UNIVERSITEIT VAN AMSTERDAM

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Tentamen

Inleiding Programmeren
Bachelor Informatica jaar 1

Tentamen
Datum: 25 oktober 2016
Tijd: 09.00-12.00

Aantal pagina's: 12 (inclusief voorblad)
Aantal vragen: 26
Maximaal aantal te behalen punten: 100
Bij iedere vraag staat het bijbehorende aantal punten vermeld.

VOORDAT U BEGINT

- **Wacht** tot u de instructie krijgt het boekje te openen.
- Controleer of uw versie van het tentamen compleet is.
- Schrijf **uw naam en studentnummer en indien van toepassing versienummer** op **elk vel papier** dat u inlevert en **nummer de pagina's**.
- U dient uw **mobiele telefoon** uit te schakelen en te bewaren in uw jas of tas. Uw **jas en tas** moeten onder uw tafel liggen.
- **Toegestane hulpmiddelen**: kladpapier, spiekbriefje (handgeschreven, één kantje).

HUISHOUDELIJKE MEDEDELINGEN

- De eerste 30 minuten en de laatste 15 minuten mag u de zaal niet verlaten, ook niet voor het bezoeken van het toilet.
- Op verzoek van de examinator (of diens vertegenwoordiger) moet u zich kunnen legitimeren met een bewijs van inschrijving of een geldig legitimatiebewijs.
- Tijdens het tentamen is toiletbezoek niet toegestaan, tenzij de surveillant hier toestemming voor geeft.
- 15 minuten voor het eind wordt u gewaarschuwd dat het inlevertijdstip nadert.
- Vul indien van toepassing na afloop van het tentamen alstublieft het evaluatieformulier in.

Succes!

Voor je ligt het eindtentamen van Inleiding Programmeren, het laatste onderdeel van dit vak.
De vragen zijn opgedeeld in verschillende hoofdstukken. Enkele tips voor je begint:

- Lees eerst rustig het tentamen een keer door van het begin tot het einde alvorens te beginnen.
- Een kort en bondig antwoord is beter dan een lang en onduidelijk antwoord.
- Een onleesbaar antwoord levert geen punten op.
- Een te lang antwoord levert ook geen punten op.
- Sommige vragen kosten meer tijd dan andere, blijf niet te lang hangen en gebruik je tijd verstandig.
  - De laatste vragen leveren relatief gezien de meeste punten op, bewaar ze dus niet voor de laatste 60 minuten!

Heel veel succes!

1 Primitieve datatypen in Java (10 punten)

Primitieve datatypen in Java zijn best fijn omdat ze op iedere computer exact hetzelfde zijn vanwege de Virtuele Machine die Java gebruikt. Zo is een integer bijvoorbeeld altijd 32 bits.

1. (1 punt) Wat is het grootste getal dat in een int past? (Indien nodig: je mag je antwoord ook geven in de vorm van een wiskunde vergelijking.)
2. (2 punten) Hoeveel bits gebruikt een float in Java? Is dat anders (minder, gelijk, meer) dan bij ints? Wat zegt dat over de waarde van ints die naar float gecast worden?
3. (3 punten) Wat zijn de waarden van de variabelen na uitvoering van onderstaande assignments?

```
int    a = 37 / 4;
byte   b = 11 / 2 * 2;
char   c = 'r' - 2;
double d = 8.5 + 2 / 5;
int    e = 12; e -= e - 7;
byte   f = (byte)(3 * Byte.MAX_VALUE) - 2;
```

4. (4 punten) Beschrijf in maximaal 5 zinnen wat onderstaande code uitprint.

```
static void vraag4()
{
    int ubnd = Integer.MAX_VALUE;
    int lbnd = ubnd;
    while ((float)lbnd == (float)ubnd)
    {
        --lbnd;
    }
    System.out.println((++lbnd) + ".." + ubnd);
}
```

2 Klassen en methoden, static en interfaces (15 punten)

Java is een object georiënteerde programmeertaal (OOP), daarover gaan onderstaande vragen.

5. (2 punten) Vul onderstaande tekst aan, kies voor iedere letter uit "static" of "niet-static".

   Bij het definiëren van een methode is het belangrijk om te letten of je hem wel of niet static maakt. Als je vanuit een [A] context zoals de main-methode een [B] methode van dezelfde klasse aanroept dan krijg je een compilerfout. Tevens is het aanroepen van een [C] methode vanuit een [D] context ook niet handig omdat je dan niet bij de instantievariabelen van het object kunt.

6. (3 punten) Wat is de uitvoer van onderstaande code?

```
public class Opgave6 {
    public static void main(String[] args) {
        printer(new int[6], 0);
    }
    static void printer(int[] p, int index) {
        if (index >= p.length)
            System.out.println(Arrays.toString(p));
        else {
            p[index] = index - 2;
            printer(p, index + 1);
        }
    }
}
```

7. (3 punten) Wat is de uitvoer van onderstaande code?

```
class Opgave7 {
    public static void main(String[] args) {
        Gast[] gasten_penthouse = {new Gast("Robin", "de Vries"),
                                   new Gast("Taylor", "Swift") };
        System.out.println(Arrays.toString(gasten_penthouse));
    }
}
class Gast {
    static String voornaam;
    static String achternaam;
    Gast(String voornaam, String achternaam) {
        this.voornaam = voornaam; this.achternaam = achternaam;
    }
    public String toString() {
        return String.format("%s %s", voornaam, achternaam);
    }
}
```

8. (3 punten) Waarom compileert onderstaande code niet?

```
int returnFirstUneven() {
    int[] p = new int[] {2, 6, 18, 0, 14, 3, 10, 6};

    for (int i : p) {
        if (i % 2 == 0)
            return i;
    }
}
```

9. (2 punten) Wat is de uitvoer van onderstaande code?

```
public class Opgave9 {
    public static void main(String[] args){
        int[] a = {8, 3};
        int b = 2, c = 7;
        swap(a, b, c);
        System.out.println("a is {" + a[0] + "," + a[1] + "}");
        System.out.println("b is " + b);
        System.out.println("c is " + c);
    }
    public static void swap(int[] p, int q, int r) {
        int temp = p[0];
        p[0] = p[1];
        p[1] = temp;
        temp = q;
        q = r;
        r = temp;
    }
}
```

10. (2 punten) Compileert onderstaande code wel of niet? En zo niet, vanwege welke reden(en)?

```
interface Opgave10Interface {
    int a();
    int b();
}

class Opgave10 implements Opgave10Interface {
    int a() { return 0; }
    private int b() { return 0; }
}
```

3 Onderdelen van imperatieve programmeertalen (9 punten)

Java is naast een object georiënteerde programmeertaal, ook een imperatieve taal. Imperatieve talen zoals C, PHP, Python en dus Java hebben allemaal overeenkomstige onderdelen bijvoorbeeld if-statements en loops. Door ze juist toe te passen kun je code vaak eleganter en korter krijgen.

11. (5 punten) Als je goed kijkt naar onderstaande code zie je dat een while-loop een wat ongemakkelijke keuze is. Herschrijf deze code naar de kortst mogelijke variant zonder while-loop. Maak gebruik van een for-loop, for-each-loop, do-while loop of recursie.
Tip: Je hoeft niet te weten wat de `PriemZoeker` klasse doet of hoe de implementatie eruit ziet, zolang je de aanroepen maar zo houdt als in onderstaande code.

```
PriemZoeker zoeker = new PriemZoeker();
int i = 0;
boolean gevonden = true;

while(i == 0 || !gevonden)
{
   zoeker.zoek();
   zoeker.herstructureer();
   gevonden = zoeker.priemIsGevonden();
   if(i < 10)
       i++;
}
```

12. (2 punten) Herschrijf onderstaande methode naar de kortst mogelijke vorm:

```
int opgave12(boolean b) {
    int a;
    if (b) {
        a = functiecall(15, false, 'b');
    }
    else {
        a = functiecall(7, false, 'b');
    }
    return a;
}
```

13. (2 punten) Leg uit wat operator overloading is en op welke manier Java dit ondersteunt.

4 Polymorfisme en overerving (17 punten)

Een taal met alleen maar losse objecten zou niet zo krachtig zijn, daarom zijn in object geörienteerde talen deze objecten vaak hiërarchisch ingedeeld.

14. (3 punten) Compileert het volgende programma? Zo ja, wat is de uitvoer? Zo nee, welke regels moeten verwijderd worden en wat is dan de uitvoer?

```
public class Opgave14 {
    public static void main(String [] args) {
1       Glub ggub = new Flub(4);
2       Glub glub = new Glub(5);
3       Flub flub = new Flub(glub);
4       System.out.println(glub);
5       System.out.println(flub);
6       System.out.println(ggub);
    }
}
class Flub {
    int blorg;
    public Flub(int i) {
        blorg = i;
    }
    public Flub(Flub other) {
        this.blorg = other.blorg;
    }
    public String toString() {
        return "Flubs " + blorg;
    }
}
class Glub extends Flub {
    public Glub(int i) {
        super(i);
    }
    public String toString() {
        return "Glubs " + blorg;
    }
}
```

15. (4 punten) Leg in maximaal 5 regels uit wat het verschil is tussen "single inheritance" en "multiple inheritance" en hoe beiden zich verhouden tot Java.
Gebruik hiervoor minstens de woorden "subklasse", "superklasse" en "interfaces".

16. (4 punten) Leg in maximaal 5 regels uit wat het verschil is tussen de `Iterable`- en de `Iterator`-interface.

17. (4 punten) Wat is de uitvoer van onderstaande code?

```
class Opgave17 {
    public static void main(String[] args)
    {
        A a = new A();
        B b = new B();
        A c = b;
        System.out.println(a.f());
        System.out.println(b.f());
        System.out.println(b.i);
        System.out.println(c.i);
    }
}
class A {
    int i = 1;
    int p() {
        return i * 10;
    }
    int f() {
        return p();
    }
}
class B extends A {
    int i = 2;
    int p() {
        return i * 50;
    }
}
```

18. (2 punten) Wat is de uitvoer van onderstaande code?

```
interface Foo { }
class X implements Foo { }
class Y extends X { }
class Z { }
public class Opgave18 {
    public static void main(String[] args) {
        X a = new X();
        X b = new Y();
        Y c = new Y();
        Z d = new Z();
        System.out.println(a instanceof Foo);
        System.out.println(b instanceof Foo);
        System.out.println(c instanceof Foo);
        System.out.println(d instanceof Foo);
    }
}
```

5 Exceptions (6 punten)

De compiler zal je beschermen tegen programmeerfouten, maar "at runtime" kunnen er ook diverse dingen misgaan. Daarvoor maken veel talen gebruik van exceptions. Aanschouw onderstaande code voor de volgende twee vragen:

```
class Opgave {
    class EersteTentamenException extends Exception {}
    class TweedeTentamenException extends RuntimeException {}
    class DerdeTentamenException extends EersteTentamenException {}


    void methode1() [A] {
        throw new DerdeTentamenException();
    }

    void methode2() [B] {
        methode1();

        try {
            methode1();
            throw new TweedeTentamenException();
        } catch (EersteTentamenException e) {
            throw new DerdeTentamenException();
        } catch (TweedeTentamenException e) {
            throw e;
        }
    }
}
```

19. (3 punten) Wat kan er **niet** op de plaats van [A]?
    (a) `throws EersteTentamenException, TweedeTentamenException, DerdeTentamenException`
    (b) `throws EersteTentamenException`
    (c) `throws TweedeTentamenException, DerdeTentamenException`
    (d) Een lege waarde voor [A]

20. (3 punten) Als we instellen [A] = `throws DerdeTentamenException`.
    Wat mag er dan **wel** op plaats [B]?
    (a) `throws TweedeTentamenException`
    (b) `throws TweedeTentamenException, DerdeTentamenException`
    (c) `throws RuntimeException`
    (d) Een lege waarde voor `[B]`

6 Algoritmiek en algoritmisch denken (43 punten)

Het laatste gedeelte van het tentamen gaat over algoritmen. Gedurende het vak hebben we er al diverse gezien: denk bijvoorbeeld aan bubble sort, gnome sort, levenshtein en Dijkstra's kortste pad algoritme. Daarnaast hebben jullie zelf ook een algoritme geschreven om polynomen te vereenvoudigen.

6.1 Recursie (16 punten)

Naast de Lucasreeks (1, 3, 4, 7, 11, ..) is er ook de "Rij van Fibonacci". Deze kent de volgende definitie:

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2} \text{ , voor n} > 1$$

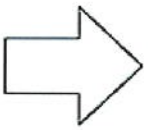
De rij begint dus als volgt: 0, 1, 1, 2, 3, 5, 8, 13, ..

21. (8 punten) Schrijf een methode **int fib(int n)** die met behulp van een **for-loop** het n-de getal uit de Fibonacci reeks uitrekent. (fib(0) is dus 0, fib(1) is 1 en fib(7) is 13)
22. (8 punten) Schrijf een methode **int fib_r(int n)** die met behulp van **recursie** het n-de getal uit de Fibonacci reeks uitrekent.

6.2 Brute force (5 punten)

23. (5 punten) Stel je krijgt een multidimensionale array van 9x9 waar een gedeeltelijk opgeloste sudoku[1] in is opgeslagen. Beschrijf in maximaal 5 zinnen hoe je met een *brute force* techniek de juiste oplossing kunt vinden.

6								1
		8		7		4		
5	9	7	4	6		8		2
					5			4
				8				
	8				2	1		7
4	2	1	7	5			6	
	7			3	4		1	5
9		5						



6	4	3	8	2	9	5	7	1
2	1	8	5	7	3	4	9	6
5	9	7	4	6	1	8	3	2
7	6	2	1	9	5	3	8	4
1	5	4	3	8	7	6	2	9
3	8	9	6	4	2	1	5	7
4	2	1	7	5	8	9	6	3
8	7	6	9	3	4	2	1	5
9	3	5	2	1	6	7	4	8

[1] Een sudoku is een puzzeltje met een 9x9 grid waarbij zowel alle rijen, als alle kolommen als de negen 3x3 grids exact éénmaal de getallen 0 t/m 9 moeten bevatten.

6.3 Polynomen vereenvoudigen (22 punten)

24. (10 punten) Onderstaande implementatie van een Polynoomklasse werkt bijna. De code compileert en geeft geen foutmeldingen bij het uitvoeren, maar ergens gaat het mis als we twee polynomen optellen: de uitvoer is namelijk niet correct. Op de volgende pagina vind je ook de `Paar` klasse en een voorbeeldprogramma met uitvoer. Waar in de Polynoomklasse zit de fout, en welke regel(s) moet je vervangen/aanvullen met wat om het programma werkend te krijgen?

```
1  class Polynoom {
2      ArrayList<Paar> termen = new ArrayList<Paar>();
3      Polynoom(int... lijst) {
4          for(int i = 0; i < lijst.length; i += 2)
5              termen.add(new Paar(lijst[i], lijst[i+1]));
6      }
7      Polynoom(ArrayList<Paar> termen) {
8          this.termen = termen;
9          vereenvoudig();
10     }
11     Polynoom telOp(Polynoom p) {
12         ArrayList<Paar> nt = new ArrayList<Paar>();
13         nt.addAll(termen);
14         nt.addAll(p.termen);
15         return new Polynoom(nt);
16     }
17     private void vereenvoudig() {
18         if(termen.size() == 0)
19             return;
20         Collections.sort(termen);
21         ArrayList<Paar> nieuw = new ArrayList<Paar>();
22         int temp_coef  = 0;
23         int temp_macht = termen.get(0).macht;
24
25         for(Paar paar : termen) {
26             if(paar.macht != temp_macht) {
27                 if(temp_coef != 0)
28                     nieuw.add(new Paar(temp_coef, temp_macht));
29                 temp_coef = paar.coefficient;
30                 temp_macht = paar.macht;
31             } else {
32                 temp_coef += paar.coefficient;
33             }
34         }
35         termen = nieuw;
36     }
37     public String toString() {
38         return Arrays.toString(termen.toArray());
39     }
40 }
```

Onderstaande Paar klasse bevat niet de fout, maar is wel handig om erbij te hebben. In dit voorbeeld worden de machten in oplopende volgorde gesorteerd.

```
class Paar implements Comparable<Paar> {
    final int coefficient;
    final int macht;

    Paar(int c, int m) {
        coefficient = c;
        macht = m;
    }
    public int compareTo(Paar o) {
        return macht - o.macht;
    }
    public String toString() {
        return String.format("%dx^%d", coefficient, macht);
    }
}
```

Onderstaande 2 polynomen (x^2+3x^4 en $-x^2+4x^4$) en de bijbehorende optelling geven de volgende uitvoer:

```
Polynoom p1 = new Polynoom(1,2,3,4);
Polynoom p2 = new Polynoom(-1,2,4,4);
System.out.println(p1);
System.out.println(p2);
System.out.println(p1.telOp(p2));
```

```
[1x^2, 3x^4]
[-1x^2, 4x^4]
[]
```

De laatste vragen bevinden zich op de achterkant van deze pagina!

25. (10 punten) Jordy komt er niet uit en besluit, geïnspireerd door één van zijn studenten om een eenvoudiger vereenvoudigingsalgoritme te implementeren met behulp van een array. Hij is goed op weg, maar er ontbreken nog drie regels code (accolades niet meegeteld). Schrijf Jordy zijn algoritme af. Je mag aannemen dat er geen paren worden toegevoegd met machten kleiner dan 0.

```
private void vereenvoudig2()
{
    if(termen.size() == 0)
        return;

    Paar hoogste = Collections.max(termen);
    ArrayList<Paar> nieuw = new ArrayList<Paar>();
    int[] coefficienten = new int[hoogste.macht + 1];

    for(Paar paar : termen)
        coefficienten[paar.macht] += paar.coefficient;

    // Regel 1
    // Regel 2
    // Regel 3

    termen = nieuw;
}
```

26. (2 punten) Geef een voorbeeld van een polynoom waarvoor deze vereenvoudigmethode onevenredig veel geheugenruimte in beslag neemt en leg uit waarom.