Objektsammlungen I

Lernziele

- Sie sammeln Erfahrung im Umgang mit der Objektsammlung ArrayList.
- Sie können einfache Problemstellungen, welche die Speicherung, Durchforstung und (selektive) Entfernung einer beliebigen Anzahl von Objekten erfordert, selbständig lösen.
- Sie setzen Klassen zur Strukturierung des Codes ein.

Aufgabe 1 (auf Papier!)

Analysieren Sie den folgenden Code und finden Sie den Fehler. Die ArrayList `eintraege` ist vom Typ ArrayList<Integer>.

```
int anzahl = eintraege.size();
for(int i = 0; i < anzahl; i++) {
      if(eintraege.get(i) < 100) {
            eintraege.remove(i);
      }
}
```

Die ArrayList ist noch leer. Deshalb gibt es noch kein Resultat. Wenn die ArrayList Daten hättte, dann wäre i für 100 Iterationen kleiner als 100. Dass heisst, dass i für 100 mal Einträge in jedem Fall entferen würde. Und zwar zuerst die erste Position, dann die 2te, dann die 3te usw. Dies ergibt keinen Sinn. Der Code Zunktioniert zwar, würde aber ein unbrauch-bares Resultat liefern.

- Tritt der Fehler immer auf? Wenn ja: Warum? Wenn nein: In welchen Fällen tritt er nicht auf?

  Die ArrayList ist noch leer. Das heisst, dass heisst Sie ist null.

  Deshalb gibt es eine Nullpointer Exception.whil

- Handelt es sich beim Fehler um einen Laufzeitfehler oder Kompilierfehler?

  Nullpointer Exception, weil die ArrayList leer ist.

Aufgabe 2 (auf Papier!)

Vervollständigen Sie die Methode `entferneFlaschenGleichenInhalts` der Klasse `Flaschenverwaltung`. Die Methode soll alle Flaschen aus der Flaschenverwaltung löschen, welche den gleichen Inhalt haben wie die als Parameter übergebene `Flasche`. Bevor Sie mit dem Vervollständige starten, beantworten Sie aber bitte noch folgende Fragen:

- Welchen Schleifentyp setzen Sie für Ihre Lösung ein?

  while-Schleife, welche eine IF-Schleife enthält.

- Wieso haben Sie diesen Schleifentyp gewählt?

Die while-Schleife kann mithilfe eines Iterators einzelne Inhalte "Suchen". Mit der enthaltenen IF-Schleife kann man nun mithilfe von 'contains' und 'remove' gefundene Duplicate löschen.

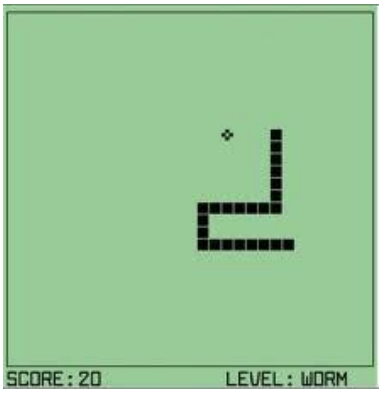
- Haben Sie einen `Iterator` eingesetzt? Wieso, resp. wieso nicht?

  Ja, dieser hat die "hasNext()" Funktion

```
public class Flasche {
	private int inhalt;
	public boolean istInhaltGleich(Flasche flasche) {
		return flasche.inhalt == this.inhalt;
	}
}

public class Flaschenverwaltung {
	private ArrayList<Flasche> flaschen;
	...

	public void entferneFlaschenGleichenInhalts(Flasche flasche) {

		Iterator it = flaschen.iterator();
		while (it.hasNext(){
		    Flasche inhalt = it.next();
		    if(flaschen.contains(inhalt)){
		       flaschen.remove()}
		}
	}
}
```

Aufgabe 3

Forken Sie für diese Aufgabe das Projekt
https://github.engineering.zhaw.ch/prog1-kurs/04_Praktikum-1_Snake.
Nutzen Sie BlueJ um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten.



In diesem Praktikum modifizieren Sie ein Grundgerüst für das Spiel **Snake** (siehe. http://de.wikipedia.org/wiki/Snake). Im Gegensatz zum echten Snake Spiel, wo sich die Schlange selbständig mit konstanter Geschwindigkeit fortbewegt und der Benutzer die Richtung steuert, bewegt sich die Schlange in unserem Grundgerüst nicht selbständig fort. Das Spiel hält nach jedem Bewegungsschritt an und wartet auf die Bewegungsangabe vom Benutzer. Diese Beschränkung werden wir nicht beheben.

Neben dieser Beschränkung hat das Grundgerüst aber noch ein paar weitere Beschränkungen, die es nun zu beheben gilt. Lesen Sie die nachfolgenden Teilaufgaben durch und lösen Sie diese anschliessend der Reihe nach. Jede Teilaufgabe macht unser Grundgerüst etwas mehr „Snake“-like.

a) Machen Sie sich mit dem Code vertraut. Folgende Fragen sollten Sie für sich beantworten können: Wie wird die Schlange gespeichert? Was passiert, wenn die Schlange bewegt wird? Wie wird das „Wachsen“ der Schlange realisiert?

   Hinweis: Bei der Bewegung und dem Wachstum der Schlange werden die dynamischen Eigenschaften der Liste, welche Objekte vom Typ Point hält, ausgenutzt. Nachfolgend ist ein Snapshot einer Schlange gezeigt, die sich bewegt und einmal um ein Element wächst:

```
|1,2| -> |1,3| -> |1,4| -> |2,4| -> |2,5|:)
|1,3| -> |1,4| -> |2,4| -> |2,5| -> |2,6|:)
|1,3| -> |1,3| -> |1,4| -> |2,4| -> |2,5| -> |2,6|:)
|1,3| -> |1,4| -> |2,4| -> |2,5| -> |2,6| -> |2,7|:)
```

b) Erweitern Sie das Grundgerüst so, dass bei Spielstart eine mittels der Methode `setAnzahlGoldstuecke` definierbare Anzahl Goldstücke auf dem Spielfeld liegen. Wird nichts angegeben, sollen 10 Goldstücke auf dem Spielfeld liegen.

   Hinweis: Studieren und nutzen Sie die Methode `erzeugeZufallspunktInnerhalb` der Klasse `Spielfeld` um Goldstücke zu platzieren.

c) Goldstücke sollen nun auch einen Wert haben, konkret ein zufälliger Wert zwischen 1 und 5. Die Schlange soll beim Einsammeln eines Goldstücks so viel wachsen, wie das Goldstück wert ist und der Wert des eingesammelten Goldstücks soll ausgegeben werden.

   Hinweis:

   - Die Methode `erzeugeZufallspunktInnerhalb` liefert Ihnen die notwendigen Hinweise, wie Sie eine zufällige Zahl zwischen 1 und 5 erzeugen können.
   - Delegieren Sie das Wachsen um x Elemente an die Schlange.

Für Fortgeschrittene (optional):

- Machen Sie das Spiel schwieriger, indem Sie die Schlange stärker wachsen lassen oder indem Sie das Programm um Hindernisse (z.B. Wände) im Spielfeld, in welche die Schlange nicht hineinkriechen darf, ergänzen.
- Verwenden Sie für die Textausgabe eine Bibliothek wie **lanterna** (http://code.google.com/p/lanterna/wiki/UsingTerminal), welche es erlauben im Textausgabefenster zu „Navigieren“: Also z.B. das aktuell dargestellte Spielfeld „in-place“ zu überschreiben, indem vor der nächsten Ausgabe wieder an den Anfang gesprungen wird. Java erlaubt das „Zurückbewegen“ mit Bordmitteln leider nicht.

  Hinweis: Für das Einlesen der Tastendrücke mit **lanterna** können Sie folgendes Codestück verwenden.

```
Key key;
do
{
    key = terminal.readInput();
} while (key == null);
return key.getCharacter();
```