Opdracht JSF32, week 16

In deze opdracht experimenteren we met:

* Java Bytecode

Om goed met bytecode te kunnen werken, is het noodzakelijk dat je een overzicht hebt van wat het is, en hoe het gebruikt wordt. Lees hiertoe eerst het artikel:

JAVA bytecode: understanding bytecode makes you a better programmer <http://www.mastercorp.free.fr/Ing1/Cours/Java/java_lesson1/doc/Articles/Decompilation/Java%20bytecode.htm>.

Voor een overzicht van alle bytecode opcodes zie:

<http://en.wikibooks.org/wiki/Java_Programming/Byte_Code>

Voor de echte diepgraver: De exacte JVM specificaties zijn te vinden in:

<https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf>

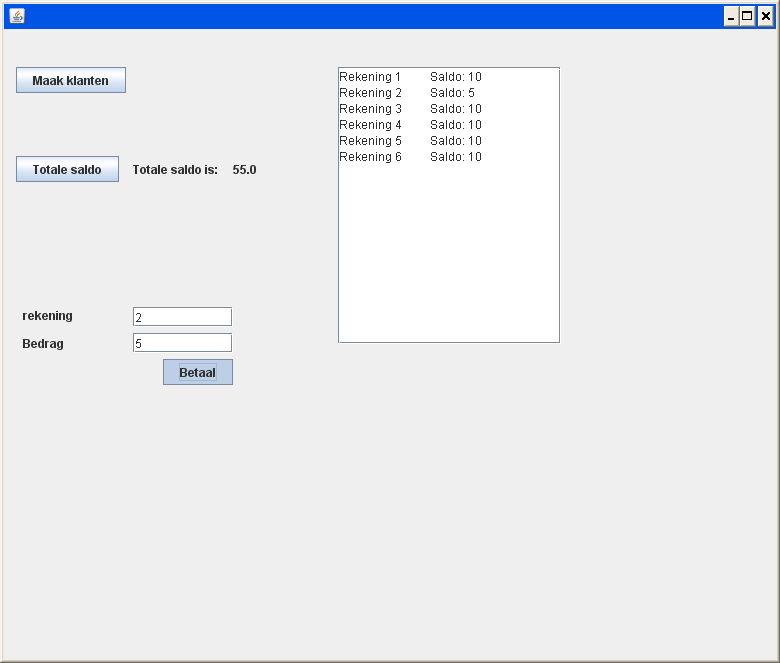
Verder heb je een bytecode editor nodig. gebruik hiervoor JBE (<http://www.cs.ioc.ee/~ando/jbe/>). Start deze door het runnen van de batchfile 'jbe.bat'.

Indien je de classfiles te ver verminkt hebt, kun je de originele .class files weer terugzetten vanuit de file 'build\_origineel.zip'.

# De opdracht

Gegeven is een applicatie OS\_ByteCodeOpgave, die je kunt runnen door het uitvoeren van de batchfile OS\_ByteCodeOpgave.bat.

Je ziet dat de sourcecode niet volledig meegeleverd is. Wel staan alle class files tot je beschikking.

Na start van de applicatie zie je het volgende window: 

Deze GUI stelt een proof of concept applicatie voor een bank voor.

Een bank bestaat uit klanten, en elke klant heeft één rekening. Na het drukken op de button 'Maak klanten' worden er 6 klanten aangemaakt, die ieder een bonus van 10 op hun rekening krijgen.

Achter het label: 'Totale saldo is:' wordt het totale saldo van alle rekeningen van de bank weergegeven, na het drukken op button 'Totale saldo'.

In de textarea rechts wordt per rekening het aanwezige saldo aangegeven.

Onderaan kun je voor één rekening een betaling uitvoeren, waardoor het saldo wijzigt.

**Opdracht 1**

In deze opdracht gaan we bytecode lezen en interpreteren.

a. Zoek in de classfiles (dus niet in de source code) uit, welke methodes aangeroepen worden bij:

het berekenen en tonen van het totale saldo (geef classfile en regelnummers in de classfile)

Bank.class 46-47

het betalen voor een rekening (geef ook weer classfile en regelnummers in de classfile)

Bank.class 49-50

b. In de BankFrame$x.class files worden ook klassen opgeslagen, die binnen de BankFrame klasse gebruikt worden.   
In welke regels van de source-code van BankFrame.java worden deze klassen gedefinieerd?

212, 291, 369

c. Welke attributen (Fields) en methoden heeft de Klant klasse (geef naam en type)?   
Methode <init> hoeft niet; dat is de constructor.

Fields:

-Naam : String

-mijnRekening : Rekening

Methoden:

-voegRekeningToe

-mijnSaldo

-getRekening

-verlaagSaldo

Reconstrueer voor elke methode de originele java-code:

methode voegRekeningToe:

0 aload\_0 // zet this op stack

1 aload\_1 // zet 1e parameter op stack

2 putfield #3 <os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening;>

// ken 1e parameter toe aan veld mijnRekening van this (this.mijnRekening)

5 return // keer terug uit methode

Dus de java-code was:

public void voegRekeningToe(Rekening rek) {

this.rekening = rek;

}

Methode mijnSaldo:

0 aload\_0 //zet this op stack

1 getfield #3 <os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening;>

4 invokevirtual #4 <os\_bytecodeopgave/Rekening/getSaldo()I>

7 ireturn

Java-code was:

Public Saldo mijnSaldo(){

Return mijnRekening.getSaldo();

}

Methode getRekening(): ByteCode:

0 new #5 <os\_bytecodeopgave/Rekening> //maak new Rekening object

3 dup //use this

4 aload\_0 /zet this op stack

5 getfield #3 <os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening;> //use this rekening

8 invokevirtual #6 <os\_bytecodeopgave/Rekening/getNummer()I> use rekening.getnummer();

11 invokespecial #7 <os\_bytecodeopgave/Rekening/<init>(I)V> puts result on stack

14 astore\_1 puts result on local variable

15 aload\_1 load rekening;

16 aload\_0 load this

17 getfield #3 <os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening;>

20 invokevirtual #4 <os\_bytecodeopgave/Rekening/getSaldo()I>

23 invokevirtual #8 <os\_bytecodeopgave/Rekening/changeSaldo(I)V>

26 aload\_1

27 areturn

Public Rekening getRekening(){

Rekening rekening = new Rekening(mijnRekening.getNummer);

rekening.changeSaldo(this.mijnRekening.getSaldo();

return rekening();

}

Methode verlaagSaldo(int bedrag) : ByteCode:

0 aload\_0 zet this op stack

1 getfield #3 <os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening;>

4 iload\_1 load int value bedrag

5 invokestatic #9 <java/lang/Math/abs(I)I> create Math.abs(bedrag) and puts result on stack

8 ineg negate int

9 invokevirtual #8 <os\_bytecodeopgave/Rekening/changeSaldo(I)V> changeSaldo on stack

12 return

Methode verlaagSaldo() : Java code

Public void verlaagSaldo(){

mijnRekening.changeSaldo(-Math.abs(bedrag));

}

**Opdracht 2**

In deze opgave gaan we de bonus bij het openen van een rekening eens misbruiken.

Zoek waar in de classfiles de rekeningen aangemaakt worden, en in welke methode de bonus op de rekening gezet wordt.

Pas deze methode aan, zodat een hogere bonus gegeven wordt. Toon dit met een hogere bonus van 100 en van 1000.

|  |
| --- |
| In de methode bonusRekening van Rekening.class veranderen we op bytecode regel 6 ‘bipush 10’ naar ‘bipush 100’ voor een toename van 100. Wanneer we dit doen met 1000 krijgen we een error. Na even gerommeld te hebben met de code, blijkt dat bipush een limiet heeft van 127 (7 bits dus). Maar na even op google te hebben gezocht, kwamen we er achter dat je ‘sipush’ kunt gebruiken voor grotere getallen. We hebben dus ‘bipush 10’ veranderd in ‘sipush 1000’ |

**Opdracht 3**

In deze opdracht gaan we ervoor zorgen, dat klanten onbeperkt kunnen opnemen van hun rekening, zonder dat hun saldo verandert. Pas hiertoe methode verlaagSaldo van de Klant klasse aan. Toon aan dat je applicatie werkt door hem te runnen.

|  |
| --- |
| In de methode verlaagSaldo van Klant.class haal ik regel 6 ‘invokevirtual os\_bytecodeopgave/Rekening/changeSaldo(I)V’ weg, waardoor de methode changeSaldo van Rekening.class niet meer wordt aangeroepen, waardoor het saldo niet meer wordt verlaagd. |

**Opdracht 4**

Hetzelfde als opdracht 3, maar nu alleen voor rekening 4. Rekening 4 mag dus niet wijzigen bij betalen, de andere rekeningen wel. Breng weer een verandering aan in verlaagSaldo van Klant.

Gebruik hiervoor een opcode die vergelijkt met een constante integer waarde (4). Deze integer waarde moet eerst op de stack gepusht worden.

|  |
| --- |
| In de methode verlaagSaldo van Klant.class voeg ik dit toe (voor de ‘originele’ code)  aload\_0  getfield os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening;  invokevirtual os\_bytecodeopgave/Rekening/getNummer()I  iconst\_4  if\_icmpne 7  return |