SAARLAND UNIVERSITY
COMPUTER SCIENCE

Proseminar "13 Wochen - 13 Sprachen"

Prof. Dr. Andreas Zeller, Jun.-Prof. Dr. Sebastian Hack
Kontakt: Martin Burger, Christoph Mallon

10. Aufgabenblatt – Eiffel

Ausgabe: 23. Juni 2009 Abgabe: 30. Juni 2009 Revision: 3368

1 Eiffel: Design by Contract

Eiffel ist eine objektorientierte, statisch typisierte Programmiersprache. Sie zeichnet sich vor allem durch "Design by Contract" aus – der Idee, das Zusammenspiel von Klassen durch "Verträge" zu regeln. Ein solcher Vertrag besteht aus

Vorbedingungen, die vom Aufrufer einzuhalten sind;

Nachbedingungen, die im Gegenzug vom Aufgerufenen garantiert werden;

Invarianten, die für alle Objekte einer Klasse vor und nach einem Methodenaufruf gelten.

Dazu stellt Eiffel differenzierte Mittel bereit, um Klassen, ihre einzelnen Funktionen sowie ganze Klassenhierarchien auf systematische Weise mit Zusicherungen auf Korrektheit der Abläufe und Zustände während der Laufzeit zu überwachen. Eiffel ist eine sehr elegantere und von Grund auf neu entworfene Sprache; es gibt sehr gute Tutorials zu Eiffel.[1]

2 Eine einfache Eiffel-Klasse

Hier ist ein einfaches Beispiel für eine Eiffel-Klasse in einer Datei `hello_world.e`, die Syntax und die Verwendung von Vor- und Nachbedingungen illustriert:

```
-- Einfaches Eiffel-Beispiel
class HELLO_WORLD
create make -- Definiert Konstruktor-Methode

feature

   -- Konstruktor
   make is
```

[1] http://www.cetus-links.org/oo_eiffel.html

```
      do
         io.put_string("Hello World!%N")
         io.put_string("9 / 3 = ")
         io.put_real(divide(9, 3));
         io.put_string("%N")
      end

   -- Methode
   divide(dividend, divisor: REAL): REAL is
      require -- Vorbedingung
         quotient_nonzero: divisor /= 0
      do      -- Körper
         Result := dividend / divisor
      ensure  -- Nachbedingung
         result_ok: dividend = Result * divisor
      end
end -- HELLO_WORLD
```

Zur Übersetzung finden Sie in den CIP-Pools den `SmartEiffel`-Compiler (s. Abschnitt 6). Rufen Sie den Eiffel-Compiler auf mit

```
$ compile hello_world
```

und rufen Sie das Programm auf mit

```
$ ./a.out
```

3 Ihre Aufgabe

Implementieren Sie eine elektronische Guthabenkarte in Eiffel. Die Klasse `PURSE` soll über die folgenden Eigenschaften verfügen:

- einen aktuellen Guthabenbetrag `balance`;
- einem maximalen Guthabenbetrag `max_balance`, den `balance` nicht überschreiten darf;
- eine `PIN` aus vier Ziffern;
- einen Konstruktor `make(max_balance: INTEGER, balance: INTEGER, pin: ARRAY[INTEGER])`, der eine Guthabenkarte mit den gegebenen Eigenschaften erzeugt;
- eine Methode `debit(amount: INTEGER): INTEGER`, mit der man einen Betrag `amount` von der Karte abbuchen kann; der erhaltene Betrag wird zurückgegeben.

Definieren Sie

- für die Attribute geeignete Invarianten, die Wertebereich und sonstige Beziehungen ausdrücken;
- für die Methoden geeignete Vor- und Nachbedingungen;
- eine Testmethode, die alle Methoden abdeckt.

4 Diskussion

Diskutieren Sie folgende Fragestellungen auf etwa einer DIN A4 Seite:

1. Welche Vorteile bringt Design by Contract? Welche Nachteile sind damit verbunden?
2. Welche Vorteile bieten die Spracheigenschaften von Eiffel im Vergleich zu Zusicherungen, wie Sie sie aus anderen Sprachen kennen? Denken Sie auch an die Integration von Contracts mit Vererbung.
3. Wie können Sie Design by Contract in herkömmlichen Sprachen nutzen? Betrachten Sie hierzu Werkzeuge wie JML oder Spec#.

5 Abgabe Ihrer Lösung

Drucken Sie Ihre Lösung aus und werfen Sie den Ausdruck bis zum **30. Juni 2009 um 9:00 Uhr** in den Briefkasten des Lehrstuhls für Softwaretechnik[2]. Vergessen Sie dabei nicht, Ihren Ausdruck mit Ihrem Namen und Ihrer Matrikelnummer zu versehen.

Bringen Sie bitte zum Proseminar ebenfalls einen Ausdruck mit, so dass Sie ihn bei etwaigen Diskussionen vorliegen haben.

6 Links und Hinweise

Zum Erstellen und Bearbeiten Ihres Programms können Sie einen gewöhnlichen Texteditor benutzen. Der GNU SmartEiffel-Compiler ist (optionaler) Bestandteil vieler Open Source-Distributionen. Im CIP-Pool müssen Sie zunächst zwei Umgebungsvariablen setzen, um ihn aufzurufen:

```
$ export SmartEiffel=/installer/import/linux/compilers/serc
$ export PATH=/installer/import/linux/compilers/SmartEiffel/bin:$PATH
```

[2] Gebäude E1 1, neben dem InfoPoint des Rechenzentrums.

Danach können Sie den Compiler aufrufen mit

```
$ compile meinProgramm
```

Das entstehende Programm können Sie dann so ausführen:

```
$ ./a.out
```

Mit Hilfe der Standard-Ausgabe `io` können Sie den Ablauf Ihres Programms verfolgen.