OO Programmeren III

Lorenz Verschingel

13 januari 2015

Samenvatting

Samenvatting OO Programmeren III HoGent

Inhoudsopgave

1 Datastructuren

1.1 Datastructuren met een vaste lengte

- Eéndimensionale arrays
- Meerdimensionale arrays

1.2 Dynamische datastructuren

De lengte van de datastructuur kan at runtime groter of kleiner worden.

Enkele voorbeelden:

- Gelinkte lijsten
- Stacks
- Queues
- Binaire bomen

Om tot deze dynamische datastructuren te komen heeft men een zelf-referentie klasse nodig:

```
1  class Node {
2    private int data;
3    private Node next;
4
5    // constructor
6    public Node (int data) {...}
7
8    public void setData (int data) {...}
9    public int getData() {...}
10   public void setNext (Node next) {...}
11   public Node getNext() {...}
12 }
```

De instantie-variabele nextNode wordt een link genoemd.

2 Collections

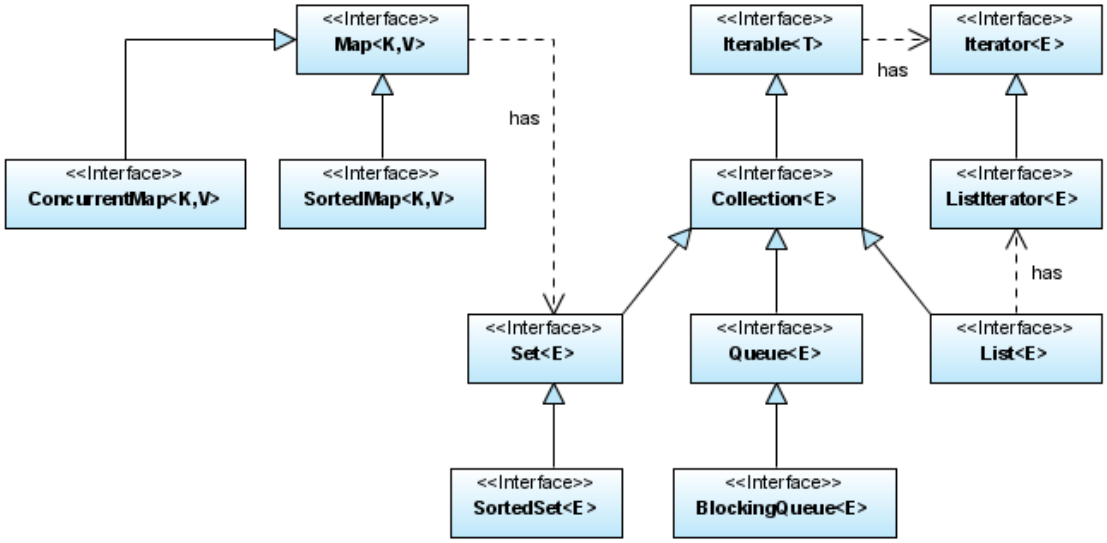
2.1 Inleiding

In het collection framework in Java zitten de meest voorkomende datastrucuren. Deze zijn gestandaardiseerd en efficient geïmplementeerd.

De collections van het Collection framework zijn niet gesynchroniseerd. Dit kan echter wel bekomen worden door het gebruik van de synchronization wrapper class.

2.2 Overzicht van het collection framework

Een collection is een datastructuur die referenties bijhoud naar andere objecten.




```
Interface Iterator<E>{
  boolean hasNext();
  E next();
  void remove();
}
```

```
Interface ListIterator<E> extends Iterator <E>{
  void add(E o);
  boolean hasPrevious();
  E previous();
  int nextIndex();
  int previousIndex();
  void set(E o);
}
```

2.2.1 Extra tools

1. Klasse Arrays:
   - methode asList
   - allerlei bewerkingen op arrays

2. Klasse Collections
    - allerlei bewerkingen op collections

2.3 Klasse Arrays

De klasse arrays voorziet static methodes om arrays te bewerken.

- binarySearch(array, searchValue)
- equals(array1, array2)
- fill(array,value)
- sort(array)
- asList(array)

2.4 Interface Collection en klasse Collections

2.4.1 Interface Collection<E>

De interface collections bevat bulk operations zoals adding, clearing, comparing...

- removeAll(Collection<?> c)
    - verwijdert alle elementen die in c zitten uit de Collection
- retainAll(Collection<?> c)
    - haalt de elementen op die zowel in c als in de Collection zitten

De interface voorziet ook een Iterator<E>, deze dient om alle elementen in een collection te doorlopen. De iterator kan ook elementen verwijderen.

2.4.2 Klasse Collections

De klasse Collections voorziet static methodes die collections manipuleren. Hierbij wordt polymorfisme ondersteund.

2.5 Interface List<E>

Een list is een geordende collection waarbij duplicaten toegelaten zijn. De list index start van nul. Er zijn verschillende implementatie klassen.

Arraylist

- Wijst effectief naar een array. Alle nodes zitten mooi na elkaar
- + Random acces is makkelijk
- - invoegen in midden ⇒ overige elementen moeten opschuiven.

LinkedList

- Iedere node zit ergens anders in het geheugen
- - Geen random acces ⇒ altijd bij de eerste node beginnen

- \+ Makkelijk toevoegen en verwijderen van nodes op een random plaats. (Enkel enkele referenties aanpassen)

Een andere implementatie is de klasse Vector. Dit is een verouderde klasse en is in essentie een gesynchroniseerde ArrayList.

2.6 Algoritmes uit het Collection framework

2.6.1 List algoritmes

- sort
  - de sorteervolgorde wordt bepaald door de compareTo-methode uit de interface comparable. Deze sorteervolgorde wordt de natuurlijke sorteervolgorde genoemd.
  - Deze volgorde kan ook aangepast worden via een klasse die de interface Comparator implementeert en waarin de methode compare wordt gedefiniëerd. Er bestaan ook voorgedefiniëerde comparators.
- binarySearch
- reverse
- shuffle
- fill
- copy

2.6.2 Collection algoritmes

- min
- max
- addAll
- frequency
- disjoint

2.7 Klasse Stack

Stack<E> is een subklasse van Vector<E>. In een stack kun je objecten plaatsen via push() en objecten afhalen via pop(). Een stack werkt volgens het LIFO-principe.

Verder zijn er ook nog de methodes empty(), peek(), search()...

Er dient wel opgemerkt te worden dat de klasse Stack sinds Java 7 in ongebruik is geraakt. Er wordt nu gebruik gemaakt van de klasse deque. Een deque is een dubble ended queue.

Deze laatste opmerking geldt eveneens voor de klasse Queue, die hierna besproken wordt. Ook hier werd erover gestapt naar de klasse Deque.

2.8 Interface Queue

In een queue kun je objecten plaatsen via offer() en objecten ophalen via poll(). Queues werken volgens het FIFO-principe.

2.8.1 De klasse PriorityQueue

In een PriorityQueue worden elementen gesorteerd volgens de natuurlijke ordening of volgens een Comparator-object.

2.9 Interface Set

Een Set is een collectie die unieke elementen bevat. Hierbij moet opgelet worden als men eigen klasses gebruikt: men dient hier de equals() en hashCode() te overriden.

- HashSet: niet gesorteerd
- TreeSet: gesorteerd

2.10 Interface Map

Een Map is een verzameling van key-value paren. Bij elke sluterl hoort precies 1 waarde.

- HashMap: niet gesorteerd
- TreeMap: gesorteerd

Met de methode get(K) haalt men de value op van de key. Met de methode put(K,V) voegt men een element met een bepaalde key toe aan de Map.

Om collisions te voorkomen maakt Java gebruik van "hash buckets".

Views:

- Map.keySet() → Set van uniek sleutels
- Map.value() → Collection met alle waardes
- Map.entrySet() → Verzameling van koppels

⇒ vergelijkbaar met een iterator op een map.

2.11 Wrappers

2.11.1 Unmodifiable wrappers

Convertie naar niet wijzigbare collections

2.11.2 Checked wrappers

Checked wrappers zijn collecties waarvan at runtome gecontroleerd wordt of er elementen van het juiste type worden toegevoegd.

2.11.3 Abstract implementations

Er zijn abstracte implementaties van de collection interfaces die als basis voor een zelf gedefiniëerde implementatie kunnen dienen.

3 Generics

3.1 Inleiding

Generiek programmeren is het definiëren van methodes/klassen, die voor verschillende types kunnen worden gebruikt. Hierbij wordt er hergebruik gemaakt van de code.

Er wordt pas at compile time een type toegevoegd waardoor er een veiligheid is ingebouwd.

Generiek programmeren is mogelijk sinds Java 1.5

3.2 Generieke methode

public <E> returntype methodenaam (E par)

- <E> is de parametersectie.
- E mag geen primitief datatype zijn.

3.3 Generieke klasse

```
1 public class NaamKlasse <E>{
2   private E attr
3   public void setAttr(E nieuw)
4   public E getAttr()
5 }
```

3.4 Generiek type beperkingen opleggen

Voor <E> is de default upperbound de klasse Object.

<E extends Number> laat enkel klasses toe die erven van de klasse Number.

<E extends Comparable T> laat enkel klasses toe die de interface Comparable implementeren.

Er dient opgemerkt te worden dat zowel voor klasses als interface “extends” gebruikt wordt.

3.5 Generiek programmeren met arrays

Voor arrays moet er via een omweg gewerkt worden.

```
1 public class Stack< E >{
2   private final int SIZE; // aantal elementen
3   private int top; // locatie van de top
4   E[] elements; // array die de elementen zal bevatten
5
6   // defaultconstructor
7   // creëert een stack van 10 elementen
8   public Stack() {
9     this(10);
10  }
11
12  // constructor;
13  // creëert een stack van s of 10 elementen
14  public Stack(int s) {
15    SIZE = s > 0 ? s : 10; // set size of Stack
16    top = -1; // stack is leeg
17    elements = (E[]) new Object[SIZE]; // creatie van de array
```

```
18    }
19 }
```

3.6 Wildcards

Wildcards is een andere manier om beperkingen op te leggen aan het generieke type.

public static double sum(Collection<? extends Number> list)
  = public static <T extends Number> double sum(Collection <T> list)

Men gebruikt wildcards voornamelijk als je T niet meer verder gebruikt in de body van de methode.

4 Files en streams

4.1 Inleiding

- Sequentiële File
  - Je doorloopt de gegevens in een sequentiële file van voor naar achter.
- Random Acces File
  - Je krijgt op willekeurige manier toegang tot de gegevens in een random acces file.

4.2 New Input Output

NIO is een uitbreiding op de bestaande IO sinds Java 1.4.

Het is een channel-based benadering voor IO.

- Channel: een open IO connectie.
- Buffer: bevat data.

4.3 Voorbeeld rekening applicatie

Listing 1: Rekening.java

```
1  package domein;
2
3  import java.nio.ByteBuffer;
4
5  public class Rekening {
6
7      public final static int SIZE = sizeBerekenen();
8      public final static int NAAMLENGTE = 20;
9      private int rekeningNr;
10     private String naam;
11     private double balans;
12
13     public static int sizeBerekenen() {
14         ByteBuffer buffer = ByteBuffer.allocate(100);
15         buffer.putInt(0);                    //rekeningnr
16         for (int i = 0; i < NAAMLENGTE; i++)//naam
17         {
18             buffer.putChar('A');
19         }
20         buffer.putDouble(0.0);               //balans
21         return buffer.position();
22     }
23
24     public Rekening() {
25         this(0, "", 0.0);
26     }
27
28     public Rekening(int nr, String naam, double bal) {
29         setRekeningNr(nr);
30         setNaam(naam);
31         setBalans(bal);
32     }
33
34     public void setRekeningNr(int nr) {
35         rekeningNr = nr;
36     }
37
```

```
38      public int getRekeningNr() {
39          return rekeningNr;
40      }
41
42      public void setNaam(String naam) {
43          this.naam = naam;
44      }
45
46      public String getNaam() {
47          return naam;
48      }
49
50      public void setBalans(double bal) {
51          balans = bal;
52      }
53
54      public double getBalans() {
55          return balans;
56      }
57
58      @Override
59      public String toString() {
60          return String.format("%d %s %.2f", getRekeningNr(),
61                  getNaam(), getBalans());
62      }
63  }
```

Merk op dat in de klasse rekening een static attribuut SIZE is voor zien. Deze geeft de grootte terug die de buffer moet hebben om een rekening-object te kunnen bevatten.

Listing 2: NioRekeningApplicatie

```
1  package ui;
2
3  import domein.Rekening;
4  import java.io.RandomAccessFile;
5  import java.nio.channels.FileChannel;
6  import java.io.IOException;
7  import java.nio.ByteBuffer;
8  import java.util.Scanner;
9
10 public class NioRekeningApplicatie {
11
12     private RandomAccessFile hetBestand;
13     private FileChannel channel;
14     private ByteBuffer buffer = ByteBuffer.allocate(Rekening.SIZE);
15
16     public static void main(String arg[]) {
17         new NioRekeningApplicatie().run();
18     }
19
20     public void run() {
21         maakTestBestand();
22         vraagRecordOp();
23         verhoogBalansAlsSaldoVoldoende();
24         vraagRecordOp();
25     }
26
27     public void maakTestBestand() {
28         try {
29             hetBestand = new RandomAccessFile("nio.dat", "rw");
30             channel = hetBestand.getChannel();
31             Rekening[] rekeningen
32                     = {
33                         new Rekening(1000, "jan", 100.0),
34                         new Rekening(1001, "piet", 200.0),
35                         new Rekening(1002, "joris", 150.0),
```

```
36                          new Rekening(1003, "corneel", 0.0)
37                      };
38              int recNr = 1;
39              for (Rekening r : rekeningen) {
40                  schrijfRecord(r, recNr++);
41              }
42              hetBestand.close();
43          } catch (IOException e) {
44              e.printStackTrace();
45              System.exit(1);
46          }
47      }
48
49      public void schrijfRecord(Rekening r, int volgnr) {
50          buffer.clear();
51          buffer.putInt(r.getRekeningNr());
52          StringBuffer hulp = new StringBuffer(r.getNaam());
53          hulp.setLength(Rekening.NAAMLENGTE);
54          String naam = hulp.toString().replace('\0', ' ');
55          for (int i = 0; i < Rekening.NAAMLENGTE; i++) {
56              buffer.putChar(naam.charAt(i));
57          }
58          buffer.putDouble(r.getBalans());
59          buffer.flip();
60          try {
61              channel.position((volgnr - 1) * Rekening.SIZE);
62              channel.write(buffer);
63          } catch (IOException e) {
64              e.printStackTrace();
65              System.exit(1);
66          }
67      }
68
69      public void vraagRecordOp() {
70          try {
71              hetBestand = new RandomAccessFile("nio.dat", "r");
72              channel = hetBestand.getChannel();
73              System.out.print("Geef record volgnr : ");
74              Scanner in = new Scanner(System.in);
75              int volgnr = in.nextInt();
76              channel.position((volgnr - 1) * Rekening.SIZE);
77              buffer.clear();
78              channel.read(buffer);
79              buffer.flip();
80              int nr = buffer.getInt();
81              char ar[] = new char[Rekening.NAAMLENGTE];
82              for (int i = 0; i < ar.length; i++) {
83                  ar[i] = buffer.getChar();
84              }
85              String naam = new String(ar).trim();
86              double balans = buffer.getDouble();
87
88              Rekening r = new Rekening(nr, naam, balans);
89              System.out.printf("%s\n", r);
90              hetBestand.close();
91          } catch (IOException e) {
92              e.printStackTrace();
93              System.exit(1);
94          }
95      }
96
97      public void verhoogBalansAlsSaldoVoldoende() {
98          try {
99              hetBestand = new RandomAccessFile("nio.dat", "rw");
100             channel = hetBestand.getChannel();
101             ByteBuffer buf = ByteBuffer.allocate(8);
102             channel.position(Rekening.SIZE - 8);
```

```
103             while (channel.read(buf) != -1) {
104                 buf.flip();
105                 double bal = buf.getDouble();
106                 if(bal>=150){
107                     bal*=1.1;
108                     buf.clear();
109                     buf.putDouble(bal);
110                     buf.flip();
111                     channel.position(channel.position()-8);
112                     channel.write(buf);
113                 }
114                 buf.clear();
115                 channel.position(channel.position()+Rekening.SIZE-8);
116             }
117             hetBestand.close();
118         } catch (IOException e) {
119             e.printStackTrace();
120             System.exit(1);
121         }
122     }
123 }
```

Op lijn 27 begint de methode die het test bestand aanmaakt.

Merk op dat de volgorde van werken als volgt is voor het schrijven:

1. Maak een nieuw RandomAccesFile aan en geef aan dat er gelezen en geschreven mag worden naar dit bestand. (lijn 29)
2. Haal de Channel van het bestand op. (lijn 30)
3. Schrijf de rekeningen naar het bestand.
4. Maak de buffer leeg. (lijn (0)
5. Zet alle attributen op de buffer. (lijn 51-59)
6. Zet de positie juist in de Channel. (lijn 61)
7. Schrijf de buffer naar de Channel (lijn 62)
8. Sluit het RandomAccesFile. (lijn 42)

Het lezen gaat als als volgt:

1. Maak een nieuw RandomAccesFile aan en geef aan dat er gelezen en geschreven mag worden naar dit bestand. (lijn 71)
2. Haal de Channel van het bestand op. (lijn 72)
3. Zet de positie juist in de Channel. (lijn 76)
4. Leeg de buffer (lijn77)
5. Lees de Channel in op de buffer. (lijn 78)
6. Flip de buffer. (lijn 79)
7. Lees de buffer uit. (lijn 80-86)
8. Sluit het bestand (lijn 90)

5 MVC - JavaFX

Zie slides

5.1 ListView

Listing 3: StartUp

```
import domein.HeroWorld;
import gui.HeroesFrame;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.stage.WindowEvent;

public class StartUp extends Application {

    @Override
    public void start(Stage stage) {
        HeroWorld domeinController = new HeroWorld();
        Scene scene = new Scene(new HeroesFrame(domeinController));
        stage.setScene(scene);

        // The stage will not get smaller than its preferred (initial) size.
        stage.setOnShown((WindowEvent t) -> {
            stage.setMinWidth(stage.getWidth());
            stage.setMinHeight(stage.getHeight());
        });
        stage.show();
    }

    public static void main(String... args) {
        Application.launch(StartUp.class, args);
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<fx:root type="GridPane" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://
    javafx.com/fxml/1">
  <columnConstraints>
    <ColumnConstraints hgrow="SOMETIMES" maxWidth="1.7976931348623157E308"
        minWidth="10.0" prefWidth="150.0" />
    <ColumnConstraints hgrow="SOMETIMES" maxWidth="127.0" minWidth="10.0"
        prefWidth="51.0" />
      <ColumnConstraints hgrow="SOMETIMES" maxWidth="1.7976931348623157E308"
          minWidth="10.0" prefWidth="150.0" />
  </columnConstraints>
  <rowConstraints>
    <RowConstraints maxHeight="37.0" minHeight="10.0" prefHeight="32.0" vgrow="
        SOMETIMES" />
    <RowConstraints maxHeight="1.7976931348623157E308" minHeight="10.0"
        prefHeight="300.0" vgrow="SOMETIMES" />
  </rowConstraints>
   <children>
      <Label contentDisplay="CENTER" text="Candidates" GridPane.halignment="
          CENTER" />
```

```
22        <Label text="Heroes" GridPane.columnIndex="2" GridPane.halignment="CENTER
              " />
23        <GridPane GridPane.columnIndex="1" GridPane.rowIndex="1">
24          <columnConstraints>
25            <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth
                  ="100.0" />
26          </columnConstraints>
27          <rowConstraints>
28            <RowConstraints maxHeight="145.0" minHeight="10.0" prefHeight="76.0"
                  vgrow="SOMETIMES" />
29            <RowConstraints maxHeight="224.0" minHeight="10.0" prefHeight="224.0"
                   vgrow="SOMETIMES" />
30          </rowConstraints>
31           <children>
32              <Button fx:id="btnSendLeft" alignment="CENTER" disable="true"
                      mnemonicParsing="false" onAction="#sendLeft" text="&lt;&lt;"
                      GridPane.halignment="CENTER" GridPane.rowIndex="1" GridPane.
                      valignment="TOP" />
33              <Button fx:id="btnSendRight" alignment="CENTER" mnemonicParsing="
                      false" onAction="#sendRight" text="&gt;&gt;" textAlignment="
                      CENTER" GridPane.halignment="CENTER" />
34          </children>
35        </GridPane>
36        <ListView fx:id="listCandidates" onMouseClicked="#mouseClickedCandidate"
              prefHeight="200.0" prefWidth="200.0" GridPane.rowIndex="1" />
37        <ListView fx:id="listHeroes" prefHeight="200.0" prefWidth="200.0"
              GridPane.columnIndex="2" GridPane.rowIndex="1" />
38    </children>
39 </fx:root>
```

Listing 4: FrameController

```
1  package gui;
2
3  import domein.HeroWorld;
4  import java.io.IOException;
5  import javafx.collections.ListChangeListener;
6  import javafx.collections.ObservableList;
7  import javafx.event.ActionEvent;
8  import javafx.fxml.FXML;
9  import javafx.fxml.FXMLLoader;
10 import javafx.scene.control.Button;
11 import javafx.scene.control.ListView;
12 import javafx.scene.control.SelectionMode;
13 import javafx.scene.input.MouseEvent;
14 import javafx.scene.layout.GridPane;
15
16 public class HeroesFrame extends GridPane {
17
18     @FXML
19     private Button btnSendRight;
20
21     @FXML
22     private Button btnSendLeft;
23
24     @FXML
25     ListView<String> listCandidates;
26     @FXML
27     ListView<String> listHeroes;
28
29     private final HeroWorld domeinController;
30
31     public HeroesFrame(HeroWorld domeinController) {
32         this.domeinController = domeinController;
33
34         FXMLLoader loader = new FXMLLoader(getClass().getResource("HeroesFrame.
               fxml"));
```

```
35          loader.setRoot(this);
36          loader.setController(this);
37          try {
38              loader.load();
39          } catch (IOException ex) {
40              throw new RuntimeException(ex);
41          }
42
43          listHeroes.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE)
                ;
44
45          listCandidates.setItems(domeinController.getCandidates());
46          listHeroes.setItems(domeinController.getHeroes());
47
48          domeinController.getHeroes().addListener(
49                  (ListChangeListener<String>) e -> btnSendLeft.setDisable(
                        domeinController.heroesEmpty()));
50          domeinController.getCandidates().addListener(
51                  (ListChangeListener<String>) e -> btnSendRight.setDisable(
                        domeinController.candidatesEmpty()));
52      }
53
54      @FXML
55      private void sendRight(ActionEvent event) {
56          int index = listCandidates.getSelectionModel().getSelectedIndex();
57          addHero(index);
58      }
59
60      private void addHero(int index) {
61          if (index != -1) {
62              domeinController.addHero(index);
63              listCandidates.getSelectionModel().clearSelection();
64          }
65      }
66
67      @FXML
68      private void sendLeft(ActionEvent event) {
69          ObservableList<String> list = listHeroes.getSelectionModel().
                getSelectedItems();
70          addCandidate(list);
71      }
72
73      private void addCandidate(ObservableList<String> list) {
74          if (!list.isEmpty()) {
75              btnSendRight.setDisable(false);
76              domeinController.removeHeroes(list);
77              listHeroes.getSelectionModel().clearSelection();
78          }
79      }
80
81      @FXML
82      private void mouseClickedCandidate(MouseEvent event) {
83          if (event.getClickCount() == 2) {
84              addHero(listCandidates.getSelectionModel().getSelectedIndex());
85          }
86      }
87
88  }
```

Let er hier op hoe met de setItems methode de lijst opgevuld word.

Listing 5: HeroWorld

```
1  package domein;
2
3  import java.util.Arrays;
4  import javafx.collections.FXCollections;
```

```
5  import javafx.collections.ObservableList;
6
7  public class HeroWorld {
8      private ObservableList<String> candidates;
9      private ObservableList<String> heroes;
10
11     //voorbeeld zonder databank
12     private final String[] candidatesArray = {"Superman",
13         "Spiderman",
14         "Wolverine",
15         "Thor",
16         "Police",
17         "Fire Rescue",
18         "Iron Man",
19         "Soldiers",
20         "Dad & Mom",
21         "Doctor",
22         "Politician",
23         "Teacher"};
24
25
26     public HeroWorld(){
27         candidates = FXCollections.observableArrayList(Arrays.asList(
               candidatesArray));
28         heroes = FXCollections.observableArrayList();
29     }
30
31     public ObservableList<String> getCandidates(){
32         return candidates;
33     }
34
35     public void addHero(int index){
36         heroes.add(candidates.remove(index));
37     }
38
39     public ObservableList<String> getHeroes() {
40         return heroes;
41     }
42
43     public boolean heroesEmpty(){
44         return heroes.isEmpty();
45     }
46
47     public boolean candidatesEmpty() {
48         return candidates.isEmpty();
49     }
50
51     public void removeHeroes(ObservableList<String> names){
52         candidates.addAll(names);
53         heroes.removeAll(names);
54     }
55 }
```

De methode getCandidates geeft een ObservableList terug. Dit betekent dat er vanaf nu automatisch het observer pattern wordt toegepast.

5.2 TableView

Bij een TableView gebeurt bijna hetzelfde als bij de ListView. Men geeft in de controller een ListView van objecten door aan de Table. Daarna koppel je alle kolommen aan een gewenste property van Persoon.

```
1 @FXML
2 private TableView<Person> tableView;
3 @FXML
```

```
4 private TableColumn<Person, String> firstNameCol;
5
6 tableView.setItems(personenLijst);
7 firstNameCol.setCellValueFactory(
8   cellData -> cellData.getValue().firstNameProperty());
```

In de klasse Persoon dient men hiervoor ook een aantal aanpassingen te doen. Zo kan men het attribuut firstName niet meer opslaan als een String. Hiervoor moet nu een StringProperty gebruikt worden.

```
1 private final SimpleStringProperty firstName = new SimpleStringProperty();
2
3 private void setFirstName(String fName) {
4   firstName.set(fName);
5 }
6
7 public String getFirstName() {
8   return firstName.get();
9 }
10
11 public StringProperty firstNameProperty() {
12   return firstName;
13 }
```

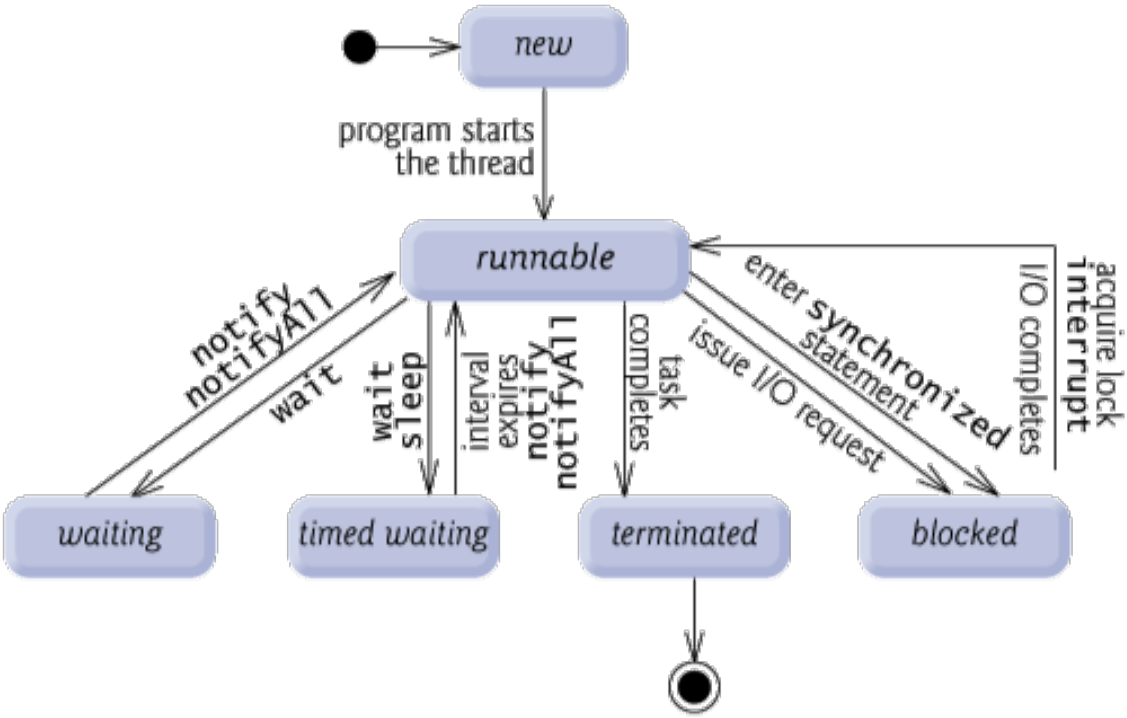
6 Multithreading

6.1 Inleiding

Threads zijn delen van het programma die in concurrentie met elkaar gelijktijdig in executie gaan. Een thread is een sequentiële besturingsstroom. Het zijn 'lichtgewicht' processen.

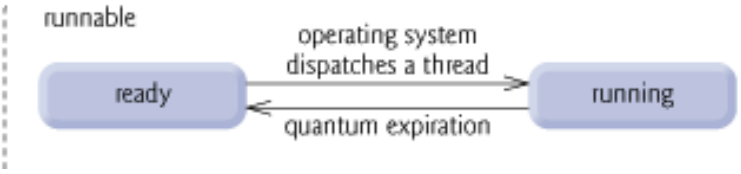
Java voorziet primitieven voor multithreading.

6.2 Toestanden van een thread



- new state
  - Thread is juist gecreëerd.
- runnable state
  - start method van thread geactiveerd.
  - Thread wordt beschouwd als in executie zijnde.
- waiting state
  - Een deel code nog niet kan uitgevoerd worden (bepaalde vereisten moeten voldaan zijn)
    - De thread activeert Object's wait methode. Keert terug naar runnable state doordat een andere thread de notify methode activeert.
    - De thread wordt geblokkeerd (lock), door bv io verzoek totdat die gedeblokkeerd wordt (unlock).

- timed waiting state
  - Wacht op andere thread of het einde van opgegeven tijdsinterval.
  - Wacht tot opgegeven "sleep" tijdsinterval is verstreken.
- terminated state
  - Thread is beëindigd (taak volbracht of exit).
  - Garbage collector kan geheugen terug vrijgeven als er geen referentie meer is naar het thread object.
- Blocked state
  - Als een taak niet onmiddellijk kan volbracht worden (vb. i/o verzoek).



- runnable state (JVM)
  - Ready state (OS)
    - Initiële state, timeslice/quantum vervallen.
  - Running state (OS)
    - Thread is toegekend aan een processor en in executie (dispatching).

Thread scheduling is functie van OS en thread priorities.

6.3 Creatie en executie van threads

Een klasse die moet kunnen runnen, moet Runnable implementeren.

In de klasse die de runnables moet laten lopen moet je gebruik maken van een threadpool, nl. ExecutorService.

6.3.1 Voorbeeld

In dit voorbeeld is het de bedoeling dat 3 willekeurige tellers tot 10 tellen.

Listing 6: main

```
1  package multithreadingtest;
2
3  import java.util.concurrent.ExecutorService;
4  import java.util.concurrent.Executors;
5
6  /**
7   *
8   * @author Lorenz
9   */
10 public class MultithreadingTest {
11
12     public static void main(String[] args) {
13         Counter counter1 = new Counter(1);
14         Counter counter2 = new Counter(2);
15         Counter counter3 = new Counter(3);
```

```
16
17          ExecutorService threadExecutor = Executors.newFixedThreadPool(3);
18          threadExecutor.execute(counter1);
19          threadExecutor.execute(counter2);
20          threadExecutor.execute(counter3);
21
22          threadExecutor.shutdown();
23      }
24  }
```

Listing 7: Counter

```
1  package multithreadingtest;
2
3  /**
4   *
5   * @author Lorenz
6   */
7  public class Counter implements Runnable {
8
9      private int id;
10
11     public Counter(int id) {
12         this.id = id;
13     }
14
15     @Override
16     public void run() {
17         for (int i = 1; i <= 10; i++) {
18             System.out.println("Counter " + id + ": " + i);
19         }
20     }
21
22 }
```

Als je deze code laat runnen zie je dat de tellers door elkaar tot 10.

Om de teller elk om de beurt een te laten bijtellen moeten we de threads dus synchroniseren.

6.4 Thread synchronisatie

Er zijn meerdere manieren om threads te synchroniseren. Er kan gebruik gemaakt worden van het keyword synchronised voor de methode, maar deze methode is verouderd. Men kan gebruik maken van Lock's en ook van de ArrayBlockingQueue.

6.4.1 Locks

Het principe bij locks is vrij simpel:

1. Men locked het object.
2. Men zorgt ervoor dat het object enkel kan uitgevoerd worden als het mag. (typisch met while-lus en .await().)
3. Het kritische stuk, de code die voor problemen zou kunnen zorgen omdat 2 objecten tegelijk 1 item zouden willen veranderen, komt na de while.
4. Als de kritische stuk is uitgevoerd, voert men een .signal() uit naar de voorwaardes die nu voldaan zijn.
5. Als laatste unlocked men het object.

Hier volgt opniew het voorbeeld met de tellers. Deze keer is het de bedoeling dan elke teller om beurt een getal telt.

Listing 8: main

```java
package multithreading.synced;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class MultithreadingSynced {

    public static void main(String[] args) {
        Writer writer = new Writer(3);

        Counter counter1 = new Counter(0, writer);
        Counter counter2 = new Counter(1, writer);
        Counter counter3 = new Counter(2, writer);

        ExecutorService threadExecutor = Executors.newFixedThreadPool(3);
        threadExecutor.execute(counter1);
        threadExecutor.execute(counter2);
        threadExecutor.execute(counter3);

        threadExecutor.shutdown();
    }

}
```

Listing 9: Counter

```java
package multithreading.synced;

/**
 *
 * @author Lorenz
 */
public class Counter implements Runnable {

    private int id;
    private Writer writer;

    public Counter(int id, Writer writer) {
        this.id = id;
        this.writer = writer;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 10; i++) {
            writer.write(id, i);
        }
    }
}
```

Merk op dat de Counter klasse hier de synchronisatie overlaat aan de klasse Writer.

```java
package multithreading.synced;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
```

```
10  *
11  * @author Lorenz
12  */
13 public class Writer {
14
15     private Lock accesLock = new ReentrantLock();
16
17     private Condition canWrite = accesLock.newCondition();
18     int accesNumber = 0;
19     int numberOfThreads;
20
21     public Writer(int numberOfThreads) {
22         this.numberOfThreads = numberOfThreads;
23     }
24
25     public void write(int counterId, int count) {
26         accesLock.lock();
27         try {
28             while (accesNumber != counterId) {
29                 System.out.println("Counter " + counterId + " tries to write");
30                 canWrite.await();
31             }
32             System.out.println("Counter " + counterId + ": " + count);
33             accesNumber++;
34             accesNumber = accesNumber % numberOfThreads;
35             canWrite.signal();
36         } catch (InterruptedException ex) {
37             Logger.getLogger(Writer.class.getName()).log(Level.SEVERE, null, ex
                   );
38         } finally {
39             accesLock.unlock();
40         }
41
42     }
43 }
```

Nu is het tijd om de klasse Writer onder de loep te nemen.

Merk op lijn 30 op dat hier het object gelocked wordt.

Vanaf lijn 33 tot lijn 36 zorgt men ervoor dat de writer enkel kan schrijven als de Counter het juiste ID heeft.

Op lijn 37 voert men het kritische stuk uit, namelijk het printen van de tekst. en hierna wordt ervoor gezorgt dat het ID van de counter die mag printen aangepast wordt.

Alle objecten die wachten op het canWrite worden op lijn 40 met .signal() terug gewekt. en controleren in de while of ze hun kritisch stuk mogen uitvoeren.

Tot slotte word op lijn 44 het object geunlocked.

6.4.2 ArrayBlockingQueue

De interface BlockingQueue bevat de methoden put en take, welke equivalent zijn met de methoden offer en poll van de interface Queue.

- Methode “put” zal een element in de BlockingQueue plaatsen (m.a.w. het element wordt achteraan in de wachtrij geplaatst.)
- Methode “take” zal een element in de BlockingQueue ophalen (m.a.w. het eerste element in de wachtrij wordt opgehaald).

De klasse ArrayBlockingQueue maakt gebruik van een array. De array wordt hier als een statische circulaire buffer gebruikt.

Hier volgt een voorbeeld:

Listing 10: main

```
import domein.BlockingBuffer;
import domein.Consumer;
import domein.Producer;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/**
 *
 * @author Lorenz
 */
public class MAIN {

    public static void main(String[] args) {
        int sizeBuffer = 3;
        BlockingBuffer buffer = new BlockingBuffer(sizeBuffer);

        Producer producer = new Producer(buffer);
        Consumer consumer = new Consumer(buffer);

        ExecutorService threadService = Executors.newFixedThreadPool(2);
        threadService.execute(producer);
        threadService.execute(consumer);

        threadService.shutdown();
    }
}
```

Listing 11: Producer

```
package domein;

/**
 *
 * @author Lorenz
 */
public class Producer implements Runnable {

    private BlockingBuffer buffer;

    public Producer(BlockingBuffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 10; i++) {
            buffer.set(i);
        }
    }
}
```

Listing 12: Consumer

```
package domein;

/**
 *
 * @author Lorenz
 */
public class Consumer implements Runnable {

    private BlockingBuffer buffer;

```

```
11      public Consumer(BlockingBuffer buffer) {
12          this.buffer = buffer;
13      }
14
15      @Override
16      public void run() {
17          for (int i = 1; i <= 10; i++) {
18              int readValue = buffer.get();
19          }
20      }
21  }
```

Listing 13: BlockingBuffer

```
1  package domein;
2
3  import java.util.concurrent.ArrayBlockingQueue;
4  import java.util.logging.Level;
5  import java.util.logging.Logger;
6
7  /**
8   *
9   * @author Lorenz
10  */
11 public class BlockingBuffer {
12
13      private ArrayBlockingQueue<Integer> buffer;
14
15      public BlockingBuffer(int size) {
16          buffer = new ArrayBlockingQueue<>(size);
17      }
18
19      public void set(int value) {
20          try {
21              buffer.put(value);
22              System.out.println("Producer writes " + value + "\nBuffersize = " +
                    buffer.size());
23          } catch (InterruptedException ex) {
24              Logger.getLogger(BlockingBuffer.class.getName()).log(Level.SEVERE,
                    null, ex);
25          }
26      }
27
28      public int get() {
29          int readValue = 0;
30          try {
31              readValue = buffer.take();
32              System.out.println("Consumer reads " + readValue + "\nBuffersize =
                    " + buffer.size());
33          } catch (InterruptedException ex) {
34              Logger.getLogger(BlockingBuffer.class.getName()).log(Level.SEVERE,
                    null, ex);
35          }
36          return readValue;
37      }
38 }
```

Merk op dat in de klasse BlockingBuffer de synchronisatie verzorgt wordt door het attribuut buffer van het type ArrayBlockingQueue.

6.4.3 Interface TransferQueue<E>

Het is een subinterface van BlockingQueue met als implementatie LinkedTransferQueue<E>.

De voornaamste extra functionaliteit wordt geboden door de transfer() methode. De blokkering blijft duren todat de overdracht volledig is. Je geeft effectief een item van de ene thread door aan de andere waarna de ene thread weer verder loopt. Dit is soms belangrijk voor de synchronisatie ("happens-before relationships" in het Java Memory Model).

6.5 Interface Callable.

Stel dat de Runnable een berekening dient uit te voeren.

De methode run is een void-methode en kan dus niet het resultaat van de berekening teruggeven. De applicatie zal een object moeten creëren en doorgeven aan de thread, zodat het resultaat in dit object kan bewaard worden. Dit zorgt voor veel overhead.

Tevens kan de methode run geen checked exception werpen. Om deze tekorten weg te werken bestaat er nu, in J2SE 5.0, de interface Callable (van package java.util.concurrent).

De interface Callable bevat één methode : de methode call. Deze methode is zoals de methode run van de interface Runnable, maar ze kan een waarde teruggeven of een exception werpen.

De interface ExecutorService voorziet de methode submit. Hierdoor kan er een thread gekoppeld worden met een Callable.

De methode submit geeft een object van type Future (van package java.util.concurrent) terug. De interface Future voorziet de methode get, zodat de waarde, die de Callable teruggeeft, kan opgevraagd worden.

6.6 De methode join() van de klasse Thread

join(): blijft wachten totdat deze thread afloopt. Werpt een InterruptedException op als een andere thread deze thread onderbreekt.

join(long): wacht tot deze thread afloopt. Er kan een time-out in milliseconden gespecifieerd worden. Een time-out van 0 milliseconden betekent: blijft wachten. Werpt een InterruptedException op als een andere thread deze thread onderbreekt.

```
1  public class Consumer extends Thread {
2    private Buffer sharedLocation;
3    private Thread producer;
4
5    public Consumer( Buffer shared, Thread producer ) {
6      super( "Consumer" );
7      sharedLocation = shared;
8      this.producer = producer;
9    }
10
11   public void run() {
12     try {
13       producer.join(); // consumer blijft wachten totdat thread "producer
             gestorven is.
14     } catch ( InterruptedException exception ) {
15       exception.printStackTrace();
16     }
17     int getal = sharedLocation.get();
18     System.err.println( String.format(%s reads %d, getName(), getal));
19   } // end method run
20 } // end class Consumer
```

6.7 Parallelle streams

Nieuw sinds Java 7.

7 JPA

7.1 JPA procedure

1. Entiteiten vastleggen
   (a) Lege database creëren
2. Persistence unit
   (a) Database
   (b) Lijst van entiteiten
3. EntityManagerFactory maken
4. EntityManager (CRUD)
5. Transactie
   (a) Begin
   (b) Commit

7.2 Voorbeeld

Het is de bedoeling dat deze applicatie eerst alle auto's met een onderhoudsbeurt zal weergeven, daarna de auto's zonder. Daarna heeft de applicatie alle onderhoudsbeurten op een bepaalde datum weer.

Listing 14: Main App

```
1  package main;
2
3
4  import domein.GarageController;
5
6  public class MAINapp {
7      private GarageController dc;
8
9      public static void main(String arg[]) {
10         new MAINapp().run();
11     }
12
13     public void run() {
14         dc = new GarageController();
15         System.out.println("Garage Applicatie gestart");
16         System.out.printf("Auto's met onderhoud : %s%s", dc.
               geefAutosMetOnderhoudsbeurt(),System.lineSeparator());
17         System.out.printf("Auto's met zonder onderhoud : %s%s",dc.
               geefAutosZonderOnderhoudsbeurt(),System.lineSeparator());
18         System.out.printf("Onderhoudsbeurten op 2014/5/10 : %s%s",dc.
               geefOnderhoudsbeurtenOpDatum(2014, 5, 10),System.lineSeparator());
19     }
20
21 }
```

De abstracte klasse Vervoersmiddel zal later de basis vormen voor de klasses Auto en LichteVracht.

Listing 15: Vervoersmiddel

```
1  package domein;
2
3  import java.io.Serializable;
```

```
4  import java.util.ArrayList;
5  import java.util.Collections;
6  import java.util.GregorianCalendar;
7  import java.util.List;
8  import java.util.Objects;
9  import javax.persistence.CascadeType;
10 import javax.persistence.Entity;
11 import javax.persistence.GeneratedValue;
12 import javax.persistence.GenerationType;
13 import javax.persistence.Id;
14 import javax.persistence.Inheritance;
15 import javax.persistence.InheritanceType;
16 import javax.persistence.NamedQueries;
17 import javax.persistence.NamedQuery;
18 import javax.persistence.OneToMany;
19
20 @Entity
21 @NamedQueries({
22     @NamedQuery(name = "AlleVervoerZonderOnderhoud",
23             query = "SELECT v FROM Vervoermiddel v "
24                     + "WHERE SIZE(v.onderhoudsbeurten)=0"),
25     @NamedQuery(name = "AlleVervoerMetOnderhoud",
26             query = "SELECT v FROM Vervoermiddel v "
27                     + "WHERE SIZE(v.onderhoudsbeurten)>0")
28 })
29 @Inheritance(strategy = InheritanceType.JOINED)
30 public abstract class Vervoermiddel implements TebetalenTaks, Serializable {
31
32     @Id
33     @GeneratedValue(strategy = GenerationType.IDENTITY)
34     private long id;
35
36     private String nummerplaat;
37
38     @OneToMany(mappedBy = "vervoermiddel", cascade = CascadeType.ALL)
39     private List<Onderhoudsbeurt> onderhoudsbeurten = new ArrayList<>();
40
41     protected Vervoermiddel() {}
42
43     public Vervoermiddel(String nummerplaat) {
44         this.nummerplaat = nummerplaat;
45     }
46
47     public String getNummerplaat() {
48         return nummerplaat;
49     }
50
51     public void setNummerplaat(String nummerplaat) {
52         this.nummerplaat = nummerplaat;
53     }
54
55     @Override
56     public String toString() {
57         return "Vervoermiddel{" + "nummerplaat=" + nummerplaat + '}';
58     }
59
60     @Override
61     public int hashCode() {
62         int hash = 3;
63         hash = 79 * hash + Objects.hashCode(this.nummerplaat);
64         return hash;
65     }
66
67     @Override
68     public boolean equals(Object obj) {
69         if (obj == null) {
70             return false;
```

```
71          }
72          if (getClass() != obj.getClass()) {
73              return false;
74          }
75          final Vervoermiddel other = (Vervoermiddel) obj;
76          if (!Objects.equals(this.nummerplaat, other.nummerplaat)) {
77              return false;
78          }
79          return true;
80      }
81
82      public List<Onderhoudsbeurt> getOnderhoudsbeurten() {
83          return Collections.unmodifiableList(onderhoudsbeurten);
84      }
85
86      public void setOnderhoudsbeurten(List<Onderhoudsbeurt> onderhoudsbeurten) {
87          this.onderhoudsbeurten = onderhoudsbeurten;
88      }
89
90      public long getId() {
91          return id;
92      }
93
94      public void addOnderhoudsbeurt(Onderhoudsbeurt o) {
95          onderhoudsbeurten.add(o);
96      }
97
98      public Onderhoudsbeurt geefOnderhoudsbeurtenOp(GregorianCalendar x) {
99          for (Onderhoudsbeurt o : onderhoudsbeurten) {
100             if (o.bevatDatum(x)) {
101                 return o;
102             }
103         }
104         return null;
105     }
106 }
```

Om te zorgen dat deze klasse persistent gemaakt kan worden, voegt men bovenaan de klasse de annotatie Entity toe. Hierna komen de namedQueries.

Op lijn 29 wordt de overerving verzorgt in de databank. Men opteert best voor joined in de meeste gevallen. Enkel in zeer specifieke omstandigheden zijn er andere, betere opties.

Op lijn 32 tot 34 wordt bepaald hoe het ID wordt gegeven. De annotatie Id geeft aan dat het attribuut Id in de klasse Vervoersmiddel het Id bevat voor het object. Lijn 33 laat de databank weten dat hij verantwoordelijk is voor de unieke nummering.

Lijn 38 beschrijft de relatie tussen Vervoersmiddel en Onderhoud, nl. een vervoersmiddel heeft meerdere onderhoudsbeurten (OneToMany). Ook kunnen we hier zien dat deze relatie door Voertuig gemapt zal worden.

Listing 16: Auto

```
1  package domein;
2
3  import javax.persistence.Entity;
4
5  @Entity
6  public class Auto extends Vervoermiddel {
7
8      protected Auto() {
9      }
10
11     public Auto(String nummerplaat) {
12         super(nummerplaat);
13     }
```

```
14
15      @Override
16      public double geefVerkeersbelasting() {
17          return 77.75;
18          //volgens cilinderinhoud
19      }
20  }
```

Listing 17: Lichte Vracht

```
1  package domein;
2
3  import javax.persistence.Entity;
4
5  @Entity
6  public class LichteVracht extends Vervoermiddel {
7
8      private double massa;
9
10     protected LichteVracht() {}
11
12     public LichteVracht(double massa, String nummerplaat) {
13         super(nummerplaat);
14         this.massa = massa;
15     }
16
17     public double getMassa() {
18         return massa;
19     }
20
21     public void setMassa(double massa) {
22         this.massa = massa;
23     }
24
25     @Override
26     public double geefVerkeersbelasting() {
27         throw new UnsupportedOperationException("Not supported yet.");
28         //volgens maximale massa
29     }
30 }
```

Listing 18: Onderhoudsbeurt

```
1  package domein;
2
3  import java.io.Serializable;
4  import java.util.GregorianCalendar;
5  import javax.persistence.Entity;
6  import javax.persistence.GeneratedValue;
7  import javax.persistence.GenerationType;
8  import javax.persistence.Id;
9  import javax.persistence.JoinTable;
10 import javax.persistence.ManyToOne;
11 import javax.persistence.NamedQueries;
12 import javax.persistence.NamedQuery;
13 import javax.persistence.Temporal;
14 import javax.persistence.TemporalType;
15
16 @Entity
17 @NamedQueries({
18     @NamedQuery(name = "OnderhoudsbeurtenOpDatum",
19             query = "SELECT o FROM Onderhoudsbeurt o "
20             + "WHERE NOT (:x < o.begindatum OR :x > o.einddatum)")
21 })
22 public class Onderhoudsbeurt implements Serializable {
23
```

```
24      @Id
25      @GeneratedValue(strategy = GenerationType.IDENTITY)
26      private long id;
27      @Temporal(TemporalType.DATE)
28      private GregorianCalendar begindatum, einddatum;
29      @ManyToOne
30      @JoinTable
31      private Vervoermiddel vervoermiddel;
32
33      protected Onderhoudsbeurt() {}
34
35      public Onderhoudsbeurt(GregorianCalendar begindatum,
36              GregorianCalendar einddatum, Vervoermiddel vervoermiddel) {
37          this.begindatum = begindatum;
38          this.einddatum = einddatum;
39          this.vervoermiddel = vervoermiddel;
40      }
41
42      public GregorianCalendar getBegindatum() {
43          return begindatum;
44      }
45
46      public void setBegindatum(GregorianCalendar begindatum) {
47          this.begindatum = begindatum;
48      }
49
50      public GregorianCalendar getEinddatum() {
51          return einddatum;
52      }
53
54      public void setEinddatum(GregorianCalendar einddatum) {
55          this.einddatum = einddatum;
56      }
57
58      public Vervoermiddel getVervoermiddel() {
59          return vervoermiddel;
60      }
61
62      public void setVervoermiddel(Vervoermiddel vervoermiddel) {
63          this.vervoermiddel = vervoermiddel;
64      }
65
66      boolean bevatDatum(GregorianCalendar x) {
67          return !(x.before(begindatum) || x.after(einddatum));
68      }
69  }
```

Ook bij de klasse Onderhoudsbeurt is er terug een NamedQuery. Hier dient op gemerkt te worden dat deze query een parameter zal meekrijgen vanuit de java code. In het sql statement gebruikt men dan :x om de parameter aan te spreken.

Ook zien we weer hoe de databank verantwoordelijk is voor het unieke id van de onderhoudsbeurten.

Op lijn 27 wordt aan de databank duidelijk gemaakt dat de attributen die erop volgen van het type Date zijn.

Verder wordt ook weer de relatie tussen Onderhoud en Vervoersmiddel duidelijk gemaakt aan de databank op lijn 29 en 30.

Listing 19: Garage Data

```
1  package domein;
2
3  import java.util.GregorianCalendar;
4
```

```
/**
 *
 * @author Lorenz
 */
public class GarageData {

    private final GarageBeheerder gb;

    GarageData(GarageBeheerder garageBeheerder) {
        gb = garageBeheerder;
    }

    void populeerData() {
        gb.addVervoermiddel(new Auto("auto100"));
        gb.addVervoermiddel(new Auto("auto200"));
        gb.addVervoermiddel(new Auto("auto300"));
        gb.addVervoermiddel(new LichteVracht(1500, "vracht100"));
        gb.addVervoermiddel(new LichteVracht(1600, "vracht200"));
        gb.addVervoermiddel(new LichteVracht(1700, "vracht300"));

        gb.addOnderhoudsbeurt("auto100", new GregorianCalendar(2014, 0, 20),
                new GregorianCalendar(2014, 0, 22));
        gb.addOnderhoudsbeurt("auto100", new GregorianCalendar(2014, 4, 20),
                new GregorianCalendar(2014, 4, 22));
        gb.addOnderhoudsbeurt("auto100", new GregorianCalendar(2014, 8, 20),
                new GregorianCalendar(2014, 8, 22));

        gb.addOnderhoudsbeurt("auto200", new GregorianCalendar(2014, 1, 1),
                new GregorianCalendar(2014, 1, 1));
        gb.addOnderhoudsbeurt("auto200", new GregorianCalendar(2014, 5, 10),
                new GregorianCalendar(2014, 5, 14));
        gb.addOnderhoudsbeurt("auto200", new GregorianCalendar(2014, 10, 25),
                new GregorianCalendar(2014, 10, 25));

        gb.addOnderhoudsbeurt("vracht100", new GregorianCalendar(2014, 0, 20),
                new GregorianCalendar(2014, 0, 22));
        gb.addOnderhoudsbeurt("vracht100", new GregorianCalendar(2014, 4, 20),
                new GregorianCalendar(2014, 4, 22));
        gb.addOnderhoudsbeurt("vracht100", new GregorianCalendar(2014, 8, 20),
                new GregorianCalendar(2014, 8, 22));

        gb.addOnderhoudsbeurt("vracht200", new GregorianCalendar(2014, 1, 1),
                new GregorianCalendar(2014, 1, 1));
        gb.addOnderhoudsbeurt("vracht200", new GregorianCalendar(2014, 5, 10),
                new GregorianCalendar(2014, 5, 14));
        gb.addOnderhoudsbeurt("vracht200", new GregorianCalendar(2014, 10, 25),
                new GregorianCalendar(2014, 10, 25));

    }
}
```

Zoals duidelijk werd uit de code van GarageData, zorgt deze klasse ervoor dat de databank gevuld wordt.

Listing 20: GarageBeheerder

```
package domein;

import java.util.GregorianCalendar;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

```

```java
/**
 *
 * @author Lorenz
 */
public class GarageBeheerder {

    public final String PERSISTENCE_UNIT_NAME = "Garage_PU";
    private EntityManager em;
    private EntityManagerFactory emf;
    private Map<String, Vervoermiddel> vervoerMap = new HashMap<>();

    public GarageBeheerder() {
        initializePersistentie();
    }

    public void initializePersistentie() {
        openPersistentie();
        GarageData od = new GarageData(this);
        od.populeerData();
    }

    public void openPersistentie() {
        emf = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);
        em = emf.createEntityManager();
    }

    public void closePersistentie() {
        em.close();
        emf.close();
    }

    public void addVervoermiddel(Vervoermiddel v) {
        vervoerMap.put(v.getNummerplaat(), v);

        em.getTransaction().begin();
        em.persist(v);
        em.getTransaction().commit();
    }

    public void addOnderhoudsbeurt(String nrplaat, GregorianCalendar begin,
        GregorianCalendar einde) {
        Vervoermiddel v = vervoerMap.get(nrplaat);
        Onderhoudsbeurt o = new Onderhoudsbeurt(begin, einde, v);

        em.getTransaction().begin();
        v.addOnderhoudsbeurt(o);
        em.persist(o);
        em.getTransaction().commit();
    }

    public void startTransaction() {
        GarageData od = new GarageData(this);
        od.populeerData();
    }

    public void endTransaction() {
        //TODO
    }

    List<Vervoermiddel> geefAutosZonderOnderhoudsbeurt() {
        return vervoerMap.values().stream()
                .filter(v -> v.getOnderhoudsbeurten().isEmpty())
                .collect(Collectors.toList());
    }

    List<Vervoermiddel> geefAutosZonderOnderhoudsbeurtJPA() {
```

```
77          return em.createNamedQuery("AlleVervoerZonderOnderhoud").getResultList
                ();
78      }
79
80      List<Vervoermiddel> geefAutosMetOnderhoudsbeurt() {
81          return vervoerMap.values().stream()
82                  .filter(v -> v.getOnderhoudsbeurten().size() > 0)
83                  .collect(Collectors.toList());
84      }
85
86      List<Vervoermiddel> geefAutosMetOnderhoudsbeurtJPA() {
87          return em.createNamedQuery("AlleVervoerMetOnderhoud").getResultList();
88      }
89
90      List<Onderhoudsbeurt> geefOnderhoudsbeurtenOpDatum(GregorianCalendar x) {
91          return vervoerMap.values().stream()
92                  .map(v -> v.geefOnderhoudsbeurtenOp(x))
93                  .filter(o -> o != null)
94                  .collect(Collectors.toList());
95      }
96
97      List<Onderhoudsbeurt> geefOnderhoudsbeurtenOpDatumJPA(GregorianCalendar x)
            {
98          return em.createNamedQuery("OnderhoudsbeurtenOpDatum")
99                  .setParameter("x", x).getResultList();
100     }
101 }
```

De GarageBeheerder klasse is verantwoordelijk om alle data te beheren. Hier worden de Persistence Unit, de EntityManager en de EntityManagerFactory bijgehouden.

Verder zien we ook een terugkerend patroon als er iets toegevoegd moet worden aan de databank:

```
1 verander(x)
2
3 entityManager.getTransaction().begin();
4 entityManager.persist(x)
5 entityManager.getTransaction().commit();
```

Op lijn 70 start de methode om vervoersmiddelen zonder onderhoudsbeurt weer te geven zonder gebruik te maken van de databank. Hierbij maakt men gebruik van streams.

Op lijn 76 start de methode om vervoersmiddelen zonder onderhoudsbeurt weer te geven met behulp van de databank. merk hierbij op dat de EntityManager een NamedQuery maakt met de naam van een NamedQuery die in de klasse vervoersmiddel op lijn 22 gedefiniëerd werd.

Merk bij lijn 98-99 op hoe de parameters worden doorgegeven aan de NamedQuery.

Listing 21: Garage Controller

```
1  package domein;
2
3  import java.util.GregorianCalendar;
4  import java.util.List;
5  import java.util.stream.Collectors;
6
7  /**
8   *
9   * @author Lorenz
10  */
11 public class GarageController {
12
13     private GarageBeheerder gb = new GarageBeheerder();
14
15     public List<String> geefAutosZonderOnderhoudsbeurt() {
```

```
16          //List<Vervoermiddel> li = gb.geefAutosZonderOnderhoudsbeurt();
17          List<Vervoermiddel> li = gb.geefAutosZonderOnderhoudsbeurtJPA();
18          return li.stream().map(Vervoermiddel::getNummerplaat).collect(
                Collectors.toList());
19      }
20
21      public List<String> geefAutosMetOnderhoudsbeurt() {
22          //List<Vervoermiddel> li = gb.geefAutosMetOnderhoudsbeurt();
23          List<Vervoermiddel> li = gb.geefAutosMetOnderhoudsbeurtJPA();
24          return li.stream().map(Vervoermiddel::getNummerplaat).collect(
                Collectors.toList());
25      }
26
27      public List<String> geefOnderhoudsbeurtenOpDatum(int jaar, int maand, int
            dag) {
28          GregorianCalendar x;
29          x = new GregorianCalendar(jaar, maand, dag);
30          //List<Onderhoudsbeurt> li = gb.geefOnderhoudsbeurtenOpDatum(x);
31          List<Onderhoudsbeurt> li = gb.geefOnderhoudsbeurtenOpDatumJPA(x);
32          return li.stream()
33                  .map(Onderhoudsbeurt::getVervoermiddel)
34                  .map(Vervoermiddel::getNummerplaat)
35                  .collect(Collectors.toList());
36      }
37
38      public void startTransaction() {
39          gb.startTransaction();
40      }
41
42      public void endTransaction() {
43          gb.endTransaction();
44      }
45
46  }
```

De klasse GarageController is verantwoordelijk voor het sturen van de garage applicatie.

Merk op dat bij de methode geefOnderhoudsbeurtenOpDatum, lijn 27-36, de controller verantwoordelijk is voor de correcte weergave. De controller verzorgt de weergave via een stream.

Bij dit voorbeeld hoort ook een Persistance Unit. Hierin worden alle klasses vermeld die moeten opgenomen worden in de databank. Deze PU is essentieel om communicatie mogelijk te maken tussen java en de databank.

8 Netwerkprogrammatie

8.1 Een eenvoudige server met Stream Sockets opzetten

1. Creëer een serverSocket object
   (a) ServerSocket server = new ServerSocket(portnumber, queueLength);
   (b) De contructor legt het poortnummer vast waarop de server op connecties wacht. (=binding server/poort)
   (c) Sockets verbergen de complexiteit van het netwerkprogrammeren.
2. De server luistert onafgebroken naar een poging van een client om een connectie te maken (blocks)
   (a) Het programma roept de methode accept aan om te luisteren naar een connectie van een client
      i. Socket connection = server.accept();
      ii. Deze methode levert een Socket af wanneer een connectie met een client tot stand gekomen is
   (b) Door de Socket kan de server interageren met de client
3. De OutputStream- en InputStream-objecten worden opgehaald zodat de server kan communiceren met de client door het verzenden en ontvangen van bytes.
   (a) De server roept de methode getOutputStream aan op de Socket en krijgt een referentie naar de Socket's OutputStream. Dan wordt de methode getInputStream aangeroepen op de Socket om een referentie te krijgen naar de the Socket's InputStream.
4. Tijdens de verwerkingsfase communiceren de server en de client via de OutputStream- en InputStream-objecten
5. Wanneer de transmissie afgehandeld is, sluit de server de connectie door de methode close aan te roepen op de streams en op de Socket

8.2 Een eenvoudige Client met Stream Sockets opzetten

1. de Socket constructor legt een connectie met de server
   (a) Socket connection = new Socket (serverAddress, port);
   (b) Als de connectie tot stand gebracht is, dan wordt een Socket afgeleverd
   (c) Als de connectie niet tot stand kan gebracht worden, dan wordt een IOException geworpen
   (d) Een onjuiste servernaam heeft een UnknownHostException tot gevolg
2. De client gebruikt de methoden getInputStream en getOutputStream om referenties naar InputStream and OutputStream te verkrijgen.
3. Tijdens de verwerkingsfase communiceren de server en de client via de OutputStream en InputStream objecten.
4. Wanneer de transmissie afgehandeld is, sluit de client de connectie door de methode close aan te roepen op de streams en op de Socket.

8.3 Voorbeeld met Stream Sockets

8.3.1 Server

Listing 22: Server Startup

```
1 import domein.Server;
2
3 public class ServerStartUp {
4
5     public static void main(String[] args) {
6         Server server = new Server();
7         server.run();
8     }
9 }
```

Listing 23: Server

```
1 package domein;
2
3 import java.io.IOException;
4 import java.io.ObjectInputStream;
5 import java.io.ObjectOutputStream;
6 import java.net.ServerSocket;
7 import java.net.Socket;
8 import java.util.logging.Level;
9 import java.util.logging.Logger;
10
11 public class Server {
12
13     private ObjectInputStream input;
14     private ObjectOutputStream output;
15     private ServerSocket server;
16     private Socket connection;
17     private int numberOfConnections;
18
19     public Server() {
20         numberOfConnections = 1;
21     }
22
23     public void run() {
24         try {
25             server = new ServerSocket(12345, 100);
26
27             while (true) {
28                 try {
29                     waitForConnection();
30                     getStreams();
31                     processConnection();
32                 } catch (Exception e) {
33                     System.err.println(e.getMessage());
34                 } finally {
35                     closeConnection();
36                     numberOfConnections++;
37                 }
38             }
39         } catch (Exception e) {
40             System.err.println(e.getMessage());
41         }
42     }
43
44     private void waitForConnection() throws IOException {
45         System.out.println("Waiting for connection...");
46         connection = server.accept();
47         System.out.println("Connection #" + numberOfConnections + " received
               from: " + connection.getInetAddress().getHostName());
```

```
48      }
49
50      private void getStreams() throws IOException {
51          output = new ObjectOutputStream(connection.getOutputStream());
52          output.flush();
53          input = new ObjectInputStream(connection.getInputStream());
54          System.out.println("Got connections");
55      }
56
57      private void processConnection() throws IOException {
58          String message = "Connection succes";
59          sendData(message);
60          do {
61              try {
62                  message = (String) input.readObject();
63                  System.out.println("Client says: " + message);
64              } catch (ClassNotFoundException ex) {
65                  System.err.println("Unknown Object Received");
66              }
67          } while (!message.equals("STOP"));
68          sendData("STOP");
69      }
70
71      private void closeConnection() {
72          System.out.println("Terminating connection");
73          try {
74              input.close();
75              output.close();
76              connection.close();
77          } catch (Exception e) {
78              System.err.println("Terminating failed");
79          }
80      }
81
82      private void sendData(String message) {
83          try {
84              output.writeObject(message);
85              output.flush();
86              System.out.println("ServerApp says: " + message);
87          } catch (IOException ex) {
88              Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex
                    );
89          }
90      }
91  }
```

Merk hierbij op dat in de methode run mooi de stappen gevolgd worden uit 8.1.

8.3.2 Client

Listing 24: Client Startup

```
1  import domein.Client;
2
3  public class ClientStartUp {
4
5      public static void main(String[] args) {
6          Client client = new Client();
7          client.run();
8      }
9
10 }
```

```
1  package domein;
2
```

```
3  import java.io.IOException;
4  import java.io.ObjectInputStream;
5  import java.io.ObjectOutputStream;
6  import java.net.Socket;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9
10 /**
11  *
12  * @author Lorenz
13  */
14 public class Client {
15
16     private ObjectInputStream input;
17     private ObjectOutputStream output;
18     private String message;
19     private String server;
20     private Socket client;
21
22     public Client() {
23         server = "192.168.0.196";
24     }
25
26     public void run() {
27         try {
28             connectToServer();
29             getStreams();
30             processConnection();
31         } catch (IOException ex) {
32             Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex
                   );
33         } finally {
34             closeConnection();
35         }
36     }
37
38     private void connectToServer() throws IOException {
39         System.out.println("Trying to connect...");
40         client = new Socket(server, 12345);
41         System.out.println("Connected to " + client.getInetAddress().
               getHostName());
42     }
43
44     private void getStreams() throws IOException {
45         output = new ObjectOutputStream(client.getOutputStream());
46         output.flush();
47         input = new ObjectInputStream(client.getInputStream());
48         System.out.println("Got streams");
49     }
50
51     private void processConnection() throws IOException {
52         do {
53             try {
54                 message = (String) input.readObject();
55             } catch (ClassNotFoundException ex) {
56                 System.err.println("Unknown Object Found");
57             }
58             System.out.println("Server says: " + message);
59             if (!message.equals("STOP")) {
60                 sendingData();
61             }
62         } while (!message.equals("STOP"));
63     }
64
65     private void closeConnection() {
66         System.out.println("Closing connection");
67         try {
```

```
68              output.close();
69              input.close();
70              client.close();
71          } catch (IOException ex) {
72              Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex
                    );
73          }
74      }
75
76      public void sendingData() {
77          try {
78              sendData("Hallo server");
79              Thread.sleep(1000);
80              sendData("Dit is client");
81              Thread.sleep(1000);
82              sendData("STOP");
83          } catch (InterruptedException ex) {
84              Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex
                    );
85          }
86      }
87
88      private void sendData(String message) {
89          try {
90              output.writeObject(message);
91              output.flush();
92              System.out.println("ClientApp says: " + message);
93          } catch (IOException ex) {
94              Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex
                    );
95          }
96      }
97  }
```

Merk ook hier weer op dat in de methode rund de stappen van 8.2 gevolgd worden.

8.4 Voorbeeld DatagramSocket

8.4.1 Server

Listing 25: Ping Server

```
1  package main;
2
3  import java.net.*;
4  import java.util.*;
5
6   /*
7    * Server to process ping requests over UDP.
8    */
9   public class PingServer
10  {
11     private static final double LOSS_RATE = 0.3;
12     private static final int AVERAGE_DELAY = 100;  // milliseconds
13
14     public static void main(String[] args) throws Exception
15     {  int port = 5555; // default value
16        // Get command line argument.
17        if (args.length == 1) {
18           port    = Integer.parseInt(args[0]);
19        }
20        System.out.println("Ping reply server started: uses port " + port);
21        // Create random number generator for use in simulating
22        // packet loss and network delay.
23        Random random = new Random();
24
```

```
25          // Create a datagram socket for receiving and sending UDP packets
26          // through the port specified on the command line.
27          DatagramSocket socket = new DatagramSocket(port);
28          // Create a datagram packet to hold incomming UDP packet.
29          DatagramPacket request = new DatagramPacket(new byte[1024], 1024);
30          // Processing loop.
31          while (true) {
32             // Block until the host receives a UDP packet.
33             socket.receive(request);
34
35             // Print the recieved data.
36             printData(request);
37
38             // Decide whether to reply, or simulate packet loss.
39             if (random.nextDouble() < LOSS_RATE) {
40                System.out.println("   Reply not sent.");
41                continue;
42             }
43
44             // Simulate network delay.
45             Thread.sleep((int) (random.nextDouble() * 2 * AVERAGE_DELAY));
46
47             // Send reply.
48             InetAddress clientHost = request.getAddress();
49             int clientPort = request.getPort();
50             byte[] buf = request.getData();
51             DatagramPacket reply = new DatagramPacket(buf, buf.length, clientHost
                   , clientPort);
52             socket.send(reply);
53
54             System.out.println("   Reply sent.");
55          }
56       } //einde main
57
58       /*
59        * Print ping data to the standard output stream.
60        */
61       private static void printData(DatagramPacket request) throws Exception
62       {
63          // Obtain references to the packet's array of bytes.
64          byte[] buf = request.getData();
65
66          // Wrap the bytes in a byte array input stream,
67          // so that you can read the data as a stream of bytes.
68    //       ByteArrayInputStream bais = new ByteArrayInputStream(buf);
69
70          // Wrap the byte array output stream in an input stream reader,
71          // so you can read the data as a stream of characters.
72    //       InputStreamReader isr = new InputStreamReader(bais);
73
74          // Wrap the input stream reader in a bufferred reader,
75          // so you can read the character data a line at a time.
76          // (A line is a sequence of chars terminated by any combination of \r
                and \n.)
77    //       BufferedReader br = new BufferedReader(isr);
78
79          // The message data is contained in a single line, so read this line.
80    //       String line = br.readLine();  De hele streamverpakking hoeft hier niet
             .
81          String line = new String(buf, request.getOffset(), request.getLength());
82          // Print host address and data received from it.
83          System.out.println(
84             "Received from " +
85             request.getAddress().getHostAddress() + ": " +  new String(line) );
86       } //einde printData
87    } // einde PingServer
```

8.4.2 Client

Listing 26: Ping Client

```
1  package main;
2
3  import java.io.IOException;
4  import java.net.DatagramPacket;
5  import java.net.DatagramSocket;
6  import java.net.InetAddress;
7  import java.net.SocketTimeoutException;
8  import java.net.UnknownHostException;
9  import java.text.SimpleDateFormat;
10 import java.util.Date;
11 import java.util.Timer;
12
13 public class PingClient {
14
15     private InetAddress host;
16     private String hostName = "localhost"; //default
17     private int portnr = 5555;  //default
18     private final int PINGAANTAL = 10;
19     private final int TOKEN_TIMESTAMP = 2; //positie in packet
20     private final int MAX_WAIT_TIME = 1000;
21     private long min = 999999, max = 0, gem = 0;
22     private int verloren = 0, aangekomen = 0;
23     private Timer timer;
24
25     public static void main(String[] args) {
26         new PingClient().run(args);
27     }
28
29     public void run(String args[]) {
30         try {
31             if (args.length > 0) {
32                 hostName = args[0];
33             }
34
35             if (args.length == 2) {
36                 portnr = Integer.parseInt(args[1]);
37             }
38
39             host = InetAddress.getByName(hostName);
40             DatagramSocket datagramSocket = new DatagramSocket();
41             datagramSocket.setSoTimeout(MAX_WAIT_TIME);
42
43             for(int i =0;i<PINGAANTAL;i++){
44                 SimpleDateFormat TimeNow = new SimpleDateFormat("MM/dd/yyyy HH:
                       mm:ss");
45                 String TimedStr = TimeNow.format(new Date(System.
                       currentTimeMillis()));
46                 String message = "Ping #" + i + ": " + System.currentTimeMillis
                       ()+ " ("+TimedStr+")";
47
48                 DatagramPacket ping_verzoek =
49                         new DatagramPacket(message.getBytes(), message.length()
                               ,host,portnr);
50                 datagramSocket.send(ping_verzoek);
51                 DatagramPacket ping_antwoord =
52                         new DatagramPacket(new byte[message.length()], message.
                               length());
53                 try{
54                     datagramSocket.receive(ping_antwoord);
55                     aangekomen++;
56                     printData(ping_antwoord);
57                 }catch(SocketTimeoutException e){
58                     System.out.println("Ping #" + i + ": No response was
                           received from the server");
```

```
59                      verloren++;
60                  }
61              }
62
63          } catch (UnknownHostException ex) {
64              System.out.println("Onbekende host: " + ex.getMessage());
65          } catch (IOException ex) {
66              System.out.println("Probleem: ");
67              ex.printStackTrace();
68          }
69      }
70
71      private void printData(DatagramPacket request) {
72          String response = new String(request.getData());
73          String[] tokens = response.split(" ");
74          long verzonden_timestamp = new Long(tokens[TOKEN_TIMESTAMP]);
75          long ontvangen_timestamp = System.currentTimeMillis();
76
77          long rtt = ontvangen_timestamp-verzonden_timestamp;
78
79          System.out.println(response + " Received from " + request.getAddress().
                getHostAddress()+ " (RTT= "+rtt+"ms)");
80          setRTTs(rtt);
81      }
82
83      private void setRTTs(long rtt) {
84          if(min>rtt)
85              min=rtt;
86          if(max<rtt)
87              max=rtt;
88          gem+=rtt;
89      }
90  }
```

8.5 Multithreaded server

8.6 Multicast