University of Zurich[UZH]

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

Informatik I – Eprog HS12

Übung 12

1 Aufgabe: Generics

1.1 Lernziele

1. Verwendung von Generics üben

1.2 Task 1

Formen Sie das folgende Code-Snippet um, so dass kein Cast mehr verwendet werden muss

```
1 List list = new ArrayList();
2 list.add("hello world");
3 String s = (String) list.get(0);
```

1.3 Task 2

Gegeben sind folgende zwei Klassen:

```
1  public class Car {
2
3    private int ps;
4
5    public Car(int ps) {
6      this.ps = ps;
7    }
8
9    public int getPS() {
10     return ps;
11   }
12 }
```

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class Highway {
5
6    public static void main(String[] args) {
7      List list = new ArrayList();
8      for (int i = 500; i < 503; i++) {
9        list.add(new Car(i));
10       System.out.println(list.get(i-500).getPS());
11     }
12   }
13
14 }
```

Begründen Sie ob dieser Code kompiliert und ohne Exceptions ausgeführt werden kann. Wenn nicht, ändern Sie den Code so um, dass er kompiliert und erfolgreich ausgeführt werden kann.

2 Aufgabe: FlipFlops

2.1 Lernziele

1. Verwendung des Observer Patterns üben und komplizierte indirekte Programmabläufe verstehen.
2. Verständnis von RS FlipFlops vertiefen

2.2 Aufgabenstellung

Aus der Vorlesung kennen Sie den Aufbau eines RS FlipFlops (M4-37). Sie sollen diesen Baustein nun unter der Verwendung des Observer Entwurfsmusters implementieren. Ein Entwurfsmuster ist einfach eine Art Musterlösung, die man in Abwandlung immer wieder gebrauchen kann. Das Observer Entwurfsmuster haben Sie bereits impliziet in der Übung 10 kennen gelernt. Das Zusammenspiel zwischen MessagePool und MailBox basiert auf diesem Muster. Beim Observer Entwurfsmuster registriert sich eine Klasse (Observer) mittels einer bestimmten Methode bei einer anderen Klasse (Subject). Der Observer übergibt eine Referenz auf sich selbst an die Registrierungsmethode des Subjects. Wenn nun ein bestimmtes Ereignis eintritt, ruft das Subject eine bestimmte Methode auf allen registrierten Observern auf. Der Observer kann nun auf dieses Ereignis reagieren. Ein einfaches Beispiel finden Sie in Listing 1.

Das Ziel ist es, basierend auf den 3 logischen Grundbausteinen Und, Oder und Inverter ein RSFlipFlop zu bauen. Die Herrausforderung ist es dabei, einen "realistischen" Signalfluss zu simulieren. Daher, wenn am Eingang ein Signal wechsel erfolgt, wird das neue Ausgangssignal an das Folgeelement weitergeleitet. Wenn wir mehrere Grundbausteine zu komplexeren Schaltungen verknüpfen wollen, müssen wir den Eingang eines Elementes, mit dem Ausgang eines anderen Elementes verknüpfen können. Diese Verknüpfung soll nun über das Observer Entwurfsmuster erfolgen.

Beispiel: AND-Gatter A hat Ausgang 1. OR-Gatter B soll Ausgang 1 als Eingang 1 erhalten. Was nun geschehen soll ist folgendes: Das OR-Gatter B erhält eine Referenz auf den Ausgang von AND-Gatter A, um das Signal abfragen zu können. Um nun zu erfahren, wenn sich das Signal am Ausgang (bzw. Eingang) ändert, registriert sich das OR-Gatter B als Observer beim Ausgang 1. Ausgang 1 ist also das Subject und informiert alle Observer, wenn sich sein Signal ändert. OR-Gatter B kann nun sein neues Ausgangssignal bestimmen und seinen eigenen Ausgang aktuallisiern.

```java
interface Observer{
   public void notify(String event);
}
interface Subject{
   public void register(Observer o);
}
class CalculatorObserver implements Observer {
   public void notify(String event){
      System.out.println(event + " was called");
   }
}
class Calculator implements Subject{
   private Observer myObserver;
   public void register(Observer o){
      myObserver = o;
   }
   private void add(){
      myObserver.notify("Add");
   }
}
```

Listing 1: Observer Beispiel

2.3 Implementierung

Sie finden als Anhang zu dieser Aufgabe folgende Klassen: Input.java, OrGate.java, TwoInGate.java, XORGate.java und XORTestDriver.java. Lesen Sie den Code und versuchen Sie zu verstehen, wie das Observerpattern verwendet wird und wie die Klassen interagieren. Sie werden bemerken, dass der Code nicht kompiliert werden kann, da das Interface InputChangeLister sowie die Klassen Inverter und AndGate fehlen. Erzeugen Sie diese. Die Klasse XORGate zeigt, wie die Basis Gatter verknüpft werden können, um eine komplizierteres Gatter zu erzeugen. Bauen Sie nach diesem Schema auch das RSFlipFlop in einer Klasse RSFlipFlop.

Hinweise:

1. Signallaufzeiten und Verzögerungen werden ignoriert.
2. Sie können leicht Endlosschleifen produzieren, dadurch das beim RSFlipFlop die Eingänge und Ausgänge über Kreuz verknüpft sind. Entsprechend lösen nur Signaländerungen eine Weiterleitung an die Observer auslösen.
3. Ihr RSFlipFlop werden Sie in einem sinnvollen Zustand initialiseren müssen (zum Beispiel zurückgesetzt).
4. Tipp: Zeichnen Sie sich das XORGate auf ein Blatt Papier und gehen Sie damit den Programmablauf durch.

2.4 RSLatch

Impelementieren Sie nun noch ein RSLatch. Wie Sie auf Folie M4-43 sehen, ist das nicht mehr sehr kompliziert, wenn Sie schon das RSFlipFlop haben. Erzeugen Sie dazu eine Klasse RSLatch

welches über den Konstruktor je einen Input s, einen Input r sowie einen Input t bekommt. Der Input t stellt den Takt dar. Das RSLatch verknüpft nun einfach eine RSFlipFlop Instanz mit zwei AndGate Instanzen. Das RSLatch soll keine Subklasse von RSFlipFlop sein (Auch wenn sich diese Idee evtl. auf den ersten Blick anbietet). Denke Sie darüber nach, ob es denn sinnvoll wäre, eine RSLatch Instanz als RSFlipFlop an eine Methode übergeben zu können, oder ob dies zu Probleme führen könnte.

2.5 TestDriver

Schreiben Sie eine TestDriver, der Ihre beiden FlipFlops mit verschiedenen Eingangsbelegungen testet. Orientieren Sie sich am XORTestDriver.