Practicum OO – lente 2016 **Opgave 1**

Leerdoelen

Na het maken van deze opgave kun je

- een Java programma schrijven, compileren en uitvoeren;
- eenvoudige klassen en objecten maken en gebruiken;
- werken met statische en dynamische methoden;
- basis input-output in Java gebruiken;
- werken met rijen in Java;
- basis javadoc gebruiken.

Programmeeromgeving

We moedigen je aan om in dit practicum versie 8.0 van **NetBeans** te gebruiken. Standaard werkt deze programmeeromgeving met Java 8, in een aantal gevallen zullen we gebruik maken van specifieke eigenschappen van deze versie van Java. De programmeeromgeving NetBeans hoort op alle studenten pc's beschikbaar te zijn en is gratis te downloaden van `netbeans.org`. De Java SE versie is mooi genoeg. Bij sommige opdrachten is een deel van de code voorgegeven. We testen de voorgegeven code steeds uit in deze versie van NetBeans. Hetzelfde gebeurt ook met de door jullie ingeleverde uitwerkingen. Als je wil mag je een andere programmeeromgeving gebruiken, maar eventuele problemen die dat oplevert zijn voor eigen risico.

Javadoc

Alle uitwerkingen van opgaven die je in het practicum van het vak Object-Oriëntatie maakt moeten voorzien worden van basis javadoc:

- naam van beide auteurs in iedere file/klasse;
- korte beschrijving van het doel van de klasse in iedere klasse. In het ideale geval kies je een korte naam voor de klasse die precies het doel beschrijft, meestal lukt het niet zoń naam te vinden en heb je dus wat toelichting nodig;
- korte beschrijving van het doel van iedere methode. Indien niet duidelijk uit de naam van de methode en/of argumenten, het doel van deze methode en argumenten, inclusief eventuele precondities.

Zie ook de aanwijzingen voor de te volgen programmeerstijl op Blackboard. Om je te laten kennismaken met Java klassen en objecten vragen we je in deze opgave een aantal klassen en objecten te maken.

1 Een hoofdklasse voor deze opgave

Maak in NetBeans een nieuw `java project` aan. De programmeeromgeving zal standaard een hoofdklasse met een lege methode **public** **static** **void** `main(String[ ] args)` voor je maken. Als je dit project gaat uitvoeren wordt deze `main` methode uitgevoerd. De *access modifier* **public** zegt dat alle klassen deze methode `main` kunnen gebruiken. Het keyword **static** zegt dat dit een bijzondere methode is: de methode hoort bij de klasse ('normale methoden', d.w.z. niet-statische methoden, horen bij een object). De `main` moet wel **static** zijn omdat er op het moment dat het programma gestart wordt nog helemaal geen objecten zijn. Het type **void** geeft aan dat deze methode niets oplevert. Je kent dit type uit C++. Het argument van deze methode is een rij met strings: de argumenten die de gebruiker meegeeft bij het

starten van het programma. Voorlopig hoeven we daar niet naar te kijken. Als je het Java programma start vanuit Netbeans zullen er normaal gesproken geen argumenten zijn. Je hoeft er dan ook niets mee te doen.

2 De klasse Student

De klasse `Student` heeft drie attributen: de strings voornaam en achternaam, en een integer voor het studentnummer. De constructor van deze klasse krijgt deze attributen mee als argument. De constructor moet de attributen een waarde geven op basis van die argumenten. Dat wil zeggen dat je een constructor met type `Student(String voor, String achter, int studentNummer)` maakt. Met deze constructor kun je een object van de klasse Student maken als

```
Student bob = new Student("Bob", "De Vries", 7);
```

In Java kun je een klasse zo veel constructoren geven als je nodig hebt. Als al deze constructoren maar verschillen in het type van de argumenten zal Java steeds die ene constructor kiezen die past.

Voeg aan de klasse `Student` een methode `String toString()` toe. Deze methode levert een string met daarin de voornaam, achternaam en het studentnummer gescheiden door spaties. Voor het studentnummer dient ook nog de letter `s` te staan[1].

De `toString` methode kunnen we gebruiken om informatie over een student af te drukken. Bijvoorbeeld:

```
System.out.println("De student: " + bob.toString());
```

of korter

```
System.out.println("De student: " + bob);
```

In dit laatste geval zal Java zelf zien dat de `toString` methode nodig is en de aanroep toevoegen.

Geef deze klasse ook een methode `setNaam(String voor, String achter)` die gebruikt kan worden om een `Student` object een nieuwe naam te geven. We kunnen deze gebruiken als:

```
bob.setNaam("Bob", "De Bouwer");
```

Merk op dat we altijd eerst de naam van een object gebruiken gevolgd door de naam van de methode. Anders is het niet duidelijk op welk object deze methode betrekking heeft. Alleen als je (hulp)methoden uit de eigen klasse gebruikt hoef je de naam van het object niet steeds op te geven. Als je in dat geval toch een object wil opgeven kun je daarvoor altijd **this** gebruiken.

Het is de bedoeling dat iedere klasse in Java in een aparte file staat (met extensie `.java`) waarvan de naam hetzelfde is als die van die klasse. De klasse `Student` wordt dus opgeslagen in het bestand `Student.java`. Als je in NetBeans een nieuwe klasse maakt zal NetBeans zelf de nieuwe file maken en die aan het huidige project toevoegen.

Test de klasse `Student` door enkele instanties van de klasse te maken (student-objecten) en af te drukken. Verander ook de naam van een student-object en druk dit opnieuw af.

3 De klasse Groep

De klasse `Groep` bevat een rijtje van studenten. Bij het maken van een object van deze klasse geven we de grootte van dit rijtje op. Dat wil zeggen dat we een constructor hebben van de vorm `Groep ( int aantal)`. Een nieuw gemaakte `Groep` bevat nog geen studenten.

Als je in Java een rij-object aan willen maken, gebruik je een speciale constructor waarin de grootte van de rij is opgegeven.

```
Student [ ] studenten = new Student [n];
```

De grootte van een rij ligt vast op het moment dat we die maken en kan niet veranderd worden. We kunnen wel aan `studenten` een nieuwe rij toekennen met een andere lengte.

[1] Je programmeeromgeving zal je er mogelijk op wijzen dat er al zo'n methode is en dat deze definitie de bestaande definitie zal vervangen. Netbeans zal voorstellen om de `@Override` annotatie toe te voegen. Neem dat voor kennisgeving aan; we zullen dat spoedig uitleggen.

Aan een `groep`-object kunnen we met de methode **boolean** `voegToe(Student s)` kunnen studenten toevoegen. Als de groep al vol is kan de student natuurlijk niet meer echt toegevoegd worden. Het resultaat van deze methode geeft aan of het toevoegen van de student gelukt is.

Geef deze klasse een methode `getStudent(`**int** `i)` die de i^e student uit de groep oplevert als die bestaat. Als deze student er niet is dient de methode **null** op te leveren. Dit is de Java manier om aan te geven dat er geen object is.

Geef ook deze klasse een eigen methode `toString` die alle studenten uit de groep in één string zet gescheiden door newlines. Als je deze string afdrukt komt ieder student uit de groep op een nieuwe regel.

4 De hoofdklasse

Tot slot vullen we de hoofdklasse van dit project. In de constructor van deze klasse moet het eigenlijke werk gebeuren. De eerder genoemde `main` methode hoeft (en mag) dan niet meer doen dan een object van deze hoofdklasse aan te maken.

Je programma moet het volgende doen:

1. Het programma vraagt aan de gebruiker hoe groot de groep studenten moet worden en maakt vervolgens een groep van die grootte.
2. Het programma vraagt naam en studentnummer voor alle leden van de groep en voegt die toe aan de groep totdat de groep vol is.
3. Vervolgens drukt het programma de tekst `De groep bevat nu:` gevolgd door alle studenten uit de groep af.
4. In een herhaling vraagt het programma vervolgens naar het lidnummer in de groep en een nieuwe naam voor die student: `Lidnummer en nieuwe voor- en achternaam?`. Nadat de groep is veranderd zal deze weer in zijn geheel worden afgedrukt vooraf gegaan door `De groep bevat nu:`.

   Het lidnummer van een student is haar volgnummer in de groep, dit is dus niet het unieke studentnummer van die student.
5. Het programma stopt zodra de gebruiker een negatief lidnummer opgeeft.

Invoer lezen gaat in Java meestal het handigst met een `Scanner` object.

```
Scanner scanner = new Scanner (System.in);
```

De klasse `Scanner` importeer je uit `java.util.Scanner`[2].

Uit deze scanner kun je een integer lezen met:

```
int aantal = scanner.nextInt();
```

Het lezen van strings kan een beetje lastiger zijn omdat niet altijd duidelijk is wanneer het lezen moet stoppen. Je kunt dat aangeven met patronen voor de scanner, maar dat hoeft niet. De simpelste manier om een string te lezen is:

```
String naam = scanner.nextLine();
```

Dit leest de hele regel tekst van de invoer.

Omdat er op twee plaatsen de tekst `De groep bevat nu:` gevolgd door alle leden van de groep wordt afgedrukt, dien je hier een hulpmethode voor te definiëren. Omdat deze hulpmethode voor intern gebruik bedoeld is, dient hij niet **public** te zijn. Een lokale methode heeft direct toegang tot alle attributen van de klasse, deze hulpmethode moet de groep dus niet als argument krijgen.

[2] Je kunt in NetBeans ook `Ctrl+Shift+I` intikken; NetBeans zal dan zelf de de benodigde imports toevoegen.

Inleveren

Lever **vóór zondag 7 februari, 23:59 uur, via Blackboard** je code in. Vergeet niet om **in** elke file van je uitwerking duidelijk je naam en die van je partner te vermelden als javadoc.

Lever alle de `.java` bestanden in die je zelf gemaakt of aangepast hebt. Als je met NetBeans een project maakt zet die in de `src` directory van dat project een map met alle `.java` bestanden uit het package. Maak een zip-file van de `.java` bestanden en lever die in als er meer dan een project is. Lever bij de programmeeropgaven uit dit practicum niet meer of minder in dan de `.java` bestanden. Als er meer dan een directory is moet je middels een zip file zorgen dat de deze directory structuur behouden blijft. Let op: deze map krijgt de zelfde naam als de map voor het hele project, dat hele project moet je dus *niet* inleveren. Geef de hoofdklasse een herkenbare naam.