Opleiding Applicatie Ontwikkelaar

A2 - Leerlijn Programmeren

Code indelen in Java

Auteur: *Harmen Steenbergen*

Datum: *18-12-2019*

Crebo: *25187*

Versie: *2.0*

Inhoudsopgave

Overzicht 3
Voorkennis 3
Materialen 3
Bronnen 3
Instructies 3
Beschrijving 4
Doelen 4
Beoordeling 4
Studieonderdelen 5
Gebruik van een methode 5
Oefeningen 6
Argumenten 6
Oefeningen 8
Static 9
Oefeningen 10
Scope van een variabele/methode 11
Regels 12
Oefening 13
Recursie 14
Eindopdrachten 17
Opdracht 1: Formules uitrekenen 17
Opdracht 2: String van karakters opruimen 18
Bronnen 19

Overzicht

Level: Domein A Level 2
Duur: 32 uur
Methode: Weekplanning

Voorkennis

A1 Introductie programmeren in Java.

Materialen

- Je laptop met;
- Een werkende Java IDE

Bronnen

- Zie bijlage Bronnen

Instructies

- Gebruik van methodes/functies om code te structureren
- Gebruik van argumenten om methodes te kunnen hergebruiken
- Scope van variabelen, wanneer heb je welke scope nodig.
- Recursie

Beschrijving

Doelen

Na het bestuderen van deze module:

- ben je in staat om “nette” code te schrijven.
- Kan je gebruik maken van methoden/functies
- Kan je uitleggen waarom variabelen de ene keer wel bereikbaar zijn en de andere keer niet
- Kan je gebruik maken van recursieve methoden

Beoordeling

- Na het doorlopen van deze module heb je 3 inleveropdrachten gemaakt. Deze opdrachten dien je in te leveren via magister.
- De docent zal een mondeling houden over jou gemaakte werk.
- De opdrachten en de mondeling vormen samen jouw eindcijfer voor deze module.

.

Studieonderdelen

Gebruik van een methode

Tot nu toe heb je in Java alleen een stukje code geschreven in de main-methode. Dit is de methode die in Java het startpunt van een programma aangeeft. In een Java-project is er een klasse die wordt gestart, daarin wordt gezocht naar de methode zoals hieronder:

```
public static void main(String[] args) {

}
```

Als je programma uitgebreider wordt en je zou alle code in de methode main zetten wordt dit een heel lange en onoverzichtelijke code. Het eerste wat je gaat doen om dit op te lossen is aparte methoden maken voor stukken code.

Voorbeeld

```
public class ConsoleApp {

   public static void main(String[] args) {
      int dobbel;
      dobbel = gooiDobbelsteen();
      System.out.println(dobbel);
   }

   private static int gooiDobbelsteen() {
      return (int)(1 + 6 *  Math.random());
   }

}
```

Uitleg code

Binnen de klasse *ConsoleApp* wordt een tweede methode geplaatst die een worp met een dobbelsteen nadoet en het resultaat van de worp teruggeeft.

- De *Math.random()* geeft een willekeurig kommagetal tussen 0 en 1.
- Door dit getal te vermenigvuldigen met 6 en te verhogen met 1 krijg je een getal van 1 tot 7 (maar niet 7 zelf)
- Daarna wordt het resultaat getypecast naar een integer waardoor je alleen het getal voor de komma (1 tot en met 6) overhoudt.
- Met de opdracht return wordt dit getal teruggegeven als resultaat van de functie.

In de definitie van de methode staat *private static int*:

- ***private*** betekent dat de methode alleen bestaat binnen de klasse waarin hij staat. Dus *gooiDobbelsteen* mag alleen gebruikt worden binnen de klasse *ConsoleApp.*
- ***static*** geeft aan dat deze methode gebruikt mag worden zonder eerst een object te maken van de klasse waarin de methode staat. Deze methode moet *static* zijn omdat de methode

die hem gebruikt, de main-methode, ook *static* is. Later gaan we dat aanpassen door meerdere klassen te gebruiken.

- ***int*** geeft het returntype van de methode aan. In dit geval is het resultaat van de methode een integer dus een geheel getal. De methode mag daardoor met return alleen een integer waarde teruggeven, en de methode moet een waarde van het type integer teruggeven. Zonder het juiste return statement zie je foutmelding en kun je je code niet compileren. Als je wilt dat een methode is doet maar geen resultaat teruggeeft, gebruik je als returntype *void* Engels voor leegte (niets).

Oefeningen

Oefening 1

Maak een methode met de naam *maakVierkant* die als resultaat een String geeft. De string bevat een vierkant van 5 x 5 plussen. Als je het op de console print zie het er uit als:

```
+++++
+++++
+++++
+++++
+++++
```

Oefening 2

Maak een methode met de naam evenDag die als resultaat geeft of de datum van vandaag een even getal is, resultaat is `true` of `false` (boolean).

Uit het Calendar object kun je de datum van vandaag halen.

Oefening 3

Maak een methode met de naam getOsInfo die uit de systeemeigenschappen de naam, de versie en de architectuur van het besturingssysteem (os) haalt. Als resultaat krijg je een String met daarin de gevraagde info gescheiden door een spatie.

Tip: https://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html

Argumenten

Het voorbeeld van de dobbelsteen voert de code uit en geeft het resultaat terug. Hierbij wordt bij de aanroep van de methode geen informatie meegeven, maar dat wil je vaak wel. Dit kun je doen door argumenten te gebruiken.

Voorbeeld 1

```
public class ConsoleApp {

    public static void main(String[] args) {
     int voorbeeld = 10;
     int resultaat = verhoogMetEen(voorbeeld);
         System.out.println(voorbeeld);
         System.out.println(resultaat);
    }

    private static int verhoogMetEen(int invoer) {
     return ++invoer;
    }

}
```

Uitleg code

De methode *verhoogMetEen* gebruikt de waarde die wordt meegegeven en verhoogt deze met 1, daarna wordt het resultaat weer teruggegeven.

- ++invoer verhoogt de waarde in invoer met 1 voor dat de waarde wordt gebruikt. Dit is wat anders dan invoer++, want daar wordt de waarde van invoer eerst gebruikt en daar pas verhoogt met 1. In dit voorbeeld zou invoer++ er voor zorgen dat de methode als resultaat de waarde teruggeeft dit in het argument *invoer* is meegegeven. Probeer dat zelf maar eens.
- (int invoer) geeft aan dat deze methode bij de aanroep gewacht dat er een integer waarde wordt meegegeven. Deze waarde wordt binnen de methode opgeslagen in de variabele invoer. Deze variabele kun je alleen binnen de methode gebruiken.

Voorbeeld 2

```
public class ConsoleApp {

    public static void main(String[] args) {
     int a = 10;
     int b = 5;
     int som = telOp(a,b);
     System.out.println(som);
    }

    private static int telOp(int getal1, int getal2) {
     return getal1 + getal2;
    }
}
```

Uitleg code

De methode *telOp* tel de twee argumenten bij elkaar op en geeft het resultaat (de som) terug.

- (int getal1, int getal2) de argumenten worden gescheiden door een komma. Elk argument wordt voorafgegaan door het type, in dit geval allebei int.

Oefeningen

Oefening 4

Maak oefening 1 nogmaals, maar nu moet je de grootte van het vierkant als argument aan de methode kunnen meegeven.

Oefening 5

Maak een methode met de naam getMax die als argument een array van integer waarden krijgt en als resultaat de grootste waarde uit de lijst geeft.

Oefening 6

Maak een methode met de naam countChar die als argumenten een String en een Char mee krijgt. De methode telt hoe vaak het teken in het tweede argument (char) in het eerste argument (String) voorkomt en geeft de waarde als resultaat.

Oefening 7

Maak een methode met de naam reverseString die als argument een String mee krijgt. Het resultaat van de methode is de invoer in omgekeerde volgorde, dus ‘oefening’ wordt ‘gninefeo’.

Static

We hebben in de voorbeelden steeds *static* gebruikt omdat de methode main anders de methode niet kan gebruiken. Uiteindelijk willen we zo min mogelijk *static* methoden hebben. De methode main kun je zien als een klein startpunt van je programma dat alleen de objecten maakt om het programma verder te laten lopen. Daarom gaan we in het project een tweede klasse maken en daaruit een methode gebruiken.

Voorbeeld

```
// ConsoleApp.java
public class ConsoleApp {

   public static void main(String[] args) {
      Dobbelsteen dobbelsteen = new Dobbelsteen();
      int dobbel;
      dobbel = dobbelsteen.gooi();
      System.out.println(dobbel);
   }
}
```

```
// Dobbelsteen.java
public class Dobbelsteen {

   public int gooi() {
      return (int)(1 + 6 * Math.random());
   }
}
```

Uitleg code

Er wordt nu een twee bronbestand binnen het project gemaakt voor de klasse Dobbelsteen. Het bestand moet dezelfde naam hebben als de publieke klasse die er in staat. Dus in dit geval heet het bronbestand Dobbelsteen.java (let op de hoofdletters).

In de klasse Dobbelsteen maken we de methode gooi met de code die een dobbelsteen worp nadoet. Deze methode hoeft niet `static` te zijn, omdat we eerst een object gaan maken van de klasse Dobbelsteen. Dat gebeurt in de `main` methode van de `ConsoleApp` klasse. Een object gemaakt op basis van een klasse heet een instantie van de klasse.

Code in main

- `Dobbelsteen dobbelsteen = new Dobbelsteen();`
  Op deze regel worden drie dingen tegelijk gedaan:
  1. `Dobbelsteen dobbelsteen` - Er wordt een variabele met de naam dobbelsteen gemaakt van het type Dobbelsteen (kan een Dobbelsteen object bevatten).
  2. `new Dobbelsteen()` - Er wordt een nieuwe instantie van de klasse Dobbelsteen gemaakt, het resultaat is een Dobbelsteen-object.

3. `...` = `...` - Het nieuwe Dobbelsteen-object wordt opgeslagen in de dobbelsteen variabele

- `int dobbel` maakt een integer variabele met de naam dobbel waar we later de waarde van de gegooide dobbelsteen in kunnen opslaan.
- `dobbel = dobbelsteen.gooi()` roept de methode gooi van het object dobbelsteen aan en zet het resultaat in de variabele dobbel.

Oefeningen

Oefening 8

Maak een project met een start klasse als in het voorbeeld. Maak een tweede klasse aan met de naam Test. In de klasse test maak je een methode `containsNumbers` die een String als argument mee krijgt en test of er in de invoer getallen voorkomen. Het resultaat van de test is een `boolean`.

Oefening 9

Voeg aan de klasse Test van de vorige oefening een methode `getNumbers` toe die een String als argument mee krijgt en alleen de getallen uit de invoer als resultaat geeft. Het resultaat van de methode is een String.

Oefening 10

Voeg aan de klasse Test van oefening 8 een methode `countDecimals` toe die een double als argument mee krijgt en als resultaat geeft hoeveel getallen achter de komma (punt) staan. Het resultaat is een integer.

Scope van een variabele/methode

Bij het maken van een variabele of een methode geef je aan wat de scope is. De scope bepaald waar je de variabele of methode kunt gebruiken. Dit doe je door de gereserveerde woorden `public`, `private` of `protected` er voor te zetten. Wat `protected` betekent komt later als je meer leert over klassen en objecten.

Behalve de declaratie met public of private wordt de scope van een variabele ook bepaald door de plek waar je de variabele maakt.

```
public class Main {
   public static void main(String[] args) {
      Scope scope = new Scope();
      System.out.println(scope.string1);
      System.out.println();
      System.out.println(scope.getStringsFromVariables());
      System.out.println();
      System.out.println(scope.getStringsFromMethods());
      System.out.println();
      System.out.println(scope.getOtherString());
   }
}
```

```
public class Scope {

   public String string1 = "Een tekenreeks in scope";
   private String string2 = "Nog een  tekenreeks in scope";

   private String getString(int i) {
      if(i == 1) {
         return string1;
      } else {
         return string2;
      }
   }

   public String getStringsFromVariables() {
      return string1 + "\n" + string2;
   }

   public String getStringsFromMethods() {
      return getString(1) + "\n" + getString(2);
   }

   public String getOtherString() {
      String other = "Een andere tekenreeks";
      return other;
   }

}
```

Uitleg

- De variabele `string1` is `public` gedeclareerd en mag binnen de klasse gebruikt worden maar ook in `Main` waar een object van de klasse `Scope` is gemaakt.
- De variabele `srting2` is `private` gedeclareerd en mag binnen de klasse gebruikt worden, in de klasse `Main` is deze variabele niet beschikbaar.
- De variabele `other` is gedeclareerd binnen het blok van de methode `getOtherString()` en is alleen daar te gebruiken. Voor elke variabele binnen een methode geldt dat hij alleen te gebruiken is binnen het eerste blok waar hij in staat, dus binnen de accolades die de variabele als eerste omsluiten.
- De methode `getString(int i)` is private gedeclareerd en mag binnen de klasse gebruikt worden, daar buiten, bijvoorbeeld in `Main` is deze methode niet beschikbaar.
- De methoden `getStringsFromVariables()` en `getStringsFromMethods()` zijn `public` gedeclareerd en kunnen binnen de klasse gebruikt worden en zijn ook in `Main` beschikbaar via het object `scope` van de klasse `Scope`.

Regels

- Over het algemeen geldt; maak de scope van een variabele zo klein mogelijk.
- Geef een variabele met een grotere scope een langere beschrijvende naam.

Oefening

```
public class Main {

   public static void main(String[] args) {
      Scope scope = new Scope();
      System.out.println(scope.repeatStringHorizontal("xyz", 10));
      System.out.println();
      System.out.println(scope.repeatStringVertical("xyz", 10));
   }

}
```

```
public class Scope {

   public String repeatStringHorizontal(String repeat, int count) {
      StringBuffer output = new StringBuffer();
      for(int c = 0; c < count ; c++) {
         output.append(repeat);
      }
      return output.toString();
   }

   public String repeatStringVertical(String repeat, int count) {
      StringBuffer output = new StringBuffer();
      for(int c = 0; c < count ; c++) {
         output.append(repeat);
         output.append("\n");
      }
      return output.toString();
   }

}
```

Neem het voorbeeld over en test de code. Verplaats de `StringBuffer output = new StringBuffer()` naar boven in de klasse en verwijder hem uit de andere methode. Kijk wat het resultaat is.

Recursie

Recursie is een truc om code van met name problemen met een wiskundige achtergrond simpeler te maken. Bij recursie wordt in een methode dezelfde methode weer aangeroepen, maar dan met een andere parameter. Dit klinkt ingewikkeld, dus bespreken we een voorbeeld.

Voorbeeld: vermenigvuldigen van twee getallen

Vermenigvuldigen is iets wat je al op de basisschool hebt geleerd. Nu gaan we er iets wiskundiger naar kijken. We stappen even bewust er over heen, dat vermenigvuldigen in een programmeertaal heel gemakkelijk is (er zijn trouwens nog steeds programmeertalen die niet kunnen vermenigvuldigen, dit heeft meestal met de simpelheid van het platform te maken)

Stel je voor: je wilt 4*3 berekenen, we weten allemaal dat de uitkomst 12 is, maar bereken je dat?

Uitkomst = 4 + 4 +4

Dus: je neemt 4 (1e keer) en dan tel je daar 4 bij op (2e keer) en daar bij tel je nog een keer 4 op. Je hebt dus 3 vieren bij elkaar opgeteld. Hieronder vind je de voorbeeld code:

```
public class Vermenigvuldiging {

    public int vermenigvuldig(int aantalkeer, int getal) {
        if (aantalkeer == 0) {
            return 0;
        } else if (aantalkeer == 1) {
            return getal;
        } else {
            return vermenigvuldig(aantalkeer-1, getal) + getal;
        }
    }
}
```

Uitleg

Stel je roept deze methode aan met `vermenigvuldig(0, 3)`:

1. aantalkeer is dan 0.
2. De methode keert dan terug met 0. Immers 0*3= 0

Daarna proberen we een aanroep met `vermenigvuldig(1, 3)`:

1. aantalkeer is 1
2. De methode keert dan terug met 3, zoals we geleerd hebben bij rekenen: 1*3= 3

Tot slot de laatste mogelijkheid, de aanroep met `vermenigvuldig(2, 3)`:

1. aantalkeer is 2
2. De methode roept zichzelf aan met aantalkeer – 1 (2-1=1)
3. In de tweede ronde geldt dus aantalkeer is 1
4. De tweede aanroep keert terug met antwoord 3
5. Daar tellen we 3 bij op (3 + 3 = 6)
6. De eerste aanroep keert terug met antwoord 6

Niet oneindig doorgaan

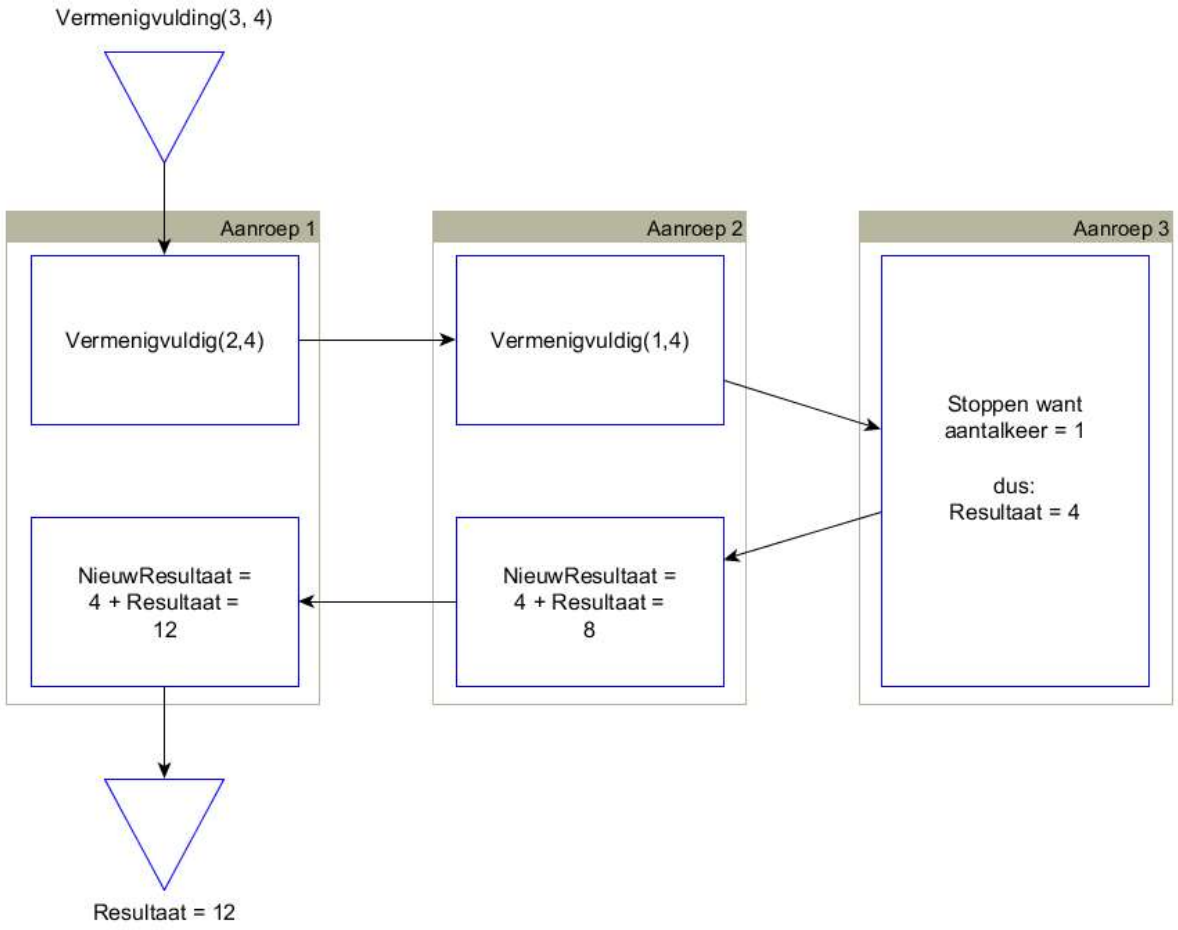
Een van de belangrijkste eigenschappen van een recursieve methode is dat er ook een stop-conditie in zit. Want anders zou het kunnen zijn dat je oneindig doorgaat, omdat je steeds weer dezelfde methode aanroept die zichzelf aanroept die zichzelf aanroept….

Uiteindelijk het programma of erger je computer vastloopt. In bovenstaande code wordt dit geregeld door de regels

```
if (aantalkeer == 0) {
    return 0;
} else if (aantalkeer == 1) {
    return getal;
```

We weten dat er elke keer -1 gedaan wordt, en dus zul je een keer bij aantalkeer = 1 aankomen, en dan springt de methode weer terug naar buiten. En aantalkeer = 0 zorgt ervoor dat het goed gaat als je met 0 begint.

Schematisch gezien loopt het programma als volgt:



Oefening

Neem de code hieronder over en bedenk nog minimaal 3 vermenigvuldigen om te controleren of het klopt.

Zet hiervoor in de klasse **Vermenigvuldiging** een paar System.out.println opdrachten zodat je kunt zien waar het programma langs loopt.

```
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Vermenigvuldiging v = new Vermenigvuldiging();

        System.out.println("4 * 3 = "+Integer.toString(v.vermenigvuldig(4, 3)));
        System.out.println("6 * 5 = "+Integer.toString(v.vermenigvuldig(6, 5)));
    }

}
```

Voorbeeld 2: Faculteit berekenen

Met dezelfde manier kun je ook een faculteit berekenen (zie ook de bronnen)

- 0! = 1
- 1! = 1
- 2! = 1 *2
- 3! = 1 * 2 *3
- 4!= 1 * 2 * 3 *4
- Enzovoort

Oefening

Werk dit ook eens uit in Java net zoals het voorbeeld hierboven.

Eindopdrachten

Je kunt de code voor de eindopdrachten ophalen op https://github.com/DrentheCollege/A2Eindopdrachten.git met een **git clone** opdracht.

Opdracht 1: Formules uitrekenen

Maak een recursieve functie die berekeningen kan doen.

De eisen:

- We volgen de standaard rekenregels
  - Dus binnen de haakjes eerst uitrekenen
  - Eerst vermenigvuldigen en daarna pas optellen en aftrekken
- We gebruiken gehele getallen (int)

Dus bijvoorbeeld:

4 + 3 =7

4 *3 + 2 = 12 + 2 = 14

(3+4)*3 = 7 * 3 = 21

((2 +1) * 4) + (2 * 3) = (3 * 4) + (2 * 3) = 12 + 6 = 18

Dat **moet** je oplossen met een recursieve methode (definitie: **public int bereken(String input)**, en vergeet niet om het resultaat steeds terug te geven. (**return**)
Je kunt dit controleren met bovenstaande sommen (gebruik die bijvoorbeeld in je code), maar bedenk er zelf ook 3 en controleer het resultaat.

```
public class Opdracht1 {
    public static void main(String[] args){
        String formule = "4+3";
        Integer resultaat = FormuleBereken.bereken(formule);
        System.out.println(formule + " = "+resultaat);

        formule = "4 *3 + 2";
        resultaat = FormuleBereken.bereken(formule);
        System.out.println(formule + " = "+resultaat);

        /* hier kun je de andere formules ook nog uitwerken en
           je eigen formules toevoegen
        */
    }
}

class FormuleBereken {

    static Integer bereken(String formule) {
        return 0;
    }
}
```

Opdracht 2: String van karakters opruimen

In deze opdracht krijg je een stuk code dat niet af is. Bedenk hier een manier om alle niet wenselijke karakters uit de string te halen. Maak een nieuwe klasse die dit gaat doen, en gebruik zoveel mogelijk methodes.

Tekst.replaceAll is niet toegestaan, het is de bedoeling dat je zelf het programma bedenkt en schrijft.

```
public class Opdracht2 {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        char[] weg = {'"', '?'};

       String tekst =
                "Dit is een tekst met \" en ** en ?? "+
                "en allerlei andere niet wenselijke tekens zoals ® etc.  ";

       System.out.println(tekst);
    }

}
```

Bronnen

- ✓ **Uitleg faculteit**
  http://www.megawetenschap.nl/faculteit.html