

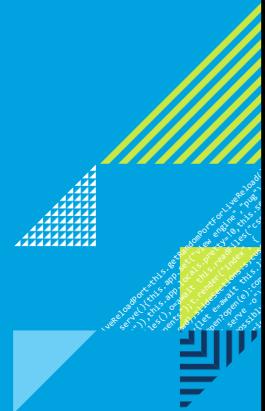
► Introduction to programming with Java

► Agenda

- Introduction
- Create new project in IntelliJ
- Working with Junit
- Types and variables
- Conditional statements
- Loop statements
- Methods
- Debugging
- Arrays
- Exception handling
- Objects



Introduction



► Who am I?

- What's your name?
- What's your background?
- What's your function?
- Do you have any experience in programming?
- How used are you to classes?
- What do you hope to achieve in this course?

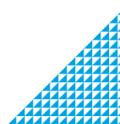
► What is a computer?

- Any ideas?
- Write your ideas on paper



► What is programming?

- Write some ideas on paper



► Main parts of a computer

- processor
- memory
 - short term (RAM)
 - long term (HDD, SSD, ...)
- interfaces
 - keyboard
 - mouse
 - screen
 - network
 - ...



Getting started

- Create new project in IntelliJ
- Working with JUnit



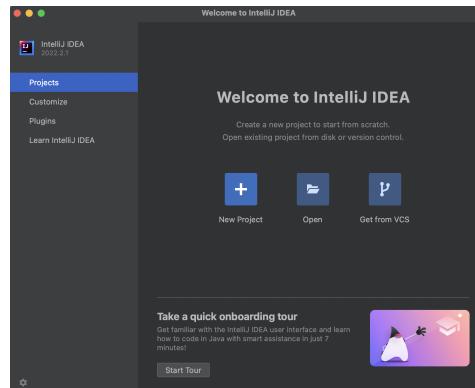
► Working with IntelliJ IDE

- IntelliJ is the Rolls Royce among the IDE's
 - Integrated Development Environment
 - large application to help you creating programs
 - very versatile
 - supports different project formats and languages
 - plugins

► Maven

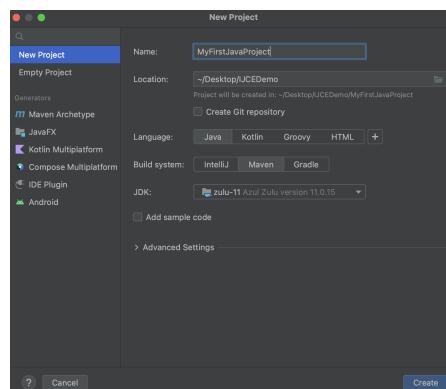
- Popular java project structure
 - dependency management and build tool
- Supported in IntelliJ
 - It has some recurring configuration
 - For now: copy and paste from the following slides!

► Start IntelliJ



➤ Choose New Project

► Enter details



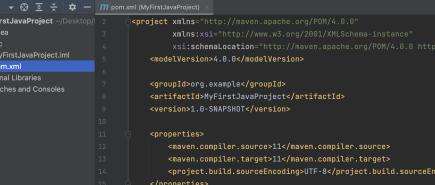
▶ Maven project created...

The screenshot shows the IntelliJ IDEA IDE with the following details:

- Project Structure:** The "MyFirstJavaProject" project is selected.
- Pom.xml Content:** The code editor displays the pom.xml file for the project, which includes XML configurations for Maven.
- Toolbars and Menus:** Standard IntelliJ IDEA toolbars and menus are visible at the top.
- SIDE BAR:** The left sidebar shows the project tree, external libraries, and scratch files.
- Bottom Navigation:** Includes tabs for Version Control, Problems, Terminal, Services, Build, and Dependencies.
- Status Bar:** Shows memory usage (1.9 GB / 31.8 GB), build time (31 sec / 855 ms), and a warning about a shared index.

- By default a lot of files/directories are created

This is a mvn project



The screenshot shows the IntelliJ IDEA IDE with the following details:

- Project Structure:** The project is named "MyFirstJavaProject".
- Pom.xml Content:**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/4.0.0/maven-xsd.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>MyFirstJavaProject</artifactId>
  <version>1.0-SNAPSHOT</version>

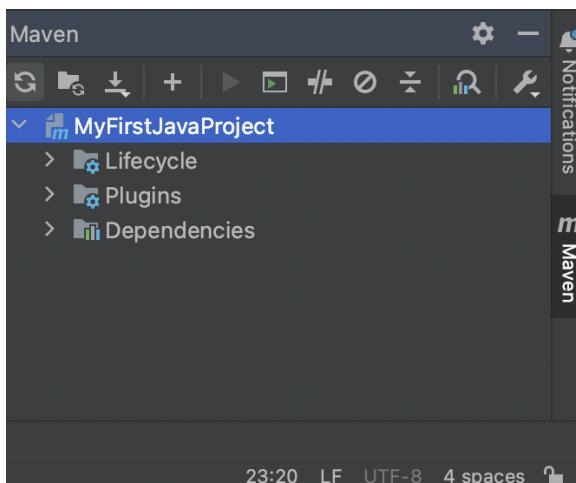
  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter</artifactId>
      <version>5.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

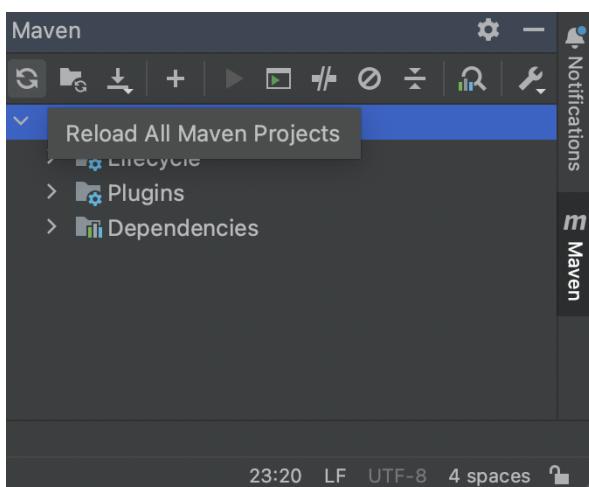
- Toolbars and Menus:** Standard IntelliJ IDEA toolbars and menus like File, Edit, View, Tools, and Help are visible.
- Sidebar:** The left sidebar shows the project tree with "MyFirstJavaProject" expanded, and "src" and "pom.xml" listed under it. It also includes External Libraries, Scratches, and Consoles.
- Bottom Status Bar:** Shows the current file as "pom.xml (MyFirstJavaProject)", the build number as "1000", and other status indicators.

- The pom.xml is the mvn configuration file
 - Add the junit-jupiter dependency

► Open the maven view



► Reload the maven project





Working with Junit

>Create class in src\test\java folder

The screenshot shows the IntelliJ IDEA interface with the following details:

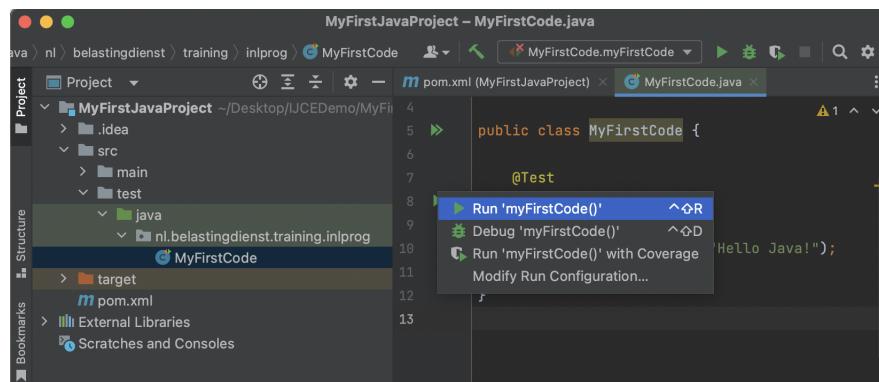
- Title Bar:** MyFirstJavaProject – MyFirstCode.java
- Toolbar:** Includes icons for file operations like Open, Save, and Run.
- Project View (Left):** Shows the project structure with a tree view of files and folders. The 'MyFirstJavaProject' folder contains .idea, src, and target. The src folder has main and test subfolders. The test folder has a java subfolder which contains the 'MyFirstCode' file.
- Editor Area (Right):** Displays the code for 'MyFirstCode.java'. The code is as follows:

```
public class MyFirstCode {  
    @Test  
    void myFirstCode(){  
        //Code goes here  
        System.out.println("Hello Java!");  
    }  
}
```

➤ Copy the code to your IDE

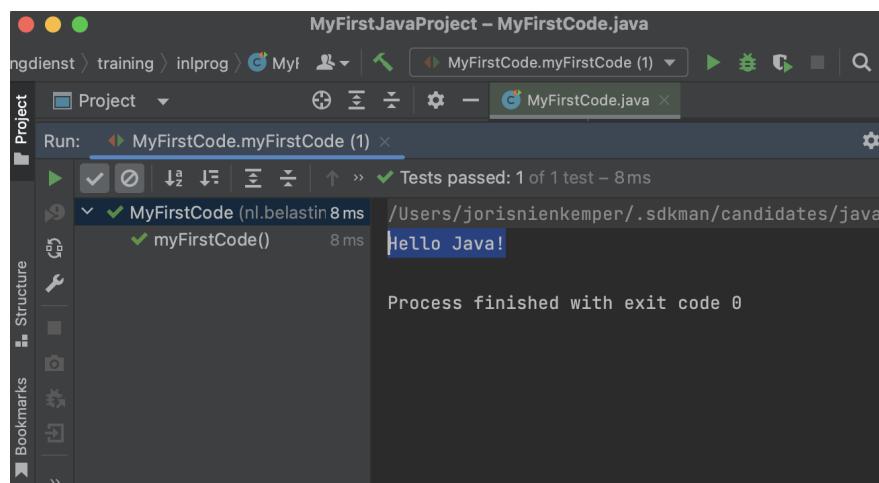
Run the code

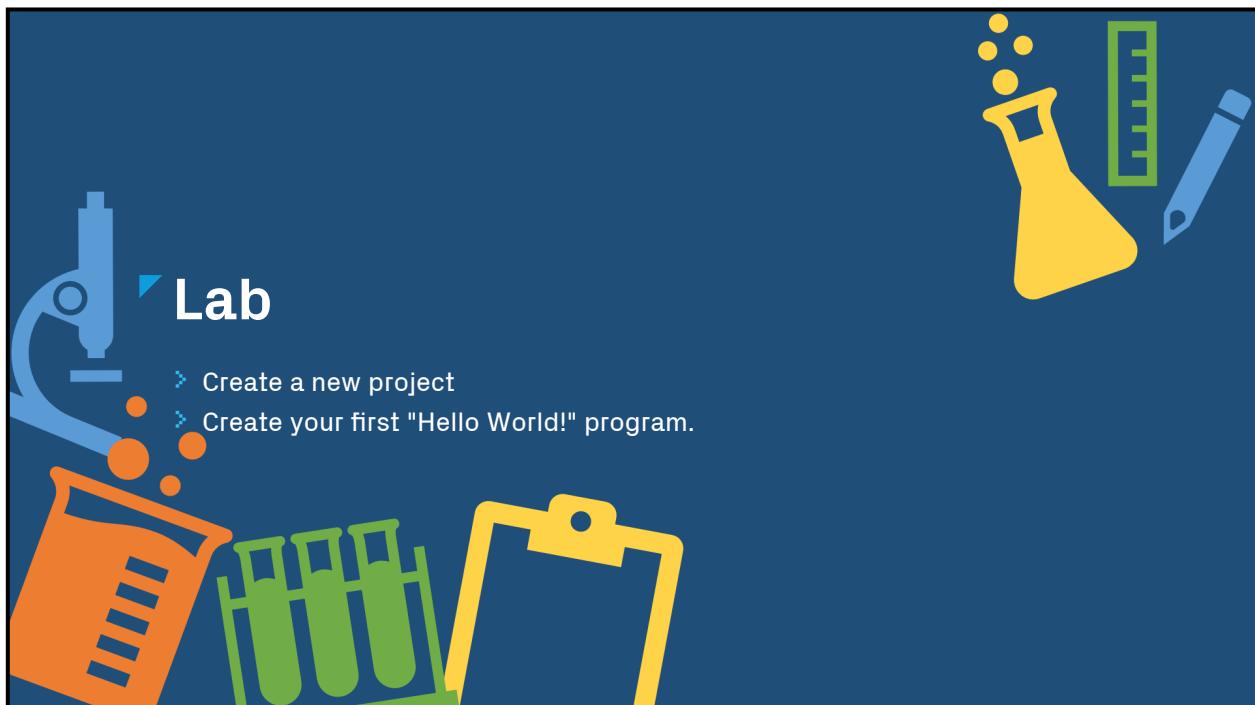
- click on the green arrow in the border



The Run window

Shows the output (if any)





»

Types and variables

► Wat is een variabele?

- een doosje
- is er voor een beperkte tijd
- doosjes specifiek voor bepaalde inhoud
- kunt er iets instoppen/uithalen
- kunt het hergebruiken
- officieel noemen we een doosje een variabele

► Type system Java

- In Java moet je als je een variabele maakt aangeven wat je er in stopt
 - Je geeft het type op van de variabele
- Hoofdtakken
 - primitive types
 - reference types
- Kijken eerst naar primitive types
 - 8 in totaal

► Variabelen

```
package org.example;

import static org.junit.Assert.*;
import org.junit.Test;

public class ExploreJavaPrimitives {

    @Test
    void skeletonToRunCode() {
        //Schrijf hier code
    }
}
```

► Primitive types:

- doosjes in java moeten getypeerd zijn
 - d.w.z. java wil van te voren weten wat je er in wilt stoppen
- we zullen allereerst stilstaan bij de primitive types
- later gaan we naar de reference types kijken

► Kleinst doosje voor gehele getallen

- byte → 8 bits

```
@Test  
void declareByteVariable(){  
    byte temperature;  
    temperature = 1; //8bits 0000 0001  
    System.out.println(temperature);  
}
```

- Is er ook een grootste waarde?
- Wat is de kleinste waarde?

► De grootste byte value

MAX_VALUE

```
@Test  
void declareByteVariable() {  
    byte minimumtemperature;  
  
    minimumtemperature = -128;      // laagste waarde, 8bits 1000 0000  
    System.out.println(minimumtemperature);  
  
    byte maximumtemperature = 127; // hoogste waarde, 8bits 0111 1111  
    System.out.println(maximumtemperature);  
}
```

short

- 2x grotere capaciteit dan byte

```
@Test
void declareShortVariable(){
    short minimumtemperature;
    //16 bits 0000 0000 0000 0001
    minimumtemperature = -32_000;
    //In de buurt van de laagste waarde
    System.out.println(minimumtemperature);

    short maximumtemperature = 32_000;
    //In de buurt van de maximale short value
    System.out.println(maximumtemperature);
    //Homework bereken 2 to the power 15
}
```

int

- 2x grotere capaciteit dan short

```
@Test
void declareIntVariable(){
    int minimumtemperature;
    //32 bits 1000 0000 0000 0000 0000 0000 0000 0000
    minimumtemperature = -2000000000;
    // Al die nullen zijn lastig te lezen
    minimumtemperature = -2_000_000_000;
    //Use underscores in int to indicate thousands
    System.out.println(minimumtemperature);

    int maximumtemperature = 2_000_000_000;
    // In de buurt van MAX_VALUE van int
    System.out.println(maximumtemperature);
    //Homework bereken 2 to the power 31
}
```

► long

- 2x grotere capaciteit dan int

```
@Test
void declareLongVariable(){
    long maximumtemperature; //64 bits
    maximumtemperature = 2_100_100_100_100_000_000L;
    //In de buurt van MAX_VALUE long
    //Hoe spreek je dit getal uit?
    System.out.println(maximumtemperature);
    //Homework bereken 2 to the power 63
}
```

► Variabelen hebben een type en een grootte

- byte/short/int/long
- Je kunt dus een waarde (a.k.a. een literal (2 of 3 of 100 etc.)) in een doosje stoppen
- Die waarde past of past niet
- In het laatste geval klaagt Java!

► Waarom heb je variabelen nodig

- Berekeningen
 - Verzin met de groep een berekening waarbij je gehele getallen gebruikt
- Voor berekeningen heb je operatoren nodig
 - Welke operatoren ken je?
 - Hoe zou je operatoren beschrijven?
 - Wat wordt er bedoeld met operanden?
- Ken je regels waar operatoren aan moeten voldoen?

► Berekening demo 1

```
@Test
void berekenTotaalAantalDeelnemers(){
    int dotNetters = 25;
    int javanen = 12;
    int totaalDeelnemers = dotNetters + javanen;

    System.out.println("Aantal deelnemers " + totaalDeelnemers);
}
```

► Berekening demo 2

```
@Test
void berekenDeDistance(){
    int velocity = 10;
    int time = 5;
    int distance = velocity * time;
    System.out.println("De afgelegde afstand " + distance);
}
```

► Berekening demo 3

```
@Test
void aantalBenodigdeWielen(){
    int auto = 32;
    int wielen = 4;

    // int aantalWielen = auto * wielen;

    System.out.println("Aantal wielen = " + auto * wielen);
}
```

► Berekening demo 4

```
@Test  
void aantalBenodigdeWielenPlusMinimumVoorraad(){  
    int wielen = 32;  
    int autos = 4;  
    int minimumVoorraad = 16;  
    // We verdubbelen het aantal auto's  
    // In de println autos*2  
    // Wat verwacht je?  
    System.out.println((wielen - minimumVoorraad)/autos*2);  
  
    //System.out.println((wielen - minimumVoorraad)/(autos*2));  
}
```

► Bestaat Meneer van Dalen nog?

- Prioriteit van operatoren
- Hoe zit het ook alweer?

► Berekening demo 5

- Wat doet de modulo operator?

```
@Test  
void operationsWithTheModuloOperator(){  
    System.out.println(1 % 2); //1  
    System.out.println(2 % 2); //0  
    System.out.println(3 % 2); //1  
    System.out.println(4 % 2); //0  
    System.out.println(5 % 2); //1  
    System.out.println(6 % 2); //0  
    System.out.println(7 % 2); //1  
}
```

► Berekening demo 6

```
@Test  
void welkePrioriteitHeeftDeModuloOperator() {  
    System.out.println( 3 * 11 % 3);  
    System.out.println((3 * 11) % 3); //0  
    System.out.println( 3 * (11 % 3)); //6  
    System.out.println( 11 % 3 * 3);  
    System.out.println((11 % 3) * 3); //6  
    System.out.println( 11 % (3 * 3)); //2  
}
```

► double

- floatingpoint rekenen
- "approximate numerics"

```
@Test
void declareDoubleVariables(){
    // Handig om te onthouden dat er bij doubles
    // (ook bij floats) vaak sprake is van een afronding
    double price = 1.9447;
    double aantalLiters = 45.68;
    double multiplier = 1.0E2; //1.0 keer 10 tot de macht 2 (100)

    System.out.println("Verkoopprijs = " + price * aantalLiters* multiplier);
}
```

► Het float type

```
@Test
public void byDefaultWordenGebrokenGetallenGezienAlsDouble(){
    float floatDoosje = 1.0F;
    System.out.println(floatDoosje);
}
```

- 1.0 zonder (F) wordt gezien als double → 8 bytes → past niet

► Het boolean type

```
@Test  
public void booleansExperiment() {  
    boolean isDoosjeLeeg, isDoosjeVol;  
    isDoosjeLeeg = false;  
    isDoosjeVol = true;  
    System.out.println(isDoosjeLeeg);  
    System.out.println(isDoosjeVol);  
}
```

- Boolean bevat 2 literal values: `true` en `false`

► Waar gebruiken we boolean variabelen voor?

```
@Test  
public void opslaanVanVergelijkingen() {  
    boolean heeftMinimumLeeftijd;  
    int minimumLeeftijd = 18;  
    int leeftijdPiet = 20;  
    heeftMinimumLeeftijd = leeftijdPiet >= minimumLeeftijd;  
    System.out.println(heeftMinimumLeeftijd);  
    //Waar gebruiken we booleans voor?  
}
```

▼ Vergelijkingen combineren

- Soms moet een conditie aan meerdere voorwaarden voldoen

1. Een persoon moet meerderjarig zijn **EN**
2. de Nederlandse nationaliteit bezitten

▼ Vergelijkingen combineren 2

```
@Test
public void beideVergelijkingenMoetenWaarZijn() {
    boolean heeftMinimumLeeftijd;
    int minimumLeeftijd = 18;
    int leeftijdPiet = 20;
    heeftMinimumLeeftijd = leeftijdPiet >= minimumLeeftijd;
    int minimumInkommen = 20_000;
    int inkomenPiet = 25000;
    boolean isBovenMinimumInkommen = inkomenPiet > minimumInkommen;
    boolean voldoetAanBeideCriteria =
        heeftMinimumLeeftijd && isBovenMinimumInkommen;
    System.out.println(voldoetAanBeideCriteria);
}
```

▼ Vergelijkingen combineren 3

```
@Test
public void eenVanDeVergelijkingenMoetWaarZijn() {
    boolean heeftDagKaart = false;
    boolean heeftMaandKaart = false;
    boolean heeftJaarAbonnement = true;
    boolean heeftGeldigToegangsBewijs =
        heeftDagKaart || heeftMaandKaart || heeftJaarAbonnement;
    System.out.println(heeftGeldigeToegangsBewijs);
}
```

▼ Het char type

```
@Test
public void spelenMetChars() {
    char charDoosje1 = 'a';
    char charDoosje2 = 'b';

    System.out.println(charDoosje1 + charDoosje2);
    System.out.println("dit is een waarde = " + 10);
    System.out.println(charDoosje1 + charDoosje2 + "");
}
```

- De + operator ziet char als getallen (unsigned)
- De expressie wordt van links → rechts geëvalueerd
- "" + 'a' → String

► Resumerend

► 8 primitive types

- gehele getallen mét sign
 - 1. byte 1 byte
 - 2. short 2 bytes
 - 3. int 4 bytes
 - 4. long 8 bytes
- gebroken getallen mét sign
 - 5. float 4 bytes
 - 6. double 8 bytes
- character (getal zónder sign)
 - 7. char 2 bytes
- waar of niet waar
 - 8. boolean

► Lab

Maak de volgende programma's.

1. Bereken je nieuwe maandsalaris na indexatie

- huidige salaris: 3105 euro / maand
- indexatie: 8,42 %

2. Bereken je energienota over 2022 a.d.h.v. de volgende gegevens:

- kosten
 - vast
 - gas: 19,44 / maand
 - stroom: 24,49 / maand
 - variabel
 - gas: 2,37 / m3
 - stroom: 0,895 / kWh
- verbruik
 - gas: 1459 m3
 - stroom: 3448 kWh, teruggeleverd 1200 kWh



Conditional statements

► Basic syntax explored

```
@Test  
void hoeZietDeJavaSyntaxErEigenlijkUit() {  
    // Met { .... } definieren we een code blok  
    System.out.println("Start van code blok");  
  
    {    //blocken kun je ook nesten  
        System.out.println("Start van inner code blok");  
        System.out.println("Einde van inner code blok");  
    }  
    System.out.println("Einde van code blok");  
}
```

► Scope van variabelen

```
void hoeZitHetMetVariabelenEnCodeBlocken() {  
    System.out.println("Start van code blok");  
    boolean outerBlokBoolean = true;  
    System.out.println("Outer block boolean = " + outerBlokBoolean);  
    {  
        System.out.println("Start van inner code blok");  
        boolean innerBlokBoolean = true;  
        System.out.println("Inner block boolean = " + innerBlokBoolean);  
        System.out.println("Outer block boolean = " + outerBlokBoolean);  
        System.out.println("Einde van inner code blok");  
    }  
    //System.out.println("Inner block boolean = " + innerBlokBoolean);  
    System.out.println("Outer block boolean = " + outerBlokBoolean);  
    System.out.println("Einde van code blok");  
}
```

► Scope van een variabele

- variable is zichtbaar binnen het block waarin het gedeclareerd wordt inclusief geneste blocken
- variabelen gedeclareerd in een genest block is buiten dat blok niet zichtbaar
- het block (inclusief geneste blocken) waarbinnen de variabele gebruikt kan worden wordt ook wel de scope van de variabele genoemd

► Een blok wel of niet uitvoeren

```
void hoeKunnenWeEenBlockConditioneelUitvoeren() {  
    System.out.println("Start van code blok");  
  
    boolean innerBlockUitvoeren = false;  
  
    //Het geneste block willen we conditioneel uitvoeren:  
    //if(conditie) { statements}  
    // - als conditie true dan statements uitvoeren  
    if(innerBlockUitvoeren)  
    {  
        System.out.println("Start van inner code blok");  
        System.out.println("Einde van inner code blok");  
    }  
    System.out.println("Einde van code blok");  
}
```

► Of het een of het andere

```
void hoeKunnenWeHetEneBlockOfHetAndereBlockConditioneelUitvoeren() {  
    System.out.println("Start van code blok");  
    boolean uitvoerenBlok1 = true;  
    if(uitvoerenBlok1)  
    {  
        System.out.println("Start van inner code blok1");  
        System.out.println("Einde van inner code blok1");  
    }  
    else  
    {  
        System.out.println("Start van inner code blok2");  
        System.out.println("Einde van inner code blok2");  
    }  
    System.out.println("Einde van code blok");  
}
```

► Veel gebruikte syntaxconventie

```
void gebruikVanSyntaxConventies() {  
    System.out.println("Start van code blok");  
  
    boolean uitvoerenBlok1 = true;  
    if(uitvoerenBlok1) {  
        System.out.println("Start van inner code blok1");  
        System.out.println("Einde van inner code blok1");  
    } else {  
        System.out.println("Start van inner code blok2");  
        System.out.println("Einde van inner code blok2");  
    }  
    System.out.println("Einde van code blok");  
}
```



Loop statements

▼ Herhalen van een blok

```
void hoeKunnenWeHetGenesteBlokEenAantalKerenHerhalen() {  
    // Met { .... } definieren we een blok  
    System.out.println("Start van code blok");  
    // Voorbeeld van oneindige loop  
    while(true) {  
        //blokken kun je ook nesten  
        System.out.println("Start van inner code blok");  
        System.out.println("Einde van inner code blok");  
    }  
    System.out.println("Einde van code blok"); // onbereikbaar  
}
```

▼ Vast aantal keer herhalen van blok

```
@Test  
public void hoeKunnenWeHetGenesteBlok10KeerHerhalen() {  
    int aantalHerhalingen = 10;  
    int loopNr = 1;  
    boolean conditie = true;  
    while(conditie) {  
        System.out.println("loopNr = " + loopNr  
                          + " aantalHerhalingen = " + aantalHerhalingen);  
        conditie = loopNr <= aantalHerhalingen;  
        loopNr = loopNr + 1;  
    }  
}
```

► Restructure code

Restructure

```
int aantalHerhalingen = 10;
int loopNr = 1;
boolean conditie = true;
while(conditie) {
    // Do useful work
    conditie = loopNr <= aantalHerhalingen;
    loopNr = loopNr+1;
}
```

into

```
while(conditie=loopNr <= aantalHerhalingen) {
    // Do useful work
    loopNr = loopNr + 1;
}
```

► Translate while into for loop

Translate

```
int aantalHerhalingen = 10;
int loopNr = 1;
while(loopNr <= aantalHerhalingen) {
    // Do useful work
    loopNr = loopNr + 1;
}
```

into

```
for(int loopNr = 1; loopNr <= aantalHerhalingen; loopNr = loopNr + 1) {
    // Do useful work
}
```

Lab

- ▷ Druk de tafel van 7 af in een nieuwe test
 - Gebruik System.out.println
 - Druk voor 1 t/m 10 af: $1 * 7 = 7$ etc.
 - Gebruik variabelen
 - Gebruik een while loop
- ▷ Maak in een nieuwe test een 2de oplossing van de tafel van 7 waarbij je een for loop gebruikt
- ▷ Maak in een nieuwe test de tafel van 8 die je wederom print naar de console

»

Methods

Codeblokken isoleren

- Start met vorige voorbeeld: tafel van 7

```
@Test  
public void uitwerkingPrintenVanDeTafelVan7() {  
    System.out.println("Start printen...");  
    int finishedLoops = 0;  
    int multiplyBy = 1;  
  
    while (finishedLoops < 10) {  
        System.out.println(multiplyBy + " * 7 = " + multiplyBy * 7);  
        multiplyBy = multiplyBy + 1;  
        finishedLoops = finishedLoops + 1;  
    }  
    System.out.println("Klaar!");  
}
```

- Veel code...

De tafel van 7

- Welk deel van de code gaat echt over de tafel van 7?
- Kunnen we dat eruit trekken?

```
public void uitwerkingPrintenVanDeTafelVan7(){  
    System.out.println("Start printen...");  
    { // start block  
        int finishedLoops = 0;  
        int multiplyBy = 1;  
  
        while (finishedLoops < 10) {  
            System.out.println(multiplyBy + " * 7 = " + multiplyBy * 7);  
            multiplyBy = multiplyBy + 1;  
            finishedLoops = finishedLoops + 1;  
        }  
    } // end block  
    System.out.println("Klaar!");  
}
```

► Geef blok een naam

```
public void uitwerkingPrintenVanDeTafelVan7() {  
    System.out.println("Start printen...");  
    tafelVan7() // When block receives a name, () are necessary  
    {  
        int finishedLoops = 0;  
        int multiplyBy = 1;  
  
        while (finishedLoops < 10) {  
            System.out.println(multiplyBy + " * 7 = " + multiplyBy * 7);  
            multiplyBy = multiplyBy + 1;  
            finishedLoops = finishedLoops + 1;  
        }  
    }  
    System.out.println("Klaar!");  
}
```

► Regels

- Het is niet toegestaan om een codeblok een naam te geven binnen een ander blok.
- Een codeblok *kan* een waarde teruggeven (return).
- In Java **moet** je het type specificeren dat een blok kan retourneren.
- Als het blok niets retourneert, **moet** je `void` als retourtype gebruiken.

► The code

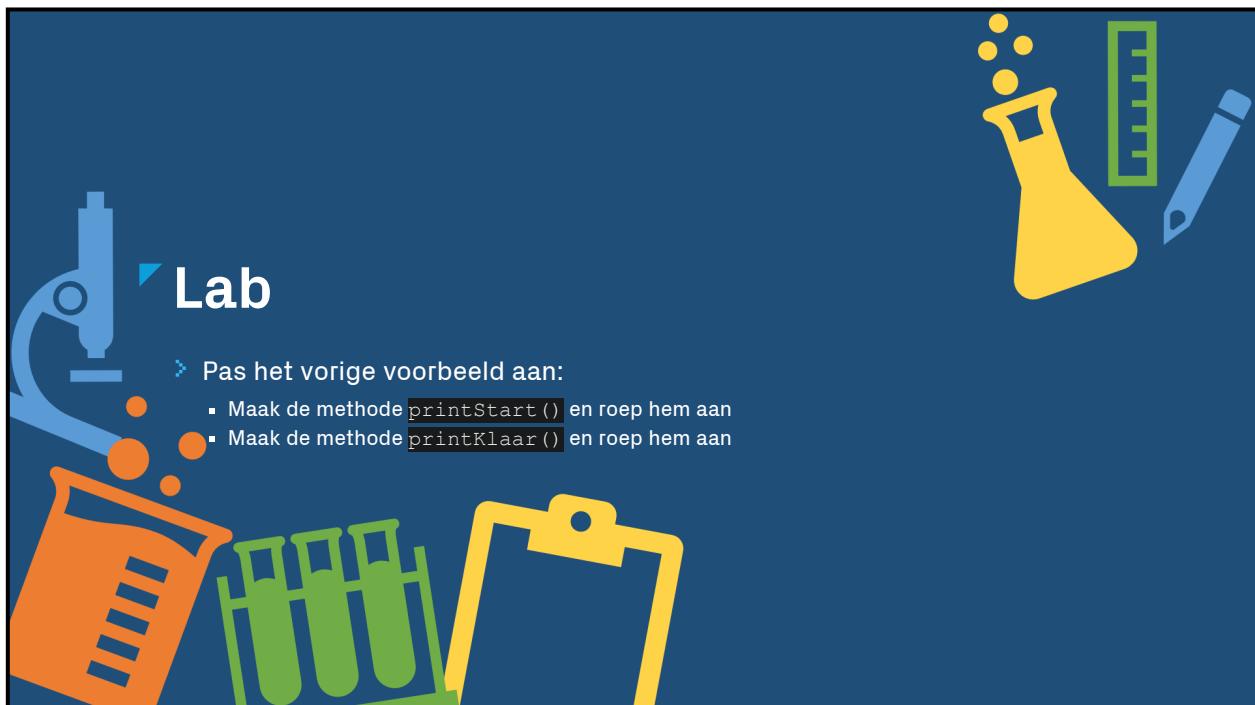
- `tafelVan7()` is een functie
 - in OO: *methode*

```
void tafelVan7() {  
    int finishedLoops = 0;  
    int multiplyBy = 1;  
  
    while (finishedLoops < 10) {  
        System.out.println(multiplyBy + " * 7 = " + multiplyBy * 7);  
  
        multiplyBy = multiplyBy + 1;  
        finishedLoops = finishedLoops + 1;  
    }  
}
```

► Aanroepen van een methode

- Methodes kunnen meerdere keren worden aangeroepen
 - Maakt hergebruik mogelijk
 - Code op één plek: beter onderhoudbaar

```
@Test  
public void generalisatie() {  
    System.out.println("Start printen...");  
    tafelVan7();  
    tafelVan7();  
    System.out.println("Klaar!");  
}  
  
void tafelVan7() {  
    // ...  
}
```



Lab

➤ Pas het vorige voorbeeld aan:

- Maak de methode `printStart()` en roep hem aan
- Maak de methode `printKlaar()` en roep hem aan

Parameter

- Een methode kan ook een of meerdere variabele waardes mee krijgen
 - parameter(s)
- Die vul je in bij het aanroepen
 - argument(en)
- Bijv. een generieke methode voor het printen van *een* bericht

```
void print(String bericht){  
    System.out.println(bericht);  
}
```



Lab

▷ Pas het vorige voorbeeld aan:

- Maak de methode `print(...)` en roep hem aan op alle plekken waar iets geprint wordt.
- Als dat werkt, zorg er dan voor dat er een uitroepstreken achter alle geprinte berichten komt.
- Zorg er daarna voor dat je met een tweede, `boolean` parameter kunt aangeven of je een uitroepstreken wilt printen of niet.



Introductie lokale variabele

```
void tafelVan() {
    int grondTal = 7; // of 5, 6, ...
    int finishedLoops = 0;
    int multiplyBy = 1;

    while (finishedLoops < 10) {
        p(multiplyBy + " * " + grondTal + " = " + multiplyBy * grondTal);
        multiplyBy = multiplyBy + 1;
        finishedLoops = finishedLoops + 1;
    }
}
```

Zou het niet handig zijn om bij het aanroepen te kunnen meegeven wat de tafel moet doen?

```
@Test  
public void voerDeTafelVan7EnVan8Uit(){  
    tafelVan(7);  
    tafelVan(8);  
}
```

► Introductie parameter

```
void tafelVan(int grondTal) {  
    int finishedLoops = 0;  
    int multiplyBy = 1;  
  
    while (finishedLoops < 10) {  
        print(multiplyBy + " * " + grondTal + " = " + multiplyBy * grondTal);  
        multiplyBy = multiplyBy + 1;  
        finishedLoops = finishedLoops + 1;  
    }  
}
```

▼ Aanroepen van het codeblok met parameter

```
@Test  
public void voerDeTafelVan7EnVan8Uit(){  
    tafelVan(7);  
    tafelVan(8);  
}
```

▼ System.out.println... veel typen!

```
@Test  
public void getRidOfSystemoutprintln() {  
    // Hier hebben we wel een block code uitgehaald  
    // Hier roepen we de methode aan  
  
    print();  
    print("nog een andere string");  
  
    // methods with the same name but with different  
    // number of parameters or type of parameter are allowed in Java  
    // This is called method overloading  
}
```

► Methods with the same name

```
// declareren we een methode
void p() {
    System.out.println("*****");
}
//String input wordt parameter genoemd
void p(String input) {
    System.out.println(input);
}
```

► Methode overloading

- methodes in java mogen de zelfde naam hebben
- methodes moeten dan wel verschillen in
 - aantal parameters
 - of type van parameters

Lab

- Maak drie varianten van de printmethode
 - `print()` print een regel sterretjes
 - `print(String b)` print het meegegeven bericht
 - `print(String b, boolean ut)` print het meegegeven bericht, eventueel met een uitroepteken
 - Druk alle tafels van 1 t/m 10 af m.b.v. een loop
 - Maak een methode om twee getallen met elkaar te vermenigvuldigen
 - krijgt twee `int` parameters
 - geeft het resultaat terug aan de aanroeper
 - gebruik deze in je eerdere code



3

Debugging

► Inzicht krijgen in code executie

```
@Test  
public void voerDeTafelVan7EnVan8Uit(){  
    tafelVan(7);  
    tafelVan(8);  
}
```

► We willen inzicht krijgen in de code

```
void tafelVan(int grondTal) {  
    int finishedLoops = 0;  
    int multiplyBy = 1;  
  
    while (finishedLoops < 10) {  
        p(multiplyBy + " * " + grondTal + " = " + multiplyBy * grondTal);  
        multiplyBy = multiplyBy + 1;  
        finishedLoops = finishedLoops + 1;  
    }  
}
```

▼ Zet een breakpoint

- De rode stip in kantlijn



▼ Klik op de start test knop

- De debug optie wordt dan getoond



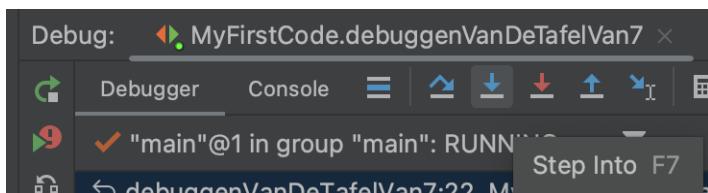
► De code wordt uitgevoerd

- De debugger pauzeert met executie code wanneer het een breakpoint tegenkomt
- Het statement dat blauw gekleurd is, is nog niet uitgevoerd



► De debugger opties

- Met **Step Into** kunnen we nu elk statement in de tafelVan methode uitvoeren



▶ Statement voor statement uitvoeren

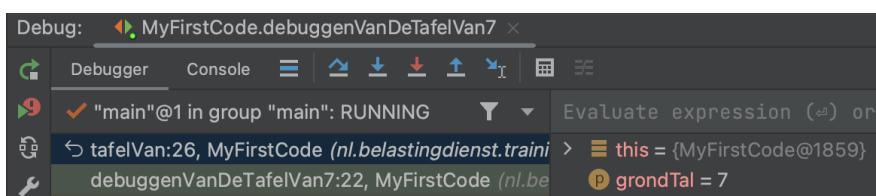


A screenshot of a Java debugger interface. A red dot at the top left indicates the current execution point. The code shown is:

```
@Test  
void debuggenVanDeTafelVan7(){  
    tafelVan( grondTal: 7);  
  
}  
  
1 usage  
void tafelVan(int grondTal) { grondTal: 7  
    int finishedLoops = 0;  
    int multiplyBy = 1;  
  
    while (finishedLoops < 10) {  
        System.out.println((multiplyBy + " * " + grondTal + " = " + multiplyBy * grondTal));  
        multiplyBy = multiplyBy + 1;  
        finishedLoops = finishedLoops + 1;  
    }  
}
```

▶ Inzicht in waarden van variabelen

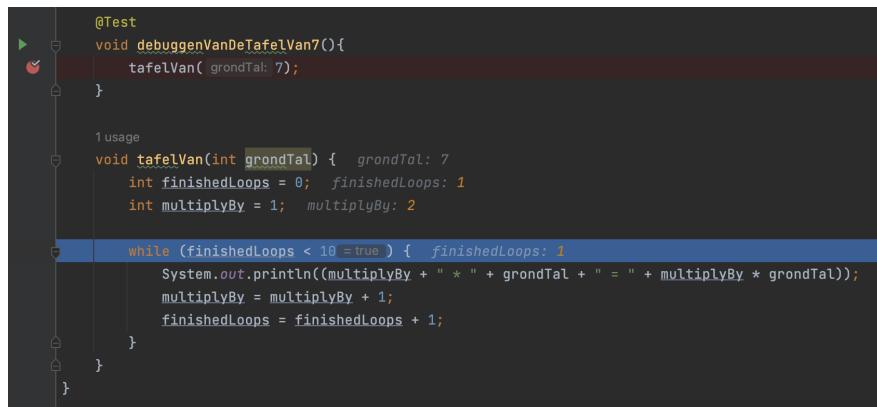
- De debugger toont ook een view met de actuele waarden van variabelen



A screenshot of the Java debugger's sidebar under the 'Debug' tab. It shows the current stack frame and variable values:

- "main"@1 in group "main": RUNNING
- ↳ tafelVan:26, MyFirstCode (nl.belastingdienst.traini)
- ↳ debuggenVanDeTafelVan7:22, MyFirstCode (nl.be)
- this = {MyFirstCode@1859}
- grondTal = 7

► Hoe vaak zijn we door de while loop gelopen?



A screenshot of a Java debugger interface. The code being debugged is:

```
@Test
void debuggenVanDeTafelVan7(){
    tafelVan( grondTal: 7);
}

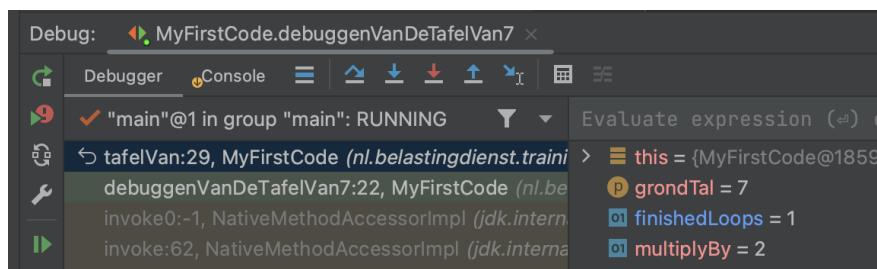
1 usage
void tafelVan(int grondTal) { grondTal: 7
    int finishedLoops = 0; finishedLoops: 1
    int multiplyBy = 1; multiplyBy: 2

        while (finishedLoops < 10 = true ) { finishedLoops: 1
            System.out.println((multiplyBy + " * " + grondTal + " = " + multiplyBy * grondTal));
            multiplyBy = multiplyBy + 1;
            finishedLoops = finishedLoops + 1;
        }
    }
}
```

The debugger shows the current execution point at the start of the while loop. The variable `finishedLoops` has a value of 1.

► Waarom heeft `finishedLoops` de waarde 0 en 1?

► De actuele waarde van variabelen



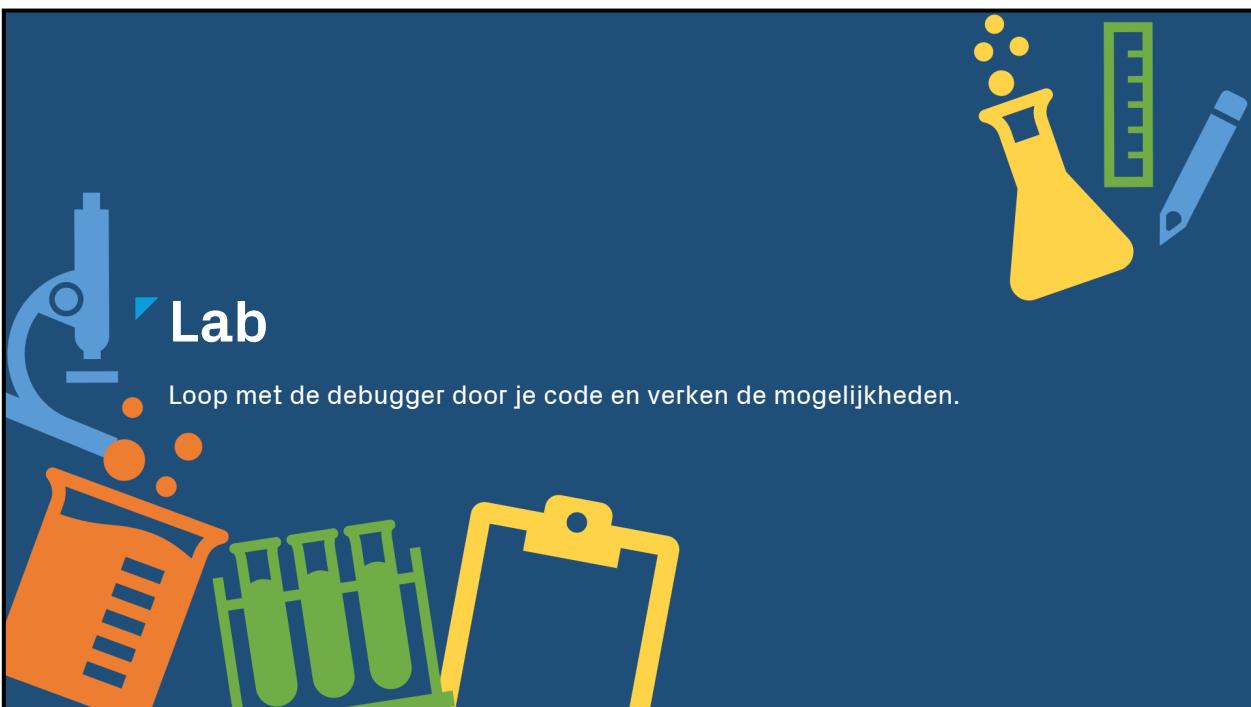
A screenshot of the Java debugger's sidebar showing variable values:

- `this = {MyFirstCode@1859}`
- `grondTal = 7`
- `finishedLoops = 1`
- `multiplyBy = 2`

► Hoe vaak zijn we door de while loop gelopen?

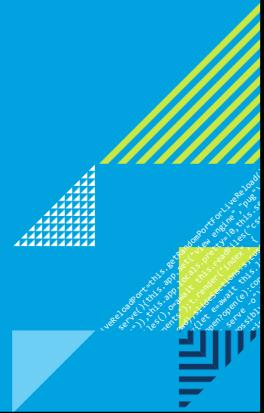
```
1 usage
void tafelVan(int grondTal) {  grondTal: 7
    int finishedLoops = 0;  finishedLoops: 6
    int multiplyBy = 1;  multiplyBy: 7

    while (finishedLoops < 10 = true ) {  finishedLoops: 6
        System.out.println((multiplyBy + " * " + grondTal + " = " + multiplyBy * grondTal));
        multiplyBy = multiplyBy + 1;
        finishedLoops = finishedLoops + 1;
    }
}
```





Arrays



► Werken met veel data

```
@Test
void slaCijfersOpInGeheugen(){
    int x0,x1,x2,x3,x4,x5,x6,x7,x8,x9;
    x0 = 4;
    x1 = 7;
    x2 = 8;
    x3 = 5;
    x4 = 7;
    x5 = 3;
    x6 = 9;
    x7 = 6;
    x8 = 6;
    x9 = 10;
    System.out.println("x0 = " + x0 + " x1 = " + x1 + " ....etc");
}
```

- Jammer dat we elk stukje data met een unieke naam moeten aanspreken
- Dit is niet echt flexibel

▶ Arrays to the rescue

```
@Test  
void slaCijfersOpInGeheugen(){  
    int[] cijfers = new int[10];  
    cijfers[0] = 4;  
    cijfers[1] = 7;  
    cijfers[2] = 8;  
    cijfers[3] = 5;  
    cijfers[4] = 7;  
    cijfers[5] = 3;  
    cijfers[6] = 9;  
    cijfers[7] = 6;  
    cijfers[8] = 6;  
    cijfers[9] = 10;  
    System.out.println("x1 = " + cijfers[0] + " x2 = " + cijfers[1] + " ....etc")  
}
```

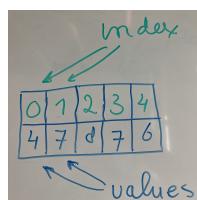
Of in één keer

```
int[] cijfers = {4, 7, 8, 5, 7, 3, 9, 6, 6, 10};
```

▶ Structuur array

➤ Eigenschappen

- heeft **vaste** lengte
- elementen hebben indexen, die beginnen op **0!**



▼ Array gebruiken in een for loop

```
@Test
void printDeAfzonderlijkeCijfers(){
    int[] cijfers = new int[10];
    cijfers[0] = 4;
    cijfers[1] = 7;
    // ... toekenning andere array-elementen weggelaten ...

    for (int index = 0; index < cijfers.length; index = index + 1) {
        System.out.println("cijfers[" + index + "] = " + cijfers[index]);
    }
}
```

➤ We kunnen dus via

- de index heel makkelijk array elementen adresseren.
- `.length` de lengte opvragen.

▼ Een array gevuld met reference types

```
@Test
void eenArrayGevuldMetDierenNamen(){
    String[] animalNames = new String[5];
    animalNames[0] = "Kat";
    animalNames[1] = "Leeuw";
    animalNames[2] = "Wolf";
    animalNames[3] = "Blauwevinvis";
    animalNames[4] = "Ekster";
    for (int index = 0; index < animalNames.length; index = index + 1) {
        System.out.println("animalNames[" + index + "] = " + animalNames[index])
    }
}
```

Lab

N.B: maak voor de opdrachten hieronder zoveel mogelijk aparte methodes.

1. Maak een array die 5 gehele getallen bevat.

- Elk getal stelt het cijfer voor van een examen waarbij maximaal 100 punten zijn te verdienen en minimaal 10

2. Maak code die

- alle cijfers print
- het laagste cijfer print
- het hoogste cijfer print
- het gemiddelde cijfer print
- controleert dat alle cijfers in de range van 10 tot 100 liggen
 - wanneer dat niet zo is, print dan de index waarvoor dat geldt en de waarde die buiten de range valt

3. Breid je oplossing uit

- de lijst kan nu een willekeurige mee te geven lengte krijgen
- de inhoud bestaat uit random cijfers tussen 10 en 100
 - tip: gebruik hiervoor `int cijfer = new Random().nextInt(10, 100);`

Lab

› Maak een array van een geschikt type, waarin je de namen kunt opslaan van je collega's.

› Maak code die vervolgens de kortste naam print.

› Breng de code die de kortste naam vindt onder in een aparte methode

- De methode verwacht als argument een array van een geschikt type
- De methode print zelf niet maar geeft de kortste naam terug

»

Exception handling

► Er gaat iets fout

- Java gooit dan een zogenaamde Exception
- Wat kan er fout gaan?
- Wie kan er iets verzinnen?

► Handling basic exceptions

- Using try and catch blocks



► Catch or throws

- Code that can throw a certain exception must be enclosed by
 1. A try/catch block, or
 2. A method that specifies that it can throw the exception.



► 1. try/catch

- Put the normal code in a try block
- Handle the exception in an separate catch block

```
try {  
    // statements that can cause SomeException  
    // ...  
} catch (SomeException e){  
    // statements to handle SomeException e  
    // ...  
}
```

► try/catch/finally

- optionally, you can add a finally to a try/catch

```
try {  
    // statements that can cause SomeException  
    // ...  
} catch (SomeException e){  
    // statements to handle SomeException e  
    // ...  
} finally {  
    // these statements are always executed,  
    // e.g. close resources  
}
```

► try/finally

- you can add a `finally` to a `try` without a `catch`

```
try {
    // statements to open a resource
    // ...
} finally {
    // these statements are always executed,
    // e.g. close resources
}
```

► 2. throws

- Specify the exception(s) thrown by a method in a `throws` clause

```
public void methodA() throws IOException, CustomException {
    // .. code that can throw IOException, CustomException
}
```

► Guidelines

- Catching
 - Arrange `catch` blocks from specific to general
 - Do not let exceptions drop off `main`
- Finally
 - `finally` blocks are always executed
 - useful for closing resources

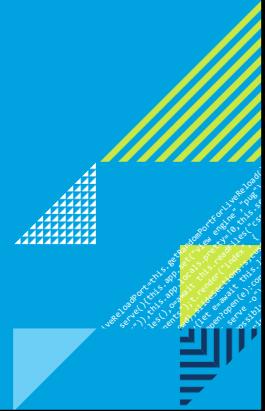


► Lab

- Maak code die een deling door nul veroorzaakt.
 - Welke fout treedt hier op?
 - Handel deze fout af.
- Wat gebeurt er als je een arrayindex gebruikt die niet bestaat?
 - Hoe los je dit op?



Objecten



Objecten

```
@Test
public void hetAanmakenVanRoald() {
    String naam; // hier declareer je variabele, een doosje

    naam = "Roald"; // hier stop je een string in
    // een string is geen primitive maar wat dan wel
    // een string is een reference naar een object
    // een object van het type String
}
```

► Wat is een object

- een logisch samenhangend geheel van
 1. data (state)
 2. gedrag (behaviour)
- de data?
 - de characters 'R', 'o', 'a', 'l' en 'd'
- het gedrag?
 - wat voor vragen kun je aan het String object vragen?
 - hier betekent het: welke methoden kun je aanroepen op "Roald"

► Welke methode heeft een String object?

Wat doen de methodes?

Wat doet:

- compareTo
- hoe moeten we het aanroepen?
- wat geeft de methode terug?
- hoe zie ik dat?



Lab

Maak afzonderlijke tests waarin je per test de volgende methodes van de String "Roald" uitprobeert?

- concat
- contains
- endsWidth
- equals
- equalsIgnoreCase

