ÜBUNGS-BLOCK 11
AUFGABEN

Aufgabe 1:

Gegeben ist die folgende Klasse:

```
public class Spielstein {

	private final Color farbe;

	public Spielstein(Color farbe) {
		this.farbe = farbe;
	}

	public Color getFarbe() {
		return farbe;
	}
}
```

Diese Klasse soll für ein Tetris-ähnliches Spiel genutzt werden. Im Spiel steuert der Spieler Paare von jeweils zwei Spielsteinen, welche von oben herabfallen und passend in das Spielfeld eingebaut werden müssen. Den Spielsteinen wird bei ihrer Erzeugung jeweils zufällig eine bestimmte (sich nach der Erzeugung niemals wieder ändernde) Farbe zugeordnet. Die möglichen Farben für einen Spielstein sind: Rot, Grün, Blau, Gelb. Als Datentyp für die Instanz-Variable der Farbe wird die in der Java Bibliothek definierte Klasse `java.awt.Color` verwendet, da sich solche Color-Objekte über bestimmte Methoden der Java-Bibliothek zum Malen der Spielsteine eignen. Zum besseren Verständnis hier ein Ausschnitt der Methode, die diesen Zeichenvorgang übernimmt:

```
@Override
protected void paintComponent(Graphics g) {
	// zeichne 50x50 Pixel großen Stein in seiner Farbe:
	g.setColor(spielstein.getFarbe());
	Point pos = calculatePosition(spielstein);
	g.drawRect(pos.x, pos.y, 50, 50);
	// ...
}
```

Durch spezielle "Bomben", welche sich von der Klasse Spielstein ableiten, kann der Spieler Spielsteine der entsprechenden Farbe zur Explosion bringen. Um auszuwerten, ob ein Spielstein auf Grund solch einer Bombe explodieren soll oder nicht, muss seine Farbe mit der Farbe der Bombe abgeglichen werden:

```
	for(Spielstein s : spielsteine){
		if(bombe.getFarbe().equals(s.getFarbe())){
			// ...
		}
	}
```

Die oben gezeigte Spielstein-Klasse lässt sich im Hinblick auf die Beschaffenheit und Funktionen der Farbe eines Spielsteins im Programm noch optimieren. Fällt dir eine Lösung ein, die der Aufgabenstellung des Spiels besser gerecht wird und es uns als Programmierern leichter macht, mit Spielstein-Objekten zu arbeiten? Implementiere ein geschickteres Design für diesen Anwendungsfall.

Aufgabe 2

In dieser Aufgabe sollst du ein paar Fragen zum Thema Generics beantworten:

1. Weshalb lässt sich der Typ-Parameter einer Klasse oder eines Interfaces nicht für statische Variablen oder Methoden verwenden?

2. Wozu sind Wildcards gut?

3. Entscheide für jeweils jede Zeile des nachfolgenden Codes, ob sie kompiliert (mit Begründung)

```
1. ArrayList<Object> listOne = new ArrayList<String>();
2. ArrayList<?> listTwo = new ArrayList<String>();
3. System.out.println(listTwo.size());
4. listTwo.add(new String());
```

Aufgabe 3

In dieser Aufgabe sollst du den praktischen Umgang mit Generics üben, und zumdem eine Vorstellung über die interne Struktur von Collections bekommen. Dazu sollst du eine Klasse "MyArrayList" Implementieren, die der Klasse ArrayList aus dem Java Collections Framework (in ihren wichtigsten Beschaffenheiten und Methoden) gleicht.

Folgende Anfordernungen müssen von der Klasse erfüllt werden:

• Die Klasse muss generisch sein und einen Typ-Parameter zur Verfügung stellen
• Die Klasse muss eine Methode contains(X) bereitstellen, um für ein zum Typ-Parameter passendes Element auszuwerten, ob sich dieses in der Liste befindet. Die Methode returned einen boolean: true, falls sich das Element in der Liste befindet, false falls nicht. Für den "Match" auf ein Element soll die Gleichheit per equals() herangezogen werden.
• Die Klasse muss eine Methode add(X) bereitstellen, um ein zum Typ-Parameter passendes Element in die Liste aufzunehmen. Das Hinzufügen von null-Referenzen ist nicht erlaubt und führt zu einer entsprechenden Exception
• Die Klasse muss eine Methode remove(X) bereitstellen, um ein zum Typ-Parameter passendes Element aus der Liste zu entfernen. Die Methode returned einen boolean: true, falls das Element geslöscht wurde, und false, falls sich das Element nicht in der Liste befand. Das Übergeben von null-Referenzen ist nicht erlaubt und führt zu einer entsprechenden Exception
• Die Klasse muss eine Methode remove(int) bereitstellen, um ein Element an der übergebenen Stelle (Index) der Liste zu entfernen. Wird ein Index übergeben, der außerhalb der gültigen Indizes für die Liste liegt (< 0 oder >= Anzahl Elemente) soll eine entsprechende Exception geworfen werden. Die Methode returned das soeben entfernte Element.
• Die Liste darf zu keiner Zeit "Lücken" enthalten: Beim Entfernen eines Elements rücken alle nachfolgenden Elemente automatisch um ein Fach nach vorne
• Die Klasse muss eine Methode get(int) bereitstellen, die das Element an dem übergebenen Index unter Verwendung des Typ-Parameter-Datentyps zurückliefert. Wird ein Index übergeben, der außerhalb der gültigen Indizes für die Liste liegt (< 0 oder >= Anzahl Elemente) soll eine entsprechende Exception geworfen werden.
• Die Klasse muss die Methoden clear() - Löschen aller Elemente in der Liste -, isEmpty() - Prüfung, ob sich mindestens ein Element in der Liste befindet oder nicht -, und size() - Ausgabe der Anzahl der Elemente in der Liste - zur Verfügung stellen.
• Die Liste soll in ihrer Kapazität nicht festgelegt sein. Es sollen theoretisch also unendlich viele Elemente per add(T) hinzugefügt werden können. (Ohne dass der Nutzer solch einer Liste selbstständig weitere Methoden aufrufen muss!)

Selbstverständlich ist die Verwendung von Klassen aus dem Java Collections Framework in dieser Aufgabe nicht gestattet! Lediglich ein Array darf zur Speicherung der Elemente verwendet werden.

Teste deine Klasse und all ihre Methoden in einem Programm "MyArrayListTest".

Anmerkung: Lasse dir bei dieser Aufgabe genügend Zeit! Es handelt sich um die komplexeste Klasse, die du bisher geschrieben hast (100+ Zeilen Code). Versuche zunächst, eine funktionierende Version zu implementieren (Testen!). Überlege anschließend, ob du den Code in Bezug auf die Lesbarkeit oder Performance noch verbessern könntest.

Aufgabe 4

Gegeben ist das folgende Programm:

```
public static void main(String[] args) {
      Set<Short> s = new HashSet<Short>();
      for (short i = 1; i <= 100; i++) {
            s.add(i);
            s.remove(i - 1);
      }
      System.out.println(s.size());
}
```

Was gibt dieses Programm bei der Ausführung aus, und *weshalb* gibt es das aus, was es ausgibt?

Wichtig: Damit du keine falschen Annahmen machst, sieh dir die Dokumentation zu der Klasse HashSet an: http://download.oracle.com/javase/6/docs/api/java/util/HashSet.html

Anmerkung: Natürlich soll der Code nicht ausgeführt werden - du sollst durch Überlegung auf das Ergebnis kommen. Man kann sich denken, dass die Ausgabe des Programms nicht das ist, was man intuitiv annimmt. In diesem Code steckt ein vom Programmierer wohl nicht beabsichtigter Bug, den es zu finden gilt.