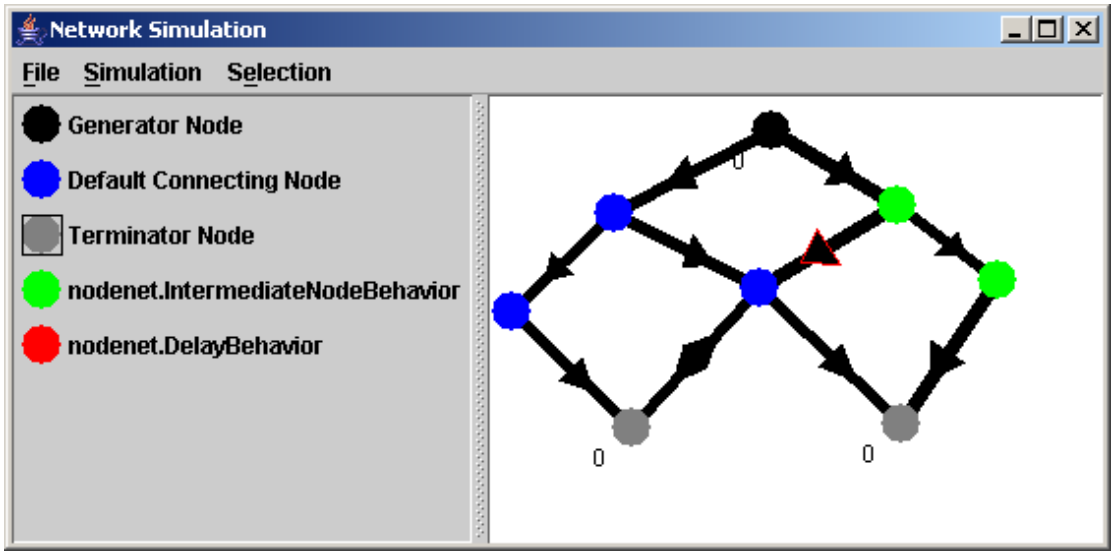
Programmieren 1 – Schmiedecke

Laborexperiment 9 – Nodenet

Diese Labor ist wieder ein Experimentierfeld. Es baut auf BallWorld aus Experiment 4 und die Nachrichtenkanäle (MessageQueues) aus Experiment 5 auf: Das Experimentierfeld erlaubt Ihnen, beliebige Netzwerke aus Verarbeitungsknoten und Nachrichtenkanälen zu bauen und zu testen; im Gegensatz zm Nachrichtenkanal gibt es nicht nur Anfangs- und Endknoten ("Sender und Empfänger"), sondern vor allem auch Mitttelknoten, die Nachrichten aus mehreren Kanälen lesen und in mehrere Kanäle schreiben können. Ihre Aufgabe besteht darin, solche Mittelknoten zu programmieren (die dadurch, dass sie das passende Interface implementieren, in die Simulationsumgebung eingebunden werden können - wie Bälle in BallWorld). Ein Netzwerk könnte zum Beispiel so aussehen:



Lernziele: Mehrdimensionale Datenbehälter, Animierte Objekte, Exceptions, Abfertigung

Vorbereitung

Lesen Sie den gesamten Aufgabentext und beantworten Sie die Vorbereitungsfragen. Bereiten Sie die Laborexperimente vor.

.

Nachbereitung: Laborprotokoll

Diesmal ist der Quelltext der verschiedenen Knoten sehr wichtig. Bitte dokumentieren Sie Ihre Überlegungen im Programmmtext in **Javadoc-Kommentaren**, erzeugen Sie für Ihre Klassen eine Dokumentation und übernehmen Sie die "Method Details" in Ihr Protokoll.

Machen Sie außerdem Screenshots von den Testnetzwerken, beschreiben Sie, welches Veralten Sie erwartet haben, welches Sie beobachtet haben, und versuchen Sie ggf., den Unterschied zu erklären.

Vorbereitungsfragen:

1. Was versteht man unter der Definition, dem Auswerfen und dem Fangen einer Exception?
2. Wenn eine Methode eine Exception auswerfen kann, welche zwei Alternativen gibt es, beim Aufruf der Methode mit dieser "Gefahr" umzugehen?
3. Für Welche Exceptions fordert Java eine Behandlung (try-catch oder throws), für welche nicht?

4. Welches sind die vier Regeln zur Reduzierung der Komplexität bei der Abfertigung und Verteilung? Wo gibt es im NodeBehavior Abfertigung?

5. Was ist der Unterschied zwischen einer gerechten und einer zufälligen Abfertigung? Skizzieren Sie die `transmitPackage`-Methode für beide Abfertigungstypen.

6. Versuchen Sie zu beschreiben, wie das Verhalten von Node-Objekten durch NodeBehavior-Objekte bestimmt werden kann. Was würde passieren, wenn Sie zur Laufzeit einem Node-Objekt ein anderes NodeBehavior-Objekt zuordnen?

Einführung:

1. Nodenet ist eine Experimentierumgebung, die dynamische Netze simuliert. Ein Netz besteht aus Anfangsknoten (GeneratorNode), Endknoten (TerminatorNode) und Mittelknoten. Zwischen den Knoten können Nachrichtenkanäle angelegt werden (die übrigens mithilfe von java.util.Vector implementiert sind; schauen Sie sich den Quellcode an – er sollte Ihnen sehr bekannt vorkommen). Knoten können interaktiv zu Netzen zusammengefügt werden. In der Simulation schicken Anfangsknoten immer neue „Pakete“ in ihre Kanäle, Endknoten absorbieren alle Pakete. Der Weg der Pakete durch die Kanäle wird visualisiert.

2. Die Knoten sind animierte Objekte, die von der Simulationsumgebung erzeugt und verwaltet werden. ***Sie müssen die Knoten nicht programmieren!*** Jedem Knoten ist ein ***Verhaltensobjekt*** zugeordnet, das Sie programmieren sollen. Es muss das Interface `nodenet.NodeBehavior` implementieren, dessen einzige Methode `transmitPacket` vom Knoten bei jedem Animationszyklus aufgerufen wird. Die Aufgabe von `transmitPacket` besteht darin, aus den Eingangskanälen Pakete zu entnehmen und diese auf die Ausgangskanäle zu verteilen. Die Zahl der Kanäle ist natürlich variabel – sie werden als Listen (Vektoren) von Kanälen übergeben.

```
public interface NodeBehavior {
   public void transmitPacket( InputChannelVector inputChannels,
                               OutputChannelVector outputChannels );
}
```

3. Ihre Aufgabe besteht darin, verschiedene Verhaltensklassen zu programmieren und ihr Verhalten in der Simulation zu testen. Ein Verhalten ist dabei eine Strategie, nach der entschieden wird, aus welchem Kanal entnommen und in welchen Kanal geschrieben wird. Denken Sie daran, dass Knoten bereits animiert sind – es muss bei jedem Aufruf nur ein Paket (oder vielleicht eine kleine feste Anzahl von Paketen) verarbeitet werden.

4. Beachten Sie: Die Kanäle selbst sind „Warteschlangen“, d.h. es kann immer nur am Ende eingetragen und am Anfang entnommen werden. InputChannel definiert dazu die Methode `readObject`, OutputCHannel die Methode `writeObject`

```
public Object readObject() throws ChannelEmptyException,
```

```
                                                ChannelDisabledException;

    public void writeObject(Object o) throws ChannelFullException,
                                               ChannelDisabledException;
```

Auf die Kanal-Vektoren dagegen können Sie an beliebiger Stelle zugreifen, und zwar mit der Methode `elementAt(int index)`. Die genaue Schnittstelle entnehmen Sie der Dokumentation. Sie finden sie in der nodenet.jar-Datei im doc-Verzeichnis. Selektieren Sie in der Dokumentation statt „All Classes“ nur das Paket nodenet – dann haben Sie schneller den Überblick.

Laborexperimente:

Experimente mit vorgegebenen Knotenverhalten

1. Öffnen Sie ein neues Projekt. Laden Sie sich nodenet.jar herunter und binden Sie es als externe Jar-Datei in ihr Projekt ein. Die Jar-Datei enthält sowohl die cs101-Bibliothek als auch die Javadoc-Dokumentation der Nodenet-Klassen.
2. Führen Sie die Klasse Main aus. Die Simulationsumgebung erscheint. Sie können neue Knoten erzeugen, indem Sie im linken Teilfenster ein Verhalten anklicken und dann in das rechte Teilfenster klicken. Wenn Sie im rechten Teilfenster einen Knoten mit Doppelklick selektieren und dann in einen anderen Knoten klicken, erzeugen Sie einen Kanal zwischen den Knoten. Korrekturen am Netz können Sie vornehmen, indem Sie ein Element anklicken und im Selection-Menü Remove wählen. Mit dem Menü Simulation können Sie die Simulation starten, unterbrechen und wieder aufnehmen. Achtung: Clear löscht ihr Netzwerk!
3. Sie können Ihr Netzwerk während der Simlation verändern – Knoten und Kanäle hinzufügen, aktivieren/deaktivieren und löschen.
4. Im Prinzip können Sie ein Netzwerk speichern (File save..), aber das funktioniert leider nicht zuverlässig. Machen Sie unbedingt Screenshots, um ein Netzwerk ggf. schnell nachbauen zu können. Ansonsten können Sie mehrere Simulationen gleichzeitig geöffnet halten...
5. Bauen Sie ein paar Netzwerke, und versuchen Sie, herauszufinden, worin sich die drei vorgegebenen Verhaltensmuster für Mittelknoten unterscheiden. Können Sie eine Rohrverstopfung erzeugen?

Eigene Verhaltensmuster:

Schreiben Sie eigene Verhaltensmuster und testen Sie Sie in verschiedenen Netzwerken. Versuchen Sie, für jedes Verhalten ein besonders einfaches charakteristisches Netzwerk zu bauen, in dem man die Besonderheit des Verhlatens deutlich erkennt. Erläutern Sie im Protokoll den Code (immer nur die Methode `transmitPackage` einfügen!), machen Sie Screenshots von den Netzwerk-Simulationen und erläutern Sie, wie das jeweilige Knotenverhalten sich auf das Netzwerk auswirkt.

6. **Verstopfung:**
   Eigene Verhaltensmuster sind Implementierungen des Interfaces `NodeBehavior`. Schreiben Sie eine erste Implementierung, in der die Methode `transmitPacket` nichts tut. Nennen Sie die Klasse z.B: Blockade, denn sie erzeugt eine sofortige Kanaverstopfung (warum?). Ordnen Sie Ihre Klasse entweder dem Paket nodenet oder einem eigenen Paket zu. Sie haben zwei Möglichkeiten, Ihre Klasse in das Simulationswerkzeug einzubinden:

- Die main-Methode der Klasse Main liest neue NodeBehavior-Klassen als Aufrufparameter ein. In Eclipse können Sie die im Run-Fenster unter Program Arguments angeben. geben Sie bitte den kompletten Paketpfad mit an, also z.B `nodenet.Blockade`.
- Während der Simaultion können Sie neue Verhalten über den Menüpunkt File – New hinzufügen. Hier wird erwartet, dass die Klassen im Paket nodenet oder im default package liegen.

7. **Gerechtigkeit**:
   Schreiben Sie ein Verhaltensmuster, das per Durchgang (max.) ein Paket durchgibt und alle Eingangs- und Ausgangskanäle gerecht abarbeitet. Denken Sie daran, dass die Lese- und Schreib-Methoden der Kanäle Exceptions auswerfen können und behandeln Sie sie entsprechend. (Die Exceptions sind im Paket nodenet definiert).

8. **Zufall:**
   Schreiben Sie ein Verhaltensmuster, das per Durchgang (max.) ein Paket durchgibt und die Eingangs- und Ausgangskanäle nach dem Zufallsprinzip abarbeitet (`Methode Math.random`).
   Bis hierher sollten Sie in der Übung mindestens kommen.

   Weitere Vorschläge – Sie können auch eigenen Ideen ausprobieren! Nur, schreiben Sie **vor dem Experiment** auf, wodurch das Verhalten charakterisiert sein soll....

9. **Verzögereung:**
   Schreiben Sie ein Verhaltensmuster, das per Durchgang (max.) ein Paket durchgibt und dieses für kurze Zeit festhält (Methode `Thread.sleep(millisec) – InterruptedException` fangen und ignorieren)

10. **Durchfluss-Regulierung:**
    Schreiben Sie ein Verhaltensmuster, das ganau einen Ausgangskanal bedient (und die anderen ignoriert) und bei jedem dritten Durchgang ein Paket weitergibt. (Wenn Sie mehrere solcher Knoten in das Netzwerk einbauen und dahinter Schlussknoten anschließen, können sie den Eindruck einer gleichmäßigen Beregnung erzeugen).

11. **Durchflussregulierung mit Überlauf-Ventil:**
    Schreiben Sie ein Durchflussregulierungs-Verhalten mit Überlauf-Ventil: Geben Sie ihrem Knoten einen Fifo-Zwischenspeicher, den Sie bei jedem Durchgang mit ankommenden Paketen füllen. Ist er voll, geben Sie alle weiteren ankommenden Pakete über einen zweiten Kanal (falls vorhanden) weiter, bis im Zwischenspeicher wieder Plalz ist. Über den ersten Kanal geben Sie einen gleichmäßigen Paketfluss aus (s.o.)

Links dieser Seite
nodenet.jar: http://www.schmiedecke.info/Prg1/Praxis-Downloads/nodenet.jar
Quellcode: http://www.schmiedecke.info/Prg1/Praxis-Downloads/nodenet-src.jar

---

Hinweis zum Copyright