DIAGNOSTISCHE TOETS

Softwaresystemen

datum: Donderdag van Week 7

- Deze diagnostische toets bevat vragen over excepties en concurrency.
- Beantwoord de vragen zo goed mogelijk **in 30 minuten**
- Bespreek vervolgens gedurende **10 minuten** je antwoorden met je buurman. Leg elkaar uit hoe je tot je antwoorden gekomen bent.
- Tijdens de laatste 5 minuten van deze bijeenkomst worden alle correcte antwoorden gegeven door de begeleider van deze bijeenkomst. Je kunt natuurlijk altijd een toelichting vragen.

Opgave 1 (Excepties)

Stel dat we het volgende stukje programmatekst hebben:

```
class Voorbeeld {

    private int i;
    private int j;
    private String s;

    public void test() {
        try {
            i = i/i;
            j = 0;
            i = s.length();
            j = 1;
        }
        catch (ArithmeticException e) {
            j = 3;
        }
        catch (NullPointerException e) {
            j = 4;
        }
        catch (Exception e) {
            if (e instanceof IllegalArgumentException) j = 6;
            else j = 5;
        }
        System.out.println("De␣waarde␣van␣j␣is␣" + j);
    }
}
```

Wat is de waarde van `j` die op het scherm geprint wordt in de volgende gevallen:

a. `i == 3` en `s.equals("tekst")`

b. `i == 0`

c. `i == 6` en `s == null`

Opgave 2 (Concurrency)

Gegeven zijn de volgende klasse `Cell`:

```
class Cell {
    int v = 1;

    synchronized public int get() {
        return v;
    }

    synchronized public void set(int v) {
        this.v = v;
    }
}

class Alex extends Worker {
    public Alex(Cell c) {
        super(c);
    }

    public void run() {
        int v=c.get();
        c.set(v+4);
    }
}

class Bram extends Worker {
    public Bram(Cell c) {
        super(c);
    }

    public void run() {
        int v=c.get();
        c.set(v*4);
    }
}
```

en een klasse `Worker` die `Cell` gebruikt:

```
public class Worker extends Thread {
    protected Cell c;

    public Worker(Cell c) {
        this.c = c;
    }

    public static void main(String[] args) {
        Cell cell = new Cell();
        Worker w1 = new Alex(cell);
        Worker w2 = new Bram(cell);
        w1.start();
        w2.start();
        try {
            w1.join();
            w2.join();
        }
        catch (InterruptedException e) {}
        System.out.println(cell.get());
```

```
    }
}
```

Beantwoord de volgende vragen:

a. Welke waarden worden mogelijk door `main` op het scherm gezet.

b. Het gehele try-catch-blok wordt verwijderd uit de methode main(). Wat zijn nu de mogelijke uitkomsten?

c. Het gehele try-catch-blok wordt verwijderd en de aanroepen `w1.start()` en `w2.start()` worden veranderd in `w1.run()` en `w2.run()`. Wat zijn nu de mogelijke uitkomsten?

d. De methode `set` van de klasse `Cell` wordt als volgt veranderd:

```
public synchronized void set(int v) {
  this.v = v;
  notifyAll();
}
```

Wat zijn nu de mogelijke uitkomsten?

e. De methoden run van de klassen Alex en Bram worden als synchronized methoden gedefinieerd. Wat zijn nu de mogelijke uitkomsten?

f. De methode `run` van de klasse `Alex` wordt

```
public void run() {
  synchronized(c) {
    c.set(c.get()+4);
  }
}
```

en de methode `run` van de klasse `Bram` wordt

```
public void run() {
  synchronized(c) {
    c.set(c.get()*4);
  }
}
```

Wat zijn nu de mogelijke uitkomsten?

g. Definieer

```
public static Object o1 = new Object ();
public static Object o2 = new Object ();
```

De methode `run` van de klasse `Alex` wordt

```
public void run() {
  synchronized(o1) {
    synchronized(o2) {
      int v=c.get();
      c.set(v+4);
    }
  }
}
```

De methode `run` van de klasse `Bram` wordt

```
public void run() {
  synchronized(o2) {
    synchronized(o1) {
      int v=c.get();
      c.set(v*4);
    }
  }
}
```

Wat zijn nu de mogelijke uitkomsten?

h. De methode `run` van de klasse `Alex` wordt

```
public void run() {
  synchronized(c) {
    if (c.get() == 1)
    try {
      c.wait();
    }
    catch (InterruptedException e) {}
    c.set(c.get()+4);
  }
}
```

De methode `run` van de klasse `Bram` wordt

```
public void run() {
  synchronized(c) {
    c.notifyAll();
    c.set(c.get()*4);
  }
}
```

Wat zijn nu de mogelijke uitkomsten?

Opgave 3 (Excepties)

Gegeven zijn de volgende subklasses van de klasse `Exception`.

```
class A extends Exception { A() { super("A"); } A(String s) { super(s); } }
class B extends A         { B() { super("B"); } B(String s) { super(s); } }
class C extends A         { C() { super("C"); } C(String s) { super(s); } }
class D extends B         { D() { super("D"); } D(String s) { super(s); } }
class E extends D         { E() { super("E"); } E(String s) { super(s); } }
```

Verder is gegeven de volgende `interface EEE`:

```
interface EEE {
     public Exception doit(Exception e) throws Exception;
}
```

Deze `interface` definieert een methode `doit` die een `Exception`-object als parameter heeft, een `Exception`-object oplevert, maar ook nog een `Exception` kan gooien. Vandaar de naam `EEE`.

We definiëren nu een klasse `Fn1` als een implementatie van `EEE`, die alleen de methode `doit` implementeert.

```
class Fn1 implements EEE {
    public Exception doit(Exception e) throws Exception {
        try                 { throw e;              }
```

```
        catch (C c)          { throw doit(new B()); }
        catch (B b)          { try             { throw e;              }
                               catch (E ee)    { return new A();       }
                               catch (D dd)    { throw  new B();       }
                               catch (C cc)    { throw  b;             }
                               catch (B bb)    { return new C();       }
                               catch (A aa)    { return doit(new D()); }
                             }
        catch (Exception x) { return new E();      }
    }
}
```

De methode `doit` van klasse `Fn1` wordt in de volgende klasse getest.

```
public class ExceptionOpgave {
    /**
     * Probeert eee.doit(e) uit te voeren.
     *  -  Als er GEEN exceptie gegooid wordt door eee.doit(e) zal
     *     "r-" plus de String-representatie van de return Exceptie
     *     van eee.doit(e) opgeleverd worden.
     *  -  Als er WEL een exceptie gegooid wordt door eee.doit(e)
     *     dan zal er "e-" plus de String-representatie van de
     *     gegooide Exceptie opgeleverd worden.
     */
    public static String tryit(EEE eee, Exception e) {
        String    prefix;
        Exception res;
        try                    { res = eee.doit(e) ; prefix = "r-"; }
        catch (Exception ee)   { res = ee          ; prefix = "e-"; }
        return (prefix + res.getMessage());
    }

    public static void main(String[] args) {
        Fn1 f1 = new Fn1();
        Exception[] a = { new A(), new B(), new C(), new D(), new E() };

        for (int i=0; i < a.length; i++) {
            System.out.println(a[i].getMessage() + "␣:␣" + tryit(f1, a[i]));
        }
    }
}
```

De methode `tryit` probeert dus gegeven een `EEE`-object `eee` en een `Exception`-object `e`, de methode `eee.doit(e)` aan te roepen. Merk op dat de methode `getMessage` van de klasse `Exception` de `String` oplevert die bij de constructie van het `Exception`-object meegegeven is.

Geef voor elke aanroep van `tryit` in `main` aan wat de uitvoer zal zijn. `5` regels met eerst de naam van `Exception`-klasse, dan een dubbele punt, gevolgd door of `r-` of `e-`, en tenslotte een hoofdletter.

Als je meent dat er bij een bepaalde `tryit` géén uitvoer zal worden afgedrukt (als bijvoorbeeld een `StackOverflowException` gegooid wordt), dan moet je dit bij de desbetreffende regel als antwoord noteren. Van de overige `tryit`-aanroepen dien je dan nog wel de uitvoer te vermelden, alsof het programma niet voortijdig beëindigd was.