Uitwerking 3e deeltentamen Imperatief programmeren

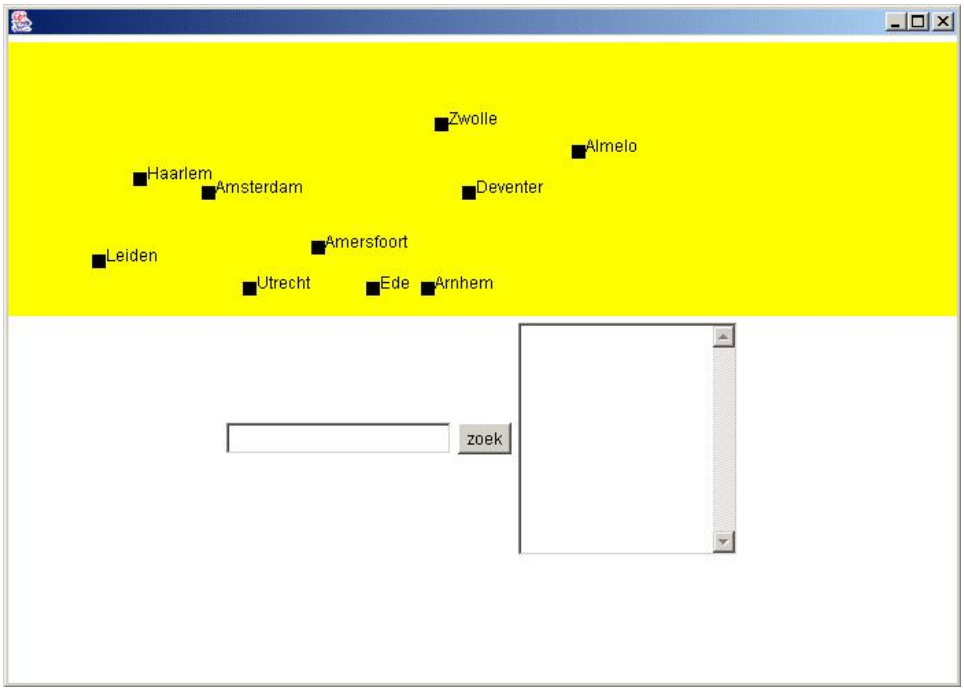
5 november 2004

Opgave 1

Zie je eigen practicumuitwerking J

Opgave 2

a. De variabelen k en a kunnen niet lokaal gedeclareerd worden in de constructor, want die zijn ook nog nodig in methode actionPerformed. De variabelen t en b kunnen wel lokaal gedeclareerd worden, want die zijn niet expliciet nodig in andere methoden.

b. 

c. Maak een subklasse van WindowAdapter, en definieer daarin alleen de niet-lege methode, dat is windowClosing. Maak een new object van die klasse, en gebruik dat als parameter van addWindowListener in plaats van this.

d. De cast is nodig, omdat getSource als resultaat een Object heeft, terwijl we hier de specifieke TextComponent-methode getText willen aanroepen. Dat kan hier veilig, want er is in dit programma maar één component dat een Kaart-object als ActionListener heeft, en dat is inderdaad een TextField (de Button heeft een Test-object als ActionListener).

e. De opdrachten achter try worden achtereenvolgens uitgevoerd. Als daarbij een Exception optreedt, wordt dit onderbroken en gaat het verder achter catch. Treedt er geen Exception op, dan worden de opdrachten achter catch overgeslagen.
In plaats hiervan kan in de header van de methode ‘throws Exception’ worden geschreven; de aanroeper van de methode moet de exception dan opvangen.

Opgave 3

a.
```
steden = new LinkedList();
```
Of ArrayList, of HashSet, of TreeSet. Maar niet: new Collection() of new List(), want van een interface kun je geen nieuwe oibjecten maken: het moet een concrete implementatie ervan zijn. HashMap is ook niet goed, want dat is geen implementatie van Collection (al moet ik toegeven dat met een HashMap opgave c veel fraaier kan worden opgelost!)

b.
```
public void mousePressed(MouseEvent e)
{    steden.add( new Stad( naam, e.getPoint() ));
     repaint();
}
```
Of in plaats van e.getPoint(): new Point(e.getX(), e.getY())

c.
```
public Point waar(String t)
{    for (Iterator i=steden.iterator(); i.hasNext(); )
     {    Stad s = (Stad) i.next();
          if (s.naam.equals(t))
               return s.locatie;
     }
     return null;
}
```

d. In schrijf:
```
PrintWriter pw = new PrintWriter(
                    new FileWriter(naam));
for (Iterator it=steden.iterator(); it.hasNext(); )
{    Stad s = (Stad)it.next();
     pw.println(s.locatie.x + " " + s.locatie.y
                              + " " + s.naam      );
}
```

In lees:
```
BufferedReader br = new BufferedReader(
                       new FileReader(naam));
String s;
while ( (s=br.readLine()) != null )
{    StringTokenizer st = new StringTokenizer(s, " ");
     int x = Integer.parseInt(st.nextToken());
     int y = Integer.parseInt(st.nextToken());
     steden.add( new Stad( st.nextToken()
                          , new Point(x,y) ));
}
```

e.

```
class Routekaart extends Kaart
{
     List route = new LinkedList();

     public void paint(Graphics g)
     {    super.paint(g);
          Iterator i;
          Point p, q;
          i = route.iterator();
          if (i.hasNext())
          {    p = (Point) i.next();
               while (i.hasNext())
               {    q = (Point) i.next();
                    g.drawLine(p.x, p.y, q.x, q.y);
                    p = q;
               }
          }
     }

     public void zoek(String namen)
     {    StringTokenizer st;
          Point p;
          route.clear();
          st = new StringTokenizer(namen, " \n");
          while (st.hasMoreTokens())
          {
               p = this.waar(st.nextToken());
               if (p!=null)
                    route.add(p);
          }
          this.repaint();
     }
}
```

f. Vlak voordat het programma afsluit, schrijven we de kaart naar een file, door in methode windowClosing toe te voegen:

```
k.schrijf("steden.txt");
```

In de constructormethode van Test lezen we de file met dezelfde naam weer in:

```
k.lees("steden.txt");
```