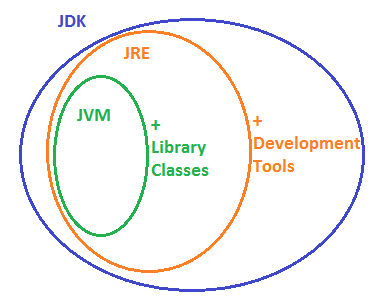
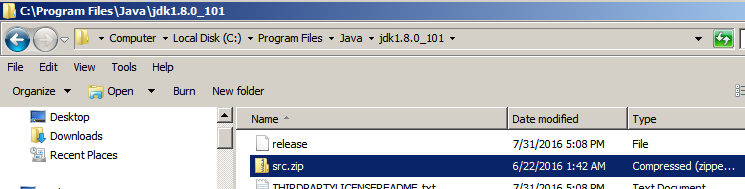
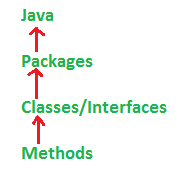
* Java is a Programming Language and a Platform
* Java is high level and object oriented programming language
* Any hardware or software environment in which program runs, called “Platform”. Java has its own Run Time Environment (JRE) & APIs. So it is also called Platform
* JDK:
* Java Development Kit
* JDK is used to Develop and Run Java applications
* It provides JRE (Runtime Environment) + Development tools (e.g. compiler + small applications which required to develop java applications/programs)
* We can develop, compile, execute and modify existing java applications in JDK
* JRE:
* Java Runtime Environment
* JRE provides Runtime Environment; does not provide development tools
* It contains JVM + APIs/Library Classes
* It is responsible to execute the java program
* We cannot develop and modify existing java application in JRE
* JVM:
* Java Virtual Machine
* JVM is responsible to execute Java program by converting that byte code (.class) into current machine’s OS language
* JVM resides in the RAM (whenever java application runs)
* JVM reads .class files and convert it into machine code
* JDK = JRE + Development tools
* JRE = JVM + Library classes



* .java:
* When we create and save a Java Program, it creates .Java file
* .class:
* When JDK compiles a .java program, it creates .class file which contains byte codes of the program
* .jar:
* It is compressed format of complied java programs i.e. contains #of .class files.
* JVM reads and execute the .jar files by converting them into machine code
* Packages:
* Java contains group of Packages
* Packages contains group of related Classes/Interfaces
* Classes/Interfaces contain group of methods
* Packages nothing but a physical Folder structure which contains group of classes
* You can see Source of java since it’s open source application
* Source can get at Src.zip file under JDK folder:



* If you unzip src.zip, you can see multiple folders/subfolders. Those are nothing but different Packages/sub-packages
* Under those packages you can find multiple files. Those are nothing but different classes



* E.G. we define the package as

Package Java.lang

i.e. ‘lang’ package is present in Java folder.

* 2 Types of packages: **1) Pre-defined packages 2) User-defined packages**
* User defined package:
* Packages created by users
* Only one package statement will be there in a source file
* Package declaration statement will be in the first line of source file
* Package declaration:
* package<<reverse order of domain name>>.<<Projectname>>.<<modulename>>;
* e.g.package com.tcs.bank.deposit;
* Pre-defined Package:
* Packages which are pre-defined/standard/out of the box packages

Access Modifiers:

* Access Modifiers defines Scope of Classes, Constructors, Methods and Variables.
* 4 Access Modifiers:
  + Default
  + Protected
  + Private
  + Public

‘Default’ access modifiers:

* If you don’t specify access modifiers for Class, Methods, Variables then by default ‘Default’ access modifiers will be considered.
* ‘Default’ modifier is accessible within same Package.

‘Protected’ access modifiers:

* Protected access modifiers can be accessed within the Package.
* It can be accessed outside the Package but only by Inheritance.
* Class and Interface cannot be Protected

‘Private’ access modifiers:

* If a Variable, Method or Constructor is declared as Private, then it is accessible only within class in which they are declared.
* Private access modifiers hides the data from outside world
* It has very limited scope
* Class and Interface cannot be Private
* Variables that are declared as Private, can be accessed outside of the class if public getter method is present in the class

‘Public’ access modifiers:

* If a Variable, Method, Class, Constructor are declared as ‘Public’, then it can be accessed from anywhere in the Program
* It has widest scope
* We can access public variables, methods, class in other programs by importing the packages in which they are declared.

public static void main(String[] args)

* If main method is not declared in the class, JVM will throw run time error while execution (not in compilation).
* Why Java Programs required ‘public static void main(String[] args)’ method?
* JVM is configured with instructions which looks for ‘public static void main(String[] args)’ method. Once this method found, jvm will start executing the program line by line.
* We can give another name instead of main(), but for that we need to customize JVM.
* Explain syntax of main():
* ‘public’ => it is access modifiers for main() to call by JVM from anywhere
* ‘static’ => without existing of an object, JVM has to call main(). Main() is not related to any object.
* ‘void’ => main() won’t return anything to JVM
* ‘main()’ => this is name which is configured in JVM
* ‘String[] args’ => command line arguments
* Syntax of main() is very strict to identify/run by JVM
* Still some modifications to main() can be acceptable
* Order of access modifiers is not important i.e. you can write ‘public static’ or ‘static public’
* You can define array in any acceptable format e.g. ‘String[] args’, ‘String []args’, ‘String args[]’
* You can give any name instead of ‘args’
* You can use var arg e.g. ‘String… args’
* main() can be declared with other modifiers e.g. ‘final’, ‘synechronized’,’strictfp’
* following main() is acceptable, there won’t be any runtime error:

class A {

final static synchronized strictfp public void main(String… xyz)

{

Syste.out.println(“Valid main”);

}

}

Followings are three different cases for main():

Case-1: Overloaded method. Overloaded method is possible for main(). But JVM always calls main(String[] args).

If you run the program you will get output “String[]”.

main(int[] args) method should be called explicitly just like other methods.

Class A

{

public static void **main**(String[] args)

{

System.out.println(“String[]”);

}

public static void **main**(int[] args)

{

System.out.println(“int[]”);

}

}

Case-2: Inheritance

If you save following program as “P.Java”, it will create two different classes i.e. P.class and C.class. For each Class in a program, java creates separate class files.

If you run ‘/> java p’, output will be “Parent”

If you run ‘/>java C’, output will be “Parent” 🡪 in this case, jvm will search psvm() in child first, if not found, it will search in parent.

Class P

{

Public static void main(String[] args)

{

System.out.println(“Parent”);

}

}

Class C **extends** p

{

}

Case-3: Method Hiding

Since main() is used twice, it seems method overriding. But it’s special case of main(); it’s call Method Hiding in case of main().

If you run ‘/> java p’, output will be “Parent”

If you run ‘/>java C’, output will be “Child”

Class P

{

Public static void **main**(String[] args)

{

System.out.println(“Parent”);

}

}

Class C **extends** p

{

Public static void **main**(String[] args)

{

System.out.println(“Child”);

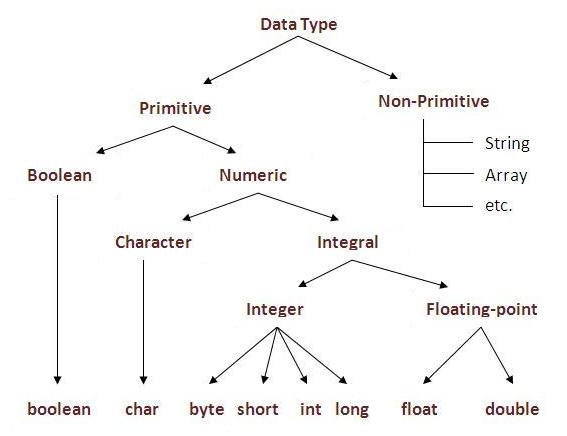
}

}

Explain System.out.println(“Hello World”):

* ‘System’ => it is class present in java.lang package
* ‘out’ => it is a static variable of type printstream present in ‘System’ class
* ‘println’ => it is a method in printstream class.

Variables:

* Variable is the name of reserved memory location
* 3 types of variables
* Local variable – which are declared inside the method
* Instance variable – which are declared inside the Class but outside of method
* Static variable - ??
* There are 2 types of data types
* Primitive – it is single value e.g. Number, character
* Non-primitive –
* 

Heap Memory:

* Heap memory is the storage for Java objects.
* It got created when JVM starts-up (runtime)
* Heap memory contains Live and Dead objects
* Live objects are those which are accessible by any java application/program
* Dead objects are those which are not accessible by java application/program
* Garbage collector runs automatically and clear out dead objects from heap memory

Non-Heap memory: is it a Stack memory???

* In non-heap area, classes and other metadata loaded (e.g. variable and method data, code for methods and constructors

Stack Memory:

* Stack memory is used in execution of thread
* When method is called and run, it creates a block in stack memory and stores the method data and reference of the objects of that method.
* When method ends, it freed up memory from stack.
* Blocks are created and removed from stack as First In First Out basis.

package com.journaldev.test;

public class Memory {

public static void main(String[] args) { // Line 1

int i=1; // Line 2

Object obj = new Object(); // Line 3

Memory mem = new Memory(); // Line 4

mem.foo(obj); // Line 5

} // Line 9

private void foo(Object param) { // Line 6

String str = param.toString(); //// Line 7

System.out.println(str);

} // Line 8

}

