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:

<http://docs.oracle.com/javase/specs/jvms/se5.0/html/Instructions2.doc.html>

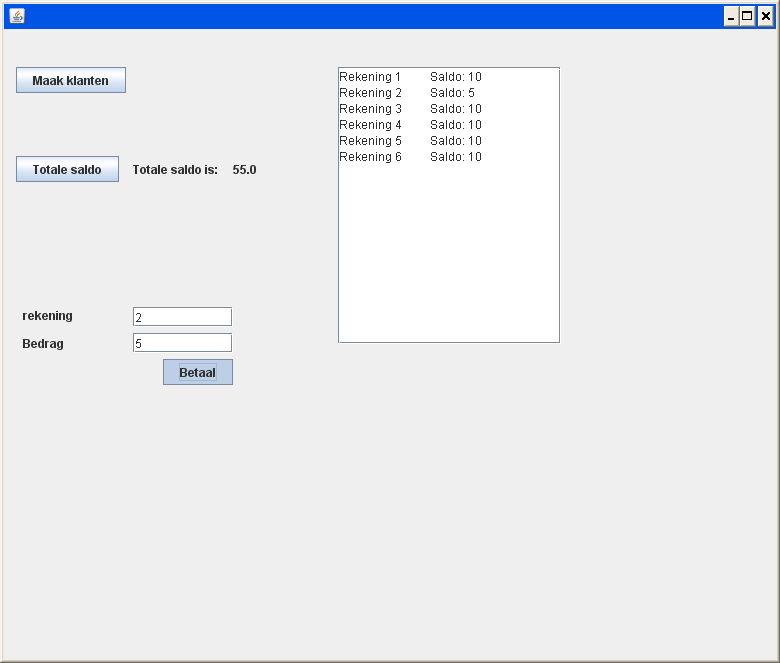
Verder heb je een bytecode editor nodig. gebruik hiervoor JBE (<http://www.cs.ioc.ee/~ando/jbe/>). Start deze door het runnen van de batchfile 'jbe.bat'.

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)

|  |  |
| --- | --- |
| **Methode** | **Aanroepplaats** |
| Bank.getTotalSaldo() |  |
| Klant.mijnSalso() | Bank.getTotalSaldo() regel 16 |
| Rekening.getSaldo() | Klant.mijnSalso() regel 3 |

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

|  |  |
| --- | --- |
| **Methode** | **Aanroepplaats** |
| Bank.betaal() |  |
| Klant.getRekening() | Bank.betaal() regel 13 |
| Rekening.getNummer() | Klant.getRekening() regel 5 |
| Rekening.getSaldo() | Klant.getRekening() regel 11 |
| Rekening.changeSaldo() | Klant.getRekening() regel 12 |

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?

43,55,63,115

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:Void

mijnSaldo:Int

getRekening:Rekening

verlaagSaldo:Void

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;

}

**mijnSaldo**

aload\_0 //zet this op de stack

getfield os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening; //haalt het veld mijnRekening op van het type Rekening

invokevirtual os\_bytecodeopgave/Rekening/getSaldo()I //voert de funtie Rekening.getSaldo() uit

ireturn //returned het resultaat

public Int mijnSaldo() {

return mijnRekening.getSaldo();

}

**getRekening**

new os\_bytecodeopgave/Rekening //Creëert een nieuw Rekening object

dup //Maakt een kopie van het Rekening object

aload\_0 //zet this op de stack

getfield os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening; //haalt het veld mijnRekening op van het type Rekening

invokevirtual os\_bytecodeopgave/Rekening/getNummer()I //voert mijnRekening.getNummer() uit.

invokespecial os\_bytecodeopgave/Rekening/<init>(I)V //voert de constructor van een nieuw Rekening object uit met mijnRekening.getNummer() als Parameter.

astore\_1 //Slaat het Rekening object op in een lokale variabele.

aload\_1 //Zet het rekening object op de Stack

aload\_0 //Zet this op de Stack

getfield os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening; //haalt het veld mijnRekening op van het type Rekening

invokevirtual os\_bytecodeopgave/Rekening/getSaldo()I // voert mijnRekening.getSaldo uit()

invokevirtual os\_bytecodeopgave/Rekening/changeSaldo(I)V //veranderd de waarde van de nieuwe rekening naar mijnRekening.getSaldo()

aload\_1 //zet de nieuwe rekening bovenaan de stack

areturn //returned de nieuwe rekening

public Rekening getRekening() {

Rekening rek = new Rekening(mijnRekening.getNummer);

rek.changeSaldo(mijnRekening.getSaldo();

return rek;

}

**verlaagSaldo**

aload\_0 //Zet this op de stack

getfield os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening; //haalt het veld mijnRekening op van het type Rekening

iload\_1 //Haalt de eerste parameter van het type Int op.

invokestatic java/lang/Math/abs(I)I //Voert de statische methode java.lang.Math.abs() uit met de eerste parameter als parameter.

ineg //Maakt de Int een negatief getal

invokevirtual os\_bytecodeopgave/Rekening/changeSaldo(I)V //voert de functie changeSaldo() uit met als parameter het resultaat van de 2 vorige berekeningen.

return //Return de methode.

public void verlaagSaldo(int bedrag) {

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.

**100**

wide

bipush 100

**1000**

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.

aload\_0

getfield os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening;

iload\_1

invokestatic java/lang/Math/abs(I)I

ineg

return

**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.

aload\_0

getfield os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening;

invokevirtual os\_bytecodeopgave/Rekening/getNummer()I

iconst\_4

if\_icmpeq 12

aload\_0

getfield os\_bytecodeopgave/Klant/mijnRekening Los\_bytecodeopgave/Rekening;

iload\_1

invokestatic java/lang/Math/abs(I)I

ineg

invokevirtual os\_bytecodeopgave/Rekening/changeSaldo(I)V

return