Hoofdstuk 1

PSVM  
SOUT **Always reformat code by ctrl alt L to gain +2 intelligence , alles terug te vinde in java AAPJE**

Long declareren: long a = 5161L;

String = “ “

Char = ‘ ‘ (1character)

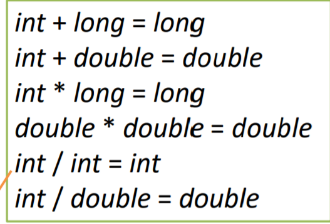
Primitieve types: geheugen voor bv int, long en char zitten rechtstreeks in het vrijgemaakte geheugen

Referentietypes: ipv de effectieve waarde in het geheugen bevat het een locatieadres naar waar een string gevestigd is

Meerdere shit in 1 print = sout(getal + “ “ + waarde)

Casten of typeconversie = veranderen van variabeltype

Int a = (int) 645,55; ,55 valt weg, doet java vanzelf door (int) toe te voegen



Boolean = true of false **rangorde**

Char = 1 character double 1

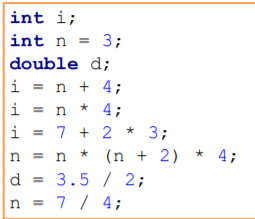
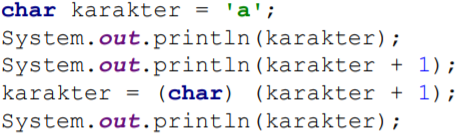
Int= klein geheel getal long 2

Long= grote getal zo int 3

Double= kommagetal char 4

7/2 = gehele deling (3)

7/2.0 = kommadeling (3.5)

 a

98

b

7

12

13

60

1.75

1

Char zet bv a om achterliggend naar zijn ascii waarde, dus er kan + getal gedaan worden

Strings zijn onveranderbaar

Variabele(reader in voorbeeld hieronder).next(); .next leest 1 woord, nextline meerdere woorden  
 .nextline(); neemt de enter van vorige inputlijn mee, waardoor we niks kunnen ingeven

Bovenstaande voor strings, onderstaande voor ints en doubles

 .nextInt();

.nextDouble();

Common mistakes with nextline: [Link](https://christprogramming.wordpress.com/2014/01/30/java-common-mistakes-1/)

Input: wanneer er een foutcode bij ingave voorkomt, just change . to ,

import java.util.Scanner;

Scanner reader = new Scanner(System.in); // Reading from System.

System.out.println("Enter a number: ");

int n = reader.nextInt(); // Scans the next token of the input as an int.

//once finished

reader.close();

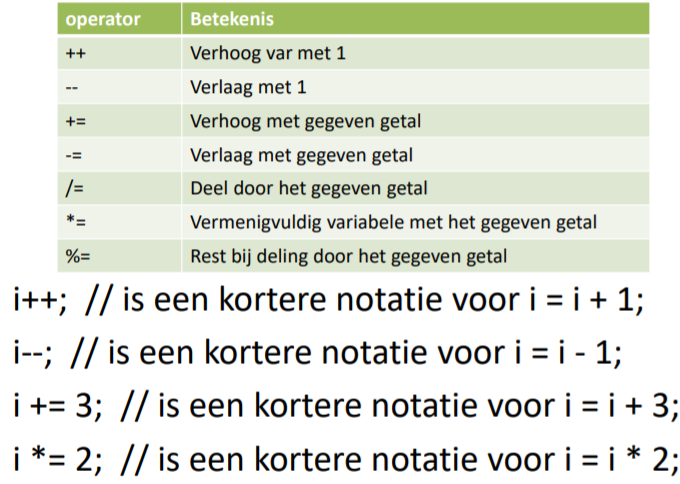
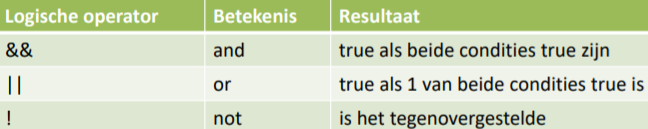
wanneer we Scanner typen en enteren voegt java automatisch de import toe vanboven

oef voornaam en achternaam:

Scanner input = new Scanner(System.*in*);  
System.*out*.print("geef voornaam in:");  
String voornaam = input.next();  
System.*out*.print("geef achternaam in");  
String achternaam = input.next();  
System.*out*.println(voornaam + achternaam);

input.close();

Print voor op dezelfde lijn input te typen, println om op de volgende lijn input te typen



µ

if (statement<statement) {

sout(“bullshit”)

} else {

}

Inhoud van 2 string vergelijken = equals() of waarde.equals

While (statement) {

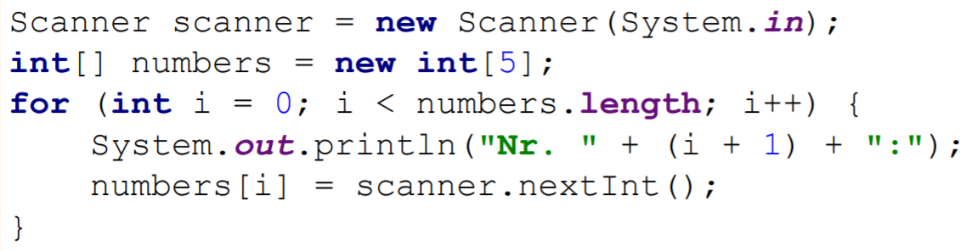
}

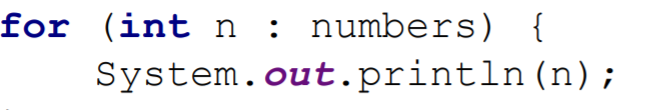
For (int count = 1; count < 5; count++ of count+3) { begint bij 1, blijft doorgaan als het kleiner dan 5 is en doet elke stap +1 ofte wel een ander getal

}

Om enkel 1 character in te laten lezen= Input.next().charAt(0)

Arrays: moeten allemaal van hetzelfde data type zijn (behalve bij array list)  
new = word gebruikt om objecten te creeren (dus new array) int [] numbers = new int[10]; reserveert 10 plaatsen maar mag ook een variabale zijn

Int[] numbers = {1, 3, 6, 9} of new int[] {1, 8, 12}

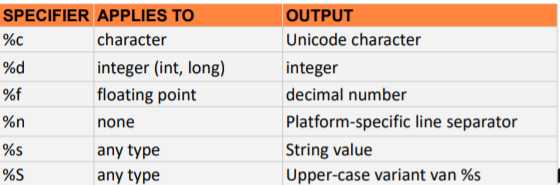


Om 123456789 deel per deel uit de printen =

**double** getal = 0;  
**for** (**int** i = 1; i <= 9; i++){  
 getal = getal \* 10 + i;

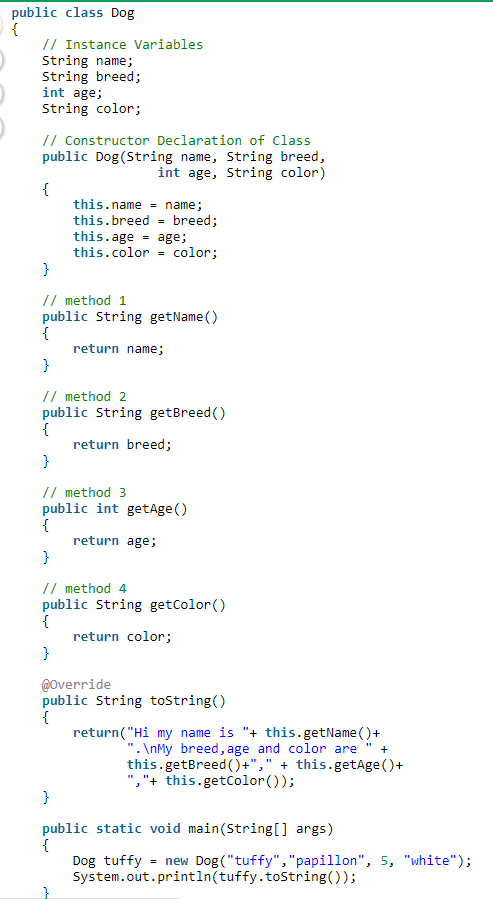
*getal = 0, en i = 1  
 0 \* 10 + 1 = 1  
 getal = 1, en i word dan 2  
 doen we 1 \* 10 + 2 = 12  
 getal = 12, en i word 3  
 12 + 10 = 120 + die 3 van i is 123*

String formatering



- = links uitlijnen

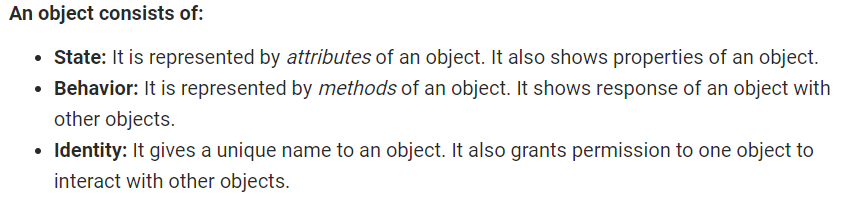
H2 <https://www.w3schools.com/java/java_classes.asp>

In klasses is er geen psvm, een programma bestaat uit meerdere mains

klassen staan boven objecten

objecten kunnen gemaakt worden op basis van een klasse

(klasse is een blauwdruk)



Constructor is dezelfde naam als onze klasse!

Default constructor is altijd aanwezig, maar er kan er extra  
aangemaakt worden. Let op als je een nieuwe constructor aanmaakt  
met variabalen dan verwijdert java de default. Willen we 2 construct-  
ors maken met 1 default moeten we die zelf aanmaken.

Default constr =   
public ‘naam’ () {

}

Eigen contstr =   
public ‘zelfdenaam’ (int x, int y, int z…..){

Je voegt gwn waardes hierin toe, die deze opvraagt }

Code –> generate-> …

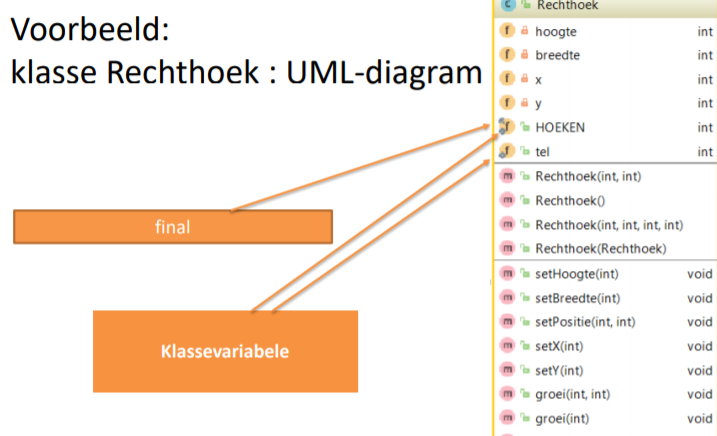
Klasse = zelfde voor alle onderdelen (bv standaard hoeveelheid wielen) worden altijd met STATIC aangemaakt dus “private **static** int wielen = 4” you cant have more or less…

Instantievariabele = ver anderlijke, als in namen, personen, voorwerpen

Klasses kunnen niet aangepast worden en gelden dus voor heel het programma.

Wanneer we “PRIVATE STATIC FINAL INT naam = 4” aanmaken betekent het dat deze nooit of te nimmer aangepast kunnen worden. Zal foutmeldingen geven wanneer we hoeveelheid willen aanpassen, ***deze opvragen voor te printen = Rechthoek.naam*** (de naam hoort als static altijd in caps te zijn.) we roepen de overkoepelende naam op van de klasse NIET rechthoekzoveel.naam!!!

Plus importante: we krijgen onze examenvragen zo, dus moeten we weten wanneer een static aan te maken en wanneer een final aan te maken. Klasse = enkel STATIC, de final in deze is STATIC FINAL



Bij een constructor geen teller++ in de default, enkel een teller in de constructor met opgeroepen variabelen.

Een constructor voor een vorige ingave te copien:

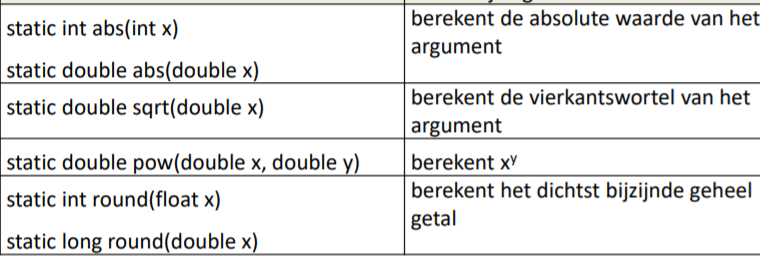
**public** Tijd(**Tijd** bestaande) {**this**(bestaande.uren, bestaande.minuten, bestaande.seconden); }

declareren het classe programma in onze variabele naam

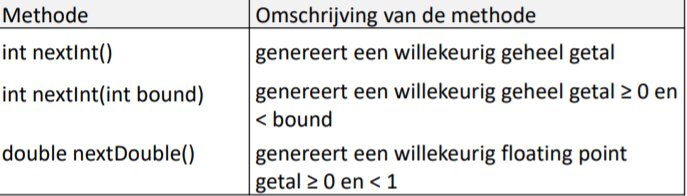
**H3**

Math oproepen met Math.naamprogr.   
ingave met een getal tot de 3de , dat afgerond wordt:

**Scanner** input = **new** Scanner(**System**.in);  
**System**.out.println("geef getal in: ");  
**int** getal = input.nextInt();  
**System**.out.println(**Math**.*round*(**Math**.*pow*(getal, 3)));

Math.abs, sqrt…….

Op 2 getallen na de komma afronden:   
**System**.out.println(**Math**.*round*(getal \* 100) / 100.0);

Random:

Java.util.Random randomgetallen = new java.util.Random();

Of import vanboven zoals bij scanner en declaratie vanonder;  
 Random rand = new Random();

nextInt = bovengrens niet inc, 0 -5 is eigl 0 – 4

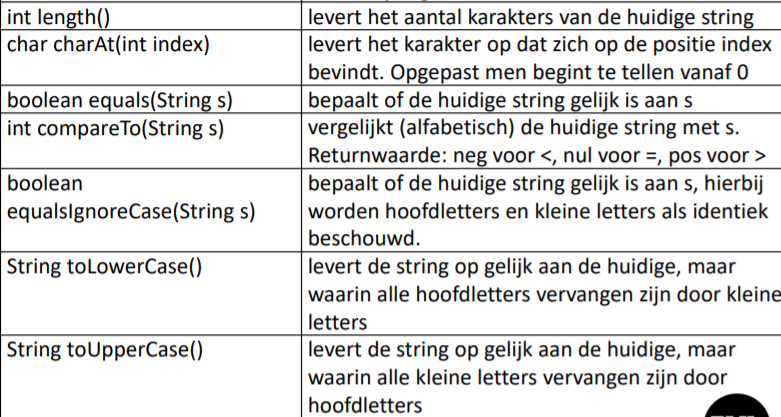
getal2 = rand.nextInt(10) + 1;

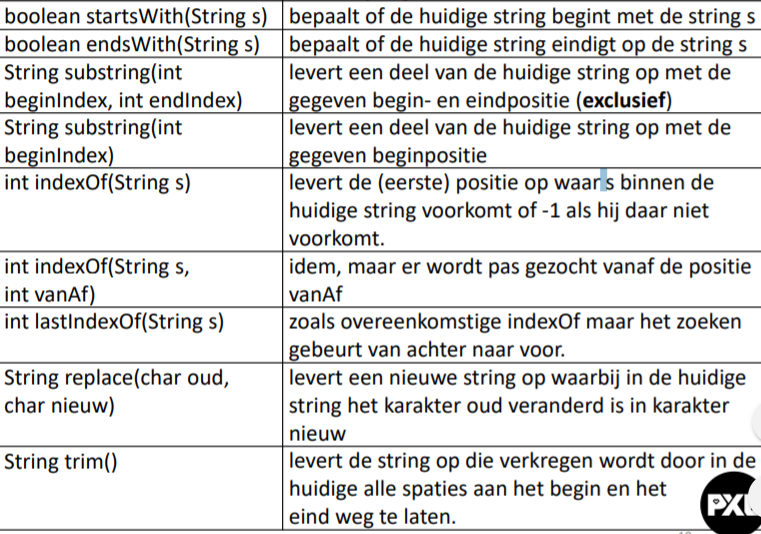
om de bound 0 NIET inc te krijgen, en 10 wel.

1 willekeurige char. Ascii a – z is getal 97 tot 122 so:

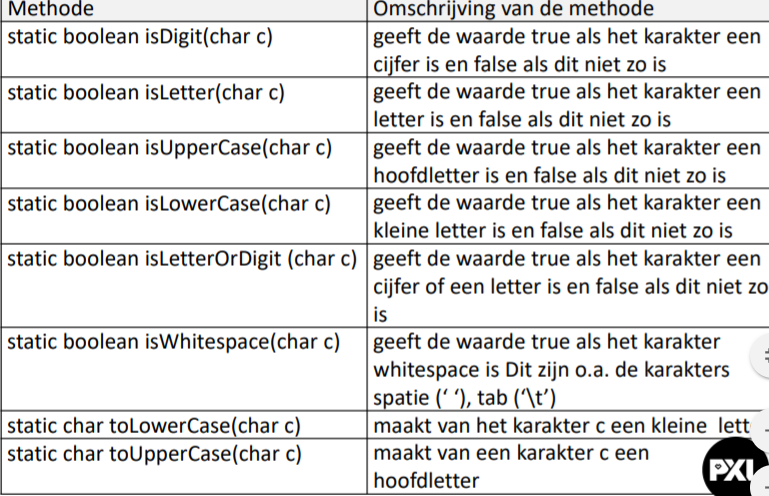
**char** getal4 = (**char**)(rand.nextInt(26) + 97);  
**System**.out.println(getal4);

strings zijn niet veranderbaar, om deze aan te passen horen we nieuwe variabelen aan te maken om de Naam.toUpperCase etc. te gebruiken. **String** str ="abc";

**String** str2 = str.toUpperCase();

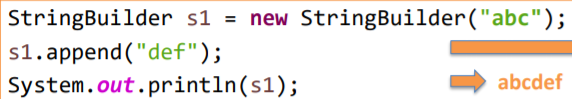


.equals(); om strings te vergelijken niet ==!! == vergelijkt opgeslagen plaats van de string

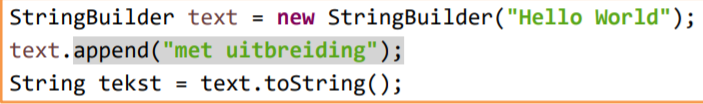


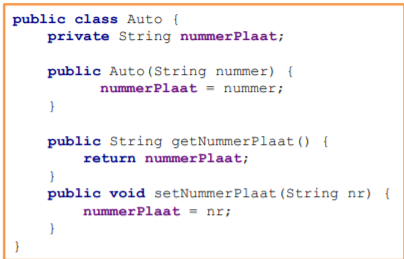
Chars:

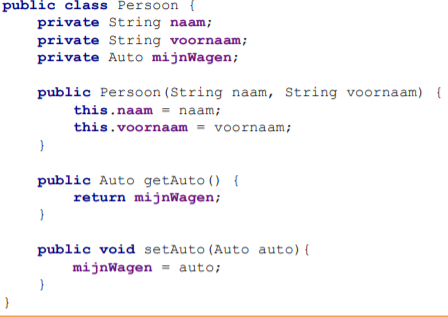
Stringbuilder: wanneer we een string aanpassen worden oude strings opgeslagen, geheugenverlies dus, stringbuilder behoud huidige string en voegt nieuw deel toe.



Best van stringbuilder – toString() om bv naar uppercase of andere methoden van string te gebruiken

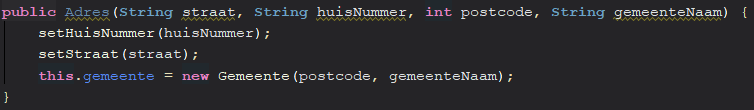


H5 associaties

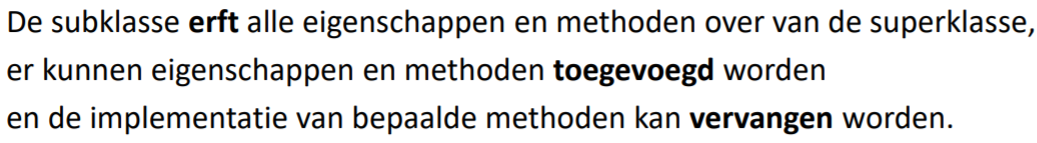


Klasse naar een andere klassen ^^ ^^ ^^   
gebruik de get en set.

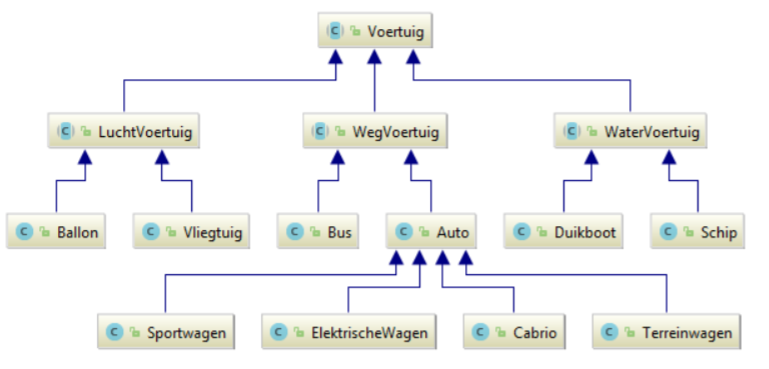
Om een een klasse in een nieuwe klasse constructor op te roepen zonder een SET; de NEW!!!

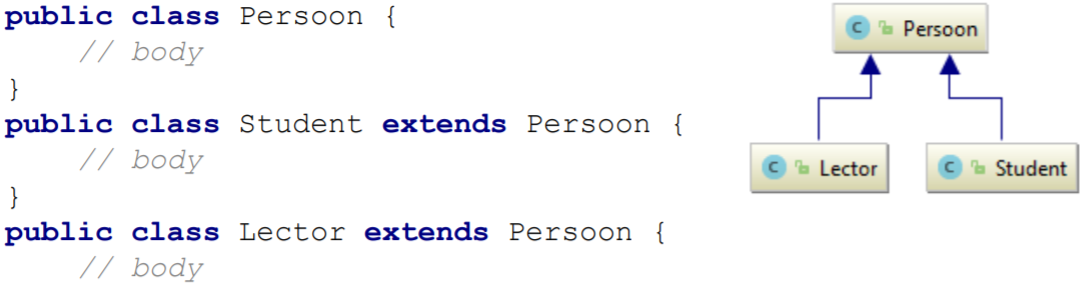


H6

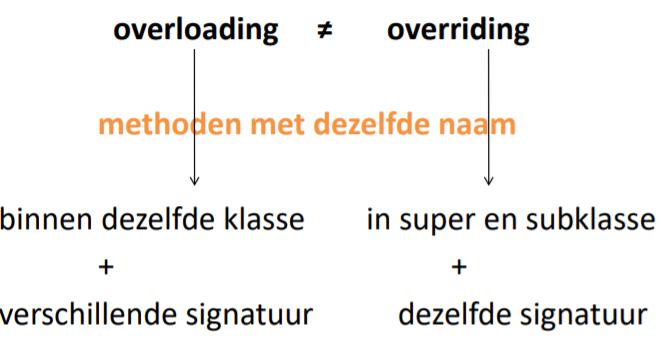


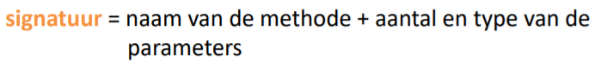
@override om een bestaande methode te overschrijven en niet aan te passen in de subklasse, anders wordt er een nieuwe methode aangemaakt in superklasse

abstracte klasse looks like dis:  
dit is een overkoepelende klasse  
zitten ‘standaard’ waarde  
in die subklassen kunnen  
gebruiken, met abstracte klasses  
kunnen we moeilijk een app maken, te weinig info

aanmaken:

er is maar 1 superklasse!

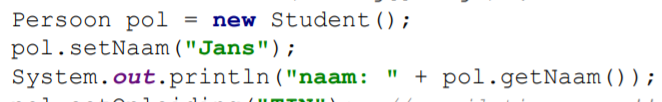


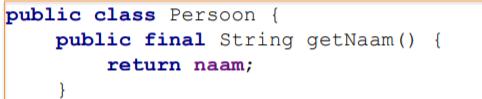


Om override te gebruiken: bv toString(int “ “, String “ “) zo overnemen  
van superklasse!

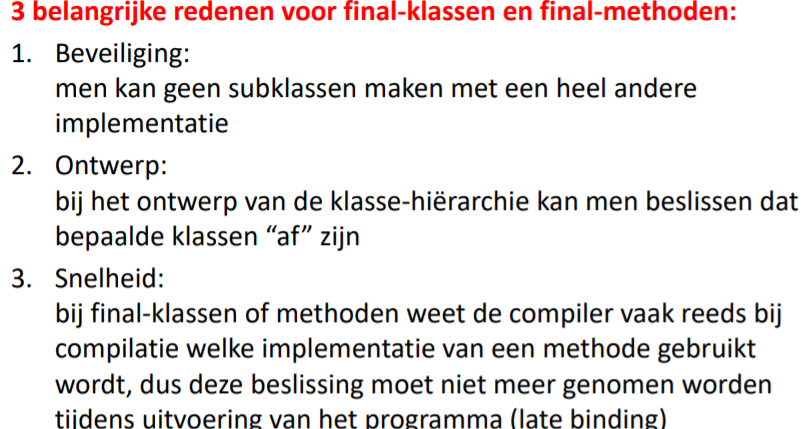
Print vanuit een superklasse gebruiken, zonder dit krijgen we foutmelding omdat alles niet static is



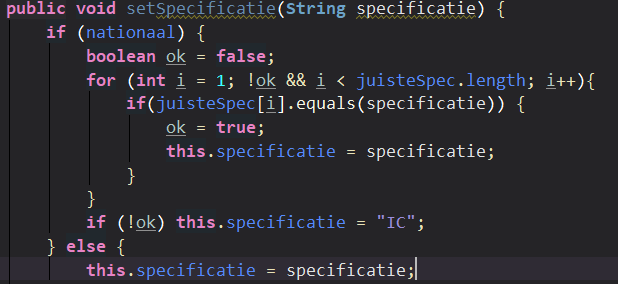


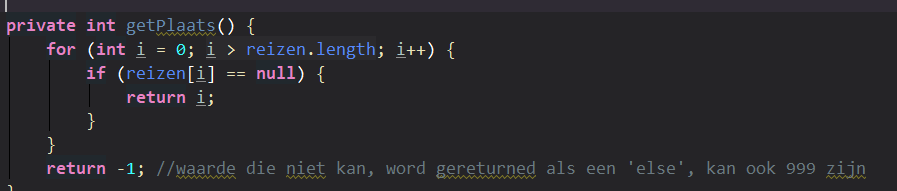
Polymorph ^^^^ fucking Lulu

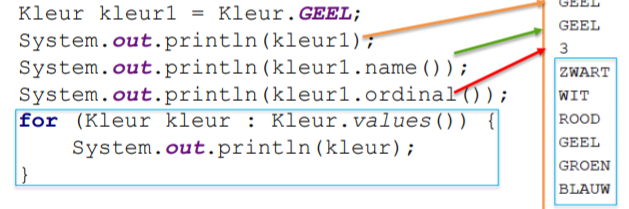
Final methoden:

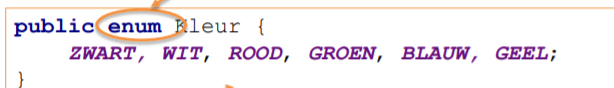
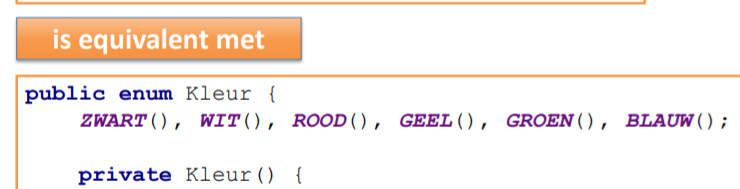


Array doorlopen en stoppen midden in de for loop wanneer obj gevonden is:



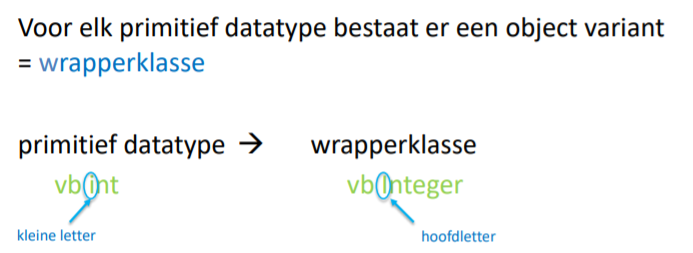
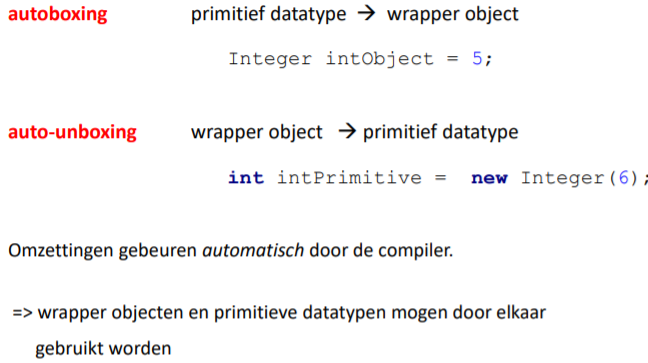






Ordinal = positie van de enum

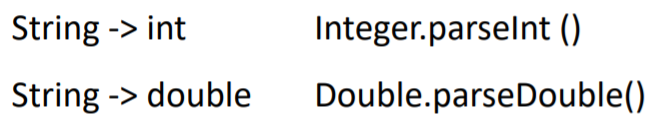
**H8**

****

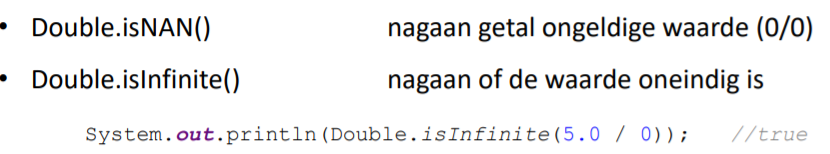
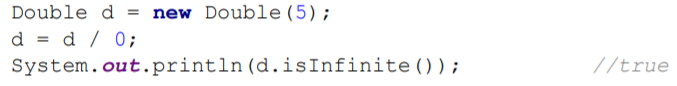
Sout heeft altijd een ToString erin automatisch  
tostring en equals zijn voorgeprogrammeerd in Object (superklasse)

Wrapper obj zijn immutable zoals strings waardoor er achterliggend elke keer nieuwe obj worden aangemaakt

**Overloading en overriding = @override overschrijven in superklasse = examenvraag**

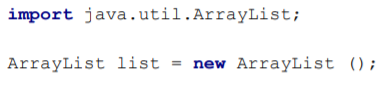
zelfde naam maar verschillende parameters

Parse = “123” 🡪 maakt van deze string 123

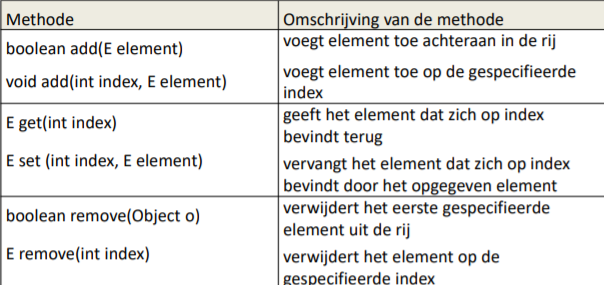
 Deze werken ook op een niet-klasse!



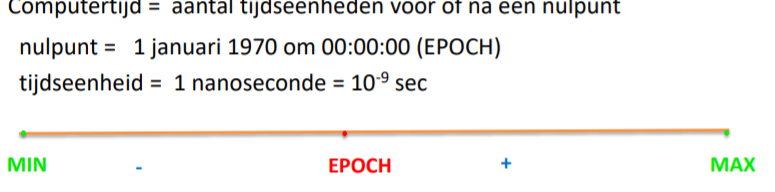
Faculteit van een getal is het getal \* alle onderliggende getalen doen= 5 wordt dan 5 \*4, \*3, \*2, \*1

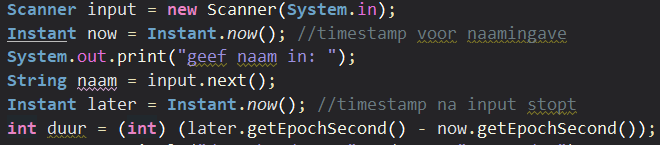
 Arraylists:

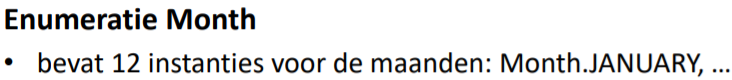
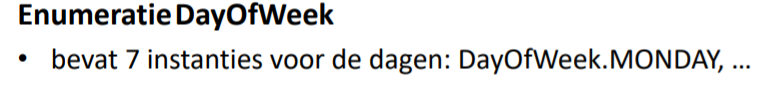
 Of java.util.\*;

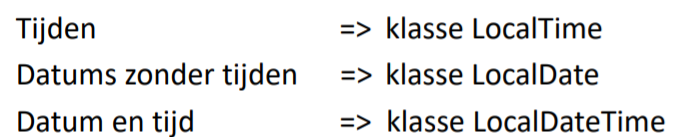
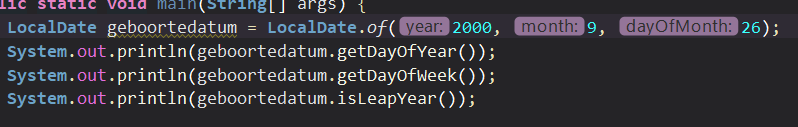


Datums en tijden; import java.time

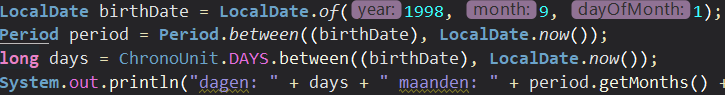


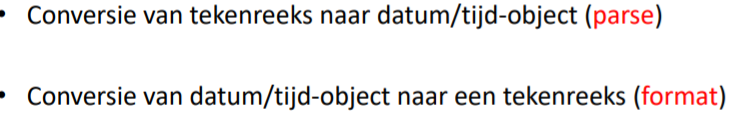
Kunnen geen Datum datum = new Datum…. Aanmaken zoals de andere klasses  
maar bv Instant now = Instant.now(); ---- maakt een ‘timestamp’ op moment dat java hierover komt. Om te berekenen hoelang een programma duurt of een ingave in dit geval:  


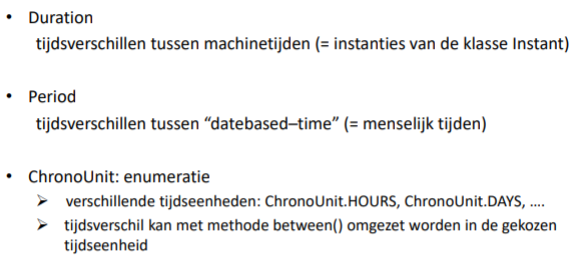




They made so many classes for us twankyouw jwava UwU

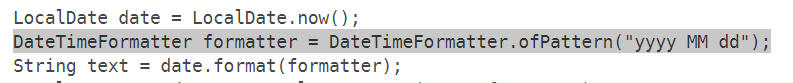


Chronounit is volledige tijd tussen 2 stamps, in dagen of …  
hier de period.months of years telt enkel aantal maanden   
verschil in dat jaar, dus bv hier  
4, niet het totaal aantal maanden tussen 2 stamps

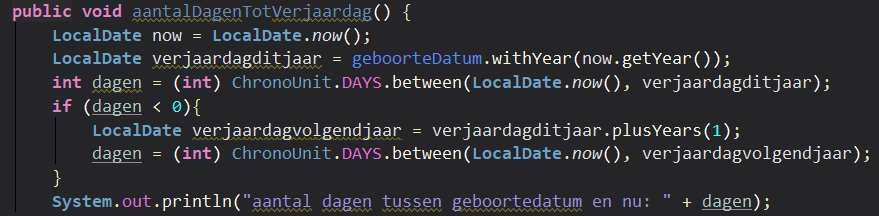




Dollarteken verwijst naar die ene datum, ipv t 6 keer te typen

procenttekens zorgen voor ‘opvulbare’ gaten SEE PSVMFFFFF

aantal dagen tot volgende verjaardag + check wanneer verjaardag al geweest is (uitkomst was dan -)

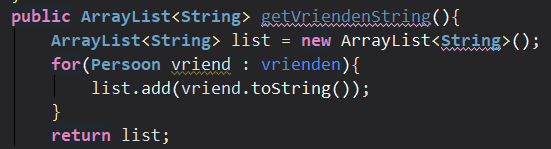


Checken of bepaald item, in dit geval naam van Persoon al in een arraylist zit:

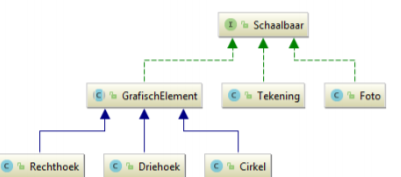
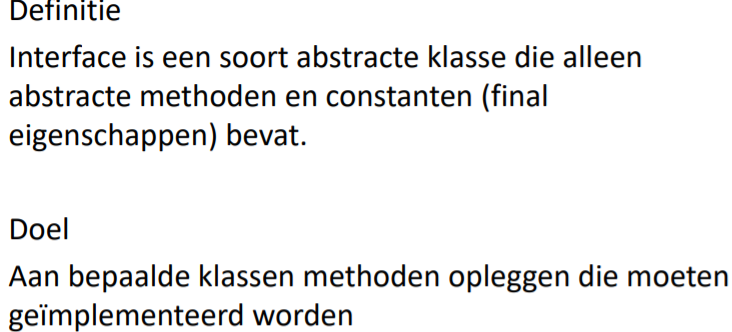
Uitkomst van indexof wanneer niet gevonden is -1

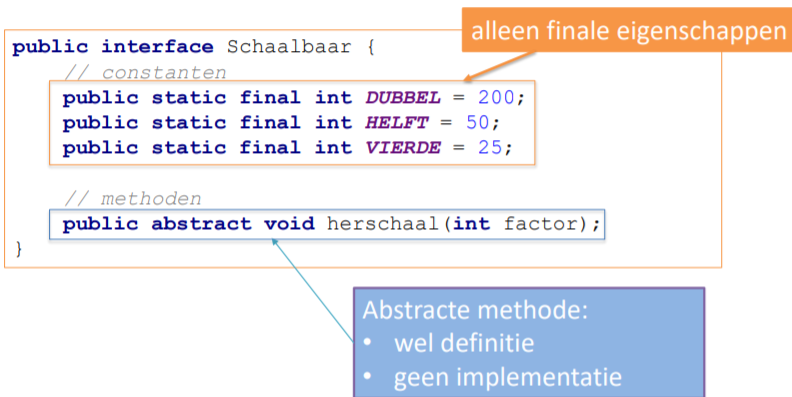
**public void** VoegVriendToe(**Persoon** p) {  
 **if** (vrienden.size() >= MAX\_AANTAL\_VRIENDEN) {  
 **System**.out.println("ya full");  
 } **else if** (  
 vrienden.indexOf(p) != -1) {  
 **System**.out.println("deze naam is al toegevoegd");  
 } **else** {  
 vrienden.add(p);  
 }

arraylist in strings omzetten en zo terug geven:



H9

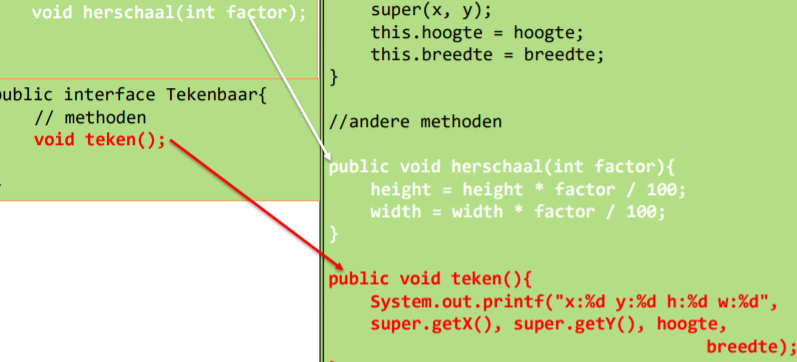
in uml schema aangegeven met streepjeslijn



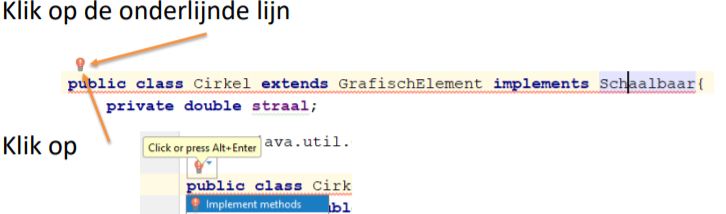
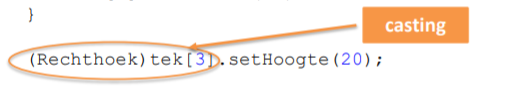
Enkel public!! Of package

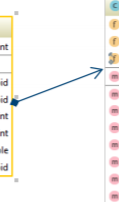


Altijd implement gebruiken



Methodes moeten altijd in klasse of psvm uitgewerkt worden,  
in interface enkel naamgeving



associatie HAS A verband zie H5 niet extenden

Macht tot de, is Math.pow(2 of meer, getal of formule)



= **double** afstand = **Math**.*sqrt*(**Math**.*pow*(2,(**this**.x - punt2.getX())) + **Math**.*pow*(2,(**this**.y - punt2.getY())));

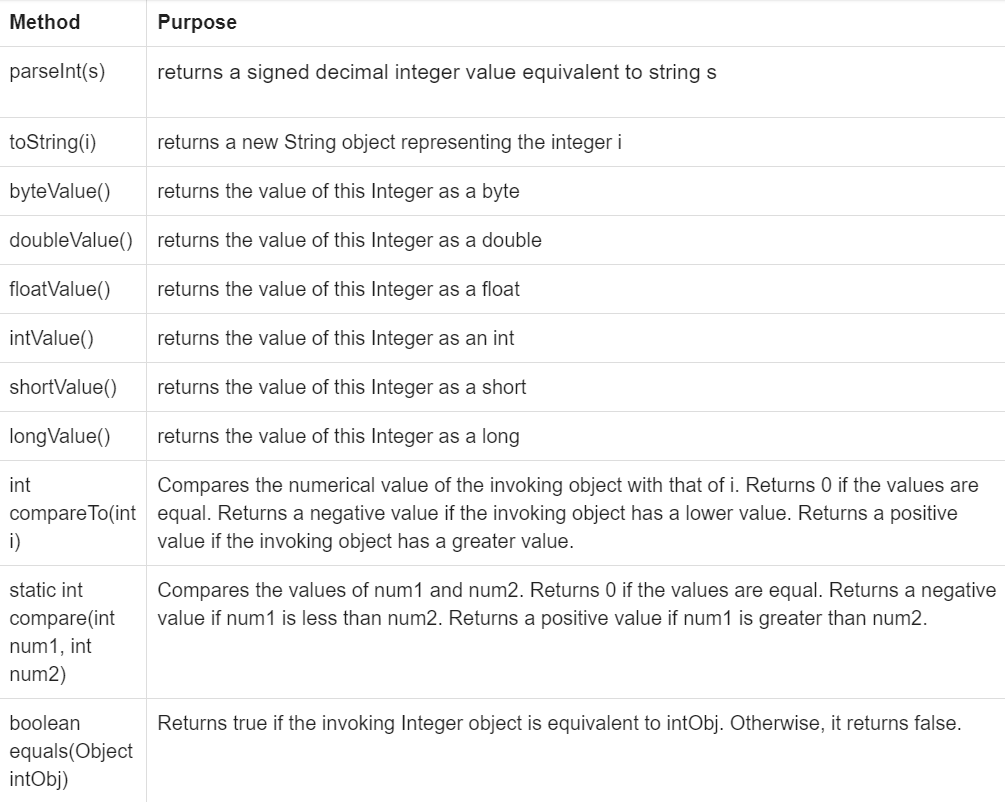
 abstracte methode wordt niet uitgewerkt

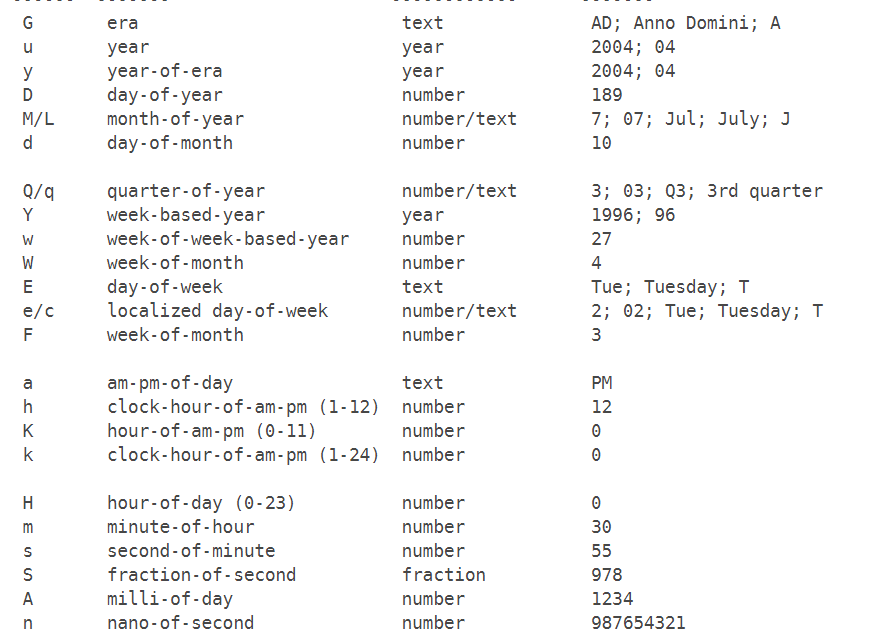
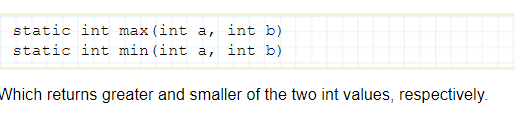
Is a -> extends

 ---

^^ abstract symbooltje

Extra wrapper functies:





// A class defines the attributes (fields) and capabilities (methods) of a real world object

public class Animal {

// static means this number is shared by all objects of type Animal

// final means that this value can't be changed

public static final double FAVNUMBER = 1.6180;

// Variables (Fields) start with a letter, underscore or $

// Private fields can only be accessed by other methods in the class

// Strings are objects that hold a series of characters

private String name;

// An integer can hold values from -2 ^ 31 to (2 ^ 31) -1

private int weight;

// Booleans have a value of true or false

private boolean hasOwner = false;

// Bytes can hold the values between -128 to 127

private byte age;

// Longs can hold the values between -2 ^ 63 to (2 ^ 63) - 1

private long uniqueID;

// Chars are unsigned ints that represent UTF-16 codes from 0 to 65,535

private char favoriteChar;

// Doubles are 64 bit IEEE 754 floating points with decimal values

private double speed;

// Floats are 32 bit IEEE 754 floating points with decimal values

private float height;

// Static variables have the same value for every object

// Any variable or function that doesn't make sense for an object to have should be made static

// protected means that this value can only be accessed by other code in the same package

// or by subclasses in other packages

protected static int numberOfAnimals = 0;

// A Scanner object allows you to except user input from the keyboard

static Scanner userInput = new Scanner(System.in);

// Any time an Animal object is created this function called the constructor is called

// to initialize the object

public Animal(){

// Shorthand for numberOfAnimals = numberOfAnimals + 1;

numberOfAnimals++;

int sumOfNumbers = 5 + 1;

System.out.println("5 + 1 = " + sumOfNumbers);

int diffOfNumbers = 5 - 1;

System.out.println("5 - 1 = " + diffOfNumbers);

int multOfNumbers = 5 \* 1;

System.out.println("5 \* 1 = " + multOfNumbers);

int divOfNumbers = 5 / 1;

System.out.println("5 / 1 = " + divOfNumbers);

int modOfNumbers = 5 % 3;

System.out.println("5 % 3 = " + modOfNumbers);

// print is used to print to the screen, but it doesn't end with a newline \n

System.out.print("Enter the name: \n");

// The if statement performs the actions between the { } if the condition is true

// userInput.hasNextLine() returns true if a String was entered in the keyboard

if(userInput.hasNextLine()){

// this provides you with a way to refer to the object itself

// userInput.nextLine() returns the value that was entered at the keyboard

this.setName(userInput.nextLine());

// hasNextInt, hasNextFloat, hasNextDouble, hasNextBoolean, hasNextByte,

// hasNextLong, nextInt, nextDouble, nextFloat, nextBoolean, etc.

}

this.setFavoriteChar();

this.setUniqueID();

}

// It is good to use getter and setter methods so that you can protect your data

// In Eclipse Right Click -> Source -> Generate Getter and Setters

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getWeight() {

return weight;

}

public void setWeight(int weight) {

this.weight = weight;

}

public boolean isHasOwner() {

return hasOwner;

}

public void setHasOwner(boolean hasOwner) {

this.hasOwner = hasOwner;

}

public byte getAge() {

return age;

}

public void setAge(byte age) {

this.age = age;

}

public long getUniqueID() {

return uniqueID;

}

// Method overloading allows you to accept different input with the same method name

public void setUniqueID(long uniqueID) {

this.uniqueID = uniqueID;

System.out.println("Unique ID set to: " + this.uniqueID);

}

public void setUniqueID() {

long minNumber = 1;

long maxNumber = 1000000;

// Generates a random number between 1 and 1000000

this.uniqueID = minNumber + (long)(Math.random() \* ((maxNumber - minNumber) + 1));

// You can cast from one primitive value into another by putting what you want between ( )

// (byte) (short) (long) (double)

// (float), (boolean) & (char) don't work.

// (char) stays as a number instead of a character

// You convert from a primitive to a string like this

String stringNumber = Long.toString(maxNumber);

// Byte.toString(bigByte); Short.toString(bigShort); Integer.toString(bigInt);

// Float.toString(bigFloat); Double.toString(bigDouble); Boolean.toString(trueOrFalse);

// You convert from a String to a primitive like this

int numberString = Integer.parseInt(stringNumber);

// parseShort, parseLong, parseByte, parseFloat, parseDouble, parseBoolean

System.out.println("Unique ID set to: " + this.uniqueID);

}

public char getFavoriteChar() {

return favoriteChar;

}

public void setFavoriteChar(char favoriteChar) {

this.favoriteChar = favoriteChar;

}

public void setFavoriteChar() {

int randomNumber = (int) (Math.random() \* 126) + 1;

this.favoriteChar = (char) randomNumber;

// if then else statement

// > < == != >= <=

if(randomNumber == 32){

System.out.println("Favorite character set to: Space");

} else if(randomNumber == 10){

System.out.println("Favorite character set to: New Line");

} else {

System.out.println("Favorite character set to: " + this.favoriteChar);

}

// Logical operators

// ! : Converts the boolean value to its right to its opposite form ie. true to false

// & : Returns true if boolean value on the right and left are both true (Always evaluates both boolean values)

// && : Returns true if boolean value on the right and left are both true (Stops evaluating after first false)

// | : Returns true if either boolean value on the right or left are true (Always evaluates both boolean values)

// || : Returns true if either boolean value on the right or left are true (Stops evaluating after first true)

// ^ : Returns true if there is 1 true and 1 false boolean value on the right or left

if((randomNumber > 97) && (randomNumber < 122)){

System.out.println("Favorite character is a lowercase letter");

}

if(((randomNumber > 97) && (randomNumber < 122)) || ((randomNumber > 64) && (randomNumber < 91))){

System.out.println("Favorite character is a letter");

}

if(!false){

System.out.println("I turned false to " + !false);

}

// The ternary operator assigns one or another value based on a condition

int whichIsBigger = (50 > randomNumber) ? 50 : randomNumber;

System.out.println("The biggest number is " + whichIsBigger);

// The switch statement is great for when you have a limited number of values

// and the values are int, byte, or char unless you have Java 7 which allows Strings

switch(randomNumber){

case 8 :

System.out.println("Favorite character set to: Backspace");

break;

case 9 :

System.out.println("Favorite character set to: Horizontal Tab");

break;

case 10 :

case 11 :

case 12 :

System.out.println("Favorite character set to: Something else weird");

break;

default :

System.out.println("Favorite character set to: " + this.favoriteChar);

break;

}

}

public double getSpeed() {

return speed;

}

public void setSpeed(double speed) {

this.speed = speed;

}

public float getHeight() {

return height;

}

public void setHeight(float height) {

this.height = height;

}

protected static int getNumberOfAnimals() {

return numberOfAnimals;

}

// Since numberOfAnimals is Static you must set the value using the class name

public void setNumberOfAnimals(int numberOfAnimals) {

Animal.numberOfAnimals = numberOfAnimals;

}

protected static void countTo(int startingNumber){

for(int i = startingNumber; i <= 100; i++){

// continue is used to skip 1 iteration of the loop

if(i == 90) continue;

System.out.println(i);

}

}

protected static String printNumbers(int maxNumbers){

int i = 1;

while(i < (maxNumbers / 2)){

System.out.println(i);

i++;

// This isn't needed, but if you want to jump out of a loop use break

if(i == (maxNumbers/2)) break;

}

Animal.countTo(maxNumbers/2);

// You can return a value like this

return "End of printNumbers()";

}

protected static void guessMyNumber(){

int number;

// Do while loops are used when you want to execute the code in the braces at least once

do {

System.out.println("Guess my number up to 100");

// If what they entered isn't a number send a warning

while(!userInput.hasNextInt()){

String numberEntered = userInput.next();

System.out.printf("%s is not a number\n", numberEntered);

}

number = userInput.nextInt();

}while(number != 50);

System.out.println("Yes the number was 50");

}

// This will be used to demonstrate polymorphism

public String makeSound(){

return "Grrrr";

}

// With polymorphism we can refer to any Animal and yet use overridden methods

// in the specific animal type

public static void speakAnimal(Animal randAnimal){

System.out.println("Animal says " + randAnimal.makeSound());

}

// public allows other classes to use this method

// static means that only a class can call for this to execute

// void means it doesn't return a value when it finishes executing

// This method can except Strings that can be stored in the String array args when it is executed

public static void main(String[] args){

Animal theDog = new Animal();

System.out.println("The animal is named " + theDog.getName());

System.out.println(Animal.printNumbers(100));

Animal.countTo(100);

Animal.guessMyNumber();

// An array is a fixed series of boxes that contain multiple values of the same data type

// How you create arrays

// int[] favoriteNumbers;

// favoriteNumbers = new int[20];

int[] favoriteNumbers = new int[20];

favoriteNumbers[0] = 100;

String[] stringArray = {"Random", "Words", "Here"};

// for(dataType[] varForRow : arrayName)

for(String word : stringArray)

{

System.out.println(word);

}

// This is a multidimensional array

String[][][] arrayName = { { { "000" }, { "100" }, { "200" }, { "300" } },

{ { "010" }, { "110" }, { "210" }, { "310" } },

{ { "020" }, { "120" }, { "220" }, { "320" } }};

for(int i = 0; i < arrayName.length; i++)

{

for(int j = 0; j < arrayName[i].length; j++)

{

for(int k = 0; k < arrayName[i][j].length; k++)

{

System.out.print("| " + arrayName[i][j][k] + " ");

}

}

System.out.println("|");

}

// You can copy an array (stringToCopy, indexes to copy)

String[] cloneOfArray = Arrays.copyOf(stringArray, 3);

// You can print out the whole array

System.out.println(Arrays.toString(cloneOfArray));

// Returns the index or a negative number

System.out.println(Arrays.binarySearch(cloneOfArray, "Random"));

}

}