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.
  Dazu müssen Sie die Aufgabe aus dem Kapitel «Empfohlen» lösen.

Aufgabe 1 (auf Papier!)

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

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

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

  Sobald eine Zahl unter 100 ist, tritt dieser Fehler auf. Der Fehler ist, das die Folge-zahl, da die aktuelle Zahl gelöscht wird, nicht kontrolliert.

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

  Es hat ein Laufzeitfehler.
  ArrayList wird immer kleiner, aber die Anzahl nicht.
  get(i) ist irgendwann out of Bound.

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 interator

- Wieso haben Sie diesen Schleifentyp gewählt?

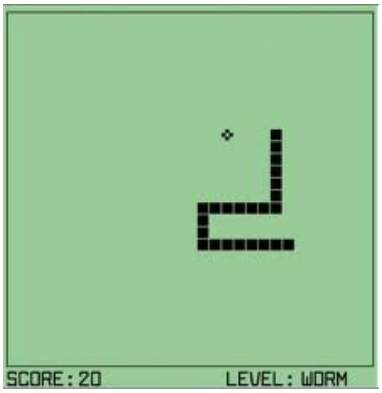
  Damit kann man relativ einfach Flaschen in der ArrayListe löschen ohne Probleme.

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

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

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

	public void entferneFlaschenGleichenInhalts(Flasche flasche) {


		Iterator<Flasche> iterator = flaschen.iterator();
	}	while(iterator.hasNext()){
}		if(flasche.istInhaltGleich(iterator.next()){
			iterator.remove();
		}
	}
```

Aufgabe 3

Forken Sie für diese Aufgabe das Projekt https://github.zhaw.ch/prog1-kurs/04_Praktikum-1_Snake.



In diesem Praktikum modifizieren Sie ein Grundgerüst für das Spiel **Snake** (siehe Snake (Computerspiel) – Wikipedia). 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) Jede Codeänderung kann neue Fehler einführen. Beantworten Sie folgende Fragen:

- Funktioniert Ihr Programm noch immer so zuverlässig wie vorher?
- Haben Sie die Anforderung exakt umgesetzt?

Hinweis: Sind alle Werte für den Parameter bei `setAnzahlGoldstuecke` zulässig? Generieren Sie exakt die gewünschte Anzahl Goldstücke, oder könnten zwei Goldstücke «aufeinander» liegen?

d) Diese Aufgabe lösen Sie nur auf Papier.
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.

- Wenn Sie dies umsetzen wollen, warum benötigen Sie eine zusätzliche Klasse?

Die Goldstücke sind in einer ArrayListe gespeichert. Die Goldstücke sollten aber selbst eine eine ArrayListe haben mit einem zufälligen Länge zwischen 1 und 5

- Welche Datenfelder und Methoden hat diese neue Klasse?

Datenfelder
-

Für Fortgeschrittene (optional):

- Bauen Sie die Goldstücke-Klasse von der vorhergehenden Aufgabe in den Code ein.

  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.
  - Finden Sie heraus, warum Sie nun beim Einsammeln des ersten Goldstücks mit einem Wert über Eins das Spiel verlieren.