University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatik I – Eprog HS13

Übung 11

1 Intro

Gegeben ist das folgende Code-Snippet, welches sich in diesem Zustand nicht kompilieren lässt:

```
1  import java.util.ArrayList;
2
3  public class GenericsMain {
4    public static void main(String[] args) {
5      ArrayList list = new ArrayList();
6      list.add(333);
7      int s = list.get(0);
8      System.out.println(s);
9    }
10 }
```

Listing 1: `GenericsMain.java`

1. Warum lässt sich dieses Code-Snippet nicht kompilieren?
2. Korrigieren Sie den Code, so dass er kompiliert. Es gibt zwei Lösungen: eine mit und eine ohne die Anwendung von Generics. Notieren Sie beide Lösungen.

2 Generische Set-Implementation

Generics werden in Java häufig angewendet, um Datenstrukturen, also Behältnisse (Collections) für mehrere Elemente, abzubilden. In dieser Aufgabe implementieren Sie eine generische Lösung zum Speichern von Objekten beliebigen Typs in einer "Set"-Datenstruktur. Sets sind insofern ähnlich wie Arrays oder Listen, als dass sie eine gewisse Anzahl Elemente abspeichern. Allerdings haben die Elemente in einem Set keine bestimmte Reihenfolge, und jedes Element kann nur einmal vorkommen. Ein Beispiel: Hätte man ein leeres Set, und fügte man dann die Elemente 2, 1, 3, 3, 4, 2 und 3 hinzu, so würde das Set die Elemente 1, 2, 3 und 4 enthalten, und zwar ohne eine bestimmte Reihenfolge.

1. Implementieren Sie eine generische Klasse `Set<T>`, in welcher Sie Objekte eines beliebigen Typs abspeichern können. Gegeben ist der folgende `SetTestDriver`, mit dem Sie Ihre Set-Implementation testen können, und aus dem sich die benötigten Features erschliessen:

```java
public class SetTestDriver {
  public static void main(String[] args) {
    Set<String> a = new Set<String>(new String[] {"a1", "a2", "a1"});
    System.out.println(a);
    // Output: [a1, a2]
    Set<String> b = new Set<String>();
    b.add("b1").add("a1").add(a).add("b2");
    System.out.println(b);
    // Output: [b1, a1, a2, b2]
    Set<Object> c = new Set<Object>().add("c1").add(5).add(true);
    System.out.println(c);
    // Output: [c1, 5, true]
    System.out.println(c.contains("c1"));
    // Output: true
    System.out.println(c.size());
    // Output: 3
    System.out.println(c.remove(5));
    // Output: [c1, true]
  }
}
```

Listing 2: `SetTestDriver.java`

Hinweise:

- Wie sie sehen, soll Ihre Set-Implementation diverse Konstruktoren und `add`-Funktionen besitzen, die es ermöglichen, ein Set auf verschiedene Arten zu instanzieren und dem Set auf verschiedene Weise Elemente hinzuzufügen. Tip: Die Typensignatur für die `add`-Funktion, welche den Inhalt eines zweiten Sets hinzufügt, lautet `public Set<T> add (Set<T> other)`.
- Es ist auch erkennbar, dass die Methode `System.out.println` den Set-Inhalt anzeigen kann. Zu diesem Zweck müssen Sie in Ihrer `Set`-Klasse die `public String toString()` Methode überschreiben, welche die `toString()` Methoden aller Elemente aufruft, und die Ergebnisse dieser Aufrufe aneinanderreiht. Im `SetTestDriver` finden Sie den jeweiligen Output der `System.out.println`-Aufrufen. Ihre Lösung sollte die gleichen Ausgaben liefern.
- Ein besonderes Feature Ihrer Set-Implementation soll sein, dass jede `add` methode anstatt `void` das Set selbst zurückliefert. Dies ermöglicht die Aneinanderreihung der `add`-Aufrufe, wie sie im TestDriver ersichtlich sind.
- Es sollten keine Änderungen an dem `SetTestDriver` ausgeführt werden!