Tentamen Object Georiënteerd Programmeren TI1200
30 januari 2013, 9.00-12.00
Afdeling SCT, Faculteit EWI, TU Delft

Bij dit tentamen mag je geen gebruik maken van hulpmiddelen zoals boek of slides.
Dit tentamen bestaat uit 5 opgaven, en heeft twee bijlagen. Aantal punten: 80.

Opgave 1 (5 * 3 punten)
Bijlage A bevat de definities van de klassen Telefoonboek en Regel, die een eenvoudig telefoonboek en een regel (met abonneegegevens) uit het telefoonboek voorstellen.

In de klasse Telefoonboek wordt de methode test() aangeroepen. Wat is na uitvoering de waarde van elk van de volgende expressies? Licht je antwoord **uitgebreid** toe!

Mocht een van de expressies aanleiding geven tot een compileer- of run timè-fout, dan dien je dat aan te geven samen met de reden van de fout.

```
a.    regel[0] == regel[2]
b.    regel[1] == regel[2]
c.    regel[1].getNaam()
d.    regel[2].getNaam()    == regel[0].getNaam()
e.    regel[2].getNummer()  == regel[1].getNummer()
```

Opgave 2 (6 + 7 + 7 punten)
Aan de klasse Telefoonboek moet een drietal methoden worden toegevoegd.
NB In for- while- en do-loops zijn break- en return-opdrachten niet toegestaan.

a. `indicesVan(nm : String) : int[]`
post: retourneert een array met **uitsluitend** de indices van de regels waarvan de naam gelijk is aan nm.

b. `voegtoe(rg : Regel)`
post: Als het telefoonboek regels bevat met rg.naam en rg.adres, is het nummer van deze regels vervangen door rg.nummer,
zo niet, dan is regel rg als laatste element aan het telefoonboek toegevoegd.

c. `dubbeleNummers (naam: String) : String[]`
post: retourneert een array dat **uitsluitend** de nummers bevat die meer dan 1 keer voorkomen in het telefoonboek

Opgave 3(4 * 3 punten)
a. Wat is het verschil tussen een **klasse** en een **interface?** Waarom is een interface handig in de Java context?
b. Welke 2 Java klassen spelen een belangrijke rol bij de implementatie van het Model-View Controller (MVC) design pattern?
c. Wat is een **Runnable**? Waarvoor kan je deze gebruiken?
d. Wat is de betekenis van het keyword **synchronized**? Leg uit wat er precies gebeurt als je dit gebruikt.

Opgave 4 (3*6 punten)
Kijk goed naar de definities uit bijlage B.

a. De code in bijlage bevat **3 fouten**, geef aan waar de fout zit en waarom de Java compiler het er niet mee eens is. Let wel, het gaat om compileerfouten, niet om semantische fouten!

Eenmaal de code in bijlage B verbeterd is, kijk dan naar de 3 programmafragmenten en bepaal:

- Of het tijdens het compileren fouten oplevert (en zo ja, welke). Hoe kan je dit oplossen (schrijf het volledig verbeterde statement op)
- Zo neen, of er tijdens het verwerken fouten oplevert (zo ja, welke). Hoe kan je dit oplossen? (schrijf het volledig verbeterde statement op!)
- Wat is na afloop de waarde dit wordt afgedrukt op het scherm? (**indien het programma goed werkt, maar ook na het oplossen van een compilerprobleem – bij een runtime-probleem moet dit niet!**)

b.
```
Voertuig auto = new HybrideAuto("Toyota Prius", 5);
HybrideAuto prius = auto;
System.out.println(prius.getMerk());
```

c.
```
Voertuig auto = new RangerExtenderAuto("Opel Ampera", 4);
ElektrischeAuto leaf = (RangeExtenderAuto)auto;
System.out.println(leaf.getMerk());
```

d.
```
Auto auto = new ElektrischVoertuig("Renault Fluence", 4);
System.out.println(auto.CO2_uitstoot());
```

Opgave 5 (15 punten)
Deze opgave speelt zich af in de klasse Getalrij:

```
public class Getalrij{
    int [] rij;
    int aantal;
    int maximaleWaarde;
    ...
}
```

De rij in een Getalrij bevat een reeks getallen die allemaal in het interval [0, maximaleWaarde] liggen. Je mag aannemen dat de array rij gecreëerd is, en één of meer getallen bevat (namelijk het aantal gegeven door het attribuut aantal). De getallen in deze rij staan door elkaar. Een getal kan meer dan één keer voorkomen, ook kan het zo zijn dat er getallen niet inzitten.

Implementeer met behulp van het zeven-stappen-plan in deze klasse de methode : `public int[] getFrequenties()`, die een array teruggeeft met de frequentie van voorkomen van de elementen.

Voorbeeld: stel dat de array volgende elementen bevat 0, 1, 1, 9, 2, 2, 3, 4, 5, 6, 7, 9 dan moet de methode die je moet implementeren de array [1, 2, 2, 1, 1, 1, 1, 1, 0, 2] teruggeven (m.a.w. 1 voorkomen van 0, 2 voorkomens van 1, 2 voorkomens van 2, enz.). Let bovendien op, het attribuut **maximaleWaarde** geeft aan wat de maximale waarde is die in de array kan voorkomen. NB. Punten voor deze opgave worden verdiend door juiste toepassing van het zeven-stappen-plan. Alleen een implementatie levert weinig op - zelfs als hij correct is.

Hieronder volgt de benoeming van de 7 stappen van het 7-stappen plan:

- Stap 1: beschrijf de situatie tijdens het algoritme
- Stap 2: beschrijf de hulpvariabelen
- Stap 3: invariant
- Stap 4: stopvoorwaarde
- Stap 5: initialisatie
- Stap 6: algoritme
- Stap 7: algoritme is correct en eindigt

Bijlage A

```
public class Regel {

      private String naam;          //naam van de abonnee
      private String adres;         //adres van de abonnee
      private String nummer;        //telefoonnummer van de abonnee

      public Regel(String naam, String adres, String nummer){
            setNaam(naam);
            setAdres(adres);
            setNummer(nummer);
      }

      public String getNaam(){            return naam;       }
      public String getAdres(){           return adres;      }
      public String getNummer(){          return nummer;     }
      public void setNaam(String nm){     this.naam   = nm;  }
      public void setAdres(String ad){    this.naam   = ad;  }
      public void setNummer(String nr){   this.nummer = nr;  }
}

public class Telefoonboek{
  private Regel[] regel;
  private int aantal;

  public Telefoonboek(){
    regel  = new Regel[1000];
    aantal = 0;
  }

  public void test(){
    Regel rg;
    rg = new Regel("Zaidman","Mekelweg 4","314159265");
    regel[aantal] = rg;
    aantal++;
    rg = new Regel("Geers","Julianalaan 132","271828182");
    regel[aantal] = rg;
    aantal++;
    rg.setNaam(regel[0].getNaam());
    rg.setNummer(regel[0].getNummer());
    regel[aantal] = rg;
    aantal++;
  }
}
```

Bijlage B

```
public abstract class Voertuig
{

	private String fMerk;
	private int fAantalDeuren;

	public Voertuig(String merk, int aantalDeuren)
	{
		fMerk = merk;
		fAantalDeuren = aantalDeuren;
	}

	public String getMerk()
	{
		return fMerk;
	}

	public int getAantalDeuren()
	{
		return fAantalDeuren;
	}
}

public interface ActieRadius
{
	public double getActieRadiusOnBattery();
}

public class Auto extends Voertuig
{
	public Auto(String merk, int aantalDeuren)
	{
		super(merk, aantalDeuren);
	}
}

public class HybrideAuto extends Voertuig implements ActieRadius{
	public HybrideAuto(String merk, int aantalDeuren)
	{
		super(merk, aantalDeuren);
	}
}

public class RangeExtender extends Voertuig
{
	public RangeExtender(String merk, int aantalDeuren)
	{
		this = new Voertuig(merk, aantalDeuren);
	}
}
```

```
public class ElektrischVoertuig extends RangeExtender, Thread
implements ActieRadius
{
	private int fCapaciteitBatterij;

	public ElektrischVoertuig(String merk, int aantalDeuren,
			int capaciteitBatterij)
	{
		super(merk, aantalDeuren);
		fCapaciteitBatterij = capaciteitBatterij;
	}

	public double getActieRadiusOnBattery()
	{
		return fCapaciteitBatterij / 3.5;
	}

	public int CO2_uitstoot()
	{
		return 0;
	}
}
```