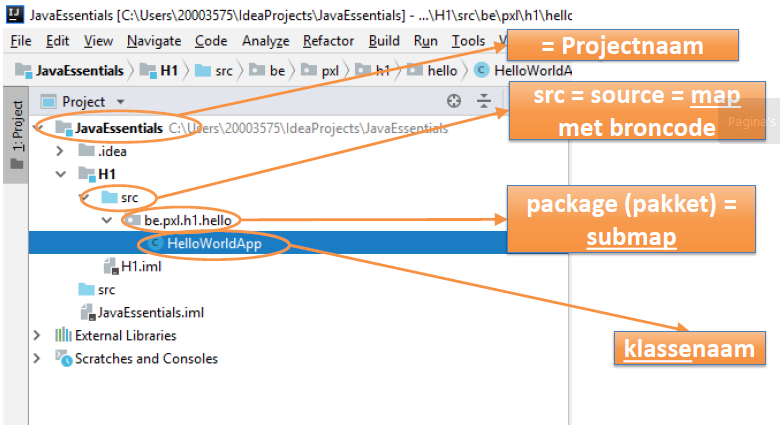
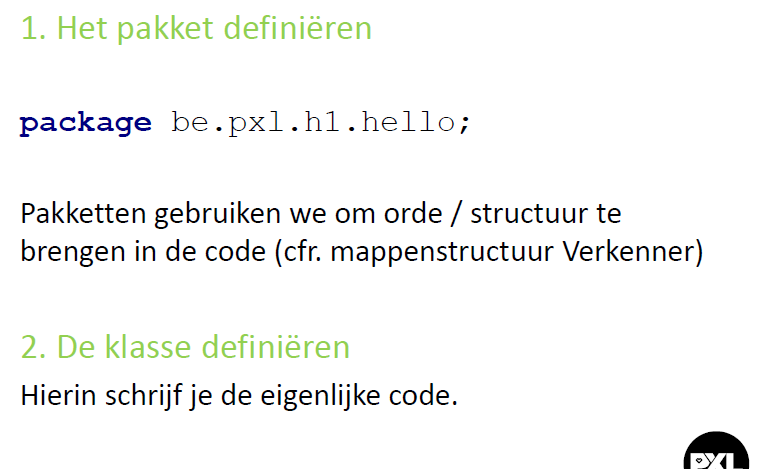
# Java Essentials

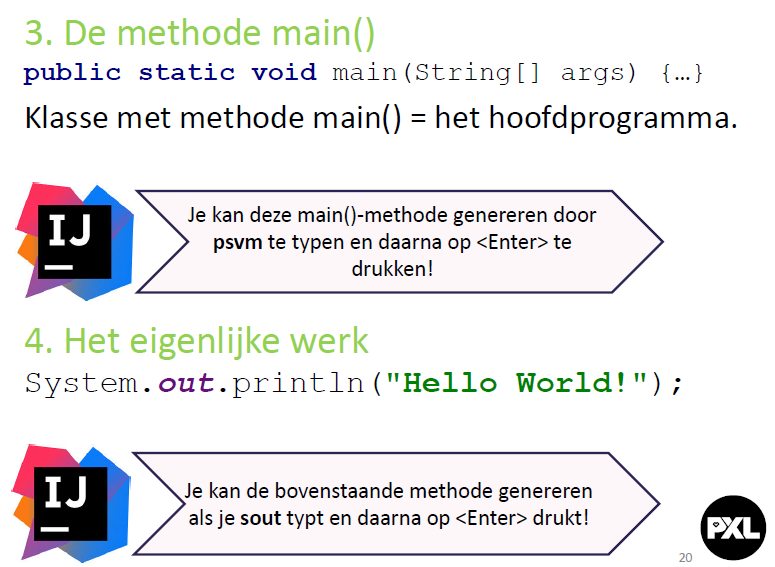
# **Hoofdstuk 1: Programmeren in Java**

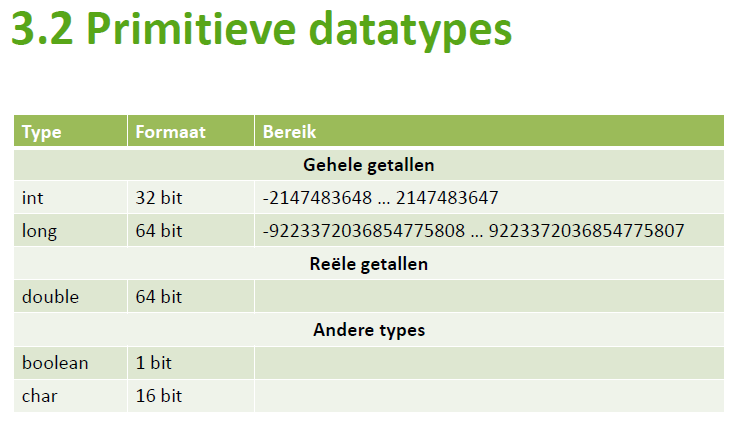
Verschil tussen JVM EN Java API

* JVM (Java Virtual Machine) interpreteerd de bytecode en gebruikt de onderliggende hardware en het os om deze uit te voeren
* Java API (Java Application Programming Interface) is een verzameling van software componenten die gebruikt kunnen worden door het java programma

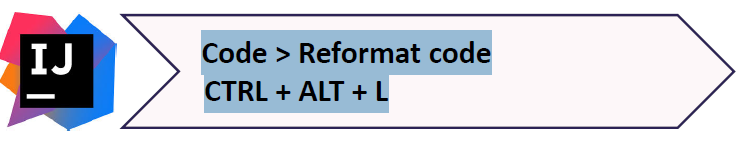
Soorten toepassingen

* Mobiel
* Point of sale systems
* Video games
* Trading applications
* Big data

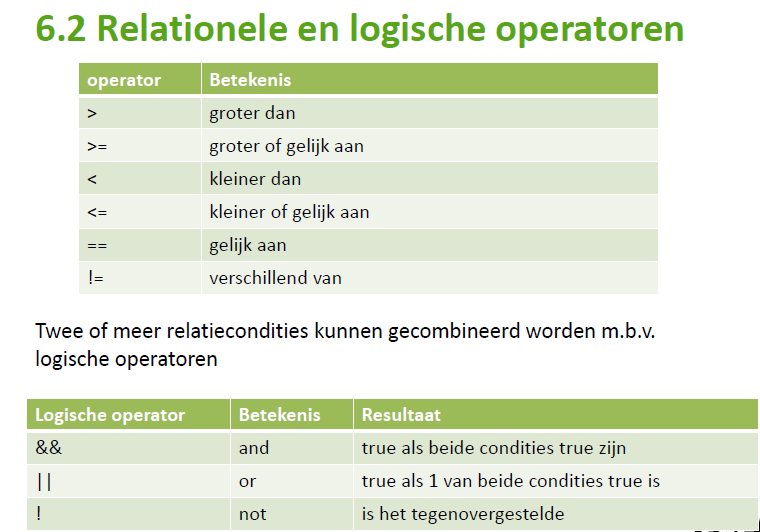


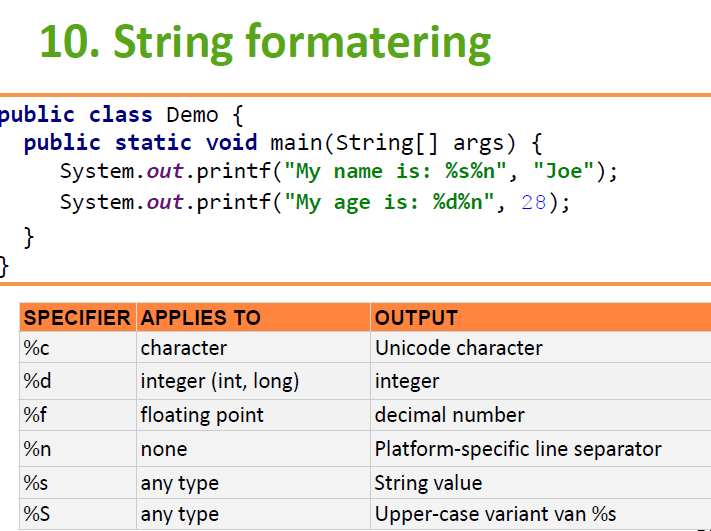












String name="sonoo";

String sf1=String.format("name is %s",name);

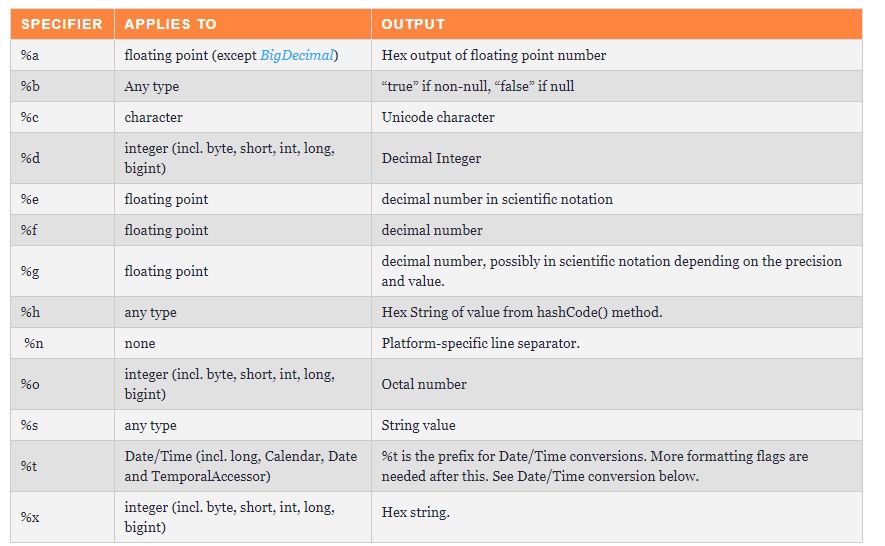
String sf2=String.format("value is %f",32.33434);

String sf3=String.format("value is %32.12f",32.33434);//returns 12 char fractional part filling with 0

System.out.println(sf1);

System.out.println(sf2);

System.out.println(sf3);



# **Hoofdstuk 2: Klassen definieeren**

Variabele private maken = encapsulation (data hiding)

Method overloading = methoden met dezelfde naam maar verschillende types of aantal parameters

Constructor overloading = meerdere constructors met verschillende parameters om een eigen taak uit te voeren

Voordeel:

Eigelijke code moet maar op 1 plaats geschreven worden

Code wordoverzichtelijker

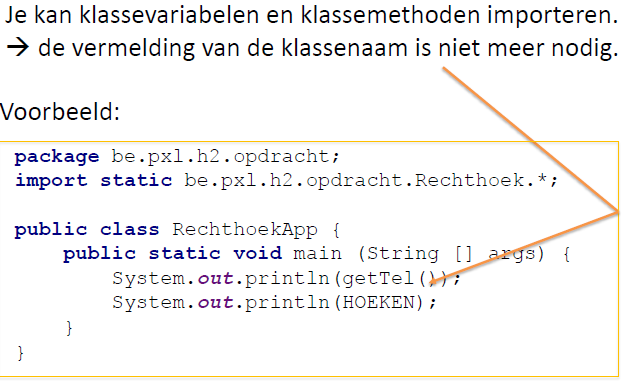
Risico op fouten daalt: aanpassingen gebeuren maar in 1 constructor

Instantievariabelen

* ieder object van een klasse heeft een eigen kopie van de variabelen met hun eigen waarden
* moet opgeroepen worden op en concreet object van de klasse

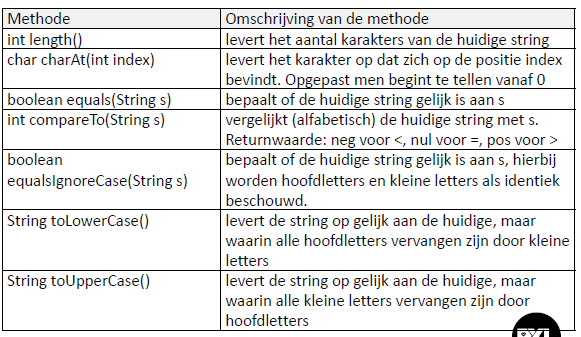
Klassevariabelen

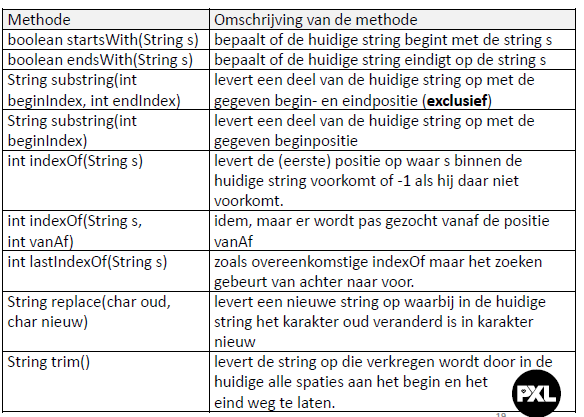
* Gemeenschappelijk voor alle objecten van dezelfde klasse
* Er is maar 1 van elke klassevariabele
* Static
* Initialisatie gebeurt bij de declaratie

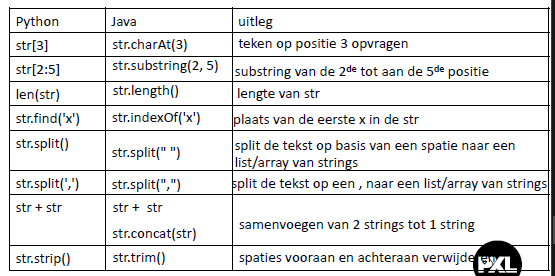


# **Hoofdstuk 3: Eenvoudige klassen (1)**

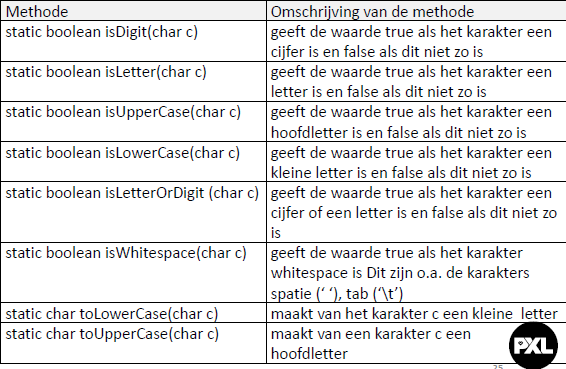
|  |  |  |  |
| --- | --- | --- | --- |
| Klasses | Uitleg | Variabelen | Methoden |
| Math | Utility klasseBevat klassevariabelen en methoden Je kan er geen object van aanmaken  Voor wiskunde berekeningen | Math.PI() Math.E() | Abs (absolute waarde) Sqrt (vierkantswortel)  Pow(x²)  Round(geheel getal) |
| Random | Utility klasse Wordt gebruikt voor het aanmaken van random getallen  Belangrijk, BOVENGRENS EXLUSIEF | / | NextInt()(willekeurig geheel getal) NextInt(int bound) willekeurig getal > 0 en < bound  NextDouble() random floating point tussen 0 en 1 |
| String | Tekst Automatisch geimporteerd in java | / | Zie onderstaande table (01 – string + 02 – string + 03 – string) |
| Character | Deze klasse reikt hulpmethoden aan om met karakters te werken (ASCII) Utility klasse | / | Zie onderstaande table (01 – character) |
| Stringbuilder | Bouwt een string om cpu tijd te besparen (idpv variabelen vervangen en weggooien) | Append() | Voorbeeld Stringbuilder s = new stringbuilder (“abc”)  s.append(“dkjlrf”) |



01 - string 1

02 - string

03 - string



01 - character

# **Hoofdstuk 4: Unittesten met Junit**

Wat is Unittesten?

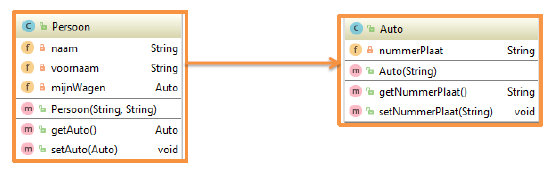
Een manier om stukjes broncode (methode’s of units) afzonderlijk te testen

Waarom Unittesten?

* Minder fouten
* Leesbaarder
* Je denkt meer na over implementatie van methoden



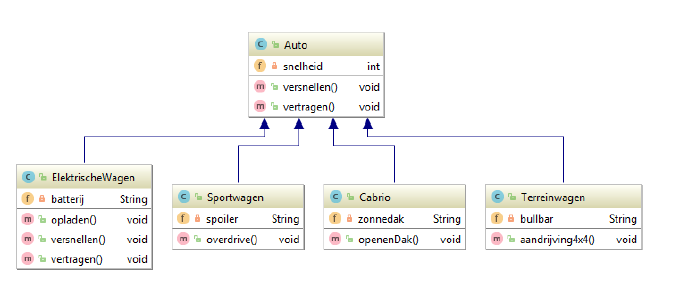
# **Hoofdstuk 5: Associaties**

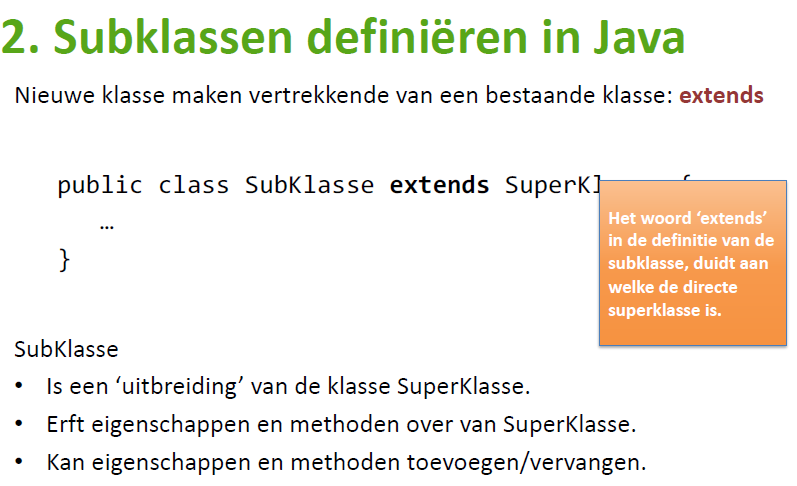
= algemene term voor een relatie tussen klassen of objecten, dit is een “HAS A”- relatie

# **Hoofdstuk 6: Overerving en klasse hierarchie**

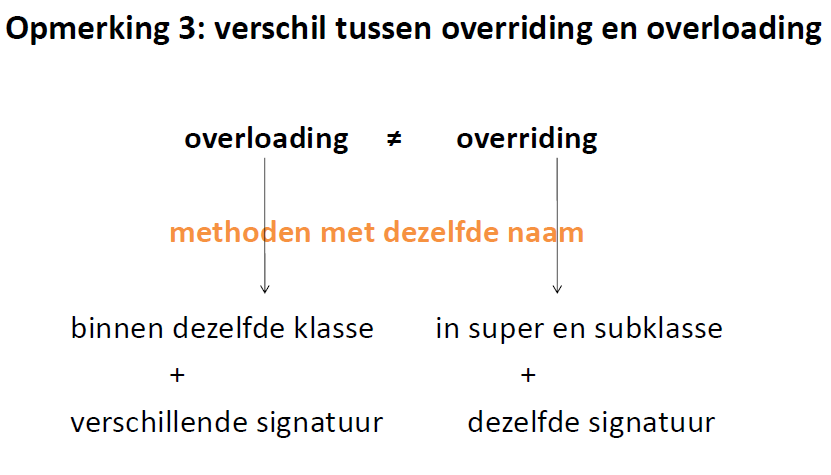
= een subklase ergt alle eigenschappen en methoden over van een superklasse, er kunnen eigenschappen en methoden toegevoegd worden. Sommige methoden kunnen vervangen worden. Dit is een “Is A”-relatie.

Eigenschappen subklassen:

* Ze erven alle eigenschappen en methoden over
* Extra eigenschappen en methoden toevoegen
* Methoden kunnen vervangen worden





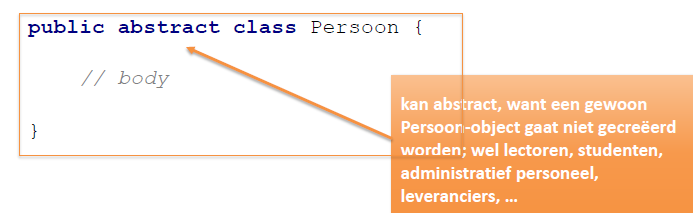


Final klassen en methoden:

* Kunnen geen subklassen hebben
* Kunnen niet overschreven worden in subklassen
* Waarom final klassen en final methoden?
  + Beveiliging
  + Ontwerp (beslissen dat bepaalde klassen af zijn)
  + Snelheid

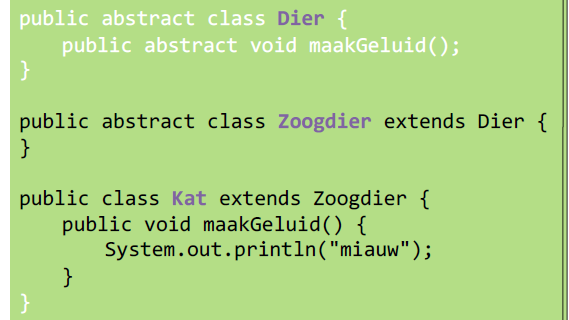
Abstracte klassen:

* Er kunnen geen objecten van gemaakt worden
* Dienen om zoveelmogelijk gemeenschappelijke code onder te brengen die overgeerft zal worden
* In UML schema (c)



Abstracte methoden:

* Enkel abstracte klassen kunnen abstracte methoden hebben
* Geen implementatie in de abstracte klasse, wel in de concrete klasse
* Een subklasse van een abstracte klasse, die zelf ook abstract is moet niet alle abstracte methoden implementeren

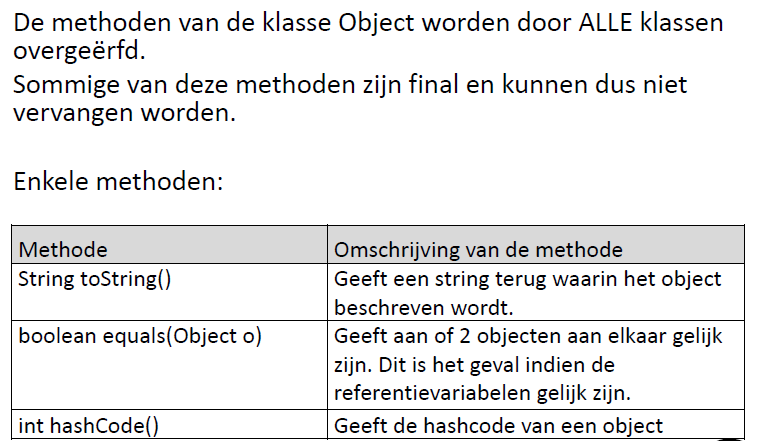


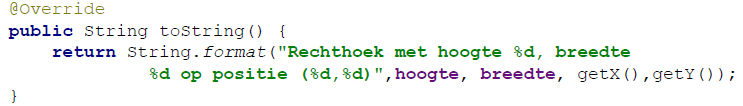
Superklasse OBJECT:

De superklasse van alle klassen is de klasse Object, iedere klasse is rechs of onrechtstreeks een subklasse van object. Dus public class A is hetzeflde als public class A extends Object

Instanceof :

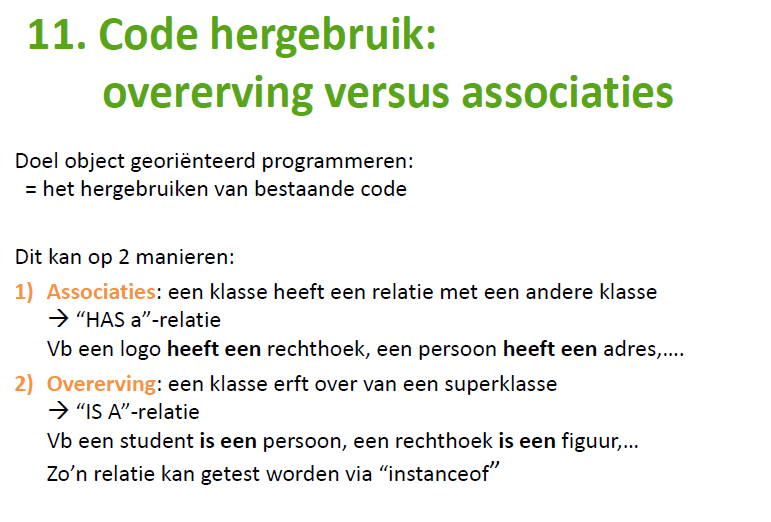
Met deze operator kan je nagaan of het object een instantie is van 1 of andere (super)klasse

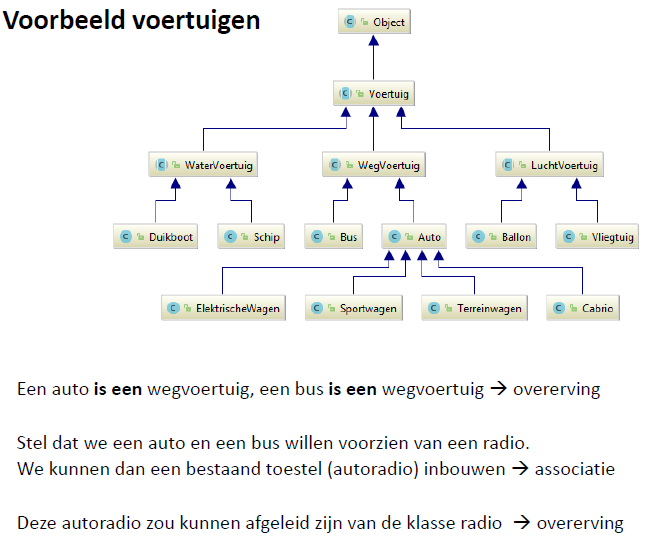


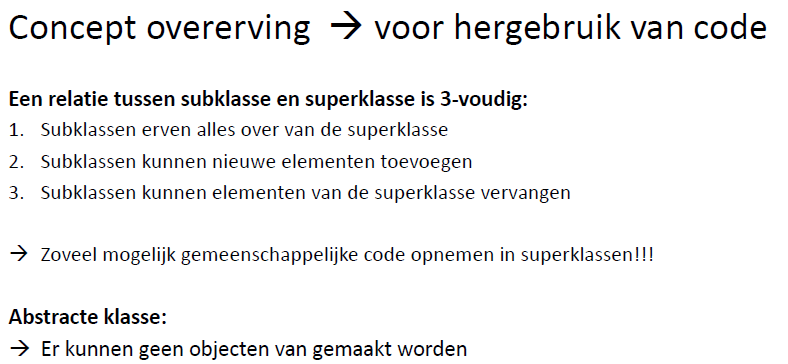


Polymorfisme:

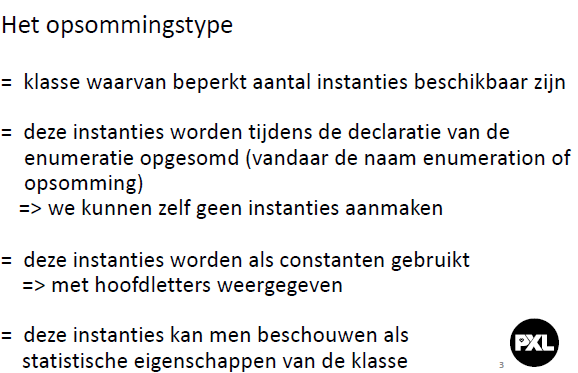
= een object dat veek vormel kan aannemen (gedeclareerd als type van een superklasse of decreerd als type van een subklasse)







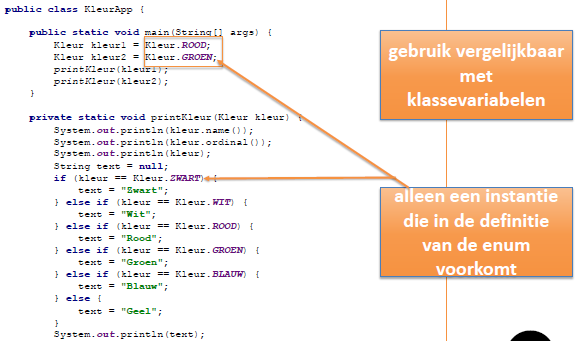
# **Hoofdstuk 7: De opsomming**

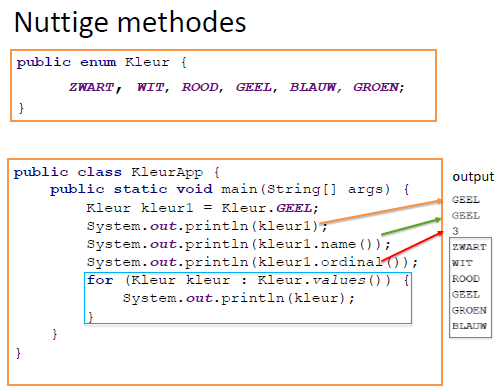


Enum:



Is een opsommingstype, elke enum is een subklasse van de abstracte klasse enum





Eigenschappen enums:

* Beschikken over eigenschappen, methoden en constructors
* Erft eigenschappen over van zijn superklasse Enum
* Eigenschappen en methoden kunnen toegevoegd of vervangen worden

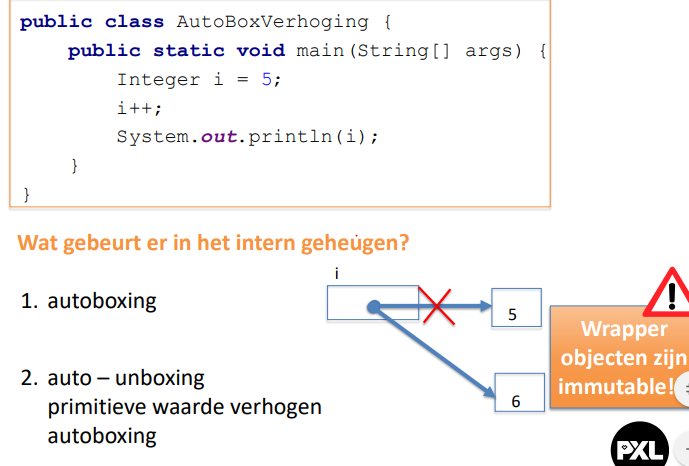
# **Hoofdstuk 8: Eenvoudige klassen (2)**

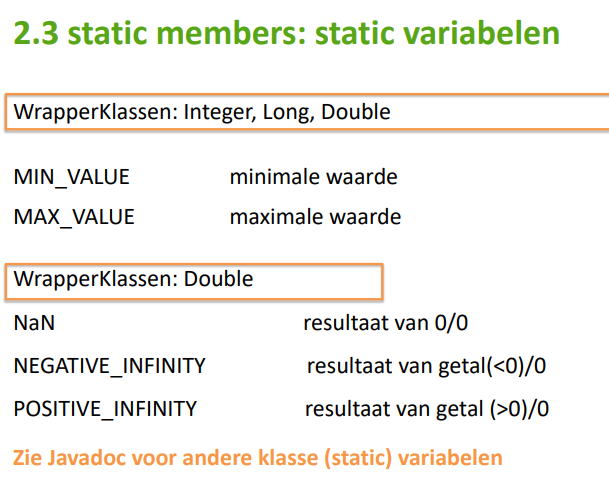
Wrapperklasse = de object variant van een primitief datatype (bv Integer bij int)

Enkele voorbeelden: Integer, Long, Double, Boolean, Character

Autoboxing 🡪 Een primitief datatype in een object steken

Auto-Unboxing 🡪 Een wrapper object in een primitief datatype steken







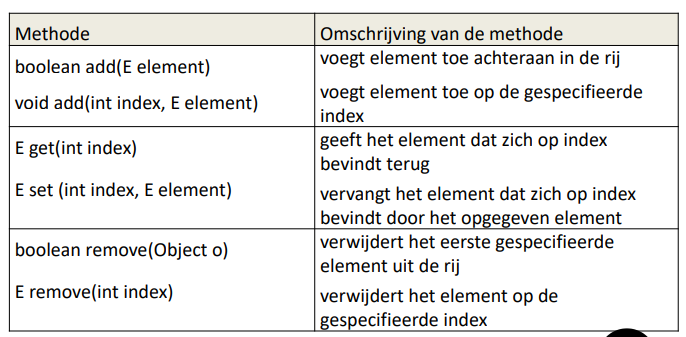
Conversiemethoden String (primitief datatype)

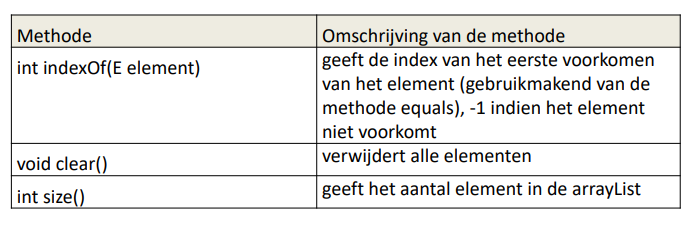
String naar Int 🡪 Integer.parseInt()

String naar double 🡪 Double.parseDouble()

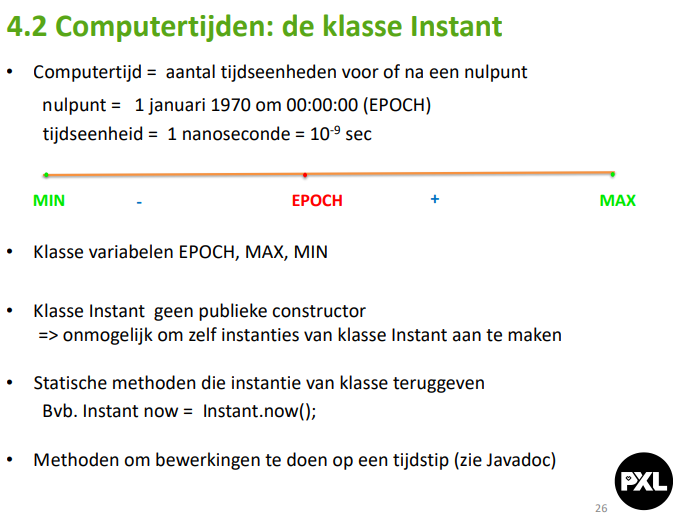
ArrayList 🡪 geordende verzameling van objecten, waarbij het aantal objecten op voorhand niet vast ligt.

ArrayList list = new ArrayList<>();





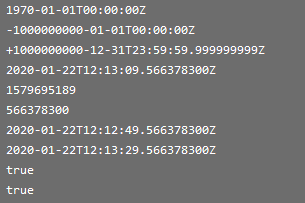
Datums en tijden:



import java.time.Instant;

public class InstantApp {

public static void main(String[] args) {

**System.out.println(Instant.EPOCH);**

**System.out.println(Instant.MIN);**

**System.out.println(Instant.MAX);**

Instant now = Instant.now();

**System.out.println(now);**

**System.out.println(now.getEpochSecond());**

**System.out.println(now.getNano());**

Instant earlier = now.minusSeconds(20);

**System.out.println(earlier);**

Instant later = now.plusSeconds(20);

**System.out.println(later);**

**System.out.println(now.isAfter(earlier));**

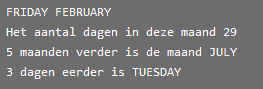
**System.out.println(now.isBefore(later));**

}

}

import java.time.DayOfWeek;

import java.time.Month;

public class EnumMontDay {

public static void main(String[] args) {

Month maand = Month.FEBRUARY;

DayOfWeek dag = DayOfWeek.FRIDAY;

**System.out.println(dag + " " + maand);**

**System.out.print("Het aantal dagen in deze maand ");**

**System.out.println(maand.length(true🡨 true = schrikkeljaar));**

**System.out.println("5 maanden verder is de maand " + maand.plus(5));**

**System.out.print("3 dagen eerder is “ + dag.minus(3));**

}

}

public static void main(String args[]) {

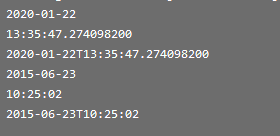
LocalDate nowDate = LocalDate.now();

LocalTime nowTime = LocalTime.now();

LocalDateTime nowDateTime = LocalDateTime.now();

LocalDate otherDate = LocalDate.of(2015, 6, 23);

LocalTime otherTime = LocalTime.of(10, 25, 2);

LocalDateTime otherDateTime = LocalDateTime.of(otherDate, otherTime **System.out.println(nowDate);**

**System.out.println(nowTime);**

**System.out.println(nowDateTime);**

**System.out.println(otherDate);**

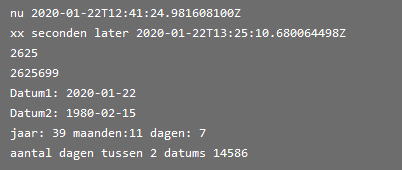
**System.out.println(otherTime);**

**System.out.println(otherDateTime);**

}

Tijdsduur:

Klassen:

* Duration (tijdsverschillen tussen machinetijden (=instanties van de klasse Instant)
* Period (tijdsverschillen tussen datebased-time (menselijke tijden)
* ChronoUnit (enumeratie)
  + Berekent verschillende tijdseenheden
  + Methode between() wordt omgezet in de gekozen tijdseenheid

public class DurationApp {

public static void main(String [] args) {

Instant now = Instant.now();

**System.out.println("nu " + now);**

Instant later = now.plusSeconds(500).plusMillis(2125698).plusNanos(456398);

**System.out.println("xx seconden later " + later);**

Duration duration = Duration.between(now, later);

**System.out.println(duration.getSeconds());**

long milliseconds = ChronoUnit.MILLIS.between(now, later);

**System.out.println(milliseconds);**

LocalDate nowDate = LocalDate.now();

LocalDate thenDate = LocalDate.of(1980, 2, 15);

**System.out.println("Datum1: " + nowDate);**

**System.out.println("Datum2: " + thenDate);**

Period period = Period.between(thenDate, nowDate);

**System.out.println("jaar: " + period.getYears() + " maanden:" + period.getMonths() + " dagen: " + period.getDays());**

long days = ChronoUnit.DAYS.between(thenDate, nowDate);

**System.out.println("aantal dagen tussen 2 datums " + days);**

}

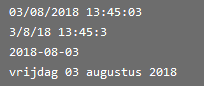
}

Datetime formatter

public static void main(String[] args) {

DateTimeFormatter myFormatter1 =

DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");

DateTimeFormatter myFormatter2 =

DateTimeFormatter.ofPattern("d/M/yy H:m:s");

DateTimeFormatter myFormatter3 =

DateTimeFormatter.ofPattern("yyyy-MM-dd");

DateTimeFormatter myFormatter4 =

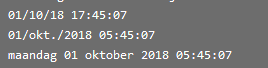
DateTimeFormatter.ofPattern("EEEE dd MMMM yyyy");

LocalDateTime dt1 =

LocalDateTime.parse("03/08/2018 13:45:03",myFormatter1);

System.out.println(myFormatter1.format(dt1));

System.out.println(myFormatter2.format(dt1));

System.out.println(myFormatter3.format(dt1));

System.out.println(myFormatter4.format(dt1));

}

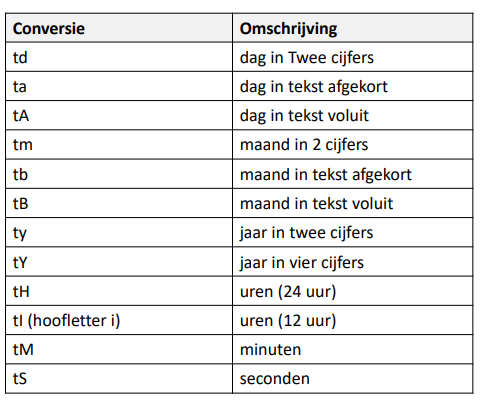
public static void main(String[] args) {

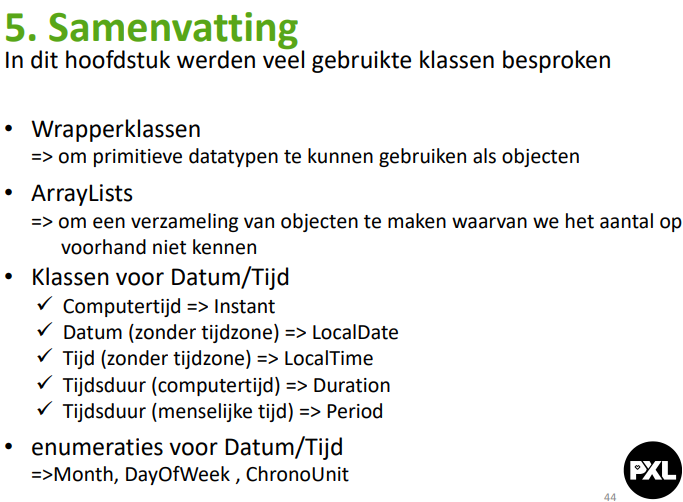
LocalDateTime datum = LocalDateTime.of(2018, 10, 1, 17, 45, 7);

System.out.printf("%1$td/%1$tm/%1$ty %1$tH:%1$tM:%1$tS \n", datum);

System.out.printf("%1$td/%1$tb/%1$tY %1$tI:%1$tM:%1$tS \n", datum);

System.out.printf("%1$tA %1$td %1$tB %1$tY %1$tI:%1$tM:%1$tS \n", datum);

}

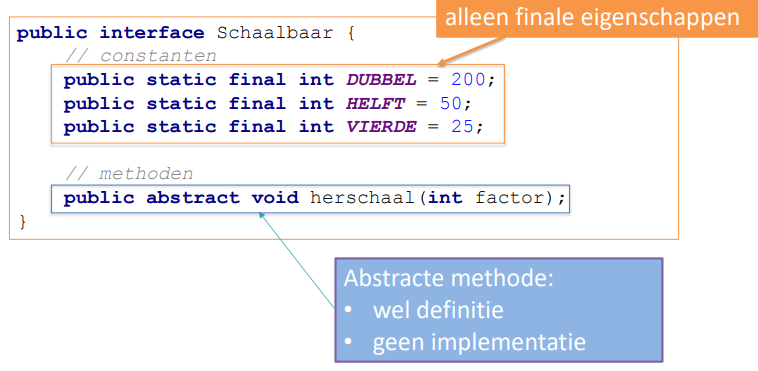


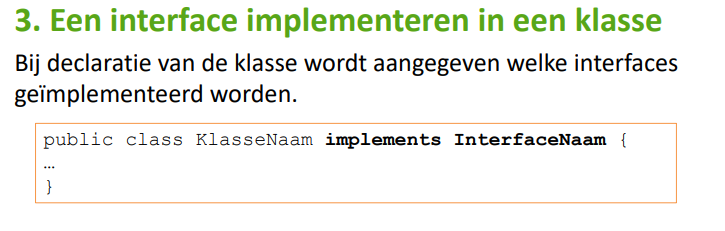
# **Hoofdstuk 9: Interfaces**

= een abstracte klasse die alleen abstracte methoden en constanten (final eigenschappen) bevat

Het doel 🡪 aan bepaalde klassen methoden opleggen die geimplementeerd moeten worden







public interface Schaalbaar {

// constanten

int DUBBEL = 200;

int HELFT = 50;

int VIERDE = 25;

// methoden

void herschaal(int factor);

}

public class Rechthoek extends GrafischElement implements Schaalbaar {

private int hoogte;

private int breedte;

public Rechthoek (int x, int y, int h, int b) {

super(x, y);

this.hoogte = h;

this.breedte = b;

}

//andere methoden

public void herschaal(int factor) {

hoogte = hoogte \* factor / 100;

breedte = breedte \* factor / 100;

}

}

