FileEditor` zeilenweise einlesen und dabei alle Vorkommen eines bestimmten Wortes durch ein anderes Wort ersetzen. Welche Worte das sind, soll der Benutzer durch Kommandozeilenargumente selbst angeben können.

Die modifizierten Zeilen sollen mittels der Klasse `StringBuilder` wieder zusammengesetzt und ausgegeben werden.

Ein Beispielaufruf mit einer fiktiven Datei (nicht im Lieferumfang enthalten):

```
java FileEditor alleMeineEntchen.txt Wasser Bierglas
```

Und die Ausgabe:

```
Alle meine Entchen schwimmen auf dem See
schwimmen auf dem See.
Köpfchen in das Bierglas
Schwänzchen in die Höh'.
```

Wenn du Hilfe brauchst, steht ein (recht schmales) Gerüst zur Klasse FileEditor auf der Tutoriumsseite bereit.

a) Erstelle dir die Klasse `FileEditor` und eine Main-Methode, welche genau drei Kommandozeilenargumente nimmt: Erstens den Namen der einzulesenden Datei, zweitens das Wort, welches später im Text ersetzt werden soll, und drittens das Ersatzwort.

b) Initialisiere nun alle wichtigen Objekte und Variablen, die du brauchen wirst: Den `BufferedReader` zum Einlesen der Datei, den `StringBuilder` zum Zusammenfügen der Zeilen etc.

c) Lies jetzt die Datei mit Hilfe einer Schleife zeilenweise ein. Welche Methoden benötigst du dafür? Auf welche Exceptions stößt du, und warum? Fange sie mit einem try-/ catch-Block ab.

Füge die eingelesene Zeile zum `StringBuilder` zu und lasse dir den Text zur Kontrolle einmal ausgeben. Achtung: Zeilenumbrüche gehen beim Einlesen verloren, die musst du selbst wieder einbauen!

d) Es ist sinnvoll, den Such- und Ersetzungsvorgang durchzuführen, bevor die eingelesene Zeile in den `StringBuilder` wandert, sonst musst du später den ganzen Text erneut durchlaufen. Erstelle dir hierzu ein Pattern mit dem vom Benutzer übergebenen Kommandozeilenargument. Wenn der String in der aktuellen Zeile gefunden wird, ersetze ihn durch den zweiten übergebenen String.

e) Gib den fertig bearbeiteten Text auf die Konsole aus. Er sollte im gleichen Format sein wie der Ursprungstext.

BONUS: Schreibe den fertig bearbeiteten Text wieder zurück in eine Datei. Tipp: Zum Schreiben in Dateien gibt es die Klassen `BufferedWriter` und `FileWriter`, die ganz ähnlich funktionieren wie die verwandten `BufferedReader` und `FileReader`.

Aufgabe 4: Dateien schreiben mit dem richtigen Encoding

Nun wollen wir das Ganze noch einmal mit der deutschen Übersetzung der Geschichte von Edgar Allen Poe versuchen. Im Deutschen haben wir das Problem, dass wir einige Sonderzeichen (z.B. Umlaute) besitzen, die in anderen Sprachen so nicht vertreten sind. Deshalb ist die Wahl des richtigen Encodings beim Einlesen und Schreiben von Dateien sehr wichtig.

a) Hole dir die Datei `poe_deutsch.txt` von der Tutoriumsseite! Schaue sie dir an – wo werden wahrscheinlich Probleme auftauchen? Versuche, sie mit deinem Programm aus Aufgabe 3 einzulesen (aber auf der Kommandozeile und ohne Eclipse)! Falls du die Aufgabe 3 nicht bearbeitet hast, findest du eine Beispiellösung auf der Tutoriumsseite.

b) Schreibe dein Programm so um, dass es den deutschen Text korrekt einliest. Welches Encoding kannst du zu diesem Zweck benutzen?

c) Gib den Text nun in eine neue Datei aus! Auch hier muss das Encoding, mit welchem der Text geschrieben werden soll, beachtet werden.

Aufgabe 5: Klassen und Objekte

Erinnerst du dich an die Aufgabe 1? Dort haben wir uns ein kleines „Schiffe versenken“ selbst programmiert. Dieses Spiel wollen wir uns als Inspiration für die nächste Aufgabe nehmen. Angenommen, wir wollten das Spiel ein wenig erweitern und mehrere Schiffe einführen, die sich in den folgenden Attributen unterscheiden:

- Ihr Name.
- Ihre Größe auf dem Spielfeld.
- Ihre Koordinaten auf dem Spielfeld (Tipp: hier ist es sinnvoll, die Startkoordinaten anzugeben. Die Endkoordinaten bei einem langen Schiff ergeben sich durch die Größe).
- Ihr Status, welcher angibt, wie stark beschädigt das Schiff schon ist (als int).
- Ihre Punktzahl, die der Spieler bekommt, wenn er sie versenkt.

a) Schreibe eine Klasse `Ship` und den zugehörigen Konstruktor, mit dem du Schiffs-Objekte nach den obigen Vorgaben generieren kannst. Du sollst ein neues Schiff mit dem folgenden Aufruf erstellen können:

```
new Ship(„Schlachtschiff“, 4, 50)
```

b) Außerdem soll jedes Schiff die folgenden Methoden anbieten:

- getName, getSize, getPoints – um die Attribute des Schiffs auszulesen.
- setName, setPoints – um die Attribute nachträglich verändern zu können.
- SetCoordinates – um später seine Koordinaten auf dem Feld zu speichern. Tipp: Du musst auch überprüfen, ob das Schiff aus dem Spielfeld herausragt!
- Hit – um bei einem Treffer den Gesundheitsstatus des Schiffs zu reduzieren.
- getStatus – um zu überprüfen, wie stark beschädigt das Schiff schon ist.
- isSunk – um zu überprüfen, ob das Schiff schon versenkt wurde.

Benötigst du zusätzliche Attribute, um all diese Methoden implementieren zu können?

c) Schreibe eine neue Klasse mit einer Main-Methode, um deine Schiffsklasse zu testen.

Aufgabe 6: Vererbung

Lass uns nun eine kleine Klassenhierarchie für unser Spiel basteln: Die Klasse `Ship` soll dabei zu einer abstrakten Klasse werden. Sie wird die Schablone für die unterschiedlichen Schiffe, die es in unserem Spiel geben soll.

Jetzt ist Kreativität gefragt: Welche Schiffe möchtest du gerne in deinem Spiel haben? Ein Schlachtschiff mit einer Größe von 4 Feldern? Ein kleines U-Boot auf nur einem Feld, so wie in Aufgabe 1? Oder vielleicht sogar etwas, was es im „normalen“ Schiffe versenken nicht gibt: Einen Flugzeugträger mit den Maßen 2x3? Was könnten die Besonderheiten jedes dieser Schiffe sein?

a) Ändere die Klasse `Ship` in eine abstrakte Klasse. Definiere einen Konstruktor. Welche Methoden werden später in allen Schiffen benötigt? Definiere sie als abstrakte Methoden mit dem Schlüsselwort `abstract`!

b) Implementiere für jedes deiner Schiffe eine eigene Klasse. Denke daran, dass du den Konstruktor der Oberklasse `Ship` mit dem Aufruf `super()` aufrufen kannst.

BONUS: Überlege dir darüber hinaus mindestens eine Besonderheit, welche jedes Schiff ausmacht, und implementiere sie. Sei kreativ! Keine Angst, du musst nicht die Funktionalität implementieren, nur die Idee dahinter!

c) Erstelle dir eine Main-Methode und teste deine Schiffe, indem du dir aus deinen Klassen Objekte erstellst. Das könnte zum Beispiel so aussehen:

```
public class Main {
	public static void main(String[] args) {

		// instantiate ships
		Battleship s1 = new Battleship(„Schlachtschiff 1“, 4, 50);
		Submarine s2 = new Submarine(„U-Boot 1“, 1, 150);

		// place ships
		s1.setCoordinates(3, 4, „diagonal“);
		s2.setCoordinates(6, 7, „horizontal“);

		// do something with them
		System.out.println(s1.getName, s1.getCoordinates);
		s1.getStatus();
		s2.useRadar(2, 3);                          // BONUS
	}
}
```

Aufgabe 7: Collections

Ein Systemadministrator benötigt ein Programm, das die Zugangsdaten der Benutzer (Studenten) eines Systems speichert. Dieses Programm soll die Möglichkeit bieten, neue Benutzer anzulegen und zu ihnen jeweils die folgenden Informationen zu speichern: den Nachnamen, einen speziellen Benutzernamen, die Matrikelnummer und ein Passwort.

Der Administrator gibt beim Erstellen eines neuen Benutzers seinen Namen, die Matrikelnummer und das Passwort mit. Das Programm soll dann, beim Erstellen des neuen Benutzers automatisch einen Benutzernamen kreieren, der sich nach folgendem Schema zusammensetzt: Die ersten zwei Buchstaben des Namens und die ersten vier Ziffern der Matrikelnummer (Beispiel: aus `Schmidt, 123456` wird `sc1234`).

a) Erstelle eine neue Klasse `AccountManager`. Diese Klasse soll zwei `HashMaps` und eine `ArrayList` als private Datenfelder besitzen, in denen die Zuordnungen Name – Benutzername und Name – Passwort gespeichert werden. Die Liste soll einfach eine Liste aller angelegter Benutzer enthalten. Der Konstruktor soll diese Felder (HashMaps) initialisieren.

b) Erstelle die Methode

`createAccount(String name, String matrikelnummer, String passwort)`

welche die folgenden Aufgaben übernimmt:

- Zunächst den Benutzernamen durch eine private Methode `createUsername()` erstellen lassen
- Dann jeweils einen Eintrag in die beiden HashMaps und die ArrayList machen.

c) Erstelle außerdem die weiteren öffentlichen (public) Methoden:

- `getPassword(String name)` – gibt das Passwort zu einem Namen zurück
- `getUsername(String name)` – gibt den Benutzernamen zu einem Namen zurück
- `printUserData(String name)` – druckt alle Angaben (Name, Benutzername und Passwort) eines Benutzers auf die Konsole aus.
- `getAllUsers()` - druckt alle angelegten Benutzer mit ihren Namen auf die Konsole aus.

d) Die ArrayList könntest du auch weglassen. Wie kannst du dann alle Benutzernamen auf einmal ausgeben lassen?

e) Wann treten bei der Nutzung deines Programms Probleme auf, die durch die Verwendung von HashMaps bedingt sind?