TENTAMEN

PROGRAMMEREN 2

vakcode: 213505
datum: 18 juni 2008
tijd: 9.00–12.30 uur

Algemeen

- Bij dit tentamen mag alleen gebruik worden gemaakt van de boeken van Niño/Hosch en van der Linden en één ander Java-leerboek naar keuze, van de practicumhandleiding van Programmeren 2 (in het bijzonder bijlage B), en van uitgeprinte kopieën van de hoorcollegesheets.
- Het aantal punten voor het tentamen wordt meegenomen in de berekening van het eindcijfer, op de manier zoals aangegeven in de handleiding.
- Dit tentamen bestaat uit 5 opgaven, waarvoor in het totaal 100 punten behaald kunnen worden. Het minimaal aantal punten per opgave bedraagt 0 punten.

Opgave 1 (20 punten)

Bij deze opgave zijn we geïnteresseerd in het totaal aantal manieren waarop een bedrag (in centen) gewisseld kan worden in munten.

Voorbeeld. Beschouw een bedrag van 6 cent. Als we de beschikking hebben over munten met de waarden 1, 2 en 5 cent, dan kan het bedrag van 6 cent op 5 verschillende manieren gewisseld worden. Namelijk: $\{5,1\}$, $\{2,2,2\}$, $\{2,2,1,1\}$, $\{2,1,1,1,1\}$ en $\{1,1,1,1,1,1\}$.

Gevraagd wordt nu om een methode `wisselen` te schrijven, die het totaal aantal manieren oplevert waarop een bedrag gewisseld kan worden. U dient hierbij een *recursieve* oplossingsmethode te gebruiken. Het is hierbij toegestaan om een hulpmethode te gebruiken. De heading van de methode `wisselen` dient er als volgt uit te zien:

```
/**
 * Deze methode bepaalt het totaal aantal manieren waarop 'bedrag'
 * (in centen) kan worden gewisseld in verschillende 'munten'.
 * De array 'munten' geeft de mogelijke munteenheden (ook in centen).
 * @require munten != null
 */
public static int wisselen(int bedrag, int[] munten)
```

Hieronder staat een voorbeeld van het gebruik van de methode `wisselen`.

```
public static void main(String[] args) {
    int[] euromunten = { 1, 2, 5, 10, 20, 100, 200 } ;
    System.out.println("6:␣␣␣" + wisselen(6,   euromunten));
    System.out.println("8:␣␣␣" + wisselen(8,   euromunten));
    System.out.println("10:␣␣" + wisselen(10,  euromunten));
    System.out.println("50:␣␣" + wisselen(50,  euromunten));
    System.out.println("100:␣" + wisselen(100, euromunten));
}
```

Het programma zal de volgende uitvoer genereren:

```
6:   5
8:   7
10:  11
50:  450
100: 4112
```

Hint: Voor het bepalen van de recursie-stap merk op dat het aantal manieren om een bedrag b te wisselen gegeven de munten $m_0 \ldots m_n$ de som is van

- het aantal manieren om een bedrag b te wisselen gegeven de munten $m_1 \ldots m_n$, en
- het aantal manieren om een bedrag $(b - m_0)$ te wisselen gegeven de munten $m_0 \ldots m_n$.

Merk ook dat deze recursie-stappen eindigen met b gelijk of kleiner dan 0, of als er met alle munten is geprobeerd.

Opgave 2 (25 punten)

Een student ontwikkelt een Java-programma om de verjaardagen van zijn vrienden bij te houden. Daartoe definieert hij eerst een klasse `Dag` voor het weergeven van een dag in het jaar. Merk op dat voor het bijhouden van verjaardagen het (geboorte)jaar niet relevant is.

```
public class Dag {
    public final int dag, maand;

    /** @require isGeldigeDag(dag, maand) */
    public Dag(int dag, int maand) {
        this.dag   = dag;
        this.maand = maand;
    }

    /** Levert true op als deze Dag dezelfde dag is als other. */
    public boolean equals(Object other) {
        if (other instanceof Dag) {
            Dag ander = (Dag) other;
            return (this.dag == ander.dag) && (this.maand == ander.maand) ;
        }
        else
            return false;
    }

    /** Levert de hashcode van deze Dag. */
    public int hashCode() {
        return dag;
    }

    public String toString() {
        // ... implementatie verder weggelaten
    }

    /** Levert true op als het paar (dag, maand) een geldige dag is.
      * Geldige dag: 1 <= maand <= 12 && dag >= 1, en
      *              als maand in {1, 3, 5, 7, 8, 10, 12} dan dag <= 31
      *              als maand in {4, 6, 9, 11}           dan dag <= 30
      *              als maand in {2}                     dan dag <= 29 */
    public static boolean isGeldigeDag(int dag, int maand) {
        // ... implementatie verder weggelaten
    }
}
```

Daarnaast definieert hij een klasse `Verjaardagen`.

```
public class Verjaardagen {
    private  Map<String,Dag> vdagen;

    public Verjaardagen() {
        // ... body nog toe te voegen
    }

    public void voegtoe(String naam, Dag d)   { vdagen.put(naam, d);          }
    public Dag verjaardagVan(String naam)     { return (Dag) vdagen.get(naam); }
    public Set<String> alleNamen()            { return vdagen.keySet();        }
    public Collection<Dag> alleVerjaardagen() { return vdagen.values();        }
    public String toString()                  { return vdagen.toString();      }
}
```

De verjaardagen worden bijgehouden in de instantievariabele `java.util.Map<String,Dag> vdagen`: bij elke naam (de *key*: een `String`) hoort een `Dag` (de *value*).

De student is ook geïnteresseerd in de sterrenbeelden van zijn vrienden. Daarom schrijft hij een klasse `Sterrenbeeld` met de klasse-constante `Map<Dag,String> sterrenbeelden` met daarin voor elke `Dag` van het jaar het bijbehorende sterrenbeeld (als `String`).

```
public class Sterrenbeeld {
    public final static String[] STERRENBEELD = {
        "Waterman", "Vissen", "Ram", "Stier", "Tweelingen", "Kreeft", "Leeuw",
        "Maagd", "Weegschaal", "Schorpioen", "Boogschutter", "Steenbok",
    };

    // keys: alle Dag-en van het jaar, values: bijbehorende sterrenbeelden
    public final static Map<Dag,String> sterrenbeelden;     // Dag -> String

    // ... initialisatie van sterrenbeelden is weggelaten
}
```

Bij deze opgave wordt u gevraagd verschillende methoden van de klasse `Verjaardagen` te definiëren. Hieronder staat alvast een voorbeeld van het gebruik van de klasse `Verjaardagen` en de gevraagde methoden.

```
    public static void main(String[] args) {
        Verjaardagen vd = new Verjaardagen();
        vd.voegtoe("Amalia", new Dag( 7, 12));
        vd.voegtoe("Jan-Peter", new Dag( 7,  5));
        vd.voegtoe("Jamai" , new Dag(30,  6));
        vd.voegtoe("Britney"  , new Dag( 2, 12));
        vd.voegtoe("Edgar" , new Dag(13,  3));
        System.out.println(vd);

        if (vd.verschillendeDagen()) {
            Map<Dag,String> dagnaam = vd.mDagNaam();
            System.out.println(dagnaam);
        }
        else
            System.out.println("Verjaardagen␣zijn␣niet␣allemaal␣verschillend!");

        Map<String,String> sbeelden = vd.mSterrenbeeld();
        System.out.println(sbeelden);
    }
```

Het voorbeeld zou de volgende uitvoer (kunnen) genereren:

```
{Britney=2-dec, Jan-Peter=7-mei, Edgar=13-mar, Amalia=7-dec, Jamai=30-jun}
{2-dec=Britney, 13-mar=Edgar, 30-jun=Jamai, 7-dec=Amalia, 7-mei=Jan-Peter}
```

```
{Britney=Boogschutter, Jan-Peter=Stier, Edgar=Vissen, Amalia=Boogschutter, Jamai
=Kreeft}
```

Let bij de implementaties van onderstaande methoden ook op de efficiëntie van de algoritmen. Met name teveel (impliciete) iteraties en/of (onnodige) cast-operaties worden aangerekend.

a. *(3 pnt.)* Geef de body van de constructor van de klasse `Verjaardagen`.

b. *(7 pnt.)* Implementeer de methode `verschillendeDagen` binnen de klasse `Verjaardagen`:

```
/** Levert true op als de personen allemaal op verschillende
 *  dagen jarig zijn, anders false.  */
public boolean verschillendeDagen()
```

c. *(8 pnt.)* Implementeer de methode `mDagNaam` binnen de klasse `Verjaardagen`:

```
/** Levert een omgekeerde Map op: de keys zijn nu Dag-objecten
 *  en de values zijn Strings.
 *  @requires verschillendeDagen()
 *  @ensures  result.values().equals(alleNamen()) &&
 *            result.keySet().equals(alleVerjaardagen()) &&
 *            voor alle (d,n) in result geldt: d.equals(verjaardagVan(n)) */
public Map<Dag,String> mDagNaam()
```

d. *(7 pnt.)* Implementeer de methode `mSterrenbeeld` binnen de klasse `Verjaardagen`:

```
/** Levert een Map op waarbij de keys en values allebei Strings zijn.
 *  De keys zijn de oorspronkelijke namen uit alleNamen() en de
 *  values zijn de bijbehorende sterrenbeelden.
 *  Maakt gebruik van de Map Sterrenbeeld.sterrenbeeld (Dag -> String),
 *  die voor elke Dag het bijbehorende sterrenbeeld oplevert.
 *  @ensures  result.keySet().equals(alleNamen())
 *            voor alle (n,s) in result geldt:
 *            s.equals(Sterrenbeeld.sterrenbeeld.get(verjaardagVan(n))) */
public Map<String,String> mSterrenbeeld()
```

Opgave 3 (15 punten)

Om de methode `mDagNaam` van de klasse `Verjaardagen` van opgave 2c te kunnen gebruiken moeten we altijd eerst controleren of aan de preconditie van deze methode voldaan wordt. In deze opgave definiëren we een *wrapper*-methode `mDagNaamE` die de methode `mDagNaam` aanroept. De methode `mDagNaamE` heeft zelf geen preconditie maar gooit in plaats daarvan een `VerjaardagenException` als niet aan de preconditie van `mDagNaam` voldaan wordt.

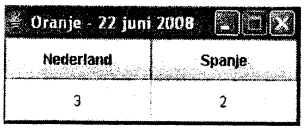
a. *(8 pnt.)* Definieer een `Exception`-klasse `VerjaardagenException`. Implementeer de methode `mDagNaamE` die `mDagNaam` aanroept als aan de preconditie van `mDagNaam` voldaan wordt, maar die een `VerjaardagenException` gooit als niet aan deze preconditie voldaan wordt.

b. *(7 pnt.)* Pas het `if`-statement van de methode `main` van opgave 2 zodanig aan dat nu de methode `mDagNaamE` gebruikt wordt. De uitvoer van `main` moet wel precies hetzelfde blijven.

Opgave 4 (20 punten)

Een enthousiaste Java programmeur en chauvinistische voetbalfan wil graag de tussenstand van voetbalwedstrijden tijdens het EK bijhouden. Hij heeft daartoe een klasse `ScoreJFrame` geschreven die de tussenstand van een wedstrijd van zijn favoriete team op het scherm laat zien. De klasse `ScoreJFrame` ziet er als volgt uit:

```
public class ScoreJFrame extends JFrame
                         implements ActionListener {
    private static final String VOOR  = "VOOR";
    private static final String TEGEN = "TEGEN";

    private class Score {
        private int[]   score = { 0, 0 };
        private String  wij, zij;

        public Score(String wij, String zij) {
            this.wij = wij;
            this.zij = zij;
        }

        public void   voorDoelpunt()    { score[0]++;       }
        public void   tegenDoelpunt()   { score[1]++;       }
        public int    voor()            { return score[0];  }
        public int    tegen()           { return score[1];  }
        public String wij()             { return wij;       }
        public String zij()             { return zij;       }
    }

    private JButton     btVoor, btTegen;
    private JTextField  tfVoor, tfTegen;
    private Score       wedstrijd = new Score("Nederland", "Spanje");

    public ScoreJFrame() {
        super("Oranje - 22 juni 2008");
        init();
    }

    public void init() {
        Container c = getContentPane();
        c.setLayout(new GridLayout(2,2));

        btVoor = new JButton(wedstrijd.wij());
        btVoor.setActionCommand(VOOR);
        btVoor.addActionListener(this);

        btTegen = new JButton(wedstrijd.zij());
        btTegen.setActionCommand(TEGEN);
        btTegen.addActionListener(this);

        tfVoor = new JTextField(5);
        tfVoor.setText(String.valueOf(wedstrijd.voor()));
        tfVoor.setHorizontalAlignment(JTextField.CENTER);
        tfVoor.setEditable(false);

        tfTegen = new JTextField(5);
        tfTegen.setText(String.valueOf(wedstrijd.tegen()));
        tfTegen.setHorizontalAlignment(JTextField.CENTER);
        tfTegen.setEditable(false);

        c.add(btVoor); c.add(btTegen);
        c.add(tfVoor); c.add(tfTegen);

        setSize(250,100);
        setResizable(false);
```



Figuur 1: Screenshot van de klasse `ScoreJFrame` in actie.

```
            setVisible(true);
60      }

        public void actionPerformed(ActionEvent ev) {
            String act = ev.getActionCommand();
            if (act.equals(VOOR)) {
65              wedstrijd.voorDoelpunt();
                tfVoor.setText(String.valueOf(wedstrijd.voor()));
            } else if (act.equals(TEGEN)) {
                wedstrijd.tegenDoelpunt();
                tfTegen.setText(String.valueOf(wedstrijd.tegen()));
70          }
        }

        public static void main(String[] args) {
            new ScoreJFrame();
75      }
    }
```

De klasse `ScoreJFrame` is een `JFrame` met vier componenten. Er zijn twee `JButtons` waarmee de doelpunten van het eigen team en de tegenstander kunnen worden bijgehouden. De twee `JTextFields` geven steeds het aantal doelpunten voor het eigen team en de tegenstander weer. Figuur 1 toont een screenshot van het programma `ScoreJFrame`.

Hoewel een aardige poging, is de klasse `ScoreJFrame` niet volgens het *Model-View-Controller* principe ontworpen. In deze opgave moet de klasse `ScoreJFrame` worden opgesplitst in drie `public` klassen `Score`, `ScoreView` en `ScoreController`, zodat de ontstane applicatie wel volgens het *Model-View-Controller* patroon ontworpen is. De `public` klasse `Score` houdt de score van een wedstrijd bij. De `public` klasse `ScoreController` dient de interface `ActionListener` te implementeren. De `public` klasse `ScoreView` bevat een methode `main` die er als volgt uit ziet:

```
public static void main(String[] args) {
    Score model = new Score("Nederland", "Spanje");
    ScoreController controller = new ScoreController(model);
    ScoreView view = new ScoreView(controller, model.wij(), model.zij());
    model.addObserver(view);
    view.update(model, null);
}
```

Geef het nieuwe ontwerp van dit programma volgens het *Model-View-Controller* patroon zoals hierboven beschreven.

Hint: Het nieuwe ontwerp komt er grotendeels op neer dat stukken van `ScoreJFrame` naar `Score`, `ScoreView` en `ScoreController` verhuizen. Het is niet nodig om alle code van `ScoreJFrame` over te schrijven. U kunt volstaan met aan te geven welke gedeelten naar welke klasse verhuizen en wat er veranderd dient te worden. Uw antwoord dient echter wel *ondubbelzinnig* en *volledig* te zijn: het moet duidelijk zijn hoe de diverse Java klassen er precies uit komen te zien, want deze vraag gaat over welke methoden aangepast moeten worden en hoe.

Opgave 5 (20 punten)

Beschouw de volgende klassen.

```
public class SpoorwegOvergang {
    public static void main(String[] args) {
        (new Auto("Honda")).start();
        (new Auto("Fiat")).start();
        (new Auto("VW")).start();
        (new Trein(4)).start();
    }
}

class Auto extends Thread {
    private String merk;
    public Auto(String merk) { this.merk = merk; }

    public void run() {
        while (true)
            System.out.print(merk + "␣");
    }
}

class Trein extends Thread {
    private int wagons;
    public Trein(int wagons) { this.wagons = wagons; }

    public void run() {
        while (true) {
            System.out.print("LOCO␣");
            for (int i=1; i<=wagons; i++) {
                System.out.print("WAGON" + i +"␣");
            }
        }
    }
}
```

De klasse `SpoorwegOvergang` kan gebruikt worden om een spoorwegovergang te modelleren en te simuleren. De `Thread`-klasse `Auto` schrijft voortdurend een `String merk` naar de standaard uitvoer. De `Thread`-klasse `Trein` schrijft steeds de `String "LOCO␣"` gevolgd door een `aantal` malen de `String "WAGON"` (met daaraan vast het nummer van de *wagon*) naar de standaard uitvoer. Een gedeelte van de (oneindige) uitvoer van het programma zou er als volgt kunnen uitzien:

```
... Honda Honda VW LOCO VW Fiat WAGON1 Fiat WAGON2 Honda Honda WAGON3 VW VW Fiat
Honda VW WAGON4 VW Honda Fiat LOCO WAGON1 Fiat Honda VW WAGON2 Honda Honda ...
```

a. *(3 pnt.)* Beschrijf de uitvoer van het programma als in de methode `main`, van het statement

```
(new Auto("Fiat")).start();
```

de aanroep van `start` vervangen wordt door `run`. Motiveer uw antwoord.

b. *(7 pnt.)* Zoals te zien valt in de voorbeelduitvoer kan de uitvoer van de `Trein`-thread onderbroken worden door de uitvoer van één of meerdere `Auto`-threads. Bij een spoorwegovergang is dit uiteraard niet wenselijk.

Zorg ervoor dat het passeren van één trein (d.w.z. het afdrukken van de string `LOCO` en de bijbehorende `WAGONS`) niet langer onderbroken kan worden door de uitvoer van één van de `Auto`-threads. Synchroniseer daartoe (met behulp van `synchronized` en/of aanroepen van de methoden `wait`, `notify` en `notifyAll`) de methoden `run` van de klasse `Auto` en `Trein`. Het is hierbij toegestaan om één extra (instantie- of klasse-) variabele te introduceren.

c. *(10 pnt.)* In plaats van één trein, laten we nu twee treinen rijden (d.w.z. de methode `main` start twee `Trein`-threads op). Deze twee treinen moeten nu tegelijkertijd (naast elkaar) de spoorwegovergang kunnen passeren. De volgende uitvoer zou dan dus mogelijk zijn:

```
... Honda Fiat VW LOCO WAGON1 WAGON2 LOCO WAGON1 WAGON3 WAGON2 WAGON3
WAGON4 WAGON4 VW Honda LOCO WAGON1 WAGON2 WAGON3 WAGON4 Fiat VW ...
```

Synchroniseer met behulp van `synchronized` en/of aanroepen van de methoden `wait`, `notify` en `notifyAll` de methoden `run` van de klasse `Auto` en `Trein` zodat de uitvoer van beide `Trein`-threads elkaar kan overlappen. De uitvoer van de treinen mag uiteraard niet onderbroken wordt door de uitvoer van een `Auto`-thread. Het is hierbij toegestaan om twee extra (instantie- of klasse-) variabelen te introduceren.

Hint: Merk op dat in dit geval moet er bijgehouden worden hoeveel treinen op een gegeven moment de spoorwegovergang passeren.