Programmieren 1 – Schmiedecke

Laborexperiment X5 – Nachrichtenakanal

In diesem Labor schreiben Sie wieder (fast) alles selbst. Nur die sehr einfache grafische Oberfläche zum Anzeigen der Ergebnisse gebe ich Ihnen vor. Auch wenn Sie recht viele Klassen schreiben sollen, ist der Programmieraufwand verhältnismäßig klein, aber es gibt wieder Raum für Experimente, mit denen Sie das Verhalten unabhängiger animierter Objekte verstehen lernen können.

Lernziele: Datenbehälter, Exceptions und Animierte Objekte

Vorbereitung

Lesen Sie den gesamten Aufgabentext, ehe Sie beginnen!

Nachbereitung: Laborprotokoll

Der Quelltext der Programme ist nicht besonders aufwändig, aber erläutern Sie bitte für sich selbst noch einmal, ***wodurch ein Objekt animiert wird***. Sehr wichtig ist diesmal eine Darstellung der gesamten ***Programmstruktur*** (welche Klassen kooperieren?) und eine **Deutung der Ergebnisse** der verschiedenen Experimente.

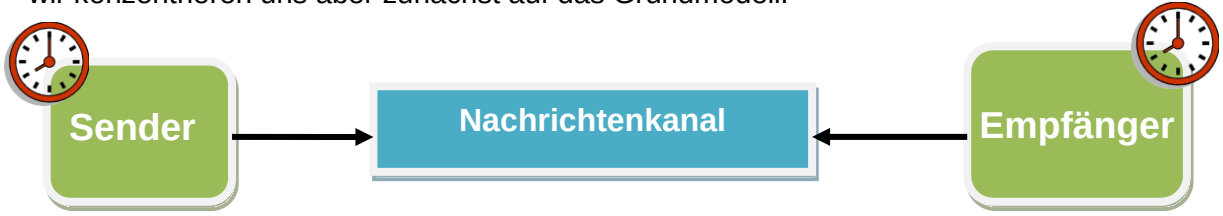
Außerdem sollen Sie ***jede Klasse zunächst separat testen*** – erläutern Sie, was Sie testen wollen. *(„Take care of the edges, the middle will see to itself“)*

Einführung:

Das Prinzip hinter dieser Aufgabe ist sehr einfach und sehr grundlegend, nämlich die asynchrone Kommunikation, wie wir sie z.B. von Emails her kennen:

Zwei Teilsysteme kommuizeren miteinander über einen Nachrichtenkanal. Das eine System schreibt Nachrichten hinein, wann es will, das andere liest die Nachrichten aus, wann es will. Der Nachrchtenkanal sorgt dafür, dass der Empfänger die Nachrichten in der Reihenfolge erhält, in der sie gesendet wurden.

Die Teilsysteme können z.B. Netzwerkknoten oder Prozesse sein. In unserem Fall sind es aktive Java-Objekte („animierte Objekte“, symbolisiert durch die Uhr). Und natürlich können solche Teilsysteme kreuz und quer über verschiedene Kanäle miteinander verknüpft sein, wir konzentrieren uns aber zunächst auf das Grundmodell.



1. Das Grundmodell besteht aus einer Queue (Warteschlange) für Strings und zwei animierten Objekten, Sender umd Empfänger. Das erste schreibt Texte in die Warteschlange, das zweite liest die Texte aus und zeigt sie an.

   - Das animierte Objekt Sender liest Zeichenketten von der Konsole (Console.readln()) und schreibt sie in die Warteschlange.
   - Das animierte Objekt Empfänger liest Zeichenketten aus der Warteschlange und schreibt sie in ein GUI-Fenster (nicht auf die Konsole, damit man die Ausgaben verschiedener Empfänger unterscheiden kann). Das Gui-Fenster ist ein Objekt der (fertigen) Klasse TextGUI, in das Sie mit der Methode write()Texte ausgeben können.
   - Die Warteschlange speichert die Texte zwischen und fungiert so als Kanal.

2. Für die Warteschlange sollen Sie verschiedene Implementierungen verwenden (können). Alle müssen das Interface TextContainer implementieren (als FIFO!)

```
public interface TextContainer {
   public void enter(String s);  // evtl. RuntimeException
   public String remove() throws EmptyException;
   public boolean empty();
}
```

   Verschiedene Implementierungen von TextContainer sollen austauschbar sein, z.B. eine mit einem Array oder eine mit einer generischen ArrayList<String>. Sie starten mit einer fertigen Implementierung namens TextContainerPrototype (s.u.).

3. Die Experimente bestehen darin, dass Sie eine oder mehrere Nachrichtenkanal-Konfigurationen "aufbauen", indem Sie die Warteschlangenimplentierungen austauschen und mehrere Sender und Empfänger auf eine Warteschlange ansetzen. Überlegen Sie vorher, welches Verhalten Sie erwarten, und werten Sie die Ergebnisse aus! Sie werden "Ungerechtigkeiten" feststellen – überlegen Sie, ob Sie etwas dagegen tun können. Natürlich können Sie auch einmal einen Stack statt einer Warteschlange einbinden...

4. Klassen zählen:
   Ihr Projekt wird aus mindestens 8 Klassen bestehen:
   Dem Interface TextContainer, mindestens einer Implementierung davon, einer animierten Klasse Sender, einer animierten Klasse Receiver, zwei Exceptions für den vollen und den leeren Nachrichtenkanal, der Klasse TextGUI und der Main-Klasse, in der Sie alles zusammenbauen..
   Das sind eine Menge Fehlerquellen: Testen Sie daher Ihre Klassen, ehe Sie sie mit den anderen kombinieren. Dazu schreiben Sie in jede Klasse eine Main-Methode, die die programmierten Methoden ausführt.

Laborexperimente:

Versuchsaufbau:

1. Erstellen Sie ein neues Projekt mit der Bibliothek cs101-lib.Legen Sie das Interface TextContainer und die TextContainer-Implemenierung TextContainerPrototype in Ihrem Projekt an und übernehmen Sie den Inhalt aus diesem Aufgabenblatt. Sie werden auch noch zwei Exception-Klassen anlegen müssen, eine vom Typ Exception, und eine vom Typ RuntimeException.

2. Testen Sie die Klasse TextContainerPrototype mithilfe der vorhandenen Main-Methode. Funktioniert die Klasse erwartungsgemäß?

3. Implementieren Sie einen animierten Sender, der Zeilen von der Konsole liest und sie in ein Objekt vom Typ TextContainer einträgt. Den TextContainer soll er im Konstruktor "kennenlernen". Bereits im Konstruktor sollte der Sender eine Eingabeaufforderung auf die Konsole schreiben.
   Schreiben Sie auch hier eine Main-Methode, um diese Klasse zu testen!

   Tipp:
   Vergessen Sie nicht, den AnimatorThread des aktiven Objekts zu starten (startExecution())!

4. Implementieren Sie einen animierten Empfänger, der – falls vorhanden – Zeichenketten aus dem TextContainer ausliest und im TextGUI-Fenster anzeigt. Auch er soll seinen TextContainer im Konstruktor "kennenlernen". Ein eigenes TextGUI-Ausgabefenster erhält er, indem er (am besten im Konstruktor) ein Objekt der Klasse TextGUI erzeugt.

   Achtung: Asynchrone Objekte kommen bisweilen aus dem Takt: Falls Ihr Empfänger startet. ehe die Kommunikation mit der Warteschlange richtig aufgebaut ist, kann es beim Zugriff auf die Warteschlange zu einer NullPointerException kommen: Fangen Sie sie auf und ignorieren Sie sie: `catch(NullPointerException x) { /* nix tun */ }`*. Beim nächsten Aufruf von act() stimmt dann wahrscheinlich schon alles.*

5. Schreiben Sie eine Main-Klasse, die einen TextContainer, einen Sender und einen Empfänger erzeugt und miteinander verknüpft. Das Programm sollte sichtbar starten (Konsolausgabe), sobald die Objekte erzeugt sind. Dann geben Sie dem Empfänger etwas zu lesen (tippen Sie etwas ein) – und sehen Sie, was passiert!
   Tipp: Schreiben Sie keinen langen Texte, sondern versuchen Sie schneller zu sein als Ihr Nachrichtenkanal, nur dann können Sie interessantes Verhalten beobachten.

6. Implementieren Sie den TextContainer jetzt selbst, zunächst mit einem Array, dann mithilfe einer typ-parametrisierten Klasse aus dem Java-Collections-Framework (z.B. ArrayList<String>) und binden Sie sie in den obigen Versuchsaufbau ein. Vergessen Sie nicht, Ihre Klassen vorher zu testen!

Bis hierher sollten Sie in der Übung kommen!

Weitere Experimente für die Schnellen:

7. Lassen Sie mehrere Sender eine Schlange "füttern" und mehrere Empänger an ihr "nagen". Welche Verteilung auf die Ausgaben erwarten Sie, und wie sieht das Ergebnis aus? (Alle TextGUI-Fenster liegen übereinander – ziehen sie sie mit der Maus zur Seite.)

8. Erzeugen Sie in einem Programm mehrere Nachrichtenkanal-Kombinationen mit unterschiedlichen TextBehaeltern. Verhalten sich die verschiedenen TextBehaelter-Implementierungen gleich?
   Tipp: Ändern Sie Ihren Sender so ab (erweitern!), dass er eine Eingabe in mehrere TextBehaelter füttern kann – sonst können Sie evtl. nicht schnell genug tippen, um die Kanäle zu versorgen.

9. Machen Sie Ihre eigenen Experimente, binden Sie Stacks ein, oder starten Sie die Empfänger deutlich verzögert, damit Sie vorher Zeit haben, den Kanal zu füllen. Sie können die Nehmer auch erst starten, wenn Ihr Textbehälter einen bestimmten Füllstand erreicht hat (readln mitzählen). Es gibt genug Raum für Kreativität.

10. Wie wäre es, eine Klasse zu definieren, die je ein Sender- und ein Empfänger-Objekt enthält und so als kombinierter Sender-Empfänger dienen kann?

Und hier der Screenshot der Klasse ***TextContainerPrototype zum Abtippen***:
Wir lernen „durch die Finger“ – deshalb ist es wichtig, dass jeder diese Klasse selbst abtippt!

```
/** Simple container for Strings, to be used as a prototype
public class TextContainerPrototype implements TextContainer {
    private String[] list ;
    private int size,
                top = 0;

    public TextContainerPrototype() {
        this(25);
    }
    public TextContainerPrototype(int size) {
        this.size = size;
        list = new String[size];
    }
    public void enter(String s) {
        if (top == size) throw new FullException();
        list[top] = s;
        top++;
    }
    public String remove() throws EmptyException {
        if (top == 0) throw new EmptyException();
        top--;
        return list[top];
    }
    public boolean empty() {
        return top == 0;
    }
    // main method for testing purposes only:
    public static void main(String[] args) {
        TextContainerPrototype chn = new TextContainerPrototype(4);
        try {   chn.enter("1"); chn.enter("2"); chn.enter("3");
                System.out.println(chn.remove());
                System.out.println(chn.remove());
                System.out.println(chn.remove());
                System.out.println(chn.remove());
        } catch (EmptyException e) { System.out.println("leer"); }
    }
}
```