Java Introduction

Name of the file

```
public class HelloWorld
{

    public static void main (String[] args)
    {
        System.out.println ("Hello World");
    }
}
```

Declare new class

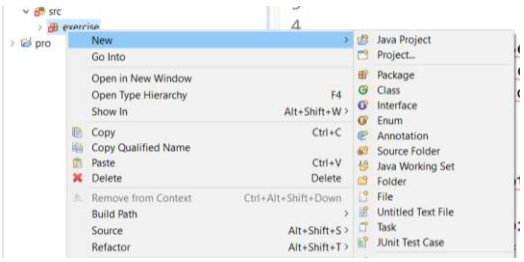
This line defines main method

→ it is important to run the class

- Capital letters and small letters are different in Java (Case Sensitive).
- Naming: Capital each letter of each word and don't use spaces.

Packages

File → New → Package (In Src folder)



- It is used to define different workspaces in your class.
- A program for two cases → use them.

Notes

- To change the font of the editor:

  Window → Preferences → General → Appearance → Colors and fonts → Java editor text font
- To make a comment:

  /* then enter Or //

Variables in Java

- Definition of variables: Names we declare to store a value
- Use All capital letters for name and underscore between words.
- Declaring a variable:

```
int __ = __;
```

Name of variable ← → Value

To print the value on the screen:

```
System.out.println (name);
```

Data types

Byte	(number, 1-byte) "smallest"
Short	(number, 2-byte)
Int	(number, 4-byte)
Long	(number, 8-byte)
float	(float_number, 4-byte)
double	(number, 8-byte) "It can store decimal numbers"
Char	(number, 2-byte)
Boolean	(true or false, 1-byte)

Note:

```
float __ = (float) 4.5
```

- When using double no need for the ( )
- To define char:

```
char __ = '    ';
```

Scanner class

```
Scanner NameOfVar = new Scanner (System.in);
```

This line allows us to take an input from the user.

- To avoid error in Scanner: input Scanner (right click on Scanner).
- To ask the user to enter a value

```
int __ = NameofVar.nextInt();
```

Input ← → Name → You can put any type but with Capital. letter at first

Note:

.print is different from .println: it prints without breaking the line.

- If you are entering a string use:

```
NameOfVar.nextline();
```

- To make your answer like this:

```
Answer = _____;
```

From program

You can use:

```
int answer = ___ ;
System.out.println("Answer = " + answer);
```

Math Operators

Division:

- Make sure that the answer type matches the answer wanted.
- Note: the remainder is from (%)

Incrementation:

-

```
x = x + 1, OR x = ++x, OR x = x++
```

Post-incrementation

Pre-incrementation

-

```
x+ = 5 → x = x + 5
```

Short Format

Loops

- While:

statement

```
While (    )
 {
    ___________;
 }
```

- Do – While:

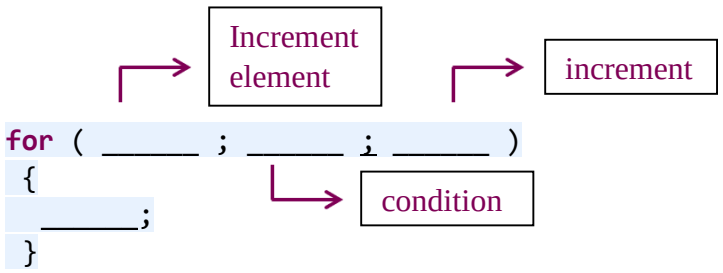
```
do
 {
    ___________;
 }
while (     );
```

Difference between while and do-while:

While: evaluation → execution

Do-while: execution → evaluation

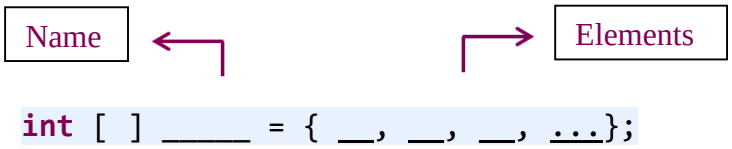
- For:



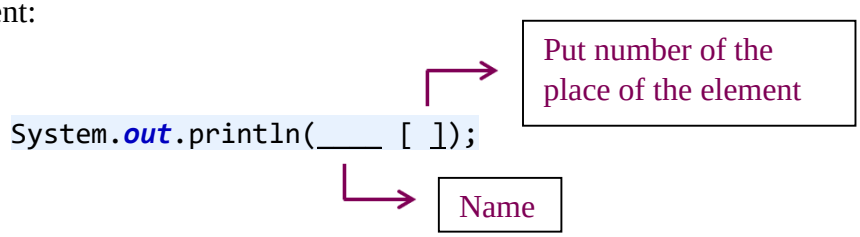
Example:

```
for ( int i = 0 ; i < 2 ; i++ )
```

Arrays



How to point to an element:

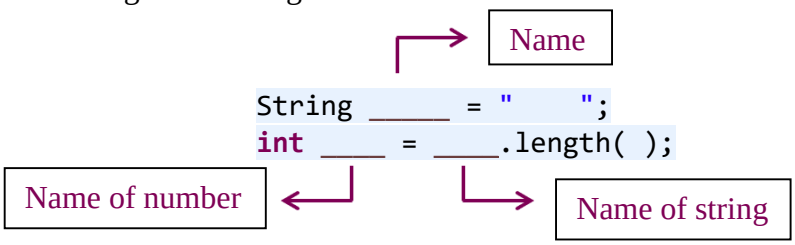


To print elements:

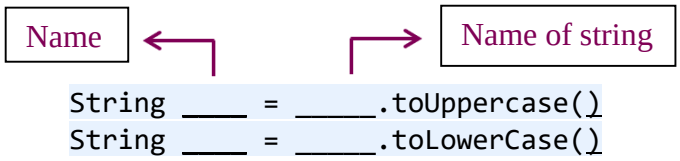
```
for (int i = 0; i < arr.length; i++) {
     System.out.print(arr[i] + " ");}
```

String

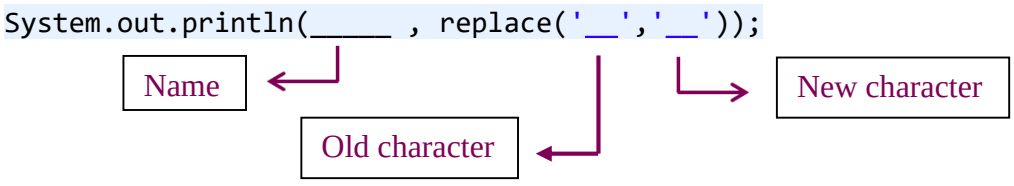
- To calculate the length of a string:



- To print in Capital or small letters use:



- You can replace characters by:



Java Inheritance

- Taking this simple calculator code:

```
public class Calculater
{

    public int add(int i , int j)
    {
          return i+j;
    }
}
```

- Now let's say I want to make a calculator that adds and subtracts too, I will need to use the Inheritance

⇒ First: I will create another class and make it do the subtractions

⇒ Second: I will use inheritance syntax

Syntax:

```
public class CalcAdv extends Calculater
```

➔ This sentence allows the class CalcAdv to do two functions. Its own and the class Calculater function too

The code of the class CalcAdv will be:

```
public class CalcAdv extends Calculater
    {
          public int sub(int i, int j)
          {
                return i-j;
          }
    }
```

⇒ Third: I will print results using code:

```
public class inheritance
{     public static void main(String[] args)
    {
    CalcAdv c1 = new CalcAdv();
    int result1 = c1.add(3,4);
    int result2 = c1.sub(5,3);

    System.out.println(result1);
    System.out.println(result2);
    }
}
```

Comparison in Java

We can either use Boolean variable or put the comparison directly in the print method.

- Using Direct method:

```
public class GreaterLessThan {
      public static void main(String[] args) {
            double creditsEarned = 176.5;
    double creditsOfSeminar = 8;
    double creditsToGraduate = 180;

    double creditsAfterSeminar=  creditsEarned+  creditsOfSeminar;

    System.out.println(creditsToGraduate < creditsAfterSeminar);
      }
}
```

- Using a Boolean variable:

```
double creditsAfterSeminar =  creditsEarned +  creditsOfSeminar;
boolean comparison = creditsToGraduate < creditsAfterSeminar;
System.out.println(comparison);
```

Equality in java

- For equality use symbol ==
- For Non-equality use symbol !=

Example:

```
public class EqualNotEqual
{
      public static void main(String[] args) {
            int songsA = 9;
    int songsB = 9;
    int albumLengthA = 41;
    int albumLengthB = 53;
    boolean sameNumberOfSongs = songsA==songsB;
    boolean differentLength = albumLengthA != albumLengthB;
      }

      }
```

Errors in java

Types of errors in java:

- Syntax error: (compile error)

It results from errors in code construction such as mistyping a keyword, forgetting some necessary punctuation such as semicolon, or using an opening brace without a corresponding closing brace.

- Run time error:

For example, happens when the user enters a value that does not suit the value you assigned or when there is a division by zero.

- Logic error:

Occurs when a program does not perform the way it was intended to (Answer is wrong).

Polymorphism

Definition: The ability of an object to have different forms

- Base class
- Derive class

You can point the reference to a base class to any object of the derived class

Casting: reference of parent class points to the object of the subclass

Syntax:

```
ParentClassName NewObjectName = new SubClassName();
```

Note (extra):

- To Checki if two strings are the same or not we use a built-in method called : .equals()

Syntax:

```
System.out.println(stringName1.equals(stringName2));
```

Classes

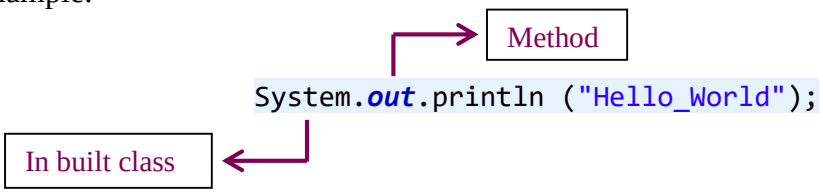
Syntax:

Definition: set of instructions that describe how an instance can behave and what information it contains.

Classes can be:

In built classes

Example:



Or

Costumed by user

Example:

```
public class Store{}
```

- The word public means that other classes can interact with this class (it will be explained later in Inheritance)

Note: that the class needs to have a main method to run.

Objects:

- They are also called: java instances.
- Classes define the state and behavior of the object.
  - State: comes from *object fields* declared inside the class.
  - Behavior: comes from *methods* defined in the class.

To understand this lets see an example of a simple code.

```
public class Car
```
→ Class declaration

```
{
    String color;
    boolean isRun;
    int velocity;
```
→ Definition of variables that we will use in the object **(these are called fields)**

```
    public Car(String carColor, boolean carRun, int carVel)
```
→ Declaration of an object and **parameters** that will be used in it

↑ This is called a **constructor method** and its named after the class

```
    {
        color= carColor;
        isRun= carRun;
        velocity=carVel;
    }
```
→ Assigning parameters to fields

```
    public static void main(String[] args)
```
→ Main method

```
    {
        Car ferrari= new Car("red",true,27);
        Car renault = new Car("blue",false,70);
```
→ New values passed into the main method
Note that Ferrari and Renault are **objects**

```
        System.out.println(ferrari.color);
```
→ We use the dot (.) to access the variables and methods of an object or

```
    }
}
```

When you run the code, the computer will enter the class, to the main method to run it then it will move to Car, and run it lastly it will return to main.

The output of this code will be: red

Methods

- They are used to make your code clearer and shorter
- A method signature gives the program some information about the method

Syntax:

This means that other classes can access this method

```
public void NameOfMethod() {}
```

This means there is no output. But if our method returns a value we will use String, int, double…

Syntax for calling a method in the main

```
NameOfInstance.NameOfMethod();
```

If the method passes parameters we put them between brackets

There are three ways to use methods:

1. Void, NO Parameter: in this case the method does not pass or return any value but it prints or assign values.
2. Void, with Parameters: in this case the method does pass parameters but it does not return any value.
3. Passes Parameters and return values: in this case the method will return and pass values.

To understand methods and how can we use them let's see an example simple code.

```
public class SavingsAccount
{

   int balance;

  public SavingsAccount(int initialBalance)

  {
    balance = initialBalance;
  }
```

This is called a constructor (its name **must** match the class name)

```
public void checkBalance()
{
  System.out.println("Hello!");
  System.out.println("Your balance is "+ balance);
}
```

This is called a method (its name **does not** match the class name)
This method will print to the user the balance
Its void because it does not need to return any value and has no parameter because it does not receive any input since balance is defined for all the class

```
public void deposit(int amountToDeposit)
{
  balance= balance+amountToDeposit;
  System.out.println("you have deposited "+ amountToDeposit);
}
```

This method will receive the value that has been added to the account, add it to the balance and print to the user the new value of the balance
Here, it's also void because it does not need to return any value but it has a parameter because it receives an integer which is the value deposited

```
public int withdraw( int amountToWithdraw)
{
  balance= balance-amountToWithdraw;
  return amountToWithdraw;
}
```

This method will receive the value that has been taken from the account, substract it from the balance and return the new value of the balance to the main when called

```
public static void main(String[] args)
{
  SavingsAccount savings = new SavingsAccount(2000);
```

Here we added an instance called savings with an initial balance of 2000

```
  savings.checkBalance();
   int result = savings.withdraw(600);
  System.out.println("you have just withdrawed "+ result);
  savings.checkBalance();
  savings.deposit(400);
  savings.checkBalance();
  }
}
```

Here we call the methods by using the syntax I mentioned before.
If you created another instance (savings2 for example) you will only need to repeat the calling lines with changing the instance name.

The output is:

```
Hello!
Your balance is 2000
you have just withdrawed 600
Hello!
Your balance is 1400
you have deposited 400
Hello!
Your balance is 1800
```

You are probably asking yourself why using method?

Let me answer your question with this code that we used no method in it

```
public class SavingsAccount {

  int balance;

  public SavingsAccount(int initialBalance){
    balance = initialBalance;
  }
  public void checkBalance(){


  public static void main(String[] args){
    SavingsAccount savings = new SavingsAccount(2000);

    //Check balance:
    System.out.println("Hello!");
    System.out.println("Your balance is "+savings.balance);

    //Withdrawing:
    int afterWithdraw = savings.balance - 300;
    savings.balance = afterWithdraw;
    System.out.println("You just withdrew "+300);

    //Check balance:
    System.out.println("Hello!");
    System.out.println("Your balance is "+savings.balance);

    //Deposit:
    int afterDeposit = savings.balance + 600;
    savings.balance = afterDeposit;
    System.out.println("You just deposited "+600);

    //Check balance:
    System.out.println("Hello!");
    System.out.println("Your balance is "+savings.balance);

    //Deposit:
    int afterDeposit2 = savings.balance + 600;
    savings.balance = afterDeposit2;
    System.out.println("You just deposited "+600);

    //Check balance:
    System.out.println("Hello!");
    System.out.println("Your balance is "+savings.balance);


  }
}
```

In this code, if you needed to create another instances for other accounts you will need to repeat all these lines

But with methods you will only need to repeat the calling line and change the instance name

So methods makes your code shorter and more efficient to use

Conditional statement

- If statement

Statement

```
if (      )
  {
     excute if true (put any statement you want)
  }
else
  {
     excute its false
  }
```

- The comparison tools

Comparison tool	Meaning
==	Equal to
!=	Not equal to
>,<	Greater, smaller
>=,<=	Greater or equal, smaller or equal

- You can use and `&&` and `||`

```
if ((state 1) && (state 2))
{
    _ _ _ _ _ _ _ _ ;
}
```

- Switch:

Name of variable (int, byte, short, char)

value

Means that it breaks out of switch

```
switch (      )
 {
    case ___ : _ _ _ _ _ _ _ _ _ ; break;
    case ___ : _ _ _ _ _ _ _ _ _ ; break;
    default  : _ _ _ _ _ _ _ _ _ ; break;
 }
```

If you want to print a statement when no statement of the above is true.

If you don't add the break it keeps doing the rest of the code until end.

In this code you can see how we used the switch statement and the equivalent if statement for it.

```
public class Order
{
        boolean isFilled;
        double billAmount;
        String shipping;

        public Order(boolean filled, double cost, String shippingMethod) {
                  if (cost > 24.00) {
            System.out.println("High value item!");
          } else {
            System.out.println("Low value item!");
          }
          isFilled = filled;
          billAmount = cost;
          shipping = shippingMethod;
        }

        public void ship() {
          if (isFilled) {
            System.out.println("Shipping");
          } else {
            System.out.println("Order not ready");
          }

          double shippingCost = calculateShipping();

          System.out.println("Shipping cost: ");
          System.out.println(shippingCost);
        }
```

```
public double calculateShipping() {
  double shippingCost;
  switch (shipping) {
    case "Regular":
      shippingCost = 0;
      break;
    case "Express":
      shippingCost = 1.75;
      break;
    default:
      shippingCost = .50;
  }
  return shippingCost;
    }
```

```
public double calculateShipping()
 {
   if (shipping =="Regular")
      return 0 ;
   else if (shipping ="Express")
      return 1.75;
   else
      return 0.5;
```

```
        public static void main(String[] args) {
            Order First = new Order(false , 15 , "Express");
            First.ship();
            First.calculateShipping();
        }
}
```

The output of this code is:

```
Low value item!
Order not ready
Shipping cost:
1.75
```

Conditional operators

- These operators only use Boolean values.
- There are three operators: and, or, !

Operator	Meaning
&& (And)	All conditions need to be true
|| (Or)	Only one condition needs to be true
! (Not)	If the single condition is opposite to which it's applied (!false= true , !true=false)

In this code you can see how we used the conditional operators.

```
public class Reservation
{
  int guestCount;
  int restaurantCapacity;
  boolean isRestaurantOpen;
  boolean isConfirmed;

  public Reservation(int count, int capacity, boolean open)
  {
    if (count < 1 || count > 8)
    {
      System.out.println("Invalid reservation!");
    }
    guestCount = count;
    restaurantCapacity = capacity;
    isRestaurantOpen = open;
  }
```

Here in the if statement, the operator used is OR which means that if the count is less than 1 or larger than 8 then the statement is true and Invalid reservation is printed.

```
public void confirmReservation()
  {
   if (restaurantCapacity >= guestCount && isRestaurantOpen)
    {
      System.out.println("Reservation confirmed");
      isConfirmed = true;
    }
   else
    {
      System.out.println("Reservation denied");
      isConfirmed = false;
    }
  }

   public void informUser()
  {
    if (!isConfirmed)
    {
      System.out.println("Unable to confirm reservation, please contact
      restaurant.");
    }
     Else
    {
      System.out.println("Please enjoy your meal!");
    }
  }

  public static void main(String[] args)
  {
     Reservation P1 = new Reservation(5,10,true);
     P1.confirmReservation();
     P1.informUser();
  }
}
```

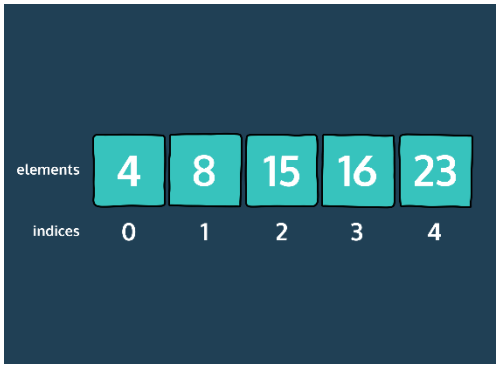
Here in the if statement, the operator used is AND which means that the two conditions are required in order to print Reservation confirmed and the boolean variable isConfirmed will be set to true

The output of this code is:

```
Reservation confirmed
Please enjoy your meal!
```

Arrays

- They are **data lists**
- it holds fixed number of values of one type such as: double, int, boolean and string.
- Each place in the array has an index:



- The length of the array shown is 5.
- The highest index is 4.
- To declare an array:

```
Type of Array[] NameofArray = {element1, element2, …. };
```

- For example:

```
double[] prices = {13.15, 15.87, 14.22, 16.66};
```

In order to have a more descriptive printout of the array we need a toString() method that is provided by the Arrays package in Java.

To do this we need to put the following line before defining a class:

```
import java.util.Arrays;
```

This line imports the Arrays package which has many useful methods to use such as:

```
Arrays.toString()
```

This method let us see the contents of the array printed out

The following code is an example of the use of this method:

```
import java.util.Arrays;

public class Numbers(){

  public static void main(String[] args){
    int[] DiffNumbers = {5, 8, 20, 25, 62};
    String NumPrintout = Arrays.toString(DiffNumbers);
    System.out.println(NumPrintout);
  }

}
```

To get values out of an array we use brackets []. For example:

```
System.out.println(DiffNumbers[1]);
```

We can also define an empty array and fill it. For example:

```
int[] Numbers = new int[4];
Numbers[0] = 1;
Numbers[1] = 45;
Numbers[2] = 22;
Numbers[3] = 3;
```

To obtain the length of the previous array:

```
System.out.println(Numbers.length);
```

Arraylists

Sometimes we need to create an array with no fixed size (dynamic list) these are called Arraylists.

These lists allow us:

- Store elements of the same type (just like arrays)
- Access elements by index (just like arrays)
- Add elements
- Remove elements

To use an ArrayList at all, we need to import them from Java's util package as well:

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

To declare an arraylist, we define the type of objects in it:

```
ArrayList<TypeOfList> NameOfList;
```

< > these are called generics

When we put the type of the list we need to use difference syntax than usual. For example: String, Integer, Double and Char.

```
NameOfList = new ArrayList<TypeOfList>();
```

To Add an item to the arraylist. See the following example:

```
ArrayList<String> toDoLiconsst = new ArrayList<String>();

String toDo1 = "Water plants";
String toDo2 = "Do homework";
String toDo3 = "play sport";

toDoList.add(toDo1);
toDoList.add(toDo2);
toDoList.add(toDo3);
```

To access an index we use method get(). For Example:

```
System.out.println(toDoList.get(2));
```