Java

2023 – 2024 : les 1

jeroen.devos01@ap.be

AP HOGESCHOOL ANTWERPEN

AP.BE

Planning OLOD Java

- Week 1 : de basis
- Week 2 : generics, collections, arrays & loops
- Week 3 : equals, sorting, recursie & iteratie
- Week 4 : optional, lambda & streams
- Week 5 : files, exceptions, stack trace & debugging
- Week 6 : logging & annotations

Planning OLOD Java

- Week 7 : JPA & Hibernate
- **Week 8 : JAX-RS, RestEasy & Undertow**
- Week 9 : Threading
- Week 10 : Unit testing & Test Driven Development
- Week 11 : TBD
- Week 12 : Herhaling / Q&A / Exameninfo

Digitap

- Info over het OLOD
- Elke week
  - Presentatie
  - Voorbeeldcode
  - Labo opgave
  - Labo oplossing
  - Eventuele extra informatie
  - Cursus (in opbouw)
  - Zelftest (in opbouw)

Lesdoelen

- Wat is Java
- IDE
- Naming conventions
- Start coding with Java
  - package / class / method / variabele
  - access level modifiers
  - final, static, void, null
  - data types + enum
- Dates
- abstract / interface / get & set
- Console
- Running the program : main

WHAT IS JAVA

Wat is Java?

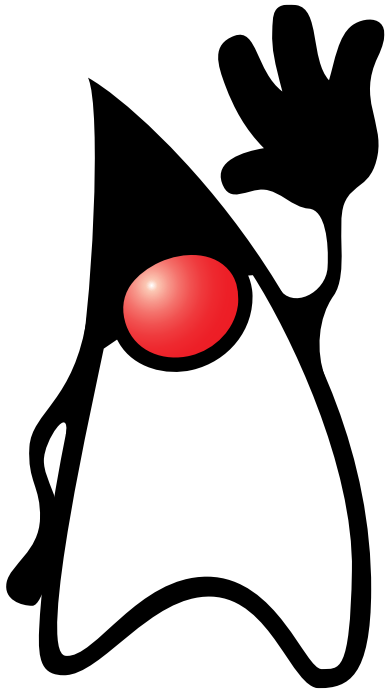
- Object georiënteerde programmeertaal
- 3GL taal (third-generation language)
- Write once , run anywhere

→ Java code wordt gecompileerd naar byte code en uitgevoerd in een Java Virtual Machine (JVM)

- Meest populaire taal voor backend server development (bronnen: Github en Google Trends)

Geschiedenis

- 1995 : Java released door Sun Microsystems
  - Bedacht door James Gosling
  - Alternatief voor C / C++
  - Focus : OO, security en geheugengebruik
  - Bedoeld voor interactieve televisie en bankautomaten
  - Nadien heroriëntatie naar web browser (applet)
- 2006 : Java wordt Open Source
- 2010 : Sun Microsystems opgekocht door Oracle
- 2016 : Applet niet langer ondersteund



Terminologie

- Platformen
  - Java Card -> SIM kaart, bankkaart, paspoort …
  - Java ME (Micro Edition) -> sensoren, printers, PDA …
  - Java SE (Standard Edition) -> pc
  - Jakarta EE (Enterprise Edition) -> server
  - Java FX -> user interface, vooral financiële/medische wereld
- Java Virtual Machine (JVM)
- Java Development Kit (JDK)
- Java Runtime Environment (JRE)

Versies

Vanaf Java 9 :

- 2 releases per jaar
- Maart en september

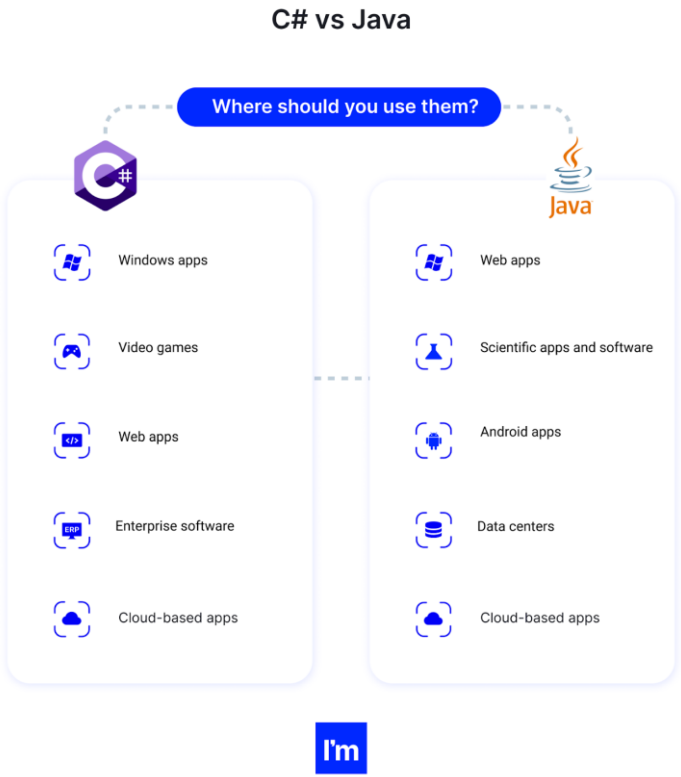
LTS : Long Term Support

Industrie volgt LTS versies

Wij werken met **Java 17**

Versie	Datum
JDK 1.0	Januari 1996
JDK 1.1	Februari 1997
J2SE 1.2	December 1998
J2SE 1.3	Mei 2000
J2SE 1.4	Februari 2002
J2SE 5	September 2004
Java SE 6	December 2006
Java SE 7	Juli 2011
Java SE 8 (LTS)	Maart 2014
Java SE 9	September 2017
Java SE 10	Maart 2018
Java SE 11 (LTS)	September 2018
Java SE 12	Maart 2019
Java SE 13	September 2019
Java SE 14	Maart 2020
Java SE 15	September 2020
Java SE 16	Maart 2021
Java SE 17 (LTS)	**September 2021**
Java SE 18	Maart 2022
Java SE 19	September 2022
Java SE 20	Maart 2023
Java SE 21 (LTS)	September 2023


Java
C#

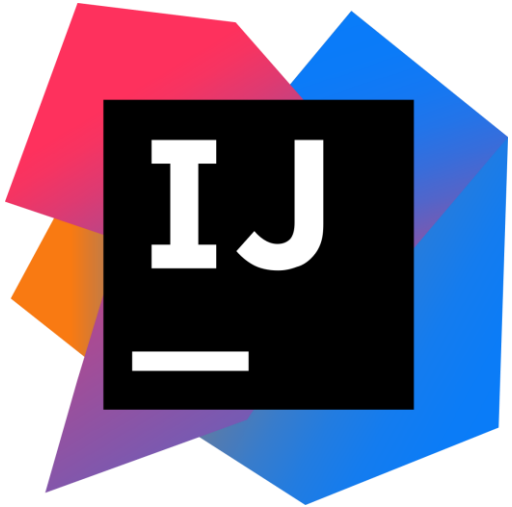


Java wordt (o.a.) gebruikt door:

- Google
- Facebook
- Netflix
- Amazon
- Instagram
- LinkedIn

En alle Android devices…

IDE : Integrated Development Environment



Welke IDE ?

- Voor Java development keuze uit:
  - IntelliJ
  - Eclipse
  - Visual Studio Code
  - ...
- Leer met zoveel mogelijke IDE’s werken !
- In de werkomgeving : meestal opgelegd door bedrijf / team

- Tijdens de lessen / labo’s zal **IntelliJ** gebruikt worden.


JAVA
PROGRAMMERS
INTELLIJ
ECLIPSE
imgflip.com


WHEN YOU FINALLY DISCOVERED INTELLIJ
AFTER BEING FUCKED BY ECLIPSE FOR SO LONG
imgflip.com

Setup (links ook te vinden op digitap)

- JDK 17

https://www.oracle.com/java/technologies/downloads/#java17

- IntelliJ Ultimate

https://www.jetbrains.com/idea/download

→ Gratis voor studenten!

- Tijdens labo extra uitleg over IntelliJ


Java Naming Conventions
Java
www.educba.com

Java Naming Conventions

Het correct toepassen van **naming conventions** onderscheidt de professional van de amateur !

package	flatcase	com.example
class / interface	PascalCase	AnimalProperties
method	camelCase	makeSound()
attribute	camelCase	dateOfBirth
constant	SCREAMING_SNAKE_CASE	MAX_AGE

https://www.oreilly.com/library/view/java-8-pocket/9781491901083/ch01.html

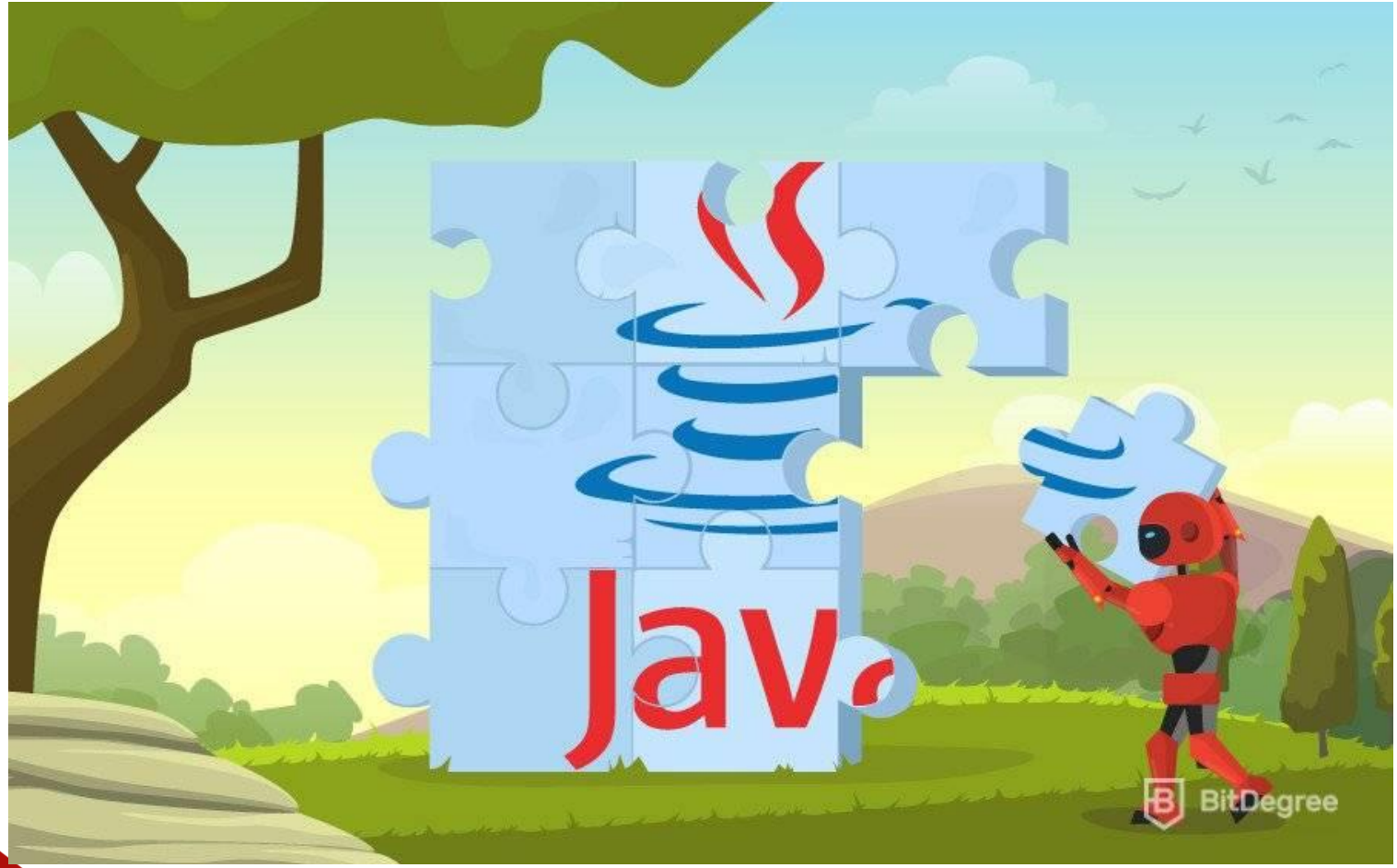
Afspraken

1. Pas de **correcte NAMING CONVENTIONS** toe

2. Schrijf je code in het **ENGELS**

Bij evaluatie zijn deze regels mee onderdeel van het 'boeteblad'

- Niet volgen naming conventions : -2
- Niet consistente naamgeving (Nederlands vs Engels) : -1


Jav
BitDegree

De eerste class

```
package be.ap.wk01;

public class HelloWorld {

}
```

- keyword package :

Een package wordt gebruikt om classes te groeperen. Vergelijk het met een folder voor bestanden.

- keyword class :

Een class is een object. De **classnaam** moet **hetzelfde** zijn als de **bestandsnaam** : HelloWorld.java

Intermezzo : access modifiers

```
package be.ap.wk01;

public class HelloWorld {

}
```

Access modifiers bepalen de zichtbaarheid doorheen de code

- public : zichtbaar voor iedereen
- protected : enkel zichtbaar binnen dezelfde package/subpackage
- private : enkel zichtbaar in dezelfde class
- ‘leeg’ : enkel zichtbaar binnen dezelfde package (package-private)

Opmerking : Class is altijd public of package-private

De eerste variabele

```
public class HelloWorld {
        private String greeting = “Hello World”;
}
```

Declaratie : de variabele greeting wordt aangemaakt van data type String en access modifier private.

Initialisatie : de variabele greeting wordt opgevuld (geïnitialiseerd) met de waarde Hello World.

Toewijzingsoperator (=) wordt gebruikt om een waarde aan een variabele toe te wijzen (initialiseren).

Intermezzo : code statement end

```
public class HelloWorld {
        private String greeting = “Hello World”;
}
```

Elk java code statement moet eindigen met een puntkomma (;).

Een code statement kan over meerdere lijnen geschreven worden voor de leesbaarheid, pas helemaal op het einde van het statement zet je de ‘;’.

De eerste method

```
public class HelloWorld {
        public String getGreeting() {
                return “Hello World“;
        }
}
```

Een method heeft :

- een access modifier : public
- een naam : getGreeting
- na de naam haakjes ()
- een return data type : String
- keyword return : om een waarde terug te geven van het return data type.

Intermezzo : code block

```
public class HelloWorld {
        public String getGreeting() {
                return “Hello World“;
        }
}
```

Een blok code die bij elkaar hoort begint en eindigt met een accolade of curly brace { en } .

Een class is een blok code, een method ook.

Blokken code kunnen genest worden.

Tip : zet steeds al je eind accolade als je een begin accolade zet.

De tweede method

```
public class HelloWorld {
        public String getPersonalGreeting(String firstName , String lastName) {
                return “Hello “ + firstName + “ “ + lastName;
        }
}
```

Een method heeft optioneel één of meerdere parameters als input.

Deze input parameters worden gescheiden door een komma (,).

Een input parameter heeft een data type.

Merk op: een input variabele heeft géén access modifier.

Variabelen - scoping

```
public class HelloWorld {
        private String greeting = “Hello World”;

        public String getPersonalGreeting(String firstName , String lastName) {
                String greeting = “Hello “ + firstName + “ “ + lastName;
                return greeting;
        }
}
```

Wat is de return value van getPersonalGreeting(“Jeroen”, “De Vos”) ?

- “Hello World”
- “Hello Jeroen De Vos”

Variabelen – soorten

```
public class HelloWorld {
        private String greeting = “Hello World”;

        public String getPersonalGreeting(String firstName , String lastName) {
                String greeting = “Hello “ + firstName + “ “ + lastName;
                return greeting;
        }
}
```

- Class attribute : greeting
- Method parameter : firstName en lastName
- Local variable : greeting

Tip : probeer te vermijden dat je variabele namen herbruikt

De derde method

```
public class HelloWorld {
        private String greeting = “Hello World”;

        public void updateGreeting(String firstName , String lastName) {
                greeting = “Hello “ + firstName + “ “ + lastName;
        }
}
```

keyword void : speciaal return type dat aangeeft dat er géén return value is.

Merk op : het keyword ‘return’ mag niet in de method staan!

Bonus: wat is de waarde van ‘greeting’ na het uitvoeren van de method?

Intermezzo : final

```
public class HelloWorld {
        private final String greeting = “Hello World”;

        public void updateGreeting(String firstName , String lastName) {
                greeting = “Hello “ + firstName + “ “ + lastName;
        }
}
```

Blijft van mijn class attribute ‘greeting’ af !

keyword final : de variabele mag na toewijzing niet meer wijzigen.

Merk op dat deze code een compilatiefout zal geven!

Intermezzo : static final

```
public class HelloWorld {
        private static final String GREETING = “Hello World”;

        public String getGreeting() {
                return GREETING;
        }
}
```

Een static final variabele is een constante.

keyword static : een static object zal slechts één keer worden aangemaakt en wordt gedeeld over alle objecten van hetzelfde type

Primitive data types

Een primitive data type bevat enkel zijn waarde en is **geen object**.

Data Type	Size	Omschrijving
boolean	1 bit	Bevat true of false
char	2 bytes	Bevat één letter of ASCII waarde
byte	1 byte	Bevat een geheel getal van -128 tot 127
short	2 bytes	Bevat een geheel getal van -32.768 tot 32.767
int	4 bytes	Bevat een geheel getal van -2.147.483.648 tot 2.147.483.647
long	8 bytes	Bevat een geheel getal van …
float	4 bytes	Bevat een getal tot 7 cijfers na de komma
double	8 bytes	Bevat een getal tot 15 cijfers na de komma

Non-primitive data types

Een non-primitive data type is **een object** en heeft methods.

Elk primitive data type heeft een overeenkomstig non-primitive type:

- int -> Integer
- long -> Long
- double -> Double
- boolean -> Boolean

Daarnaast ook nog:

- String
- Class
- …

Enum : the special data type

```
public enum GreetingType {
        FORMAL, INFORMAL
}

public class HelloWorld {
        public void sayGreeting(GreetingType type) {
        }
}
```

- keyword enum
- Is geen class, maar volgt de naamgeving (PascalCase)
- Bevat constanten, gescheiden door komma
- Wordt gebruikt in een class / method als data type

Class constructor

```
public class Greeting {
        public Greeting() {
        }
}
```

```
public class HelloWorld {
        public void doGreeting() {
                Greeting greeting = new Greeting();
        }
}
```

- Nieuwe class Greeting
- public Greeting() { } => **constructor**
- De constructor is de code die wordt uitgevoerd als de class wordt aangemaakt via het keyword new.
- Een constructor zonder extra method parameters = default constructor

Class constructor overloading

```
public class Greeting {
        private final String fullName;

        public Greeting(String fullName) {
                this.fullName = fullName;
        }
}
```

```
public class HelloWorld {
        public void doGreeting() {
                Greeting greeting =
                        new Greeting("Jeroen De Vos");
        }
}
```

- Constructor kan één of meerdere method parameters krijgen => typisch wanneer je wilt dat bij aanmaak van een class een waarde MOET gekend zijn omdat de class anders niet goed kan werken.
- Keyword this : hiermee selecteer je een class attribute, this gebruik je dus om onderscheid te maken tussen local variable en class attribute.

En wat met methods?

```
public class Greeting {

        public void sayHello() {
        }

}
```

```
public class HelloWorld {
        public void doGreeting() {
                Greeting greeting = new Greeting();
                greeting.sayHello();
        }
}
```

- Method sayHello() van class Greeting kan gebruikt worden in de HelloWorld na initialisatie van de class.
- Gebruik een punt (.) om een methode van de geïnitialiseerde class op te roepen.

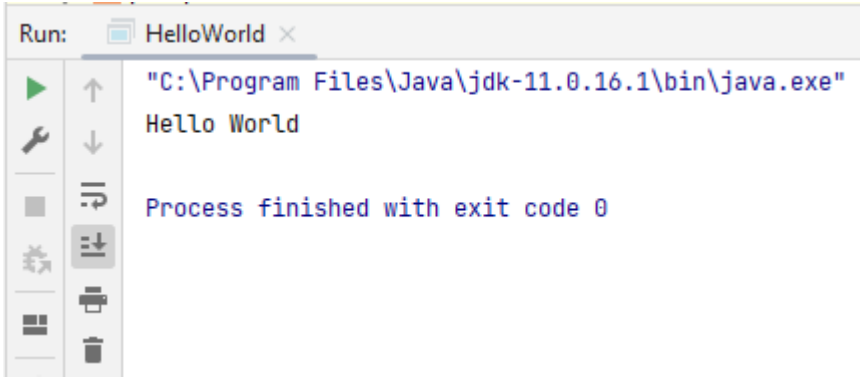
Werken met de console

```
public class Greeting {
        public void sayGreeting() {
                System.out.println(“Hello World”);
        }
}
```

De Java console is een debugging tool.

Deze console kan je gebruiken om bv tekst in te schrijven.

System.out.println() : schrijft één lijn tekst in de console.



Data type null

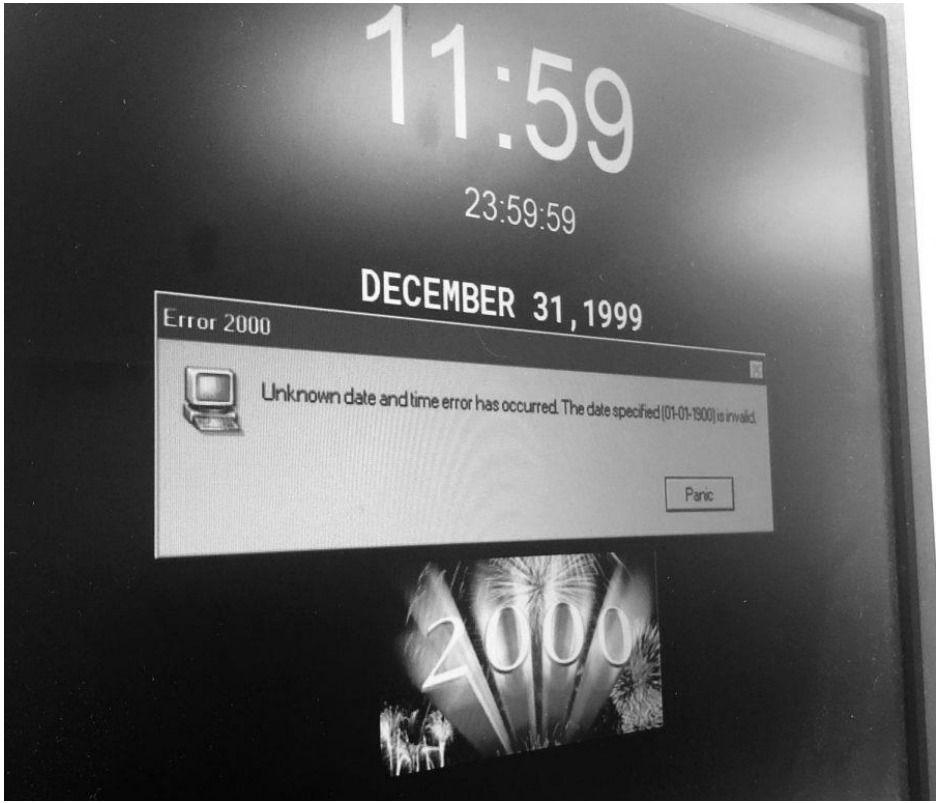
```
public class HelloWorld {
	private String hello;

	public void doGreeting() {
		Greeting greeting;
	}

}
```

- Wat is de waarde van class attribute hello of local variable greeting ?
- Deze zijn niet geïnitialiseerd -> waarde is **null**

Opgepast : als je een method van een ‘null’ object aanroept -> fout

Dates


REMEMBER
Turn your computer off
before midnight on
31/12/99.


11:59
23:59:59
DECEMBER 31, 1999
Error 2000
Unknown date and time error has occurred. The date specified (01-01-1900) is invalid.
Panic
2000

java.util.Date

```
Date theOldWay = new Date();
```

- Onderdeel van Java vanaf versie 1.1
- Voor Java 8 was dit dé manier om een nieuw datum object aan te maken
- Heel veel code gebruikt nog java.util.Date

Legacy … gebruik dit **niet meer** !

-> onderdeel van boeteblad

java.util.Date – design flaws

```
// 1 januari 2023
Date theOldWay = new Date(123, 0, 1);
```

- Geen datum, maar een milliseconde teller vanaf 1 januari 1970
  - Geen tijdszones – maar in toString wordt de lokale tijdzone genomen
  - Negatieve milliseconden voor 1970
- Jaar = int vanaf 1900 : bv 2023 = 123
- Maand = int 0 tot 11 : januari = 0, december = 11
- Dag = int 1 tot 28/29/30/31


OK, WE SCREWED UP WITH JAVA.UTIL.DATE
LET'S CREATE A CALENDAR CLASS

java.time (vanaf java 8)

```
LocalDate nowOnlyDate = LocalDate.now();
LocalTime nowOnlyTime = LocalTime.now();
LocalDateTime now = LocalDateTime.now();
ZonedDateTime zoned = ZonedDateTime.now();
OffsetDateTime offset = OffsetDateTime.now();
```

- LocalDate : datum zonder tijdzone bv 2022-01-01
- LocalTime : tijd zonder tijdzone bv 10:25:30
- LocalDateTime : datum/tijd zonder tijdzone bv 2022-01-01 10:25:30
- ZonedDateTime : datum/tijd met tijdzone
- OffsetDateTime : datum/tijd met tijdsverschil t.o.v. GMT

java.time

- Gebruik LocalDate / LocalTime / LocalDateTime om een datum / tijd te stockeren.
  Bv: geboortedatum, vervaldatum, aankoopdatum

- Gebruik ZonedDateTime wanneer het exacte moment belangrijk is.
  = LocalDateTime + TimeZone

- Gebruik OffsetDateTime om over een datum te communiceren tussen systemen of met de database… als de tijdzone relevant is.
  = LocalDateTime + offset

Rekenen met tijd

```
LocalDateTime from = LocalDateTime.of(2022, 1, 1, 0, 0, 0);
LocalDateTime to = LocalDateTime.of(2022, 1, 31, 23, 59, 59);

System.out.println(from.isAfter(to));

Duration duration = Duration.between(from, to);
System.out.printn(duration.toHours());
```

- Datums vergelijken : isAfter / isBefore / isEqual
- Duration : duurtijd tussen twee datums berekenen

Opgepast: Duration werkt enkel met LocalDateTime. Gebruik atStartOfDay() om van LocalDate naar LocalDateTime te gaan.

Parse Date - Time

```
String date = “01-01-2022 12:30”;
DateTimeFormatter formatter = DateTimeFormatter.ofPattern(“dd-MM-yyyy HH:mm”);
LocalDateTime dateTime = LocalDateTime.parse(date, formatter);
```

DateTimeFormatter

- Gebruik ISO datum formaat om patroon te bepalen

LocalDateTime.parse:

- Datum als string
- Formatter

Format Date - Time

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern(“dd-MM-yyyy HH:mm”);
LocalDateTime dateTime = LocalDateTime.of(2022, 1, 1, 12, 30, 0);
String date = dateTime.format(formatter);
```

DateTimeFormatter

- Gebruik ISO datum formaat om patroon te bepalen

dateTime.format:

- Formatter


STAR WARS
Get ready! Here we go!

Packages en imports

- Een package groepeert classes die bij elkaar horen. Vergelijk het met een folder.
- Binnen een package moet een class een unieke naam hebben.
- Binnen een package kent een class elke andere class.
- Een package kan één of meerdere sub-packages bevatten.

- Ook de Java broncode is opgesplitst in packages.

Denk eraan : packages worden in flatcase geschreven.

Packages en imports (cont.)

```
import starwars.characters.DarthVader;
import starwars.characters.LukeSkywalker;

public class StarWars {
   DarthVader vader = new DarthVader();
   LukeSkywalker luke = new LukeSkywalker();
}
```

- keyword import :

Zorgt ervoor dat een class uit een andere package gekend is in de class.

Packages en imports (cont.)

```
import starwars.characters.*;

public class StarWars {
   DarthVader vader = new DarthVader();
   LukeSkywalker luke = new LukeSkywalker();
}
```

Indien meerdere classes uit dezelfde package gebruikt worden, kan gekozen worden om een wildcard (*) teken in de import te zetten.

Opgelet : dan zijn ineens alle classes uit die package gekend.

OOP

- Encapsulatie / inkapselen (encapsulation)
- Overerving (inheritance)
- Polymorfisme (polymorphism)

Encapsulatie

```
public class DarthVader {
   private Side side = Side.DARK;
   public Side getSide() {
      return side;
   }
   public void setSide(Side side) {
      this.side = side;
   }
}
```

Interne werking is verborgen.

We beslissen zelf welke informatie we aan de buitenwereld kenbaar maken.

Voor class attributes → typisch : get en set

Overerving

```
public class Character {
   private final boolean forceUser;

   public Character(boolean forceUser) {
      this.forceUser = forceUser
   }
}
```

```
public class DarthVader extends Character {

   public DarthVader() {
      super(true);
   }

}
```

- keyword extends : erf over van een super class
- keyword super : met super roep je een attribuut of method van de super class aan, super() roept de constructor van de super class op.

Polymorfisme

```
public class DarthVader {

   public String favoriteFightingStyle() {
      return “throw lightsaber”;
   }

}
```

```
public class HanSolo {

   public String favoriteFightingStyle() {
      return “shoot blaster”;
   }

}
```

Elke class heeft dezelfde methode, maar het resultaat is afhankelijk van de class.

Probleem : hoe afdwingen dat elke class deze methode voorziet?

Polymorfisme – abstract class en methode

```
public abstract class Character {

   public abstract String favoriteFightingStyle();

}
```

```
public class DarthVader extends Character {

   public String favoriteFightingStyle() {
      return “throw lightsaber”;
   }

}
```

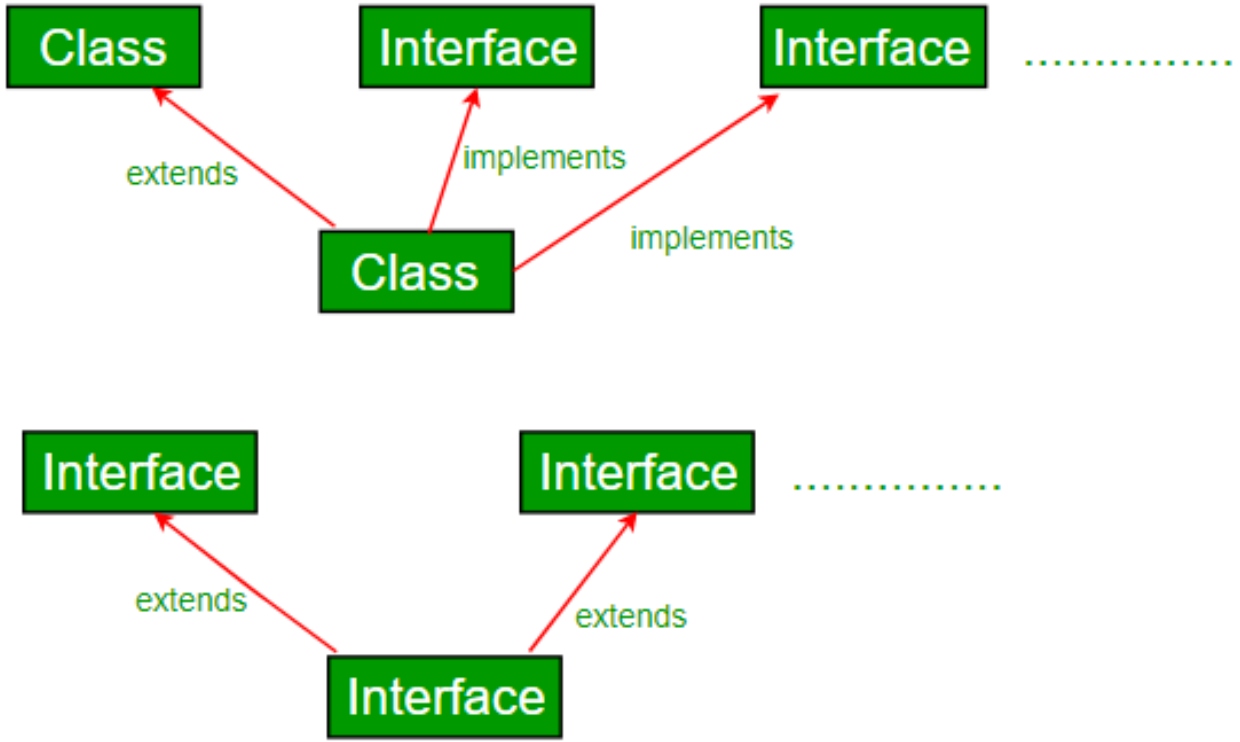
- keyword abstract class: een abstract class kan niet zelf worden aangemaakt, m.a.w. “new Character()” zal niet werken.
- keyword abstract method: kan enkel bestaan in een abstract class. De abstract method heeft geen body en moet door de subclasses worden aangemaakt.

Polymorfisme – interface

```
public interface FightingStyle {

   String favoriteWeapon();

}
```

```
public class DarthVader implements FightingStyle {

   public String favoriteWeapon() {
      return “lightsaber”;
   }

}
```

- keyword interface : een interface is een contract waaraan een class moet voldoen. De interface bevat verder geen code.


Class
Interface
Interface
.....
extends
implements
implements
Class
Interface
Interface
.....
extends
extends
Interface


ALMOST THERE
WE ARE
makeameme.org

Running your program : main method

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

        }
}
```

keywords public static void main(String[] args)

Deze speciale method zorgt ervoor dat de JRE je code zal kunnen uitvoeren. Bij opstart van je programma wordt de code in die main method eerst uitgevoerd.

En omdat Java developers zichzelf niet altijd even serieus nemen….


FOR TODAY
ANYWAY...