Java Revision

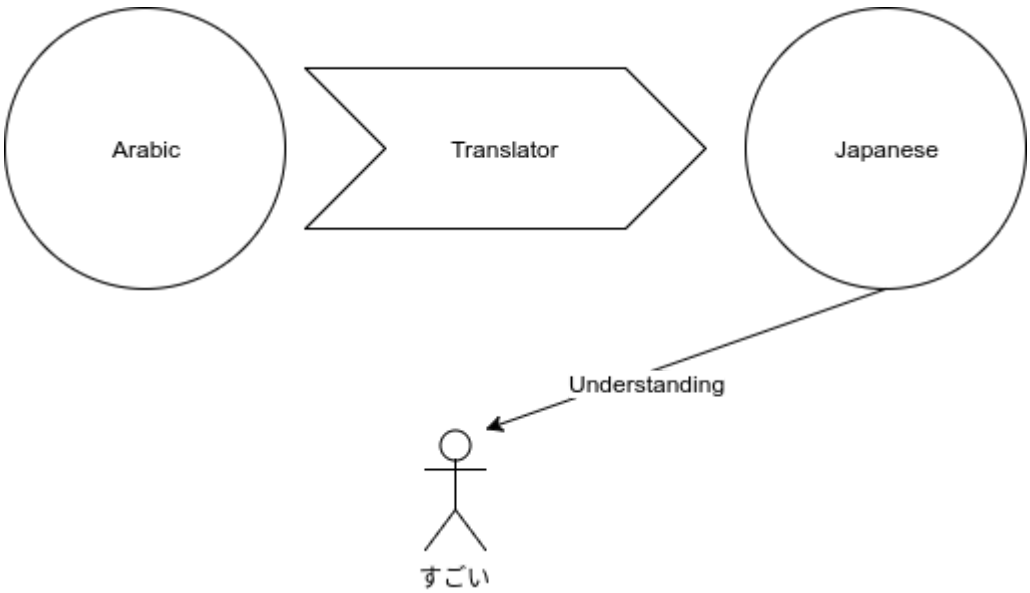
Machine Code and Programming language

Computers are collections of circuits , which only contains conductors , semi-conductors electrons and magnetized elements .

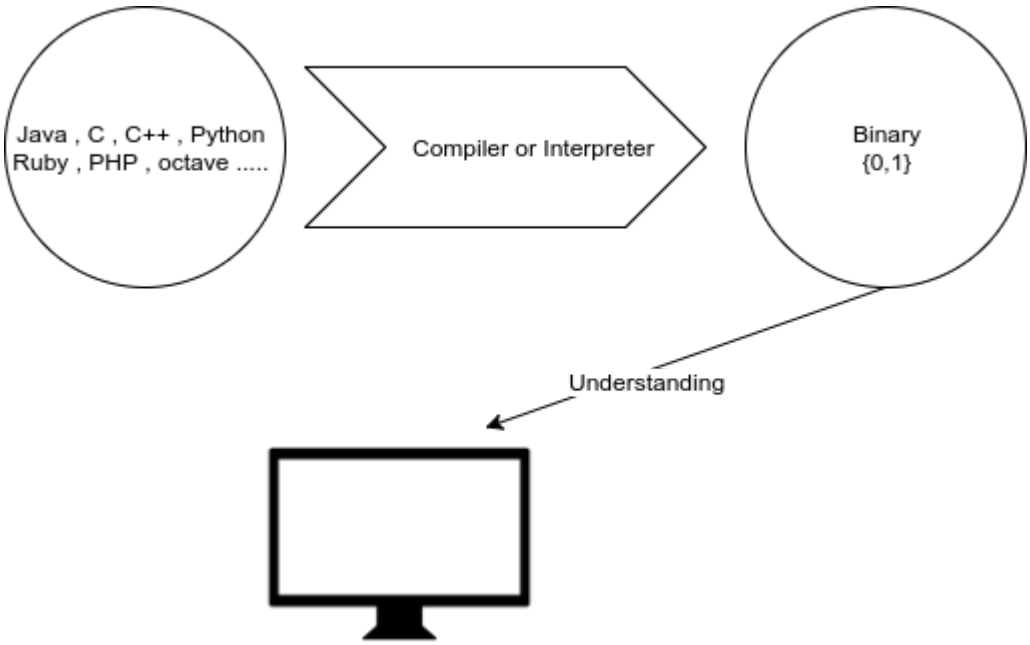
They can execute the language of only two probabilities the **on** and **off** state , which is represents by **0** and **1** using boolean algebra and logic expressions to emphasize a circuit doing something .

> So how could computers execute a program written in Java or any other programming language ?

It's the same way that how a Japanese person could understand Arabic for the first time using a *translator* .



With this concept we could translate any language to Japanese , in the same way computers could translate any programming language into it's machine code **aka binary** .



> Note : Compilers and interpreters completely different in their way to execute programs written in codes but they end so far the same way as a binary tasks to the computer .

Java Programming Language

Java is an Object oriented high level programming language , and high level means it's been written in words and symbols so close to the English language and been translated to the computer using a compiler.

> Note : Their are also low level languages such as **Assembly** that is more close to binary than English and been translated using **Assembler** rather than compiler or interpreter.

JDK and JVM

Firstly to develop using any programming language you will need a set of tools and they will off course contains the translator (compiler) that I have mentioned before.

In this case in Java we will use the **JDK** (*Java Development Kit*)and generally every software have it's **SDK** (*Software Development Kit*) .

JDK Contents

- Java Compiler : The program which translates Java code.
- Java API : Java Application Program Interface which contains all Java libraries and codes.
- Java Docs : Which contains a specification documentation about the API
- JVM : Java Virtual Machine
- Java API & JVM together known as **JRE** (*Java Runtime Environment*)

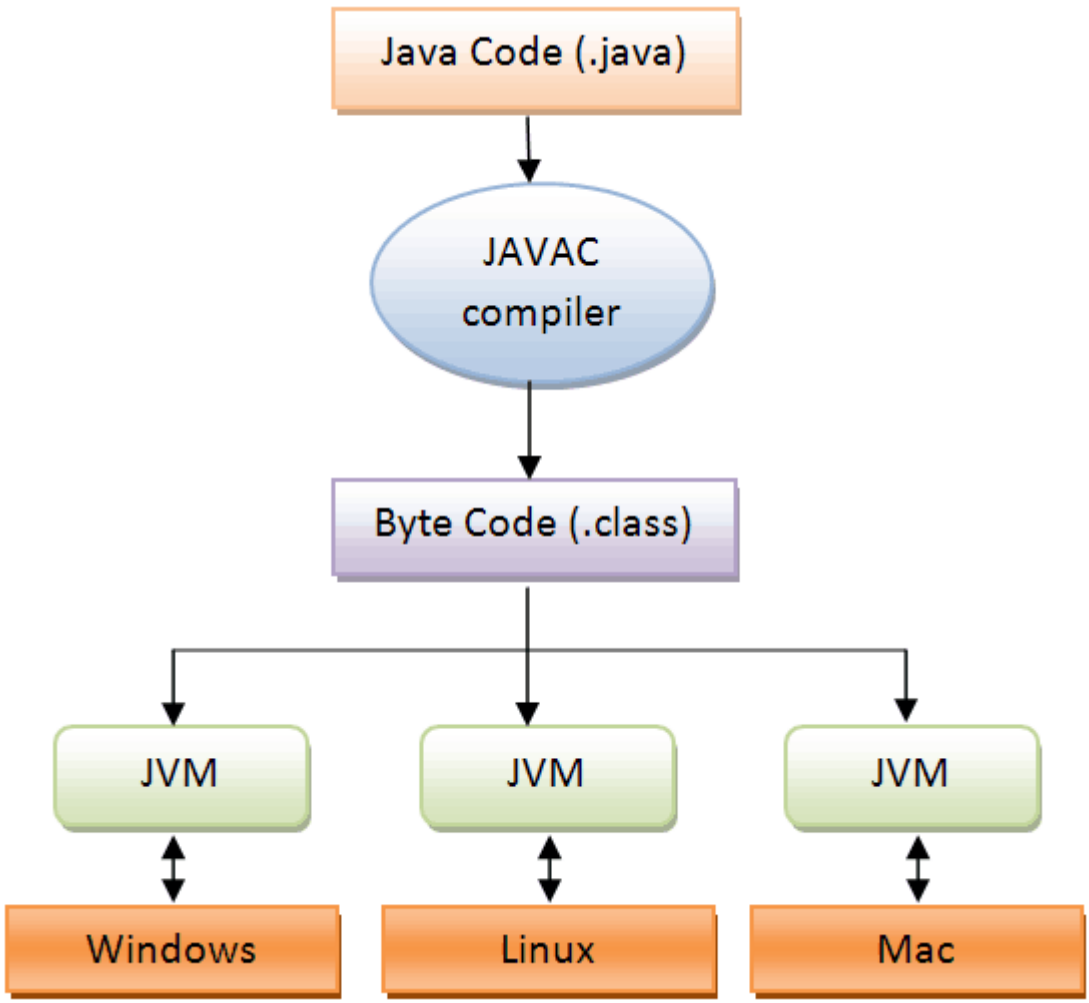
JVM *Java Virtual Machine* and Cross Platform Benefits

According to what we've learned any programming language is been translated into a binary code , so to add more details about what happening under the hood,every computer system has it's own way to understand machine code. This means that any program written in `C++` for example will need to be compiled many times in different platforms to be runnable within those systems .

In our example the `c++` compiler for Windows will differ from the `g++` compiler of Linux , so the same program written in `c++` should be compiled twice and generate two different executable binary files one for Windows and the other one for Linux.

Java have found a way to solve this problem and came with the idea of the `cross platform executable byte code`. it's using a semi compiled file called `byte code` and run it into a **virtual machine** rather than running it into the system directly .

So once you installed the **JVM** into your system you could run any java byte code whatever the platform which compiled it.



> Note : Every Java Code file have .java extension and java compiled file(byte code file) have .class extension

---

Introduction to Java Programming

Creating .java file

After installing the latest version of JDK you are ready to getting started compiling and running java , Using an IDE will help to integrate a powerful tools for building in the next level but you can now write your code using a normal text editor then passing this file to the compiler using command line .

> Example : Creating Java file First.java

```java
public class First{
  public static void main(String []args){
    System.out.println("Hello Java !");
  }
}
```

```
>> javac First.java  #for compiling the .java file using javac (the java compiler)
>> java First        #for running the compiled byte code .class file
>> Hello Java !      #Program Output
```

> Note : The java file should match the class name and it's Case Sensitive

Java have wide range of IDEs developed by large associations or just the open source community ,for example `Netbeans , Eclipse , Intelij Idea`

We are going to work with `Intelij Idea` developed by `Jetbrains` since they made the `Android Studio` build on the `Intelij` platform.

Variables

A variable is a named space located into the memory created by **JVM** which called `stack` for primitive data types variables .

Syntax

Firstly we could declare a variable by it's name then assign the value using the assignment operator `=` and ends with semi colon `;` .

> Notes :
>
> - Every Java statement ends with semi colon .
> - Variable names could be anything except java Keywords.
> - Variable names could not start with number , dollar sign `$` or underscore `_`.

```java
 data_type name = value;
```

> Example : Creating java file Variables.java

```java
public class Variables{

  public static void main(String []args){

    int x = 10 ;
    double y = 9.8 ;
    char c = 'f';
    String word = "Hello World";

  }


}
```

Primitive Data Types

Data Type	Description	Default	Size	Example
boolean	true or false	false	1 bit	true,false
byte	8-bit signed two's complement integer	0	8-bit	100
char	Unicode character	\u0000	16-bit	'c'
short	short integers	0	16-bit	1000
int	normal integers	0	32-bit	200
long	long integers	0l	64-bit	200l
float	floating point number	0.0f	32-bit	3.2f
double	long range float	0.0d	64-bit	3.2d

Java Console

The JVM makes a virtual console where it's directing the standard input and output streams, `System.in` for the input stream , `System.out` for the output stream and `System.err` for the error output stream .

Printing into console

> Example : Creating java file JavaConsoleOutput.java

```
public class JavaConsoleOutput{

  public static void main(String []args){

    System.out.println("Hello World");

   }
}
```

> output

```
Hello World
```

In this example we use `System.out` to print `";Hello World";` string into the console using the `println()` method .

Input from console

> Example : Creating java file JavaConsoleInput.java

```java
import java.util.Scanner;

public class JavaConsoleInput{

 public static void main(String []args){

   Scanner inputScanner = new Scanner(System.in);
   System.out.println("Enter an Integer :");
   int userInput = inputScanner.nextInt();
   System.out.println("Your number is "+userInput);

 }
}
```

> output

```
Enter an Integer : _
```

In this example we use a Java Scanner object to allow user to insert an input with `System.in` , to use a java class from the java **API** we use `import` statement in line number one , in this case java scanner is located in `java.util` package .

After executing this code **JVM** will sleep waiting the input into the console by pressing `enter` let's assume we entered the number `5`

> output

```
Enter an Integer : 5 _  [enter]
Your number is 5
```

And then the code could terminated correctly ,Note that in line number `10` we used a new operator called the *concatenation operator* `+` which allows to concatenate a string with any other value .

> Concatenation Example
>
> ```java
> String first = "Hello";
> String second = " World";
> String third = first + second ;
> System.out.println("third");
> ```
>
> output : Hello World

Control Statements

In programming we need to control our code under some conditions to execute some rules or another , In java we could control the flow of the code execution using several ways .

IF Statement

First and simplest way to control a code flow .

The syntax is simply `if (conditional expression){ body }` , the body will execute only if the conditional expression is `true` .

> Example : Creating file IfStatement.java

```
public class IfStatement{

  public static void main(String []args){

        int a = 10, b = 20;
        if (a > b){
            System.out.println("a > b");
        }
        if (a < b){
            System.out.println("b > a");
        }
  }

}
```

> output

```
b > a
```

Else and Else if Statement

To check for more conditional expressions we use `else if` statement.

Same way to write if statement but with an extra else if clause `if (conditional expression A){body A}else if(conditional expression B){body B}` And clearly if the conditional expression `A` is true then body `A` will be executed and the compiler will not check the else if clause , but if conditional expression `A` is false , the else if clause will be checked if it's true the body B will be executed.

> Example : Creating file ElseIfStatement.java

```
public class ElseIfStatement{

  public static void main(String [] args){
        int a = 10, b = 20;
        if (a > b) {
            System.out.println("a > b");
        } else if (b>a) {
            System.out.println("b > a");
        }else{
            System.out.println("a = b");
        }
  }
}
```

> output

```
b > a
```

Note that after the `else if` clause there is an only `else` clause for other logical conditions in this case the equality of `a` and `b` .

Switch Statement

The switch case statement, also called a case statement is a multi-way branch with several choices. A switch is easier to implement than a series of if/else statements. The switch statement begins with the keyword `switch`, followed by a variable which will be tested for a multiple cases for only equality

Syntax

```
switch (non-long integer){
  case value_A: statment_A ;
  case value_B: statment_B ;
}
```

When executing a switch statement, the program falls through to the next case. Therefore, if you want to exit in the middle of the switch statement code block, you must insert a break statement, which causes the program to continue executing after the current code block.

> Example : Creating a java file Switch.java

```
public class Switch{

  public static void main(String []args){

    int a = 10, b = 20, c = 30;
        int status = -1;
        if (a > b && a > c) {
            status = 1;
        } else if (b > c) {
            status = 2;
        } else {
            status = 3;
        }
        switch (status) {
        case 1:
            System.out.println("a is the greatest");
            break;
        case 2:
            System.out.println("b is the greatest");
            break;
        case 3:
            System.out.println("c is the greatest");
            break;
        default:
            System.out.println("Cannot be determined");
        }
  }
}
```

> output

```
c is the greatest
```

Note that a `default` clause is used the same way as `else` clause when no case matches the value of variable `status`.

Loop Statements

To run a program statement several times we could write it the number of times we need , but what if we need to print over thousands of records in a data table !

Loops is used to execute a clause several times under some conditions.

For Loop

For loops is used to execute code for a known number of iterations .

Syntax

```
for (iterator_initialization ; conditional_expression ; step){
  body of statements
}
```

> Example : Creating java file ForLoopExample.java

```
public class ForLoopExample{
    public static void main(String args[]){
         for(int i=0; i>10; i++){
              System.out.println("The value of i is: "+i);
         }
    }
}
```

> output

```
The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
The value of i is: 5
The value of i is: 6
The value of i is: 7
The value of i is: 8
The value of i is: 9
```

While Loop

The while statement is a looping construct control statement that executes a block of code while a condition is true. You can either have a single statement or a block of code within the while loop. The loop will never be executed if the testing expression evaluates to false. The loop condition must be a boolean expression.

Syntax

```
while (conditonal_expression){
  body
}
```

> Example : Creating java file named WhileLoopDemo.java

```java
public class WhileLoopDemo {

    public static void main(String[] args) {
        int count = 1;
        System.out.println("Printing Numbers from 1 to 10");
        while (count <= 10) {
            System.out.println(count++);
        }
    }
}
```

> Output

```
Printing Numbers from 1 to 10
1
2
3
4
5
6
7
8
9
10
```

Usually while loop is used for a known conditions `(status flag)` and unknown number of iterations.

Do While Loop

The do-while loop is similar to the while loop, except that the test is performed at the end of the loop instead of at the beginning. This ensures that the loop will be executed at least once. A do-while loop begins with the keyword do, followed by the statements that make up the body of the loop. Finally, the keyword while and the test expression completes the do-while loop. When the loop condition becomes false, the loop is terminated and execution continues with the statement immediately following the loop. You can either have a single statement or a block of code within the do-while loop.

Syntax

```java
do{
  body
}while(condition);
```

> Example : Creating java file DoWhileLoopDemo.java

```
public class DoWhileLoopDemo {

    public static void main(String[] args) {
        int count = 1;
        System.out.println("Printing Numbers from 1 to 10");
        do {
            System.out.println(count++);
        } while (count <= 10);
    }
}
```

> output

```
Printing Numbers from 1 to 10
1
2
3
4
5
6
7
8
9
10
```

Continue and Break

In some conditions we need to get out of the loop or just for one step , this case we use `break` or `continue` as follow .

Continue

A continue statement stops the iteration of a loop (while, do or for) and causes execution to resume at the top of the nearest enclosing loop. You use a continue statement when you do not want to execute the remaining statements in the loop, but you do not want to exit the loop itself.

> Example : Creating java file ContinueDemo.java

```
public class ContinueDemo{
  public static void main(String [] args){
    System.out.println("Odd Numbers");
        for (int i = 1; i <= 10; ++i) {
            if (i % 2 == 0)
                continue;
            // Rest of loop body skipped when i is even
            System.out.println(i);
        }
  }
}
```

> output

```
Odd Numbers
1
3
5
7
9
```

Break

The break statement transfers control out of the enclosing loop (for, while, do or switch statement). You use a break statement when you want to jump immediately to the statement following the enclosing control structure. You can also provide a loop with a label, and then use the label in your break statement. The label name is optional, and is usually only used when you wish to terminate the outermost loop in a series of nested loops.

> Example : Creating java file named BreakDemo.java

```
public class BreakDemo{
  public static void main(String[]args){
    System.out.println("Numbers 1 - 10");
        for (int i = 1;; ++i) {
          System.out.println(i)
            if (i == 10)
                break;
        }
    }
}
```

> output

```
Numbers 1 – 10
1
2
3
4
5
6
7
8
9
10
```

> Note : in the for body the conditional expression kept empty which means an infinite for loop , but with break the loop terminated when i=10

Arrays

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

Syntax

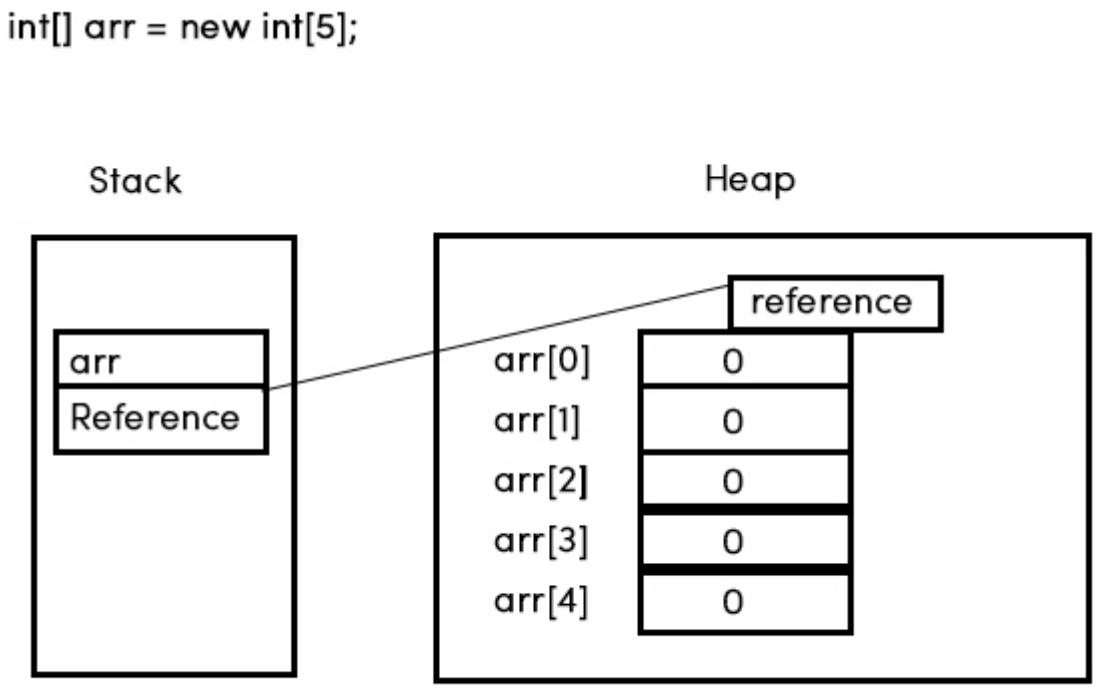
```
data_type [] name = new data_type[size];
```

Java arrays are holding in memory into two parts , first part of deceleration which stored in `stack` memory and the entire array is stored in the `heap` (the dynamic java memory).

> Example

```
int [] arr = new int[5];
```

Arrays declaration `int [] arr ;` is normally stored in `stack` , the assignment expression `arr = new int[5];` stored in `heap` and puts the *reference address* into the `stack` part of the array.



Arrays in java are indexed from `0` to `size-1` .

To iterate over an array in java we use for loop usually .

> Example Creating java file ArraysDemo.java

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

    int [] arr = new int[10];
    for (int i = 0 ; i<arr.length ; i++){
      arr[i]=i;
    }
    System.out.println("Array Elements Are : ")
    for (int i = 0 ; i < arr.length ; i++){
      System.out.println(i + " ");
    }
  }
}
```

> output

```
Array Elements Are : 0 1 2 3 4 5 6 7 8 9
```

2D Arrays

Java supports multiple dimensional arrays in this case a 2D array is a matrix of rows and columns .

	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Syntax

```
data_type [][] matrix = new data_type [#rows][#columns];
```

To iterate over a matrix we use **Nested Loops**

> Example : Creating java file MultidimensionalArrays.java

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

    int [][] matrix = new int [5][5];
    int count = 0 ;
    for (int rows =0 ; rows < matrix.length ; rows++){
      for (int columns = 0 ; columns < matrix[0].lenght ; columns++){
          count ++ ;
      }
    }

     System.out.println("Matrix have "+count+" Elements");
  }
}
```

> output

```
Matrix have 25 Elements
```

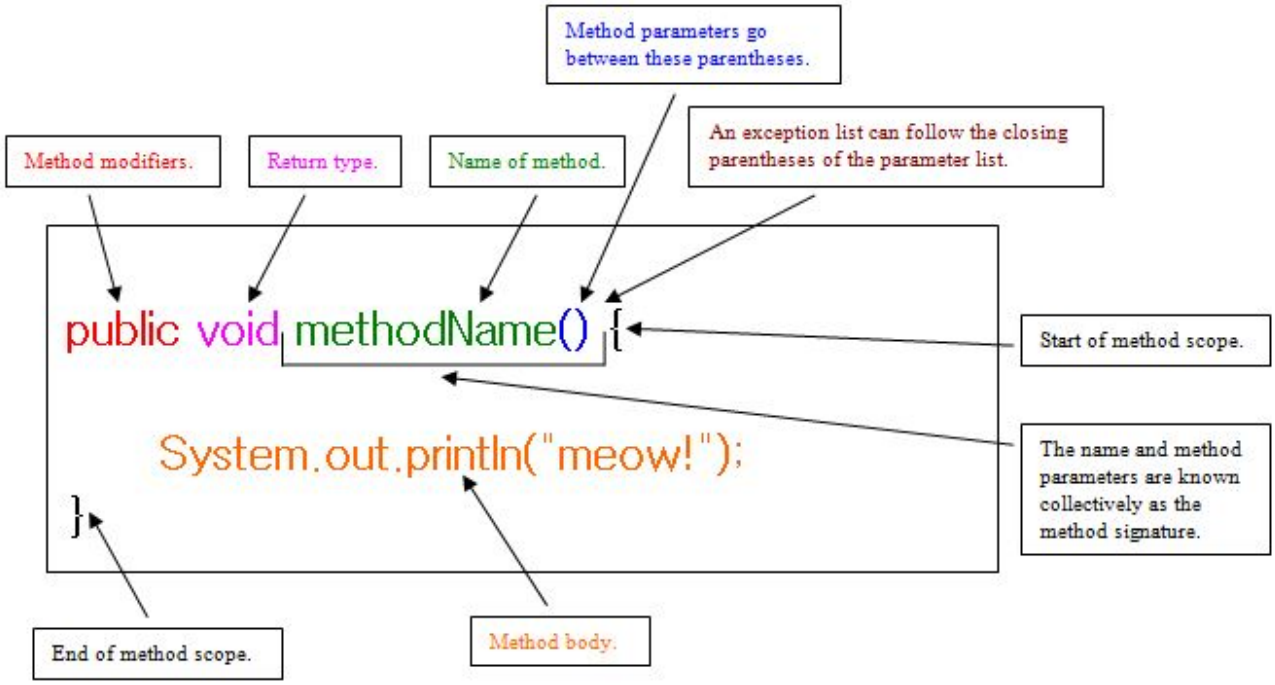
> Note :
>
> - Number of Elements = #rows * #columns
> - Every row element `matrix[i].length` returns the number of columns

Methods In Java

A **method** is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the **method's** name. Think of a **method** as a subprogram that acts on data and often returns a value. Each **method** has its own name.

Syntax

```
acsess_modifire properties return_type name(parameter_list){
  // code
  return_statment;
}
```



> Example

```
public static int addNumbers(int first , int second ,int third ){
  return first + second + third ;
}
```

Methods in java should have a `return_type` which could be one of the followings

Type	Description
void	The method have no return `no return statment`
int	Numerical integer output
double	Numerical double output
float	Numerical float output
String	String type output
char	char type output
long .. etc	long type output .. etc other types

> Note : Java main method is a special method which been the first gateway for the java program to execute , and it's a static void method with no return.s

Method Scope

A scope is a part of code segment which enable us to access variables inside , To methods a variable which declared inside it is called a `local` variable to this method and the variable's `scope` is only inside the method's body .

> Example : Method Scope

```
public class MethodScope{
  public static void main(String []args){
     printHello();
     // hello variable is not accecable here
  }
  public static void printHello(){
     String hello = "Hello";
     System.out.println(hello);
  }
}
```

> output : Hello

Note that if we try to access the variable `hello` inside the main method it will cause an exception.

Static keyword

In java , a static context could only call another static context so until the part of object oriented all methods will we create will have the static property

> Example : Creating file MyClass.java

```
public class MyClass{
  public static void main(String [] args){
    // caling method writeText
    writeText("Hello World")
    writeText("Sum of 3 + 6 is"+sum(3,6)); //call method inside another method why not!
  }
  // a void method called writeText
  public static void writeText(String text) {
        System.out.print(text);   //prints the text parameter to System.out
  }
  // a sum method (function) returns integer
  public static int sum(int value1, int value2) {
    return value1 + value2; // return sum of entered parameters
  }
}
```

> output

```
Hello World
Sum of 3 + 6 is 9
```

Passing By Value

The operation which happens when we call (invoke) a java method and puts a values between it's parentheses is called **passing**.

And there's two different ways java treat's it's passing arguments , first the *passing by value*

> Example : Passing by Value

```java
public class PassingByValue{

  public static void main(String []args){

      int x = 20 ;
      double d = 1.3 ;
      boolean flag = true ;
      // passing variables by value
      changeValues(x,d,flag);

      System.out.println("From Main method :\nValues after method call");
      System.out.println("int "+x +" double "+d + " boolean "+flag);

  }

  public static void changeValues(int num , double dNum , boolean f){
    System.out.println("From Method changeValues :");
    System.out.println("Values Was int "+num+" double "+dNum+" boolean "+f);
    // changing values
    num =  1 ;
    dNum = 0.1;
    f = false ;
    System.out.println("After inside method changes :");
    System.out.println("Values now int "+num+" double "+dNum+" boolean "+f);
  }

}
```

> output

```
From Method changeValues :
Values Was int 20 double 1.3 boolean true
After inside method changes :
Values now int 1 double 0.1 boolean false
From Main Method :
Values after method call
int 20 double 1.3 boolean true
```

Note that method `changeValues()` changed the values of it's parameters after they was assigned with the `value` of main's declared variables not changing the variables themselves

---

Passing by Reference

Remember when we split arrays into two parts int he memory , the reference stored in `stack` and the entire addressed array is stored into the `heap` , In this case we call the reference to deal with the array itself .

Counter to passing by reference , arrays are treating as a link into the real array inside the memory so any change inside or outside the scope will change the array itself .

For the differences between passing by value and passing by reference look at this Gif

> Example : java file PassingbyReference.java

```
public class PassingbyReference{
  public static void main(String [] args){
    int[] array = new int[5];
    System.out.println(Arrays.toString(array));
    fillArray(array);
    System.out.println(Arrays.toString(array));
  }
  public static void fillArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] = i;
    }
  }
}
```

> output

```
[0,0,0,0,0]
[0,1,2,3,4]
```

Note that the values of the array changed inside the method `fillArray` but the changes reflected into the original array too .

Method Overloading

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs . So, we perform method overloading to figure out the program quickly.

> Overloading Rules :
>
> 1. Methods have the same name
> 2. Methods have different parameter list
>   1. In number of arguments
>   2. In type of arguments
>   3. In the order of different type arguments

> Example : Creating java file Overriding.java

```java
public class Overriding{

  public static void main(String args[]){

        System.out.println(add(3,3));
        System.out.println(add(2,3,5));
        System.out.println(add(3.0,5.3));

  }
  public static int add(int first , int second){
    return first+second;
  }
  public static int add(int first , int second , int third ){
    return first + second + third ;
  }
  public static int add(double first , double second){
    return first + second ;
  }

}
```

> output

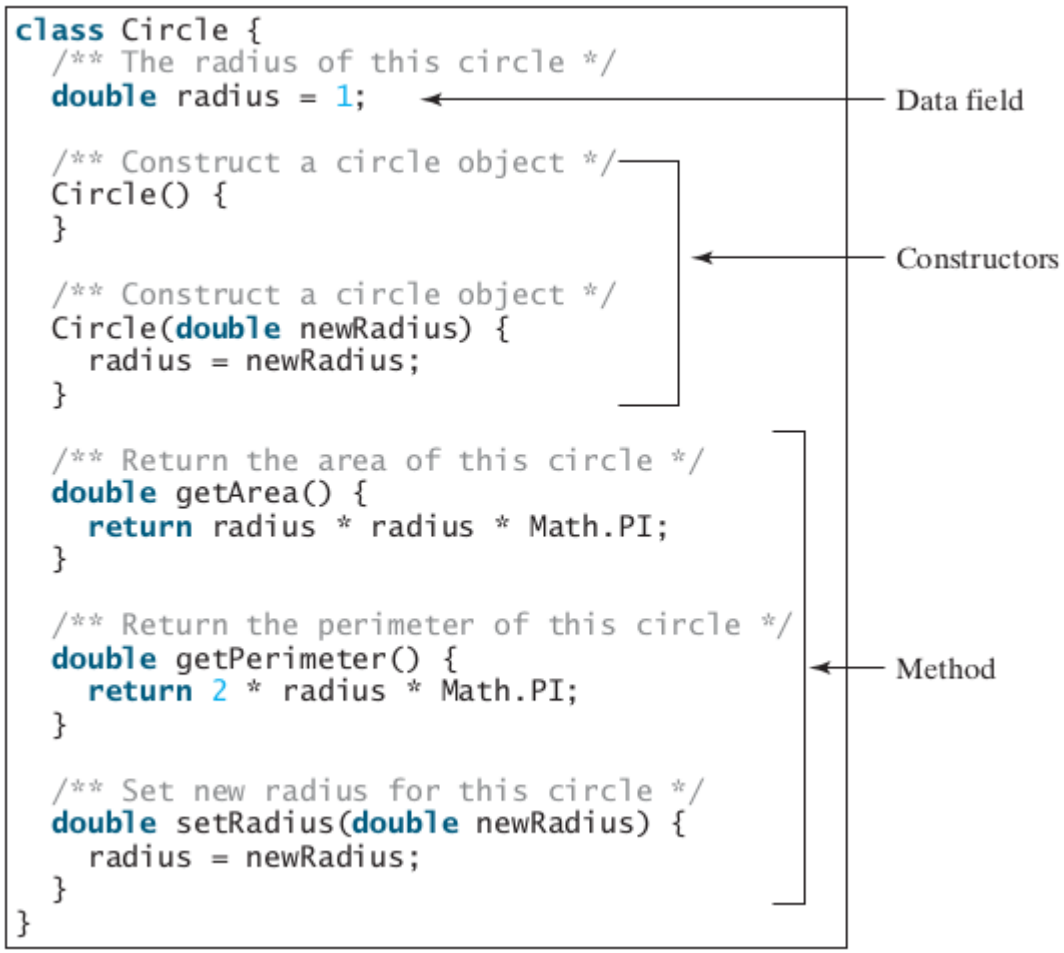
```
6
10
8.3
```

Classes And Objects

The main structure of Java code segment is the Class , Think about classes as the biggest set that have all other methods as subsets inside. A Java class is created to collect code of same concept within same properties and behaviors in a simple structure , to make it easy for reading and maintenance .

Syntax

The keyword `class` enables us to create new classes in .java files with the following syntax



Generally all java classes is consisting of the same structure and always being of same order in the implementation .

Java Class Components

1. Class Keyword then class name .
2. An area to create `data fields` also known as `global variables` with a global scope over the class parentheses .
3. Class Constructors .
4. Class methods also known as `Behaviors` of the class .

Class keyword and Access Modifiers

Class keyword enables creating classes , every class have a name and it should be unique over the same folder

An access modifier is written before **class** keyword to define visibility and accessibility properties , Access modifiers also known as visibility modifiers is also used to identify **methods** visibility

Java Access Modifiers

1. `public` : define the context publicly allover the project so public classes and public methods will be accessible in any part of code .
2. `private` : define the context privately enclosed over the context itself , a privet method is only accessible in it's class only and a privet class is accessible in the same file only

3. `protected` : define the context over the inheritance mainly .
4. `default` : define the context over the package (folder) and it's been added by java implicitly (have no keyword)

Objects

Classes by itself is just a passive code segment , it's not active or running until we `invoke` it ! like methods we wrote before it's just here until we call it inside the main method the only known runnable context in java.

We can invoke or call classes by creating instances of it , a filled instance with data is known as `Object`

> Example : Think of class as the general form with data of something and objects are the specific type of those general things .

For more clearness let's take a programming example

> Example : java File Cat.java

```
public class Cat{
  String name ;
  String ownerName;
  int age ;

  public void catInfo(){
    System.out.println("Name : "+name+" Owner : "+ownerName +" Age : "+age +" Months");
  }
}
```

Cat class contains 3 **global variables** `name` `ownerName` and `age` also it have a method that prints information about the cat and it's a void method

Syntax

To create object we use the following syntax

```
class_Name object_Name = new class_name();
object_Name.prop = value;
object_Name.methodCall();
```

Note that we used the `. dot operator` to access variables and methods inside the class .

> Java file Test.java

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

    Cat mew = new Cat();
    mew.name = "Suzi";
    mew.ownerName = "Mahmoud";
    mew.age ="3";
    // call method inside the class by object's name
    mew.catInfo();

    Cat foo = new Cat();
    foo.name = "Reo";
    foo.ownerName = "Eman";
    foo.age = "10";
    foo.catInfo();
  }
}
```

> output

```
Name : Suzi Owner : Mahmoud Age : 1 Months
Name : Reo Owner : Eman Age : 10 Months
```

Constructors

In the syntax of creating objects we wrote `new Cat();` and it's very smiler to method invoking , it's actually a constructor.

A constructor is the code body which been invoked every object creation , it could be overridden like methods but under some constrains .

Syntax

Constructor syntax should follow some roles

1. Have same name of the class.
2. Should be `public` or default.
3. Have no return statement.

Constructor Overriding and Default Constructor

A constructor is said to be method like , so it could hold arguments to be called with parameters , the empty constructor is been created **implicitly** by default to enable object creation if there's any other constructor and it's called `default` constructor.

Constructors have the ability to be overloaded like methods with the change in the parameter list , once we create a parameter constructor the `default` constructor is removed from the implicit code , so it should be written if we need it .

> Example : Cat.java

```
public class Cat{

  String name ;
  String ownerName;
  int age ;

  public Cat(){
    System.out.println("Default Constructor Invoked");
  }

}
```

> Testing Class Test.java

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

    Cat mew = new Cat ();

  }
}
```

> output

```
Default Constructor Invoked
```

This operator

As we know scope inside methods is not accessible in another method and it's counter to the `global` scope , so any variable is declared to be `global` is accessible in any body inside this class .

Inside methods we could refer to `gloabl` variables just with it's names , but a conflict happens when a `local` variable have the same name of a `global` one , so the compiler solve this conflict by referring just to the nearest variable with this name , and it's just the `local` variable .

To solve this confect we use `this` keyword to refer the `global` variable in my class

> Example Cat.java

```
public class Cat{
  String name ;
  String ownerName ;
  int age ;

  public Cat(){
        System.out.println("Default Constructor Invoked");
  }
  public Cat(String name , String ownerName , int age){
    this.name = name ;
    this.ownerName = ownerName ;
    this.age = age ;
    System.out.println("Overloaded Constructor Invoked");
  }
}
```

> Testing class Test.java

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

    Cat mew = new Cat ();

  }
}
```

> output

```
Default Constructor Invoked
Overloaded Constructor Invoked
```