TECHNISCHE UNIVERSITEIT DELFT
Faculteit Elektrotechniek, Wiskunde en Informatica

TU Delft

TI1206 Object Georiënteerd Programmeren (schriftelijk deel)
28 oktober 2015, 9:00-11:00

Bij dit tentamen mag je *geen* gebruik maken van hulpmiddelen zoals boek of slides. Digitale middelen zoals telefoon, tablet, ... stop je in je boekentas/rugzak.

Ter informatie:

- Elke vraag heeft 1 juist antwoord
- Er zijn 25 vragen, elke vraag heeft hetzelfde gewicht.
- Dit tentamen = 8 bladzijden + 1 bladzijde appendix.
- Je vult je antwoorden in op een apart blad dat automatisch verwerkt zal worden
  - Vraag een nieuw invulblad bij verkeerd ingevulde antwoorden.
  - Vergeet niet naam en studentennummer op het antwoordformulier in te vullen!
- De versie moet niet aangevinkt worden op het antwoordformulier. Ook dag/avond moet je niet aanvinken.

Vraag 1 De primitieve types *float* en *double* worden gebruikt voor reële getallen. Welke uitspraak is waar?

a. Zowel float als double kunnen alle mogelijke reële getallen representeren.
b. Float en double maken gebruik van respectievelijk een 32 en 64 bit voorstelling. Bij beide types wordt er in de voorstelling van het getal rekening gehouden met 1) het teken, 2) de exponent en 3) de mantisse.
c. Float en double maken gebruik van respectievelijk een 32 en 64 bit voorstelling. Bij beide types wordt er in de voorstelling van het getal rekening gehouden met 1) het teken, 2) de exponent en 3) de mantisse. Het verschil in beide voorstellingen zit hem enkel in het aantal bits dat beschikbaar is voor de mantisse.
d. Omdat alle nieuwe processoren toch allemaal 64 bit zijn maakt het verschil tussen een float en een double niet meer uit voor de programmeur.

Vraag 2 Een "lazy operator" kan omschreven worden als:

a. Een operator waarbij de 2^{de} operand niet meer geëvalueerd wordt als het resultaat van deze 2^{de} operand niets meer uitmaakt voor het eindresultaat.
b. De lazy operator is een andere naam voor "=", meer specifiek om het onderscheid te maken met "==".
c. De lazy operator is een andere naam voor een toString() methode. In recente versies van Java hoef je immers toString() niet expliciet meer op te roepen om het object om te zetten naar een String. Bijvoorbeeld: System.out.println(mijnKlasse);
d. De lazy operator wordt niet ondersteund door Java.

Vraag 3 Method overloading betekent (welke uitspraak is het meest correct):

a. Je kan in 1 klasse 2 of meer methoden definiëren met dezelfde naam.
b. Je kan in 1 klasse 2 of meer methoden definiëren met dezelfde naam, maar de namen van de parameters moeten verschillend zijn.
c. Je kan in 1 klasse 2 of meer methoden definiëren met dezelfde naam, maar de types van de parameters moeten verschillend zijn.
d. Je kan in 1 klasse 2 of meer methoden definiëren met dezelfde naam, maar de parameterlijsten moeten anders zijn door hun aantal. Als de aantallen wel gelijk zijn, dan moeten de types anders zijn of de volgorde van de types moet anders zijn.

Vraag 4 Wat weet je over de default constructor (de constructor van een klasse zonder parameters) (welke uitspraak is het meest correct):

a. Die moet je in iedere klasse definiëren.
b. Die moet je in iedere klasse public definiëren.
c. Die mag je in iedere klasse definiëren.
d. Die mag je in iedere klasse definiëren. Als je hem zelf niet definieert, dan zal de Java compiler hem standaard genereren. Dit proces kan je tegengaan door hem zelf als private te definiëren.

Vraag 5 Wat is de output na het uitvoeren van volgende main()?

```
public static void swap(int[] rij){
        int temp = rij[0];;
        rij[0]  = rij[1];
        rij[1]  = temp;
}

public static void main(String[] args){
        int[] rij = {3,4};
        System.out.println(rij[0] + " " + rij[1]);
        swap(rij);
        System.out.println(rij[0] + " " + rij[1]);
}
```

a. 3 4
   4 3
b. 3 4
   3 4
c. 4 3
   4 3
d. Geen van bovenstaande

Vraag 6 Wat gebeurt er met/in volgende methode?

```
public static int[] selecteer(int[] rij, int niv){
        int n = rij.length();
        int[] temp = new int[n]; int aantal = 0;
        for(int i = 0; i < n; i = i + 1){
                if (rij[i] >= niv){
                        temp[aantal] = rij[i];
                        aantal = aantal + 1;
                }
        }
        int[] res = new int[aantal];
        for(int i = 0; i < aantal;i = i + 1){
                res[i] = temp[i];
        }
        return res;
}
```

a. Deze compileert niet.
b. Deze geeft aanleiding tot een runtime fout, meer bepaald een ArrayIndexOutofBoundsException.
c. Deze selecteert alle elementen uit een rij die voldoen aan de voorwaarde dat ze groter of gelijk zijn aan een een bepaald niveau.
d. Deze selecteert alle elementen uit een rij die voldoen aan de voorwaarde dat ze groter of gelijk zijn aan een een bepaald niveau. De method geeft een array terug die net groot genoeg is om al deze elementen te bevatten.

Vraag 7 Ga uit van een klasse Punt met 2 attributen x en y, allebei van het type int. Gegeven deze equals methode:

```
public boolean equals(Punt other){
        if (other instanceof Punt){
                Punt that = (Punt)other;
                return (this.x == that.x) && (this.y == that.y);
        }
        return false;
}
```

In een testmethode staat nu dit:

```
Punt a = new Punt(3,4);
Punt b = new Punt(3,4);
assertEquals(a, b);
```

Deze code:

a. Zal niet compileren.
b. Zal aanleiding geven tot een runtime fout.
c. Geeft als resultaat een geslaagde test. ("pass")
d. Geeft als resultaat een falende test ("fail").

Vraag 8 De methode "accept()" is (meest correcte antwoord):

a. Een onderdeel van de klasse Socket en wacht op een binnenkomend verzoek tot communicatie.
b. Een onderdeel van de klasse Socket en wacht op een binnenkomend verzoek tot communicatie. Bovendien "blokkeert" deze methode het programma tot er daadwerkelijk communicatie wordt opgestart.
c. Een onderdeel van de klasse ServerSocket en wacht op een binnenkomend verzoek tot communicatie.
d. Een onderdeel van de klasse ServerSocket en wacht op een binnenkomend verzoek tot communicatie. Bovendien "blokkeert" deze methode het programma tot er daadwerkelijk communicatie wordt opgestart.

Vraag 9 De cyclomatische complexiteit (CC) van een methode is:

a. Het aantal lineair onafhankelijke paden doorheen de code. Je berekent het door het aantal condities te tellen. De CC is een ondergrens voor het aantal testen dat je voor een methode moet schrijven.
b. Het aantal lineair onafhankelijke paden doorheen de code. Je berekent het door het aantal condities te tellen. De CC is een ondergrens voor het aantal testen dat je voor een klasse moet schrijven.
c. Het aantal lineair onafhankelijke paden doorheen de code. Je berekent het door het aantal condities te tellen en telt daar 1 bij. De CC is een ondergrens voor het aantal testen dat je voor een methode moet schrijven.
d. Het aantal lineair onafhankelijke paden doorheen de code. Je berekent het door het aantal condities te tellen en trekt daar 1 vanaf. De CC is een ondergrens voor het aantal testen dat je voor een methode moet schrijven.

Vraag 10 Wat gebeurt er in onderstaand stuk code?

```
public class IntLijst{

        private int[] element;
        private int aantal;

        public IntLijst(int n){
                int[] element = new int[n];
                aantal = 0;
        }

        public int getElement(int index)
        {
                if(index < aantal)
                        return element[i].
        }
}
```

a) Deze klasse compileert niet. Java geeft een foutmelding bij de constructor.
b) Deze klasse compileert niet. Java geeft een foutmelding bij de methode getElement(int index).
c) Deze klasse compileert wel. Er treedt echter wel een Exceptie op tijdens uitvoering van de constructor.
d) Deze klasse compileert wel. Er treedt echter wel een Exceptie op tijdens uitvoering van de methode getElement(int index).

Vraag 11 Als een klasse A de interface Runnable implementeert dan:

a. Moet klasse A een methode *public static void main(String[] args)* implementeren
b. Moet klasse A een methode *public void start()* definieren
c. Moet klasse A een methode *public void run()* definieren
d. Moet klasse A een attribuut van het type Thread hebben.

Vraag 12 Een methode abstract declareren houdt in dat:

a. Deze methode geen returnwaarde kan hebben.
b. Deze methode overriden moet worden in een afgeleide klasse.
c. Deze methode niet meer overriden kan worden in een afgeleide klasse.
d. Deze methode is enkel op te roepen door methoden uit de eigen klasse.

Vraag 13 Een klasse B erft van klasse A. Dan geldt volgende constructie-volgorde:

a. Aanroep constructor superklasse. Initialisatie van de attributen van klasse B. Constructor van B.
b. Initialisatie van de attributen van klasse B. Constructor van A. Constructor subklasse.
c. Aanroep constructor van B. Aanroep van constructor van A. Initialisatie van de attributen van klasse B.
d. Aanroep constructor van A. Aanroep van constructor van B. Initialisatie van de attributen van klasse B.

Vraag 14 Welke uitspraak klopt? (er is maar 1 goed antwoord!)
Stel je hebt een klasse A met 10 methoden. 2 van die 10 methoden zijn synchronized gedefinieerd (en geen andere stukken code zijn synchronized in die klasse A).
Situatie 1: De main thread roept een synchronized methode op op een instantie van klasse A. Dit houdt in dat:

a. Een tweede thread die een niet-synchronized methode op dezelfde instantie van klasse A wil oproepen, nu moet wachten.
b. Een tweede thread die een synchronized methode op dezelfde instantie van klasse A wil oproepen, nu moet wachten.

Situatie 2: De main thread roept een niet-synchronized methode op op een instantie van klasse A. Dit houdt in dat:

c. Een tweede thread die een niet-synchronized methode op dezelfde instantie van klasse A wil oproepen, nu moet wachten.
d. Een tweede thread die een synchronized methode op dezelfde instantie van klasse A wil oproepen, nu moet wachten.

Vraag 15 Welke uitspraak is **fout**? Encapsulatie houdt in dat:

a. Je de interne details van de klasse (de implementatie) verbergt voor de buitenwereld.
b. Je de programmeur beschermt tegen zichzelf.
c. Je het moeilijker maakt om te testen.
d. Je het pass-by-value principe in Java mogelijk maakt.

Vraag 16 Wat is het verschil tussen een thread en een proces?

a. Een proces komt ruwweg overeen met een programma. Het proces krijgt van het operating system een stuk geheugen waarbinnen het kan werken.
   Een thread is een andere naam voor een proces.
b. Een proces komt ruwweg overeen met een programma. Het proces krijgt van het operating system een stuk geheugen waarbinnen het kan werken.
   Binnen een proces kan er slechts 1 thread zijn die het verloop van het programma controleert.
c. Een proces komt ruwweg overeen met een programma. Het proces krijgt van het operating system een stuk geheugen waarbinnen het kan werken.
   Binnen een proces kunnen meerdere threads zijn die parallel of sequentieel opdrachten uitvoeren binnen een proces waardoor het lijkt of het programma meerdere dingen tegelijk doet.
d. Een proces komt ruwweg overeen met een programma. Het proces krijgt van het operating system een stuk geheugen waarbinnen het kan werken.
   Binnen een proces kunnen meerdere threads zijn die elk hun eigen stukje geheugen krijgen en die parallel of sequentieel opdrachten uitvoeren binnen een proces waardoor het lijkt of het programma meerdere dingen tegelijk doet.

Vraag 17 Bekijk volgende klasse, welke uitspraak klopt?

```
public class MeerdereThreads implements Runnable {
        private Thread fThread;

        public MeerdereThreads(){
                fThread = new Thread(this);
                fThread.run();
        }
        public void run(){
                System.out.println("Other thread starts");
        }
}
```

a. Dit stuk code compileert niet.
b. Dit stuk code leidt tot een runtime fout.
c. Dit stuk code start een nieuwe thread op en schrijft op het scherm "Other thread starts".
d. Dit stuk code start geen nieuwe thread op en schrijft op het scherm "Other thread starts".

Vraag 18 Een breakpoint is een punt in je programma waar:

a. Een fout zit
b. Waar een break statement staat (om bijvoorbeeld een case van een switch te eindigen).
c. Waar de debugger even stopt.
d. Waar een iteratie van een lus-structuur eindigt.

Vraag 19 Wat is het verschil tussen een error, checked exception en unchecked exception?

a. Een error is een fout van de gebruiker, een checked exception *moet* opgevangen worden en mag niet doorgegeven worden, terwijl een unchecked exception opgevangen mag worden of mag worden doorgegeven.
b. Een error is een fout van de gebruiker, een checked exception *moet* opgevangen of doorgegeven worden, terwijl een unchecked exception opgevangen mag worden of mag worden doorgegeven.
c. Een error is een systeemfout, een checked exception *moet* opgevangen worden en mag niet doorgegeven worden, terwijl een unchecked exception opgevangen mag worden of mag worden doorgegeven.
d. Een error is een systeemfout, een checked exception *moet* opgevangen of doorgegeven worden, terwijl een unchecked exception opgevangen mag worden of mag worden doorgegeven.

Vraag 20 Bekijk de code in bijlage A. Gegeven de code in bijlage A, wat is het gevolg van volgende 3 lijnen code?

```
Id multiplier = new Multiplier(7);
Functie f = multiplier.afgeleide();
double x = f.apply(2);
```

a. Een compile-time fout
b. Een run-time fout
c. Geen fout, x krijgt de waarde 2 na afloop.
d. Geen fout, x krijgt de waarde 14 na afloop.

Vraag 21 Bekijk de code in bijlage A. Gegeven de code in bijlage A, wat is het gevolg van volgende 3 lijnen code?

```
Functie functie  = new Multiplier(6);
Constant constant = (Constant)functie;
double x = constant.apply(2);
```

a. Een compile-time fout
b. Een run-time fout
c. Geen fout, x krijgt de waarde 12 na afloop.
d. Geen fout, x krijgt de waarde 2 na afloop.

Vraag 22 Bekijk de code in bijlage A. Gegeven de code in bijlage A, wat is het gevolg van volgende 3 lijnen code?

```
Functie functie  = new Multiplier(6);
Id id = (Id)functie;
double x = functie.apply(2);
```

a. Een compile-time fout
b. Een run-time fout
c. Geen fout, x krijgt de waarde 12 na afloop.
d. Geen fout, x krijgt de waarde 2 na afloop.

Vraag 23 Voor volgende stuk code geldt:

```
private static double fRente = 3.15;

public static double geefRente(double correctiefactor)
{
        return this.rente * correctiefactor;
}
```

a. Deze compileert niet.
b. Deze zal een runtime fout opleveren.
c. De returnwaarde is 6.30 als de parameter waarde 2.0 heeft.
d. Geen van de bovenstaande.

Vraag 24 Er bestaat een basisklasse Persoon, daarvan afgeleid zijn Student en Docent. Er moet een gemeenschappelijke lijst van Studenten en Docenten komen. Er bestaat een toString methode in de klasse Persoon die stopt enkel de naam van de Persoon in een String.

Nu wordt volgende code gecompileerd.

```
1:  ArrayList personenlijst = new ArrayList();
2.  personenlijst.add(new Docent("Andy"));
3.  personenlijst.add(new Student("Tim"));
4.  boolean check = personenlijst.get(0).equals("Andy");
5.  Persoon p = personenlijst.get(1);
```

De compiler geeft een fout aan op lijn:

a. 1
b. 2 en 3
c. 4
d. 5

Vraag 25 Om een Model View Controller pattern te implementeren biedt java ondersteuning door middel van:

a. De interfaces Model en View.
b. De klassen Model en View.
c. De interfaces Model, View en Controller.
d. De interface Observer en de klasse Observable.

Bijlage A

```
public abstract class Functie{
   public double apply(double x){
      return x;
   }
}

public class Id extends Functie{ }

public class Constant extends Functie{

   public Constant (double c){
      this.c = c;
   }

   public double apply(double x){
      return c;
   }

   private double c;
}

public class Multiplier extends Id{
   public Multiplier(double f){
      factor = f;
   }

   public double apply(double k){
      return k * factor;
   }

   public Functie afgeleide(){
      return new Constant(factor);
   }

   private double factor;
}

class Compose extends Functie{
    public Compose (Functie first, Functie second){
       this.first = first;
       this.second = second;
    }

    public double apply(double x){
       double y = first.apply(x);
       return second.apply(y);
    }

    private Functie first, second;
}
```