

Labosessie 1

UML-diagrammen en Java

Theorie

Theorie die je vóór deze labosessie moet verwerkt hebben, heb je gezien in het hoorcollege van 29 september.

Doel

In dit labo verkennen we de *integrated development environment* (IDE) **BlueJ** en leren we een **eerste Java-project** opstarten. We oefenen ook op het gebruik van *Unified Modeling Language* (UML).

Formaat

Elke labosessie is onderverdeeld in drie verschillende delen: startoefeningen, labo-oefeningen en thuisoefeningen. **Startoefeningen** zijn oefeningen waarvan we verwachten dat je ze thuis oplost, vóór de labosessie. Op die manier geraak je vertrouwd met de materie en check je of je de theorie van de hoorcolleges goed begrepen en verwerkt hebt. **Labo-oefeningen** zijn oefeningen die tijdens de labosessie worden behandeld. **Thuisoefeningen** zijn oefeningen die dieper ingaan op de materie. Ze zijn optioneel voor wie sneller werkt, of als extra oefeningmateriaal bij het studeren/oefenen thuis. Neem contact op met jouw labo-assistent voor feedback op deze oefeningen.

Leerdoelstellingen

Na deze sessie moet je in staat zijn om de volgende concepten te begrijpen en te implementeren:

- Een eerste UML-diagram creëren en interpreteren
- Klassen en methoden schrijven
- Objecten creëren en eenvoudige taken op uitvoeren
- Verschillende primitieve datatypes gebruiken






Startoefeningen

Oefening 0: Installeer BlueJ

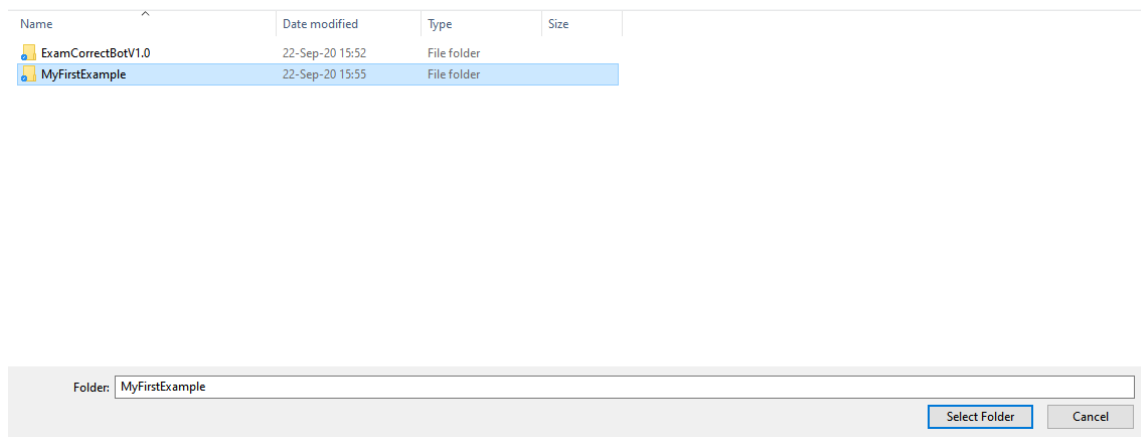
Download en installeer BlueJ vanop <https://bluej.org/>.

Oefening 1: Hello World

Download het zip-bestand "MyFirstExample.zip" van Toledo. Unzip dit bestand (dus niet gewoon openen!). Je zou de volgende inhoud moeten zien:

 MyFirstExample.class	22-Sep-20 15:54	CLASS File	1 KB
 MyFirstExample.ctxt	22-Sep-20 15:54	CTXT File	1 KB
 MyFirstExample.java	22-Sep-20 15:54	JAVA File	1 KB
 package.bluej	22-Sep-20 15:54	BlueJ Project File	1 KB
 README.TXT	22-Sep-20 15:53	Text Document	1 KB

Open vervolgens BlueJ, klik op Project → Open Project. Selecteer de map MyFirstExample. Ga niet *in* de map, *selecteer* de map en klik op "Select Folder". Dit zou er min of meer als volgt moeten uitzien:

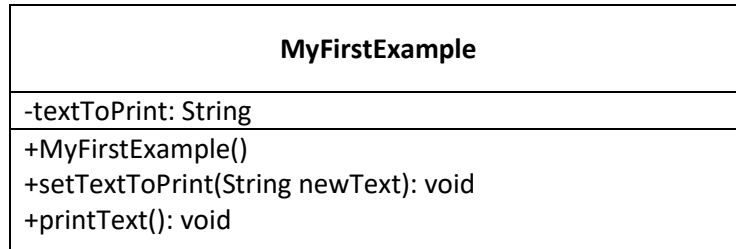


Dubbeltklik op de klasse MyFirstExample en inspecteer de code. Probeer te beredeneren wat elke lijn code doet. Sluit de broncode en maak volgende opdrachten.

1. Maak een object van de klasse MyFirstExample. Klik hiervoor rechts op de klasse en kies MyFirstExample(). Noem het myFirstObject. Je zou een rood object, met naam myFirstObject, moeten zien verschijnen in de "object bench" onderaan.
2. Voer de methode printText() uit door rechts te klikken op het object.
3. Verander de te printen tekst door gebruik te maken van setTextToPrint(). Verander de tekst in "Hello World!". Hoe heet de variabele waar je de nieuwe string in opgeslagen hebt?
4. Print de aangepaste tekst.

Oefening 2: UML-diagrammen analyseren

Dit is het UML-diagram van de klasse MyFirstExample:



Beantwoord de volgende vragen.

1. Welk *keyword* in Javacode stemt overeen met het minteken voor de variabele textToPrint?
2. Wat is de technische term voor MyFirstExample()?
3. Wat is de naam van een variabele zoals newText?
4. Wat is de rol van het *keyword* "void"?

Oefening 3: Instanties

Kijk opnieuw naar het UML-diagram van MyFirstExample. Je kan zien dat er één instantievariabele (ook *veld* genaamd) is, genaamd textToPrint. De eigenlijke waarde van deze variabele kan verschillen tussen verschillende objecten (instanties) van dezelfde klasse. Doe het volgende:

1. Maak twee objecten van de klasse MyFirstExample.
2. Kijk naar de waarde van textToPrint bij beide objecten (klik rechts op object → inspect).
3. Pas de methode setTextToPrint() toe op één van de twee objecten om de te printen tekst aan te passen.
4. Inspecteer beide objecten. Is de waarde van de variabele voor beide veranderd?

Oefening 4: Datatypes

Wat is het datatype van deze waarden?

Waarde	Datatype
56	
'm'	
2 ³⁰	
"410"	
"Hello World"	
true	
0.045	

Labo-oefeningen

Oefening 1: Methodes zelf schrijven

Open de klasse `MyFirstExample`. We zullen hierin twee methodes toevoegen. De nieuwe methodes zijn aangeduid in **vet**.

MyFirstExample
-textToPrint: String
+ MyFirstExample() + setTextToPrint(String newText): void + printText(): void + printSum(int a, int b): void + prettyPrintSubtraction(int a, int b): void

1. `printSum` berekent de som van twee parameters `a` en `b` en print het resultaat. Schrijf deze methode en test je code door een instantie aan te maken en de methode op te roepen. Vergeet niet op "compile" te klikken om je code te compileren nadat je ze geschreven hebt.
2. `prettyPrintSubtraction` trekt `a` van `b` af en print het resultaat in een zinnetje, bijvoorbeeld: "5 min 2 is gelijk aan 3". Maak opnieuw een object aan en test je code.

Oefening 2: Zelf een klasse maken

Tijd om zelf een klasse aan te maken. Het UML-diagram van de beoogde klasse vind je hieronder:

GroupTStudent
-name: String -yearsAtGroupT: int
+ GroupTStudent (String studentName) + addYear(): void + introduceMyself(): void

Voer volgende stappen uit:

1. Maak een klasse door rechts te klikken in het lege canvas en "New Class" te kiezen. Geef de klasse de correcte naam (hoofdletter!) en trek je niets aan van de andere instellingen. Open de klasse.
2. BlueJ zet automatisch een aantal velden en methoden. Analyseer wat deze methoden en variabelen doen. Je mag deze verwijderen.
3. Voeg velden `name` en `yearsAtGroupT` toe.
4. Voeg een parameter `studentName` toe in de constructor.
5. In de constructor doe je het volgende:
 - a. De parameter `studentName` opslaan in het veld `name`;
 - b. `yearsAtGroupT` initialiseren op 0.

6. Compileer je code en maak een object aan van de klasse GroupTStudent. Inspecteer het object om te controleren of alles is zoals het zou moeten zijn. Het is een goed idee om af en toe te compileren en testen als je nieuwe code schrijft, op die manier detecteer je sneller fouten.
7. Schrijf de methode addYear(). Deze methode verhoogt yearsAtGroupT met 1. Compileer, creëer object, test.
8. Schrijf de methode introduceMyself(). Deze methode print een zinnetje zoals "Mijn naam is ... en ik studeer al ... jaar op GroepT". Compileer, creëer object, test.

Oefening 4: UML-diagram zelf ontwerpen

Stel een gedetailleerd klassediagram op, op basis van volgende probleembeschrijving. Definieer de benodigde klasse, haar attributen (naam + type) en signatuur van de methodes.

Probleemomschrijving:

Men wil een systeem ontwerpen om vakken op Groep T op te slaan in een softwaresysteem, om op die manier gemakkelijker de Toledopagina van dit vak te kunnen beheren. Een vak heeft een naam en een aantal studiepunten. Vakken krijgen ook een code die je wil opslaan: b, m of p (deze staan voor respectievelijk bachelor-, master- of postgraduaatsvak). Je moet ook kunnen aangeven of het vak in Toledo al dan niet zichtbaar is voor studenten. Je moet al deze informatie van het vak kunnen uitprinten. Naam, studiepunten en code liggen vast, maar of het vak al dan niet zichtbaar is, kan aangepast worden.

Oefening 5: Temperatuurconversie

Om het chemielabo te helpen in het converteren van temperaturen, maken we een omzetter. Temperaturen kunnen van Celsius naar Fahrenheit omgezet worden, of omgekeerd. De formule voor die omzetting staat hieronder.

$$^{\circ}\text{C} = 5 * (^{\circ}\text{F} - 32) / 9$$

$$^{\circ}\text{F} = 9 * (^{\circ}\text{C}) / 5 + 32$$

1. Maak een nieuw project en een klasse TemperatureConverter.
2. Definieer een veld convertsTo van type char.
3. Schrijf een constructor voor de klasse die een character als parameter accepteert. Initieer convertsTo met deze waarde. De gebruiker maakt dus een converter aan en geeft aan naar welke eenheid deze converter zal omrekenen: 'C' of 'F'.
 - a. Bonus: maak de constructor "fool proof": als de gebruiker een waarde ingeeft die verschillend is van deze twee, kies je een standaardwaarde (bijvoorbeeld 'C').
4. Schrijf een methode convert() die een float accepteert als parameter. Dit is de waarde die geconverteerd moet worden. Afhankelijk van de waarde van het veld convertsTo, gebruik je de juiste formule. Het resultaat wordt in een zinnetje geprint, bijvoorbeeld: "1832 graden Fahrenheit is 1000 graden Celsius".

5. Maak een tweede klasse: `GeneralTemperatureConverter`. Schrijf hierin een methode `convert()` die twee parameters accepteert: een float die de te converteren waarde voorstelt, en een character, de eenheid van deze waarde. De methode doet de juiste conversie en print het resultaat.
- Waarom houdt het nu weinig steek om die eenheid op te slaan in een veld?
 - Bonus: maak de methode opnieuw "fool proof": als de gebruiker een verkeerde waarde ingeeft, wordt een foutboodschap geprint.

Je kan voor deze oefening de volgende waarden gebruiken als testcasus:

Celsius	Fahrenheit
-273.15	-459.67
-30	-22
0	32
30	86
100	212
1000	1832

Thuisoefeningen

Oefening 1: Temperatuurconversie bis

Herschrijf je code van hoger om ervoor te zorgen dat te koude waarden verworpen worden. Zorg ervoor dat temperaturen lager dan 0K (absoluut nulpunt) tot een foutmelding leiden. Omzetting naar Kelvin gebeurt zo:

$$T \text{ (in } ^\circ\text{C)} + 273.15 = T \text{ (in K)}$$

$$T \text{ (in K)} - 273.15 = T \text{ (in } ^\circ\text{C)}$$

Oefening 2: Toegang film

Gebaseerd op: <https://edabit.com/challenge/MvBYeGybFo4iEpWyp>

Maak een klasse MovieAdmissions en implementeer een methode die controleert of een person een 15+-film mag bekijken. Eén van volgende voorwaarden moet in dat geval voldaan zijn:

- De persoon is minstens 15 jaar oud;
- Of de persoon heeft zijn of haar ouders bij.

De signatuur van de methode is als volgt:

+ acceptIntoMovie (int age, Boolean supervised): void

Testoplossingen met geprinte output

acceptIntoMovie(14, true) → kom maar binnen

acceptIntoMovie(14, false) → BUITEN!

acceptIntoMovie(16, false) → kom maar binnen

acceptIntoMovie(16, true) → kom maar binnen

acceptIntoMovie(15, true) → kom maar binnen

acceptIntoMovie(15, false) → kom maar binnen

Oefening 3: Implementatie UML-diagram

Implementeer het UML-diagram dat je hebt opgesteld in oefening 4 van de labo-oefeningen.