University of Zurich
Institut für Informatik

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

Software Engineering Übung 1

Programmverstehen, Dokumentation

1 Allgemeines

1.1 Wichtige Daten

- Ausgabe Di 16.09.2008
- Abgabe So 28.09.2008 bis 23:55 Uhr
- Besprechung am Di 07.10.2008, 14:00 Uhr

1.2 Formales

Die Lösungen sollen als PDF Datei mit dem Namen **Ex[*n*]_[*NameA_NameB_NameC*].pdf** abgegeben werden, wobei [*n*] die Nummer der Übung ist und [*NameA_NameB_NameC*] die Nachnamen der Gruppenmitglieder sind.

Mailen sie Ihre Lösungen vor dem Abgabetermin an `jeanneret@ifi.uzh.ch`. Der Betreff der E-mail sollte mit **[SE EX HS08]** beginnen. Falls Sie zusätzliche Abgabematerialien (z.B. Source Code) haben, mailen Sie bitte ein Archiv (.zip-File), welches alle Dateien, einschliesslich dem PDF, enthält. Benennen sie das Archiv anhand der oben erwähnten Konventionen.

Denken Sie daran, dass verspätete Abgaben zwar korrigiert, die erreichten Punkte aber nicht angerechnet werden. Die Übungen sollen in 3er Gruppen gelöst werden. Jedes Gruppenmitglied muss über alle Teile der Lösungen Auskunft geben können.

1.3 JClusim Installation

Diese Übung basiert auf einem existierenden Java Programm, *JClusim*. *JClusim* ist ein Framework zum Entwerfen und Simulieren von Experimenten zur Clusterbildung. In diesen Experimenten gibt es eine Menge von Agenten (z.B. Roboter), welche Gegenstände verschieben, um diese anhand bestimmter Eigenschaften zu gruppieren.

JClusim kann als gepacktes Archiv von der Webseite der Übungen heruntergeladen werden. Das Archiv beinhaltet:

- den Source Code von *JClusim* (im `src` Ordner). Der Source Code umfasst das Framework und (darauf aufbauend) ein einfaches Experiment
- zwei Bibliotheken, JCommon und JFreeChart[1] (im `lib` Ordner), die von *JClusim* benötigt werden

[1] `http://sourceforge.net/projects/jfreechart/`

- eine Ant Build-Datei (`build.xml`)
- alle benötigten Dateien, sodass das Archiv als Projekt in Eclipse importiert werden kann

Kompilieren und Starten von JClusim in Eclipse

Gehen Sie in Eclipse ins *File* Menü und klicken Sie auf *Import...*. Wählen Sie dann *Existing project into the workspace* und geben Sie bei *Select archive file* das *JClusim* Archiv an. Klicken Sie auf *Finish*. Eclipse erstellt im Workspace ein neues Projekt (namens *JClusim*). Um *JClusim* zu starten, rechts klicken Sie auf das *JClusim* Projekt, und klicken Sie auf *Run as* und *Java Application*. Wählen Sie *jclusim.JClusim* als Hauptklasse.

Kompilieren und Starten von JClusim mit Ant

Extrahieren Sie das Archiv mit dem Befehl `tar xvzf JClusim.tar.gz`. Wechseln Sie dann ins Verzeichnis `JClusim`. Mit dem Befehl `ant run` können Sie das Experiment starten.

JClusim Eigenschaften

Eine *JClusim* Simulation kann im interaktiven Modus oder im Batch Modus ausgeführt werden. Im Batch Modus erhalten Sie Screenshots vom Initial- und Endzustand der Umgebung sowie einige weitere Daten. Wenn die Simulation im interaktiven Modus gestartet wird, zeigt ein Fenster eine Animation der Simulation. Der Modus wird durch ein Java Property namens *jclusim.interactive* gesetzt. Setzen Sie es auf *false* um *JClusim* im Batch Modus laufen zu lassen. Die gesammelten Daten der Simulation können dann im Verzeichnis `Results` gefunden werden.

Das Experiment, das *JClusim* simuliert, muss in einer Klasse definiert sein, welche das Interface *jclusim.base.IExperimentSettings* implementiert. Es genügt dann, den Namen dieser Klasse in dem Property *jclusim.experiment* anzugeben. *JClusim* wird diese Klasse laden und zur Konfiguration des Experimentes verwenden.

Um die Werte der beiden Properties zu ändern, können Sie, falls Sie Ant benutzen, die Zeilen 9 und 10 in der Datei `build.xml` anpassen. Mit Eclipse ist es einfacher, wenn man die Default-Werte der Properties ändert. Diese Werte sind in der Klasse *jclusim.JClusim* in den Zeilen 23 und 24 definiert.

2 Verstehen des Quellcodes (10 Punkte)

2.1 Struktur von JClusim (4 Punkte)

Betrachten Sie die zwei Packages *jclusim.base* und *jclusim.experiments*. Das erste Package beinhaltet das Framework, das zweite den Source Code eines simplen Experimentes. Zeichnen Sie ein UML Klassen Diagramm, welches Folgendes zeigt:

- alle zu diesen Packages gehörenden Java Klassen und Interfaces
- ihre Attribute und Methoden
- die Beziehungen zwischen ihnen (Vererbung, Interface Implementation, und Assoziation)

Schreiben Sie einen kurzen Text (maximal 5 Sätze), welcher den Inhalt des Diagramms zusammenfasst.

2.2 Verhalten der Agenten (3 Punkte)

Allgemeines Verhalten (2 Punkte)

Das grundlegende Verhalten der Agenten ist in der Methode *behave()* der Klasse *jclusim.base.Agent* implementiert. Benutzen Sie ein UML Zustandsdiagramm (UML state machine) oder ein UML Aktivitätsdiagramm um das generelle Verhalten eines Agenten zu beschreiben. Das Diagramm muss sich auf die abstrakten Methoden der Klasse beziehen (*considerDrop()*, *considerPickUp()* und *chooseNextCell()*).

Schreiben Sie einen kurzen Text (maximal 5 Sätze), welcher den Inhalt des Diagramms zusammenfasst.

Simples Agentenverhalten (1 Punkt)

Beschreiben Sie das spezifische Verhalten des Agenten, welches im vorhandenen Experiment (*jclusim.experiments.simple.SimpleAgent*) definiert ist, in informeller Sprache. Erwähnen Sie insbesondere, wann sich der Agent dazu entschliesst, einen Gegenstand aufzuheben oder abzustellen, und was für eine Strategie zur Fortbewegung er benutzt.

2.3 Experiment Definitionen (3 Punkte)

Initialisierung der Kollaboration (2 Punkte)

Ein wesentlicher Vorteil des Designs von *JClusim* ist, dass man die Möglichkeit hat, eigene Experimente zu definieren. Das Interface *jclusim.base.IExperimentSettings* ist dafür verantwortlich, ein Experiment zu konfigurieren, bevor es simuliert wird. Benutzen Sie ein UML Sequenzdiagramm, um die Kollaboration zu beschreiben, welche durch den Aufruf der Methode *configure()* der Klasse *jclusim.base.Experiment* initialisiert wird.

Das simple Experiment (1 Punkt)

Beschreiben Sie informal das simple Experiment, welches mit *JClusim* mitgeliefert wurde. Ihre Antwort sollte auch die Anzahl und die Startpositionen der Gegenstände und Agenten, sowie die Abbruchbedingung der Simulation beinhalten.

Tipp: Sie können *JClusim* im Batch Modus laufen lassen und die Screenshots in Ihrer Antwort verwenden.

3 Verbesserung und Erweiterung des Quellcodes (10 Punkte)

3.1 Optimierung (2 Punkte)

In *JClusim* ist das *identity* Attribut eines Gegenstandes unveränderbar, d.h. *identity* wird während der Erzeugung eines Gegenstandes definiert und kann danach nicht mehr verändert werden. Mehrere Eigenschaften des Gegenstandes *jclusim.base.Item* werden von dem Attribut *identity* abgeleitet:

- die Norm von seiner *identity* (method *getIdentityNorm()*)
- die Dimension von seiner *identity* (method *getIdentityDimension()*)
- die grösste Komponente seiner *identity* (method *getIdentityMaxValue()*)
- der Index der grössten Komponente seiner *identity* (method *getIdentityMaxIndex()*)
- die Farbe in der er dargestellt wird (method *getColor()*)

Verbessern Sie die Laufzeit dieser Methoden, indem Sie die Memoization[2] Optimierungstechnik anwenden. In anderen Worten: ändern Sie die Klasse *jclusim.base.Item*, so dass die Werte der abgeleiteten Eigenschaften während der Konstruktion des Objektes berechnet und in (zusätzlichen) Klassenfeldern gespeichert werden. Ändern Sie auch die erwähnten Methoden, damit sie die zuvor berechneten Werte zurückgeben (anstatt die Werte neu zu berechnen).

Erklären Sie kurz, warum diese Optimierung für *JClusim* geeignet ist, und was die Nachteile dieser Optimierung sind.

3.2 Dokumentation (4 Punkte)

Die abstrakte Klasse *jclusim.base.Agent* und das Interface *jclusim.base.IExperimentSettings* sind die Schlüsselelemente zum Definieren eines Experimentes. Dokumentieren Sie diese Klasse und dieses Interface (schreiben Sie Klassen-, Feld- und Methodenkommentare). Befolgen Sie dabei die Richtlinien des Java Style Guide (erhältlich auf der Webseite der Übungen). Ihre Antworten zu den vorherigen Fragen werden für diese Aufgabe definitiv nützlich sein.

3.3 Ein neues Experiment (4 Punkte)

Programmieren Sie ein neues Experiment anhand der folgenden Beschreibung. Erstellen Sie die Klassen, welche dafür benötigt werden, aber ändern Sie nicht den Code des Frameworks!

Die Umgebung soll 48 Zellen hoch und 64 Zellen breit sein. Darin sollen sich 330 Gegenstände befinden, deren *identity* Attribute aus dem folgenden Set ausgewählt werden: $\{(1,0,0),(0,1,0),(0,0,1)\}$. Von jeder Art soll es jeweils 110 Gegenstände geben. Die Gegenstände sollen zufällig in der Umgebung verteilt werden. Beachten Sie, dass nicht zwei Gegenstände gleichzeitig in derselben Zelle vorhanden sein können. Wenn eine Zelle bereits von einem Gegenstand belegt wird, soll eine andere Zelle ausgewählt werden.

Die Agenten sind spezialisiert: einige Agenten sind wählerischer als andere beim auflesen von Gegenständen, dafür bewegen sie sich aber schneller. Die Agenten erhalten hierzu ein neues Attribut vom Typ Integer namens *speed*. Der Wert wird jeweils zufällig gewählt, das Minimum ist 1, das Maximum 6. Es sollen 30 Agenten zufällig in der Umgebung verteilt werden (in einer Zelle können sich gleichzeitig mehrere Agenten aufhalten). Ein Agent bewegt sich von Zelle zu Zelle wie folgt: sei (x, y) die momentane Position. Der Agent kann sich zu einer der folgenden 9 Zellen begeben: $(\{x-speed, x, x+speed\}, \{y-speed, y, y+speed\})$. Das heisst, der Agent kann entweder auf seiner momentanen Position bleiben, oder er bewegt sich in einer oder beiden Dimensionen um die Distanz *speed*.

Wie schon der simple Agent aus Aufgabe 2, heben (/stellen) die Agenten Gegenstände mit einer bestimmten Wahrscheinlichkeit auf (/hin). Die Wahrscheinlichkeit hängt von der Dichte der Gegenstände ab, welche sich im momentanen, lokalen Gebiet befinden. Sei ρ die Dichte, welche durch den Ausdruck *Similarity.COSINUS.computeAveragedSimilarityWithNeigbhorhood* (mit einem Radius von 1) berechnet wird. Die Wahrscheinlichkeit, dass der Agent einen Gegenstand aufhebt, ist definiert als $p_{pickup} = (1-\rho)^{speed}$, und die Wahrscheinlichkeit, dass der Agent einen Gegenstand hinstellt, ist $p_{drop} = \rho$ (diese zweite Wahrscheinlichkeit entspricht derjenigen vom simplen Agenten).

[2] `http://en.wikipedia.org/wiki/Memoization`