Arrays

```
Animal [] zoo = new Animal [4];
zoo [0] = new Tiger();
zoo [1] = new Giraffe();
…

String [] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);                    //Outputs 4

int [][] myNumbers = {{1, 2, 3, 4}, {4, 5, 6}};
int x = myNumbers [1][2];
System.out.println(x);                              //Outputs 6

Arrays.sort(cars);
System.out.println(Arrays.toString(cars));
          //[BMW, Ford, Mazda, Volvo]

SORTING OBJECTS WITH MULTIPLE PARAMETERS:
NATURAL SORTING                 (Created within class)
  public class Person implements Comparable<Person>{…
  @Override
  public int compareTo(Person o) {
  return Double.compare(this.weight, o2.weight);
  }                                         ----> Use wrapperclass

Arrays.sort(listOfPeople);    /         Collections.sort(...);

ALTERNATIVE SORTING             (Created in sepparate class)
  public class SortOnName implements Comparator<Person>{
  @Override
  public int compare(Person o1, Person o2) {
  return o1.getName().compareTo(o2.getName());
  }

Arrays.sort(listOfPeople, new SortOnName());
```

Arraylist

```
import java.util.ArrayList;
import java.util.Collections;

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

    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}

cars.get(0);                    //Acces an item
cars.set(0, “Opel”);            //Change an item
cars.remove(0);                 //Remove an item
cars.clear();                   //Clear full list
cars.size();                    //Find out number of ellements

Collections.sort(cars);         // Sort cars

for (String i : cars) {
   System.out.println(i);
}
```

Converting Array to ArrayList

```
import java.util.Arrays;

String [] names = {"John", "Jack", "Jill", "Jane"};

List<String> list = Arrays.asList(names);
```

Math

```
Math.random();      //Random nr between 0.0(excl) and 1.0 (excl)

int randomNum = (int)(Math.random() * 101);       // 0 to 100

Math.sqrt(64);                  //Returns square root

a >= b                          //Greater than or equal to
a == b                          //Equal to
a != b                          //Not equal to

int nr ++;                      //nr + 1
int nr += 5;                    //nr + 5
```

Randomize list

```
import java.util.Collections;

ArrayList<String> mylist = new ArrayList<String>();
myList.add("One");
mylist.add("Two");
myList.add("Three");

Collections.shuffle(myList);            //[Two, One; Three]
```

Break / Continue

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;                   //This skips the value of 4
  }
  if (i == 6) {
    break;                      //This jumps out of the for loop
  }
}
```

If...Else

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning!");
} else if (time < 20) {
  System.out.println("Good day!");
} else {
  System.out.println("Good evening!");
}
                                //Outputs "Good evening!"

variable = (condition) ? expressionTrue :  expressionFalse;

int tine = 20;
String result = (time < 18) ? "Good day!" : "Good evening!";
System.out.println(result);
                                //Outputs "Good evening!"
```

Switch...Case

```
int day = 4;
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednessday");
    break;
  case 4:
    System.out.println("Thursday");
    break;
  case 5:
    System.out.println("Other day");
    break;
 }
```

While loop

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

Do - While loop

```
int i = 0;
do {
  System.out.println(i);
  i++
}
while (i < 5);
```

For loop

```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

For - Each loop

```
for (type variableName : arrayName) {
  // code to be excecuted
}

String [] cars = {"Volvo", "BMW", "Ford"};
for (String i : cars) {
  System.out.println(i);
}
```

Itterating

```
Iterator<String> iter = naamArrayList.iterator();

while (iter.hasNext()) {
  System.out.println(iter.next().toString());
}
```

String methods

```
String name1 = "cheatsheet";
String name2 = "exam";

name1.length();                         //10
name2.concat(name1);                    //examcheatsheet
name1.equals("cheatsheet");             //true
name1.equalsIgnoreCase(name2);          //false
name1.indexOf('e');                     //2
name1.charAt(0);                        //'c'
name1.toCharArray();
          char [] test = name1.toCharArray();
test.toString();
name1.replace(int 1, int 2, String);
          //Change chars from pos1 to pos2 by string
```

Exceptions

```
try {
  // Block of code to try
}
catch (Exception e) {
  // Block of code to handle errors
          ex: System.out.println(e.getMessage());
}
finally {
    try {
      // Block of code to try
    }
    catch (Ecxeption o) {
      // Block of code to handle errors
    }
```

TO CREATE SPECIFIC EXCEPTIONCLASS:

```
public class MijnException extends Exception{
    public MijnException (){
        super("DezeTekstAlsErrorBvb");
    }
}
```

THROW THE EXCEPTION UP:

```
public boolean check (Persoon x) throws MijnException{
    if (x.getAchternaam().equals(y.getAchternaam())){
        return true;
    }else {
        throw new MijnException();
    }
}
```

Type casting

Widening Casting (automatically) - converting a smaller type to a larger type size
byte -> short -> char -> int -> long -> float -> double

```
int myInt = 9;
double myDouble = myInt;      //Automatic casting: int to double
```

Narrowing Casting (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char -> short -> byte

```
double myDouble = 9.78;
int myInt = (int) myDouble;   //Manual casting: double to int
```

Upcasting / downcasting

```
//UPcasting:
ParentClass name1 = new ChildClass(Parameters);

Animal name1 = new Cat("Garfield");
name1.makeSound();                      //Outputs "miauw"
          //Can only acces methods defined in Parent class
            and override-methods in Child class

//DOWNcasting: only works on upcasted objects
ChildClass name2 = (ChildClass) name1;

Cat name2 = (Cat) name1;
name2.makeSound();                      //outputs "miauw"
          //Can acces all methods defined in Child class
            AND all methods in Parent class

---> Practical use to avoid errors:
if (name1 instanceof ChildClass) {
  ChildClass name2 = (ChildClass) name1;
  name2.makeSound();
  name2.uniqueChildClassMethod();
}
```

Public - Default - Protected – Private

```
public              Visible for all
default             Visible only in package
protected           Visible in class- and subclasses
private             Visible only in class
```

Keywords

```
ABSTRACT:
  public abstract class ParentClass { …
          //Blocks creating object ParentClass
          ----->  Required use of ChildClasses
FINAL:
  //Attribute       Once value is obtained, it can't be changed
          private final int a = 4;
  //Methods         Cannot be overriden by subclasses
          public final void doSomething() {…}
  //Class           Cannot be used for Childclasses (no extends)
          public final class LockedClass { …

INTERFACE:
  public interface dontForget { …
    void method1 ();
  }
          //NO attribute
          //NO constructors
          //ONLY methods without body
          //Uses implements  (NOT extends)

STATIC:
  private static long counter;
          //Fixed value for this class AND all subclasses

SUPER:
  super()                     //Calls default constructor
  super(Parameter...)         //Calls matching constructor
  !Must be first line in constructor!
```

Stacks

```
import java.util.Stack;

Stack <String> games = new Stack <String>();
games.add("Call of Duty");
games.add("Guitar hero");
games.add("Dragonball Z");

       bottom                 middle                  top
//[  Call of Duty,          Guitar Hero,          Dragonball Z ]

games.pop();                  //Takes (removes) upper from list
games.peek();                 //Shows upper from list
games.contains("");           //Boolean
games.get(0);                 //Use according to index
```

Queue

```
import java.util.Queue

Queue <String> bbqLine = new LinkedList<String>();
bbqLine.add("Jackson");
bbqLine.add("Jason");
bbqLine.add("Johnson");

//[ Jackson, Jason, Johnson ]

bbqLine.poll();               //Takes (removes) first
bbqLine.peek();               //Shows first
```

Collections

List
```
  ArrayList         //List in order of added elements
  LinkedList        //List of linked lists
```

Set
```
  HashSet           //Unsorted
  TreeSet           //Sorted            !elements use Comparable
                                         interface
```

Map
```
  HashMap           //Unsorted
  TreeMap           //Sorted by key     !key class implemented
                                         with Comparable
```