Übung „Simulator“

Aufgabe 1

Machen Sie sich mit den Paketen `de.hspforzheim.eit.simulation` und `de.hspforzheim.eit.simulation.blocks` vertraut. Hierzu stehen Ihnen der Quellcode sowie eine (rudimentäre) JavaDoc-Hilfe zur Verfügung.

Entwickeln Sie aus dem Code das Klassendiagramm. Welche Klassen sind abstrakt und warum? Warum ist die Klasse `Port` nicht abstrakt, obwohl nur Objekte der Unterklassen `Inport` oder `Outport` angelegt werden?

Aufgabe 2

Entwerfen Sie ein weiteres System (RC-Tiefpass), das sie simulieren. Verwenden Sie hierzu die bereits vorhandenen Implementierungen und ergänzen Sie die Klasse `SimTest` um die statische Funktion `systemConfig04()`, die die entsprechende Systemkonfiguration vornimmt.

Schreiben Sie die Ergebnisse (Systemeingang und –ausgang) in eine Textdatei „`System04.txt`“ und überprüfen Sie sie mit Ihren Erwartungen.

Aufgabe 3

Stellen Sie eine neue Klasse bereit, die aus einer Textdatei im CSV (Comma Separated Values)-Format (vgl. Ergebnisse in System04.txt) Stimulationswerte einliest und als Stimuli zur Verfügung stellt. Die erste Spalte enthält dabei die Zeitpunkte. Zwischen zwei Zeitpunkten ist ggf. zu interpolieren. Falls die Anzahl der Spalten nicht mit der Anzahl der Ausgänge kongruent ist soll der Block nichts tun (und ggf. eine Fehlermeldung in der Ausgabe protokollieren).

Hinweis: Machen Sie sich hierfür mit dem Java-Paket `java.io` (insbesondere den Klassen `FileReader` und `FileWriter`) vertraut. Beachten Sie die Methode `flush()`.

Aufgabe 4

Erstellen Sie eine neue Übertragungsfunktion für ein Totzeit-Glied (Verzögerung des Einganges um eine feste Zeit). Erstellen Sie eine weitere Systemkonfiguration, in der Sie das Totzeitglied (Totzeit: 0,5 Sekunden) über ein Verstärkungsglied mit Verstärkungsfaktor 0,6 zurückkoppeln und erzeugen Sie entsprechend eine Ergebnisdatei `System05.txt`.

Aufgabe 5

Die Abarbeitungsreihenfolge der Simulationsblöcke ist nicht beliebig. Ein Block kann nur berechnet werden, wenn alle seine Eingänge bereits berechnet wurden.

Schreiben Sie eine Methode `sortBlocks` für die Klasse `Simulator`, die die korrekte Reihenfolge der Blöcke ermittelt. Dazu müssen Sie für jeden Block prüfen, ob seine Eingänge bereits berechnet wurden. Dieser Vorgang muss nur einmal vor Beginn der Simulation durchgeführt werden, um die richtige Reihenfolge zu bestimmen.

Gehen Sie wie folgt vor:

1. Erweitern Sie die Klasse `Outport` um einen boolschen Member `isCalculated`. Hier wird vermerkt, ob der Port schon (virtuell) berechnet wurde. Fügen Sie außerdem einen boolschen Member `wasVisited` ein. Hier wird vermerkt, ob der `Outport` bereits geprüft wurde.
2. Die Prüfung erfolgt rekursiv für jeden Block. Ein zu prüfender `Outport` wird zunächst als `wasVisited` markiert. Wenn alle Eingänge als berechnet gekennzeichnet sind, dann wird der Block in die Liste der abzuarbeitenden Blöcke angefügt und alle Ausgänge werden als berechnet (`isCalculated = true)` markiert.
3. Wenn ein Eingang noch nicht berechnet wurde, so wird der aktuelle Block auf einen Stack gelegt und für jeden Eingang der verbundene `Outport` geprüft. Dies kann sich rekursiv fortsetzen.
4. Ein Ausgang lässt sich berechnen, wenn der entsprechende Block keinen Eingang hat (z.B. Signalgenerator).
5. Wenn man bei der Prüfung auf einen `Outport` stößt, der bereits als besucht, aber nicht als berechnet markiert wurde, so liegt eine algebraische Schleife vor. Diese ist hinter dem letzten Integrationsblock aufzubrechen.