Universität Zürich[UZH]
Institut für Informatik

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

Software Engineering Übung 1

Programmverständnis, Dokumentation

1 Informationen

1.1 Daten

- Ausgabe Di 20.09.2011
- Abgabe So 02.10.2011 bis 23:59 Uhr
- Besprechung am Di 11.10.2011 um 12:15 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. Die PDF Datei sollte ausserdem ebenfalls Ihre Namen und Matrikelnummern beinhalten.

Mailen sie Ihre Lösungen vor dem Abgabetermin an `nhoby@access.uzh.ch` und `todoran@ifi.uzh.ch`. Der Betreff der E-mail sollte mit **[SE EX HS11]** 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.

Die Übungen sollen in 3er Gruppen gelöst werden. Jedes Gruppenmitglied muss über alle Teile der Lösungen Auskunft geben können. Verspätete Abgaben werden korrigiert, aber nicht bewertet.

2 JGAP

2.1 Allgemein

Diese Aufgabe basiert auf der Bibliothek JGAP, dem Java Genetic Algoritm Package. Mit JGAP können genetische Algorithmen implementiert und zur Problemlösung verwendet werden.

Zusätzliche Informationen zur JGAP Bibliothek finden Sie auf:
`http://jgap.sourceforge.net/`

Damit alle die gleiche Ausgangslage haben, verwenden Sie bitte den auf unserer Website zu den Übungen angebotenen Quelltext.

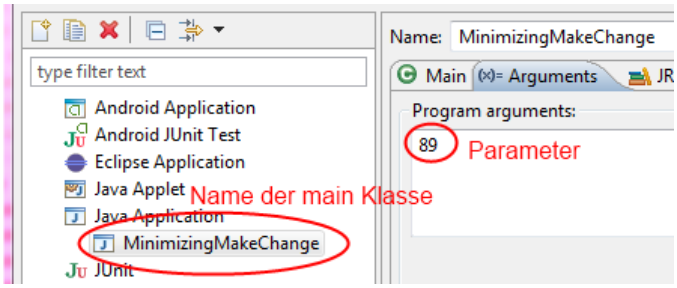

Abbildung 1: Beispielparameter für MinimizingMakeChange

2.2 Installation

In Eclipse, wählen Sie *File* und dann *Import...*. Im neuen Fenster klicken Sie auf *General* und dann *Existing Projects into Workspace*. Als Archiv wählen Sie die *jgap.zip* Datei. Klicken Sie auf *Finish*. Eclipse wird nun ein neues Projekt unter dem Namen *JGAP* im Workspace anlegen.
Im Ordner `src` befindet sich der Sourcecode des Programms, welchen Sie für die nachfolgenden Aufgaben brauchen. Der Ordner `examples/src` enthält ausführbare Beispiele, mit denen Sie verschiedene Anwendungen des JGAP-Frameworks ausprobieren können.

2.3 Hinweis zu den Beispielen

Viele der Beispiele im Ordner `examples/src` erwarten einen oder mehrere Parameter um sie auszuführen. Um die Beispiele in eclipse laufen zu lassen, müssen sie die erforderlichen Parameter in der entsprechenden *Run Configuration* angeben. Zuerst müssen Sie das gewünschte Beispiel ausgeführen, damit eine Run Configuration zu dieser Klasse erstellt wird. Wählen Sie dazu eine Klasse, die eine `main` Methode enthält. Rechtsklick auf die Datei, danach *Run as* und *Java Application* auswählen. In der Konsole sollte nun der Hinweis auf die fehlenden Parameter auftauchen. Danach wählen Sie *Run as* und *Run Configurations*. In der Liste auf der linken Seite finden Sie die Konfiguration zum gewünschten Beispiel. Wählen Sie den entsprechenden Eintrag aus. Im Tab *Arguments* können Sie unter *Program arguments* die nötigen Parameter wie vom Konsolenmodus her bekannt angeben. Für das im Abschnitt 4 verwendete Beispiel könnte die Konfiguration wie in Abbildung 1 dargestellt aussehen.

3 Verständnis des Quelltextes (14 Punkte)

3.1 Struktur (6 Punkte)

Für diese Aufgabe betrachten Sie folgende Klassen und Interfaces aus den Packages `org.jgap` und `org.jgap.impl`:

- `Population`
- `IChromosome`, `BaseChromosome`, `Chromosome`
- `Gene`, `BaseGene`, `ICompositeGene`, `CompositeGene`

Verwenden Sie ein Klassendiagramm, um die obengenannten Klassen und Interfaces, ihre Methoden sowie die Beziehungen untereinander darzustellen. *Private* Methoden brauchen Sie nicht zu berücksichtigen, ebenso das in jeder Klasse vorhandene Feld *CVS_REVISION*. Nicht dargestellt werden darüber hinaus Konstruktoren sowie Setter und Getter. Stellen Sie für jede Klasse alle Klassenvariablen dar sowie mindestens 5 Methoden (sofern vorhanden, ohne Setter/Getter).

3.2 Anwendungsbereich (2 Punkte)

Die eigentlichen Informationsträger in einem genetischen Algorithums werden - analog zur Biologie - als *Gene* bezeichnet.

Eruieren Sie, welche Arten von Genen im JGAP Framework verwendet werden können. Benennen Sie, welche Typen existieren und welche Art von Daten sie tragen. Die Gruppe der *Supergenes* (implementieren `org.jgap.supergenes.Supergene`) können Sie in der Betrachtung weglassen, ebenso alle abstrakten Klassen und Interfaces.
Inwiefern unterscheidet sich die Klasse `CompositeGene` von den anderen Gen-Klassen?

3.3 Verhalten (6 Punkte)

Bei der Ausführung eines genetischen Algorithmus wird eine künstliche Evolution durchgeführt. Zu Beginn wird, anhand eines Beispielchromosoms, eine Population von Chromosomen mit zufälligen Werten erstellt. Mittels aus der Biologie entlehnten Techniken wie Selektion, Mutation und Crossover (*genetische Operatoren*) wird ausgehend von dieser Startpopulation eine künstliche Evolution simuliert. Die eigentliche Anwendung der Operatoren erfolgt in der Klasse `org.jgap.impl.GABreeder`. Beschreiben Sie das Verhalten nach einem Aufruf der Methode ***evolve(Population a_pop, Configuration a_conf)*** in der Klasse `GABreeder`. Stellen Sie den Ablauf der **Zeilen 44-67** (d.h. nur die ersten 24 Zeilen der Methode!) in einem UML-Sequenzdiagramm dar und berücksichtigen Sie nur die folgenden Klassen:

- `org.jgap.impl.GABreeder`
- `org.jgap.Population`
- `org.jgap.Configuration`
- `org.jgap.IChromosome`
- `org.jgap.FitnessEvaluator`

3.4 UML Notation

- Hilfe zur Notation von UML Diagrammen, insbesondere den hier verwendeten Klassen- und Sequenzdiagrammen finden Sie z.B. unter **`http://www.holub.com/goodies/uml/`**
- Ein gutes Tool zur Erstellung von UML Klassendiagrammen ist das kostenlose UMLet, zu finden unter **`http://www.umlet.com/`**. Es gibt jedoch eine grosse Auswahl an möglichen Programmen.
- UML Sequenzdiagramme lassen sich ebenfalls mit einer Vielzahl an Programmen entwerfen. Einen eher ungewohnten Ansatz verfolgt die Website **`http://www.websequencediagrams.com`**, auf der sich Sequenzdiagramme programmatisch erstellen lassen. Die Beispiele helfen beim Verständnis der Notation, ein Export als PNG-Bild oder PDF-Datei ist möglich.

4 Verbessern des Quelltextes (6 Punkte)

Das JGAP Framework umfasst viele ausführbare Beispiele für Probleme, die sich mittles genetischen Algorithmen lösen lassen. Die Beispiele befinden sich im Ordner `examples/src`. Für die nachfolgenden Aufgaben verwenden wir ein Beispiel, das sich dem sogenannten *Change-making problem*[1] widmet. Dabei wird versucht, einen gegebenen Betrag durch die kleinstmögliche Anzahl gegebener Münzen darzustellen.

Das Beispiel besteht aus den beiden Klassem

[1] `http://en.wikipedia.org/wiki/Change-making_problem`

- `examples/src.examples.MinimizingMakeChange`
- `examples/src.examples.MinimizingMakeChangeFitnessFunction`

Die erste Klasse beinhaltet das ausführbare Beispiel und den Ablauf der künstlichen Evolution. Die zweite Klasse ist die zu jedem genetischen Algorithmus benötigte *Fitnessfunktion*. Eine Fitnessfunktion bewertet die Fitness (d.h. die Güte) eines Chromosoms in Bezug auf eine gewählte Problemstellung. Ein einzelnes Chromosom stellt eine mögliche Lösung für das Problem dar, mit den Werten, die in den jeweiligen Genen gespeichert sind. Während es für ein Problem wie das Münzwechseln verschiedene *mögliche* Lösungen gibt, gibt es selten mehr als eine *optimale* Lösung. Die Fitnessfunktion bestimmt, welches die optimale Lösung ist, in dem sie jeder möglichen Konfiguration eines Chromosoms einen numerischen Wert, den *Fitnesswert* zuweist. Je nach Fitnessfunktion können dabei hohe Fitnesswerte entweder für eine gute oder eine schlechte Lösung stehen. Aufgabe des Frameworks ist es dann, mittels den in 3.3 beschriebenen genetischen Operatoren diese Fitnessfunktion zu minimieren/maximieren.

4.1 Dokumentation (4 Punkte)

Die Fitnessfunktion für das Münzwechsel-Problem bewertet zwei Aspekte eine möglichen Lösung: Die Differenz zum Zielbetrag (je kleiner desto besser) sowie die Anzahl verwendeter Münzen (je tiefer desto besser).

Erklären Sie anhand der Methode *MinimizingMakeChange.evaluate(IChromosome a_subject)*, wie der Fitnesswert für ein bestimmtes Chromosom berechnet wird. Ergänzen Sie die Dokumentation des Codes, indem Sie Methodenkommentare für die folgenden Methoden der Klasse *MinimizingMakeChange* schreiben:

- *evaluate(IChromosome a_subject)*
- *changeDifferenceBonus(double a_maxFitness, int a_changeDifference)*
- *computeCoinNumberPenalty(double a_maxFitness, int a_coins)*

Vervollständigen Sie die Javadoc-Kommentare der Methoden selbst und geben Sie kurze Erklärungen innerhalb des Codes. Folgen Sie dabei den Richtlinien des Java Style Guide (erhältlich auf der Webseite der Übungen). Die zu bearbeitenden Stellen im Code sind mit `TODO` markiert.

Die Kommentare können entweder in einem separaten Dokument erstellt werden, oder direkt im Code. Wenn Sie die Kommentare direkt in den Code schreiben, geben Sie bitte nur die betroffene Datei *MinimizingMakeChangeFitnessFunction.java* ab, keinesfalls das gesamte Framework.

4.2 Verbessern des Codes (2 Punkte)

Ein genetischer Algorithmus simuliert einen Evolutionsprozess. Mittels Mutation werden ein Teil der einzelnen Chromosomen verändert und danach mittels Selektion diejenigen Chromosomen ausgewählt, die eine gute Fitness besitzen. Ein Chromosom mit guter Fitness ist, eine entsprechende Fitnessfunktion vorausgesetzt, eine bessere Lösung für das Problem, als eines mit schlechter Fitness. Der Evolutionsprozess läuft normalerweise für eine bestimmte Anzahl Generationen, nach deren Durchlauf das Individuum mit der besten Fitness als gefundene Lösung präsentiert wird.

Ein inhärentes Merkmal eines genetischen Algorithmus' ist, dass eine gefundene Lösung innerhalb ihrer Population optimal ist. Es gibt aber keine Garantie dafür, dass nicht ein anderer Ablauf der Evolution eine bessere Lösung produziert hätte. Ob und wie eine bestimmte Lösung gefunden wird, hängt von verschiedenen Faktoren ab. Einen wesentlichen Einfluss auf das Resultat haben unter anderem:

- Die Anfangspopulation, welche meist zufällig erzeugt wird

- Die Grösse der Population
- Die Wahrscheinlichkeit, mit welcher eine Mutation oder ein Crossover-Vorgang eintritt
- Die Strenge des Selektionsprozesses
- Die Anzahl Generationen, die die künstliche Evolution durchläuft

Die Klasse *MinimizingMakeChange* besitze eine Methode, um die Fitness der erhaltenen Resultate zu optimieren. Der Benutzer kann eine untere Grenze für die gewünschte Fitness festlegen, welcher im Attribut `MIN_FITNESS_VALUE` festgelegt wird. Entdeckt das Programm, dass die Resultate nach dem ersten Durchlauf der Evolution nicht die definierte Anforderung erfüllen, d.h. der Fitnesswert des besten Chromosoms liegt unter der spezifizierten Grenze, führt es die Evolution erneut durch. Maximal wird der Evolutionsprozess so viele Male wiederholt, wie durch das Feld `MAX_ATTEMPTS` vorgegeben ist. Die Überprüfung der Resultate und eine allfällige Wiederholung des Vorgangs werden in der Methode *checkMinFitness(Configuration a_configuration)* durchgeführt.

Diese Methode hat aus der Sicht des Software Engineerings mehrere Schwachstellen. Betrachten Sie die Definition, den Ablauf und die Dokumentation der Methode und geben Sie in Stichworten an, was verbesserungswürdig an ihr ist. Sie brauchen keine verbesserte Version der Methode zu schreiben.