TENTAMEN

PROGRAMMEREN 2

vakcode: 213505
datum: 9 april 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.

JAVADOC. Bij enkele opgaven van dit tentamen wordt gevraagd om, gegeven een `javadoc`-specificatie van een methode, de implementatie van die betreffende methode te schrijven. Bij het beantwoorden van een dergelijke opgave is het uiteraard niet nodig om het `javadoc`-commentaar te herhalen. Voeg alleen `javadoc`-commentaar toe als daar expliciet om gevraagd wordt.

Opgave 1 (20 punten)

Beschouw de klasse `Sentence` gedefinieerd als volgt

```
public class Sentence {
        private String text;

        public Sentence () {
                this.text = "";
        }

        /**
         * @require t != null
         */
        public Sentence (String t) {
                this.text = t;
        }
```

Stel voor dat wij een methode `indexOf (String t)` voor deze klasse willen implementeten die de positie (index) geeft van het begin van de eerste substring van `text` die correspondeert met `t`, en `-1` als deze substring niet te vinden is.

- Geef een recursieve implementatie van de methode `indexOf (String t)`, waarin alleen de methoden `length()`, `substring(int begin, int end)` en `equals(String t)` van `String` gebruikt mogen worden.
- Geef de aanroepboom en het verwachte resultaat van de aanroep van `indexOf` in

```
Sentence s = new Sentence("Programmeren");
   int n = s.indexOf("gra");
```

Hints:

- Gebruik een recursieve hulpmethode om `indexOf(String t)` te implementeren.
- Een voorbeeld van aanroepboom is te vinden in Niño/Hosch (Figuur 19.4).

Opgave 2 (20 punten)

Beschouw de `BankAccount` klasse gedefinieerd als volgt:

```
public class BankAccount {

        private String name;
        private double balance;

        public BankAccount(String name, double balance) {
                this.name = name;
                if (balance > 0)
                        this.balance = balance;
                else
                        this.balance = 0;
        }

        public boolean deposit(double amount) {
                if (amount < 0) return false;
                balance = balance + amount;
                return true;
        }

        public boolean withdraw(double amount) {
                if (amount > balance || amount < 0) return false;
                balance = balance - amount;
                return true;
        }
}
```

- Geef een nieuwe definitie van `BankAccount` waarin excepties worden gegooid in het geval een poging wordt gedaan om (i) een rekening aan te maken met een negatieve initiële saldo, (ii) een negatief bedrag over te maken en (iii) een negatief bedrag uit te halen. Deze zijn geen standaard Java excepties, d.w.z. ze dienen alle drie gedefinieerd te worden als aparte excepties. Geef de definities van deze excepties ook in uw antwoord.
- Schrijf een test-programma die alle drie excepties opvangt en hun omschrijving afdrukt. Dit programma moet alle drie gevallen testen en de excepties niet propageren!

Opgave 3 (20 punten)

De interface `java.util.Map<K,V>` is een model van een wiskundige *functie*. Gegeven een *key*, levert een `Map` immers de bijbehorende *value*. In deze opgave ontwikkelen we enkele hulpmethoden voor het gebruik van functies. Beschouw de klasse `MapUtils`.

```
public class MapUtils {

    /** Controleert of map een injectie is.
     * @return true als voor alle v in map.values() er precies
```

```
     *          een k in map.keySet() is met v == map.get(k)
     */
    public static <K,V> boolean isInjectief(Map<K,V> map) {
        // vraag a.
    }

    /** Gegeven een Map<K,V> map, levert de methode keerom een nieuwe
     * Map<V, Set<K>> op, waarin de keys en values van map "omgekeerd" zijn.
     * Aangezien de oorspronkelijke map niet injectief hoeft te zijn,
     * zijn de values van het resultaat Set-objecten.
     * @require  map != null
     * @ensure   voor alle (k,v) in map:
     *              result.containsKey(v) && result.get(v).contains(k)
     *           voor alle (k,v) in result:
     *              map.containsValue(k)
     */
    public static <K,V> Map<V, Set<K>> keerom(Map<K,V> map) {
        // vraag b.
    }

    /** Gegeven een Map<K,V> map, levert de methode maxValues een Set<V> op
     * met daarin alle values uit map.values() die het meeste voorkomen.
     * @require  map != null
     * @ensure   zij #(map,v) == aantal keren dat v in map.values() voorkomt
     *           voor alle x in result en voor alle y in map.values():
     *              #(map,x) >= #(map,y) &&
     *              #(map,x) == #(map,y) => result.contains(y)
     */
    public static <K,V> Set<V> maxValues(Map<K,V> map) {
        // vraag c.
    }
}
```

Bij deze opgave zult u de verschillende `static` methoden van de klasse `MapUtils` moeten implementeren. Hieronder staat een voorbeeld van het gebruik van deze methoden.

```
    public static void main(String[] args) {
        Map<String,String> studs = new TreeMap<String, String>();
        studs.put("Ab"  , "INF"); studs.put("Bea" , "BIT");
        studs.put("Cor" , "INF"); studs.put("Dora", "TEL");
        studs.put("Eric", "BIT"); studs.put("Flip", "TEL");
        studs.put("Gea" , "BIT"); studs.put("Henk", "INF");

        System.out.println(studs);
        System.out.println(isInjectief(studs));
        System.out.println(keerom(studs));
        System.out.println(maxValues(studs));
    }
```

Het programma zal de volgende uitvoer (kunnen) genereren:

```
{Ab=INF, Bea=BIT, Cor=INF, Dora=TEL, Eric=BIT, Flip=TEL, Gea=BIT, Henk=INF}
false
{BIT=[Gea, Eric, Bea], TEL=[Dora, Flip], INF=[Ab, Henk, Cor]}
[BIT, INF]
```

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. *(5 pnt.)* Geef een implementatie van de methode `isInjectief`.

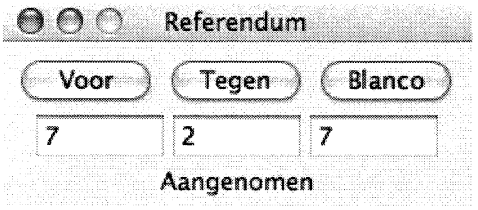
b. *(7 pnt.)* Geef een implementatie van de methode `keerom`.

c. *(8 pnt.)* Geef een implementatie van de methode `maxValues`.

Opgave 4 (20 punten)

Een ondernemende student voorziet een toename van het aantal referenda in Nederland. Hij besluit een Java-programma te ontwikkelen waarmee het eenvoudig wordt om stemmen te tellen en meteen ook de uitslag af te kunnen lezen. Zijn eerste poging ziet er als volgt uit:

```
1  public class Referendum extends JFrame implements ActionListener {
       public static final int VOOR=0, TEGEN=1, BLANCO=2;
       public static final String action[] = { "VOOR", "TEGEN", "BLANCO" };
       private int teller[] = { 0, 0, 0 } ;

6      private JButton     bVoor,  bTegen,  bBlanco;
       private JTextField  tfVoor, tfTegen, tfBlanco;
       private JLabel      lUitslag;

       public Referendum() {
11         super("Referendum");
           init();
       }

       protected void init() {
16         Container c = getContentPane();
           c.setLayout(new FlowLayout());

           bVoor   = new JButton("Voor");   bVoor.setActionCommand(action[VOOR]);
           bTegen  = new JButton("Tegen");  bTegen.setActionCommand(action[TEGEN]);
21         bBlanco = new JButton("Blanco"); bBlanco.setActionCommand(action[BLANCO]);
           bVoor.addActionListener(this);
           bTegen.addActionListener(this);
           bBlanco.addActionListener(this);

26         tfVoor   = new JTextField(5); tfVoor.setEditable(false);
           tfTegen  = new JTextField(5); tfTegen.setEditable(false);
           tfBlanco = new JTextField(5); tfBlanco.setEditable(false);
           lUitslag = new JLabel();      lUitslag.setText("Onbeslist");

31         c.add(bVoor);  c.add(bTegen);  c.add(bBlanco);
           c.add(tfVoor); c.add(tfTegen); c.add(tfBlanco);
           c.add(lUitslag);

           setSize(250, 110);
36         setVisible(true);
           setResizable(false);
       }

       public void actionPerformed(ActionEvent ev) {
41         String act = ev.getActionCommand();
           if (act.equals(action[VOOR]))
               teller[VOOR]++;
           else if (act.equals(action[TEGEN]))
               teller[TEGEN]++;
46         else
               teller[BLANCO]++;
```



Figuur 1: Screenshot van het programma `Referendum` in actie.

```
            showUitslag(teller[VOOR], teller[TEGEN], teller[BLANCO]);
        }
51
        private void showUitslag(int voor, int tegen, int blanco) {
            tfVoor.setText("   + voor);
            tfTegen.setText("  + tegen);
            tfBlanco.setText("" + blanco);
56
            if (voor > tegen)
                lUitslag.setText("Aangenomen");
            else if (voor < tegen)
                lUitslag.setText("Verworpen");
61          else
                lUitslag.setText("Onbeslist");
        }

        public static void main(String[] args) {
66          new Referendum();
        }
    }
```

Figuur 1 toont een screenshot van het programma `Referendum`. De klasse `Referendum` is een `JFrame` met zeven componenten. Bij het indrukken van één van de drie knopjes wordt er een 'voor'-, 'tegen'- of 'blanco'-stem geregistreerd. De drie `JTextFields` onder de knopjes geven het totaal aantal stemmen van de drie soorten stemmen weer. De `JLabel` onderaan geeft de uitslag weer op grond van het aantal voor- en tegen-stemmen: "Aangenomen", "Verworpen" of "Onbeslist".

Hoewel een aardige poging, is de klasse `Referendum` niet volgens het *Model-View-Controller* principe ontworpen. In deze opgave dient de klasse `Referendum` opgesplitst te worden in drie `public` klassen: `Teller`, `TellerView` en `TellerController` en een `interface Stemmen`, zodat de ontstane applicatie wel volgens het *Model-View-Controller* patroon ontworpen is. De klasse `Teller` krijgt de functionaliteit van de array `teller` en houdt de drie soorten stemmen bij. De klasse `Teller` heeft een methode `void inc(int i)` om een stem van soort `i` te tellen en een methode `int totaal(int i)` die het totaal aantal stemmen op soort `i` oplevert. Voor beide methoden geldt: `VOOR <= i <= BLANCO`. De klasse `TellerView` laat een representatie van `Teller` zien. De klasse `TellerController` handelt de `ActionEvents` af en zal overeenkomstig het `Teller`-object wijzigen. De interface `Stemmen` wordt gegeven en bevat alleen de definitie van de `int`-constanten `VOOR`, `TEGEN` en `BLANCO` en de `String`-constanten in de array `action`.

```
interface Stemmen {
    public static final int VOOR=0, TEGEN=1, BLANCO=2;
    public static final String action[] = { "VOOR", "TEGEN", "BLANCO" };
}
```

De klassen die deze constanten nodig hebben hoeven deze interface slechts te implementeren om de constanten zonder de prefix `Stemmen.` te kunnen gebruiken.

Tenslote bevat de klasse `TellerView` de volgende methode `main`:

```
public static void main(String[] args) {
    Teller teller = new Teller();
    TellerController controller = new TellerController(teller);
    TellerView view = new TellerView(controller);
    teller.addObserver(view);
    view.update(teller, null);
}
```

Hint: Het nieuwe ontwerp komt er grotendeels op neer dat stukken van de klasse `Referendum` naar `Teller`, `TellerView` en `TellerController` verhuizen. Het is niet nodig om alle code van `Referendum` over te schrijven. U kunt volstaan met aan te geven welke gedeelten naar welke klasse verhuizen en wat er veranderd dient te worden.

Opgave 5 (20 punten)

Bij deze opgave dient u de mogelijke uitkomsten van enkele programma's te bepalen. Naast 'normale uitvoer' zouden ook de volgende 'uitkomsten' kunnen optreden:

- *vertaalfout*: het programma is geen correct Java programma;
- *runtime-fout*: er wordt een `Exception` gegooid;
- *geen uitvoer*: bijvoorbeeld vanwege een deadlock.

U hoeft uw antwoorden *niet* te motiveren. U zult zien zijn dat de opgaven na opgave a. variaties zijn op het oorspronkelijke programma. Als u meent dat een variatie de uitkomst van het oorspronkelijke programma niet verandert, kunt u volstaan met "als a." als antwoord.

Beschouw het Java-programma `Groei`.

```
public class Groei {
    private String s = "#";

    public String getString()          { return s;     }
    public void   voegtoe(String toe) { s = s + toe; }

    static final Groei g = new Groei();

    public static void main(String[] args) {
        Thread ta = new Thread() { public void run() { g.voegtoe("a"); }; };
        Thread tb = new Thread() { public void run() { g.voegtoe("b"); }; };

        ta.start(); tb.start();

        try { ta.join(); tb.join(); }
        catch (InterruptedException e) {}

        System.out.println(g.getString());
    }
}
```

a. *(3 pnt.)* Wat zijn de mogelijke uitkomsten van het programma `Groei`? *Hint*: denk aan de race condities van methode `voegtoe`!

b. *(2 pnt.)* In het oorspronkelijke programma wordt het gehele `try-catch`-blok (twee regels dus) uit de methode `main` verwijderd. Wat zijn nu de mogelijke uitkomsten van het programma?

c. *(2 pnt.)* In het oorspronkelijke programma wordt het gehele `try-catch`-blok (twee regels dus) uit de methode `main` verwijderd. Daarnaast worden ook de aanroepen van `ta.start()` en `tb.start()` in de methode `main` veranderd in respectievelijk `ta.run()` en `tb.run()`. Wat zijn nu de mogelijke uitkomsten van het programma?

d. *(2 pnt.)* In het oorspronkelijke programma worden de methoden `run` van de `Threads` `ta` en `tb` beiden als `synchronized` methoden gedefinieerd. Wat zijn nu de mogelijke uitkomsten van het programma?

e. *(2 pnt.)* In het oorspronkelijke programma wordt de methode `getString` als `synchronized` methode gedefinieerd. Wat zijn nu de mogelijke uitkomsten van het programma?

f. *(3 pnt.)* In het oorspronkelijke programma wordt de methode `voegtoe` als `synchronized` methode gedefinieerd. Wat zijn nu de mogelijke uitkomsten van het programma?

g. *(3 pnt.)* In het oorspronkelijke programma wordt de methode `voegtoe` als volgt veranderd:

```
public void voegtoe(String toe) {
    s = s+toe;
    notifyAll();
}
```

Wat zijn nu de mogelijke uitkomsten van het programma?

h. *(3 pnt.)* In het oorspronkelijke programma wordt de methode `voegtoe` als volgt veranderd:

```
public synchronized void voegtoe(String toe) {
    if (s.equals("#") && toe.equals("a"))
        try { wait(); }
        catch (InterruptedException e) {}
    notify();
    s = s+toe;
}
```

Wat zijn nu de mogelijke uitkomsten van het programma?