Java Introduction

What is Java?

Java is a popular programming language, created in 1995.

It is owned by Oracle(Initially, Sun Micro System), and more than 3 billion devices run Java.

It is used for:

- Mobile applications (specially Android apps)

- Desktop applications

- Web applications

- Web servers and application servers

- Games

- Database connection

- And much, much more!

Why Use Java?

**-** Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)

- It is one of the most popular programming language in the world

-It is easy to learn and simple to use

- It is open-source and free

- It is secure, fast and powerful

- It has a huge community support (tens of millions of developers)

- Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs

- As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa

What is a Class ?

- A Java class can be defined as a template or blueprint which describes state/behavior of it’s object.

- In Other Word a class is used to create Objects.

So in Java, every Project begins with a class name, and that class must match the filename.

Let's create our first Java file, called MyClass.java, which can be done in any text editor

The file should contain a "Hello World" message, which is written with the following code:

**File Name MyClass.java**

Program:

public class MyClass {

public static void main(String[] args) {

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

}

}

**Explanation:**

Every line of code that runs in Java must be inside a class. In our example, we named the class MyClass. A class should always start with an uppercase first letter.

Note: Java is case-sensitive: "MyClass" and "myclass" has different meaning.

The name of the java file must match the class name. When saving the file, save it using the class name and add ".java" to the end of the filename. To run the example above on your computer, make sure that Java is properly installed: Go to the Get Started Chapter for how to install Java. The output should be:

## The main Method

## The main() method is required and you will see it in every Java program:

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

## Any code inside the main() method will be executed. You don't have to understand the keywords before and after main. You will get to know them bit by bit while reading this tutorial.

## For now, just remember that every Java program has a class name which must match the filename, and that every program must contain the main() method.

## System.out.println()

## Inside the main() method, we can use the println() method to print a line of text to the screen:

## public static void main(String[] args) {

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

## }

What are Objects ?

## Objects are nothing but the instance of the class.

## - A single class can create any number of unique objects.

Creating objects

- In java an object is created when someone says “new”.

- At each “new” , a new object of a class is created. E.g. new MyClass

- Objects lives in Java heap.

What is java heap?

Java heap is nothing but the memory space taken by JVM from the OS.

All objects are created in this heap(space). Whenever JVM encounters “new” keyword, it creates an object in heap.

• When there is no space in heap to create objects, then JVM throws “Out of Memory” error.

Creating First Object

• public class Student

{

public static void main(String[] args)

{

new Student();

}

}

Garbage Collection

Garbage Collection is the mechanism provided by JVM, to clean out the Heap, so that new

objects can be created.

• It destroys the objects which are “not in use” or eligible for “garbage collection”.

• Any object is said to be eligible for garbage collection (GC) if there is no “Reference

Variable” attached to it.

• So what is a “Reference Variable”??????

Java Variables and its type

• Variables are named space of memory which stores the data.

• There are two types of variables :

* + Primitive variables
  + Reference variables

• Reference variables are those variables which stores only address of an “object.

Assigning Objects to Reference Variable

• Before creating a reference variable we have to specify “object of which class” it is going to refer.

• In other word we need to specify the “class type” of the reference variable. E.g.

public class Test

{

public static void main(String[] args)

{

Test t = new Test();

//where t is the reference variable which stores the

Address of Test object in heap

}

}

Primitive and Non primitive data types

|  |  |
| --- | --- |
| **Primitive Data type** | **Non-Primitive Data type** |
| Primitive data types are defined by the programming language. | Non-Primitive (or Reference) data type are defined by programmer. |
| These are - Integer type  - Floating type  - Character   - Boolean | In this the data type of the variable is the Class whose object it is going to refer. Test t = new Test(); //where Test is the data type of |

Declaration and initialization of primitive variables

**Integer**

-int i = 234242425;

- long l = 284798247287427427428947l; // //suffix ‘l’ is must otherwise compiler will treat it as integer.

**Float**

**-** float = 2342.34f; //suffix ‘f’ is must otherwise compiler will treat it as a double

**double** = 298472847242478927.2942949274;

**Character**

Char c = ‘j’; // only single character is allowed;

**Boolean**

Boolean b = true // only true or false is allowed

Java String and String Concatenation

- In java “String” is a class and not a data type and it can be instantiated like other classes

String s = new String();

- String Concatenation is basically a way to combine two or more strings into a single string. This is done by using ‘+’ operator.

String s = “We” + “are” + “learning” + “java” + “.”

- String values can be concatenate with any other datatype. boolean b = true;String s = “this is” + “ “ + b

|  |  |
| --- | --- |
| **Arithmetic operators** | **Unary operators** |
| • ‘+’ additive operators/string concatenate • ‘-’ subtraction operator • ‘\*’ multiplication operator • ‘/’ division operator • ‘%’ remainder operator | • ‘++’ increment operator • ‘--‘ decrement operator • ‘!’ logical compliment operator |

**Java Operator Continues**

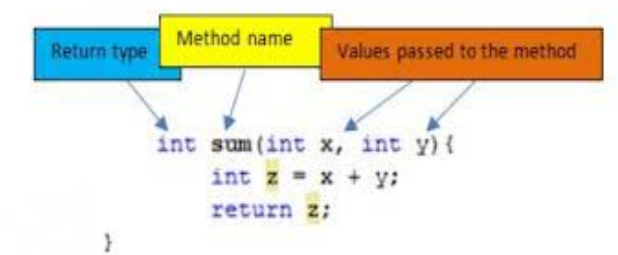
|  |  |
| --- | --- |
| **Equality and Relational** | **Conditional Operators** |
| • ‘==‘ Equal to • ‘!=‘ Not equal to • ‘>’ Greater than • ‘<‘ Less than • ‘>=‘ Greater than or equal to • ‘<=‘ Lesser than or equal to | • ‘&&’ Conditional – And • ‘||’ Conditional – Or |

**Day 2**

**What are Methods ?**

- Methods are block of statements which are used to do a specific task.

- Methods are generally used to divide a large code into manageable chunks of codes.



public int sum (int x, int y){

int z = x+y;

return z;

}

1. Access/NonAccess Modifier

2. returnType

3. MethodName ( Parameter List)

4. {

5. //method body or statement block

6. }

**Method Return Type**

• It May be possible that a method returns a value or doesn’t returns any value.

Return type of a method is nothing but the data type of the value returned by the method.

e.g.

- If method returns an integer value then it’s return type will be ‘int’, for float ‘float’, for character ‘char’, for boolean ‘boolean’ and so on.

- If a method doesn’t returns any value then it’s return type will be “void”.

• If a method returns a value then the last statement should be a “return statement”

• Any code after return statement is not reachable

**Parameter List**

• Method Parameters are the variables which are declared in the declaration of method.

e.g

**public int go(int x) {}**

• The method parameters can be nothing, one or more than one. For multiple parameters they are separated by comma.

e.g.

Public int go (int x, float y, String s, char c){ }

**Method Body**

• Body of a method starts from opening curly braces ‘{‘ and ends at closing curly braces ‘}’.

• All statements of methods goes within the pair of curlybraces.

• Execution of method stops on either ‘return’ statement or closing curly braces ‘}’.

e.g.

public int go(int x, int y)

{

Int sum = x + y;

Return sum;

}

But how to call a method???????

We see a example in the program.

**Instance variable Vs Local variable**

|  |  |
| --- | --- |
| **Instance Variables** | **Local Variable** |
| 1. Instance variables are those variables which are declared within a class. | 1. Local variable are those variables which are declared within a method. |
| 2. These are known as ‘instance variables’ because each instance of the class (object) have it’s own copy of instance variable. | 2. These are known as ‘Local variable’ because they formed within method and dies at the end of the method |

**Accessing Object’s methods and instance variables**

- The methods and instance variables of any object can be accessed by the help of dot operator (.) on reference variable

- referenceVariable.instanceVariable/method

e.g.

For class Animal having eat() method :

Animal a = new Animal();

a.eat();

**Method Arguments and method calling**

• Arguments are the values which are passed during calling of a method. These arguments are stored in the parameters of the method.

• Number of arguments must be equal to the number of parameters

• Calling a void method: a.eat();

• Calling a single argument method: a.eat(45);

• Calling a multiple argument method: a.eat(343, 43, ‘c’, “wsfsf”)

**Predefined Methods**

Predefined methods are those methods which are already defined in java and ready to use.

• We can call these methods and can use in our code directly. e.g. Math.random();

• Math.random() is a method defined in Math class which is used to generate a random number between o.oto .9 in double type

• Calling Math.random() : Int x = (int) (Math.random() \* 4);

//It will generate the number from 0 to 3

**Method Calling Methods**

- The method can call other methods in chain-like pattern.

e.g. Public **class** Test

{

**public** **void** firstMethod()

{

Test2 t = **new** Test2();

t.secondMethod(); // calling other method

}//end of method

}//end of class

**IF, IF-Else, IF-Else-IF statements**

IF statement:

This is a decision making statement, which will execute it’s codes **if** and only **if** the condition is **true**.

If(condition)

statement; // codes

• If-**else** statement:

In **this** the ‘**else**’ codes will be execute **if** and only **if** the condition is **false**. If(condition)

statement;//codes

**else**

statement; // codes

• If-**else**-**if** :

This is combination of **if**-**else** statements.

If(condition)

Statements;

Else **if** (condition)

Statements

Else **if** (condition)

Statements;

Else Statements;

**IF, IF-Else, IF-Else-IF statements**

IF statement:

This is a decision making statement, which will execute it’s codes **if** and only **if** the condition is **true**.

If(condition)

statement; // codes

• If-**else** statement:

In **this** the ‘**else**’ codes will be execute **if** and only **if** the condition is **false**. If(condition)

statement;//codes

**else**

statement; // codes

• If-**else**-**if** :

This is combination of **if**-**else** statements.

If(condition)

Statements;

Else **if** (condition)

Statements

Else **if** (condition)

Statements;

Else Statements;

**import** java.util.\*;

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

{

B t = **new** B();

Scanner scan = **new** Scanner(System.***in***); //ignore

System.***out***.println("enter a value");

**int** input = scan.nextInt(); //ignore

scan.close(); //ignore

**int** randomNum = (**int**)(Math.*random*()\* 10);

**if**(input == randomNum)

System.***out***.println("well done smarty");

**else**

t.number(randomNum);

System.***out***.println("randomNum is ----> "+randomNum);

}

**public** **void** number(**int** randomNum)

{

**if** (randomNum == 0)

System.***out***.println("Zero");

**else** **if** (randomNum == 1)

System.***out***.println("One");

**else** **if** (randomNum == 2)

System.***out***.println("Two");

**else** **if** (randomNum == 3)

System.***out***.println("Three");

**else** **if** (randomNum == 4)

System.***out***.println("Four");

**else**

System.***out***.println("Above 4");

}

}

**Day 3**

Loops

Loops are the way of repeating lines of codes until loop condition is met.!!!!

Loops in java :

¬ While loop

¬ Do-while loop

¬ For loop

¬ Enhanced for loop

**While loop**

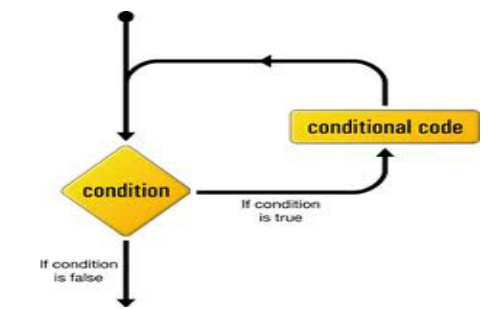
• While loop repeats a block of code until the condition is true

While (condition) {

// block of codes

}

//where condition is nothing but the Boolean expression

****

**Do-while loop**

Do-while loop is similar to the “while loop” but the only difference is that, in this the loop block is guaranteed to run at least one time!!!!

do

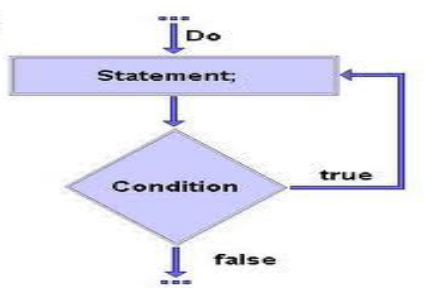
{

//all codes here

}

While (condition)

//since condition appears at the end, therefore the code block executes at least one



**For loop**

In for loop the initialization, condition checking and updating of loop element is done in a single line.

For(initialization; condition; update)

{

// codes

}

• The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables.

• Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute.

• After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables.

**Nesting of loops**

• The placing of one loop inside the body of another loop is known as Nesting of loops.

• While working with nesting loops the outer loop will change only when inner loop is completely finished.

**for** (**int** outer =0; outer<5 ; outer++)

{

**for** (**int** inner= 0; inner <3; inner++)

{

System.***out***.println("outer is " + outer + "inner is" + inner);

}//inner loop ends

}//outer loop ends

}

**Break and Continue statements**

• Break and Continue statements are used to change the normal flow of compound statement.

• The break statement immediately jumps to the end of the compound statement.

• The continue statement immediately jumps to the next iteration of the compound statement.

**for** (**int** outer=0; outer< 12; outer++) {

**if**(outer ==3)

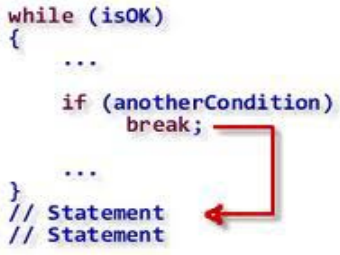
**continue**;

System.***out***.println(outer);

**if**(outer ==7)

**break**;

}



**Switch statement**

• Switch statement is the shorthand for multiple ‘if-else’ statement,

which allow us to choose a single path from a number of execution

path.

• Switch statement works with char, short, byte, int and String.

**switch**(x)

{

**case** 1:

System.out.println("case1");

**break**;

**case** 2:

System.out.println("case2");

**break**;

**case** 3:

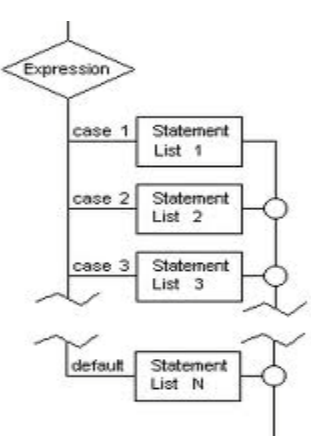
System.out.println("case 3");

**break**;

**default**:

System.out.println("default case");

}



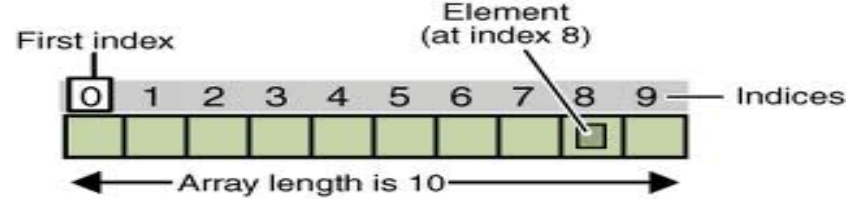
**Arrays**

• Arrays are the collection of similar data types.

• Each variable in an array is known as ‘array element.

• Each variable of array is referenced by a particular integer number which is known as ‘array index’.

• The total number variables in array decide the length of the array.



**Declaration and initialization of array**

• In java array is an object, therefore it is declared and initializes like an object.

• Declaration of array variable:

int[] array;

• Constructing the array:

new int[(length of the array)];

• Assigning array to array variable:

array = new int[(length of the array)];

• Initialization of array:

Array[0] = 34;

• Declaration and initialization in single line:

Int[] array = { 34, 56, 7, 23, 34,};

**Initialization of array using loop**

• Arrays can be initialized by using loops.

• int [] array = new int[34];

• for (int i=0;i<array.length;i++)

• {

• array[i]=i;

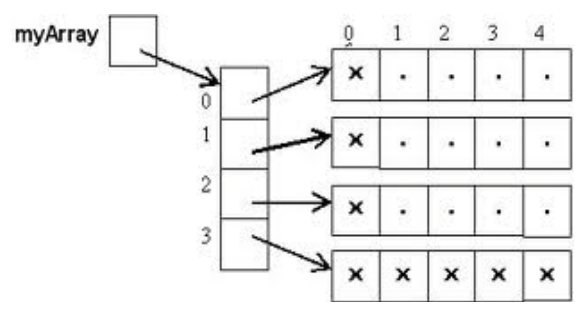
• System.out.println(array[i]);

• }

**Multidimensional arrays**

• Multi-dimensional arrays are nothing but the “array of arrays” where each element represents

a single dimensional array.



• Here ‘myArray’ is a 2-d array, whose each element contains a single dimensional array.

**Initialization and declaration of 2-d array**

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

{

**for**(**int** i=0;i<myArray2.length;i++)

{

**for** (**int** j= 0; j<myArray2[i].length;j++ )

{

myArray2[i][j]= j;

}

}

**for** (**int** i=0; i<myArray2.length;i++)

{

**for** (**int** j=0 ; j<myArray2[i].length; j++)

{

System.***out***.print(myArray2[i][j] + "\t");

//System.out.print("\t");

}

System.***out***.println();

}

}

}

**Program #1 : pyramid of Stars**

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

{

**for** (**int** outer =1; outer<=5; outer++ )

{

**for** (**int** inner = 0; inner<outer; inner++)

{

System.***out***.print("\*");

}

System.***out***.println();

}

}

}

**Enhanced for loop**

• “Enhanced for loop” is introduced in java 5, in order to simply the way to iterate a collection or

array.

• In this the loop continues till the last element of the collection or array.

**for** (**int** y : array)

{

System.out.print(y);

}

**Enhanced for loop for 2-d array**

**for**(**int**[] x : myArray2)

{

**for** (**int** y : x)

{

System.out.print(y + "\t");

}

System.out.println();

}

**Day 4**

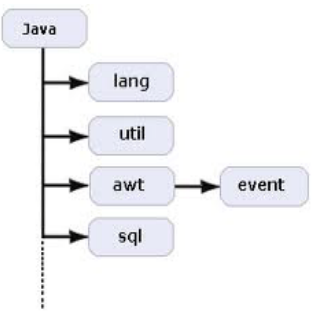
**Packages and Access Modifiers**

**Packages in java**

• A package is the grouping of related types providing access protection and named space management.

• Packages are created by using the keyword “package” and it should be first line of the source file.

• In order to use classes of other packages we have to use “import” statements.



Using import to access packages

**package package1;**

**• public class demoTest {**

**• public void go()**

**• {**

**• System.out.println("in different package");**

**• }**

import package1.\*;

• public class Test {

• int x = 6;

• public static void main(String[] args)

• {

demoTest t = new demoTest();

• t.go();

• }

• }

package first;

public class ClassInFirst {

public void go()

{

System.out.println("in first package");

}

}

package third;

import first.second.\*;

import first.\*;

public class ClassInThird {

public static void main(String[] args)

{

ClassInFirst c = new ClassInFirst();

ClassInSecond d = new ClassInSecond();

c.go();

d.go();

}

}

package first.second;

public class ClassInSecond {

public void go()

{

System.out.println("in second package");;

}

}

**Access Modifiers in java**

Access modifiers specifies access level of a java component.

• Access modifiers can be divided into two categories:

1) Class level

2) Member level

**Class level access modifiers**• Public :

If a class is marked as public then it is accessible anywhere in java world.

Public class demo {}

• Default:

If a class have no modifier, then it will be marked as ‘default’ implicitly, then it is accessible in it’s package only

Class demo()

**Class Member level access modifier**• Public:

If a member is marked as public then it is accessible in whole java world.

• Default:

If a member have no modifier, then it will be marked as ‘default’ implicitly, and accessible in it’s package only.

• Protected:

If a member is marked as protected then it is accessible in it’s package. It is also accessible outside the package but through “inheritance” only.(????)

• Private:

If a member is marked as private then it is accessible in it’s class only

**Static**

A static method means that it can be accessed without creating an object of the class, unlike public:

**Day 5**

**Object Orientation Programming - (OOPS)**

Any language is said to be object oriented if it supports following object properties:

1. Encapsulation

2. Inheritance

3. Polymorphism

4. Data abstraction (or interfaces)

**OOPS - Encapsulation**

• Encapsulation can be described as the mechanism in which we “encapsulate” our code in such a way that it can not be randomly accessed by other code outside the class.

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

• Declare class variables/attributes as private

• provide public get and set methods to access and update the value of a private variable

• If we want to include encapsulation in our code then we have to do following things:

1. Always make instance variable private.

2. Always make public accessor methods and force calling code to use these methods instead of directly calling the instance variables.

3. Use naming convention set() and get() for these

methods.

Problem scenario without encapsulation

• public class Duck {

• private int size;

• public void display()

• {

• if (size<0)

• System.out.println("incorrect size");

• else if(size>10)

• System.out.println("bigger duck!!!");

• else if (size < 10)

• System.out.println("smaller duck!!");

}

}

• public class Test {

• public static void main(String[] args)

• {

• Duck d = new Duck();

• d.size = 45;

• d.display();

• }

• }

**Getters and Setters**

Getters and Setters are nothing but the methods which are used to “set” and “get” the value of instance variables.

In previous chapter, private variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public get and set methods.

• Setters : Setters catch the “value” of instance variable in it’s parameter and “set” or assign this value to the instance variable. Setters have always parameters and no return.

• Getters: Getters returns the value of a instance variable to it’s “caller”. It only returns the already set value of a instance variable. Getters don’t have parameters and always return something

public class Duck {

private int size = 12;

public void setSize(int x)

{

if (x<=0);

else if(x>=25);

else

size = x;

}

public int getSize()

{

return size;

}

public void display()

{

if(size>10)

System.out.println("bigger duck!!!");

else if (size < 10)

System.out.println("smaller duck!!");

}

}

public class Test {

public static void main(String[] args)

{

Duck d = new Duck();

d.setSize(45);

System.out.println("the encapsulated size" +

d.getSize());

d.display();

}

}

**Benefits of Encapsulation** • Code becomes more maintainable and flexible.

• In future we can change our code, without breaking some other code, which depends on our code.

• The class have total control over what is going to be stored over it’s fields.

• The user of the class don’t know how class stores the data.

**Inheritance**

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

- By using inheritance the information becomes more manageable and in a hierarchical order.

- In other word we can say the Inheritance is the relationship between super class and subclass.

-Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

**Why use inheritance in java**

- For Method Overriding (so runtime polymorphism can be achieved).

- For Code Reusability.

**Terms used in Inheritance**

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

**The syntax of Java Inheritance**

class Subclass-name extends Superclass-name

{

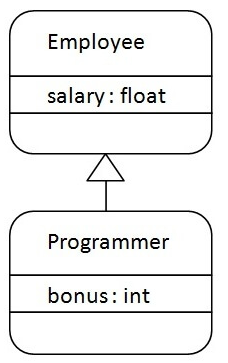
//methods and fields

}

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Java Inheritance Example



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.

class Employee{

float salary=40000;

}

class Programmer extends Employee{

int bonus=10000;

public static void main(String args[]){

Programmer p=new Programmer();

System.out.println("Programmer salary is:"+p.salary);

System.out.println("Bonus of Programmer is:"+p.bonus);

}

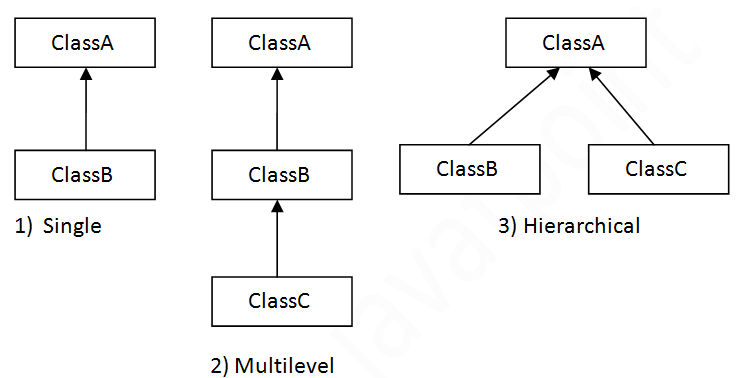
}

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

**Types of inheritance in java**

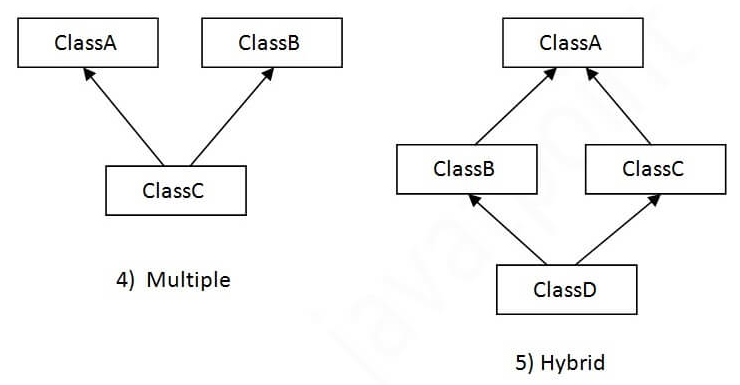
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



**Note: Multiple inheritance is not supported in Java through class.**

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



**Single Inheritance Example**

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class TestInheritance{

public static void main(String args[]){

Dog d=new Dog();

d.bark();

d.eat();

}}

**Multilevel Inheritance Example**

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class BabyDog extends Dog{

void weep(){System.out.println("weeping...");}

}

class TestInheritance2{

public static void main(String args[]){

BabyDog d=new BabyDog();

d.weep();

d.bark();

d.eat();

}}

**Hierarchical Inheritance Example**

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class Cat extends Animal{

void meow(){System.out.println("meowing...");}

}

class TestInheritance3{

public static void main(String args[]){

Cat c=new Cat();

c.meow();

c.eat();

//c.bark();//C.T.Error

}}

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

class A{

void msg(){System.out.println("Hello");}

}

class B{

void msg(){System.out.println("Welcome");}

}

class C extends A,B{//suppose if it were

public static void main(String args[]){

C obj=new C();

obj.msg();//Now which msg() method would be invoked?

}

}

**Aggregation in Java**

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

class Employee{

int id;

String name;

Address address;//Address is a class

...

}

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

**Why use Aggregation?**

- For Code Reusability.

**Simple Example of Aggregation**

|  |  |  |
| --- | --- | --- |
| Circle |  |  |
| Operation op |  | Operation |
| Double pi |  | Square (int n) |
| area (int radius) |  |  |

In this example, we have created the reference of Operation class in the Circle class.

class Operation{

int square(int n){

return n\*n;

}

}

class Circle{

Operation op;//aggregation

double pi=3.14;

double area(int radius){

op=new Operation();

int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).

return pi\*rsquare;

}

public static void main(String args[]){

Circle c=new Circle();

double result=c.area(5);

System.out.println(result);

}

}

**When use Aggregation?**

- Code reuse is also best achieved by aggregation when there is no is-a relationship.

- Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

**Understanding meaningful example of Aggregation**

In this example, Employee has an object of Address, address object contains its own informations such as city, state, country etc. In such case relationship is Employee HAS-A address.

public class Address {

String city,state,country;

public Address(String city, String state, String country) {

this.city = city;

this.state = state;

this.country = country;

}

}

public class Emp {

int id;

String name;

Address address;

public Emp(int id, String name,Address address) {

this.id = id;

this.name = name;

this.address=address;

}

void display(){

System.out.println(id+" "+name);

System.out.println(address.city+" "+address.state+" "+address.country);

}

public static void main(String[] args) {

Address address1=new Address("gzb","UP","india");

Address address2=new Address("gno","UP","india");

Emp e=new Emp(111,"varun",address1);

Emp e2=new Emp(112,"arun",address2);

e.display();

e2.display();

}

}

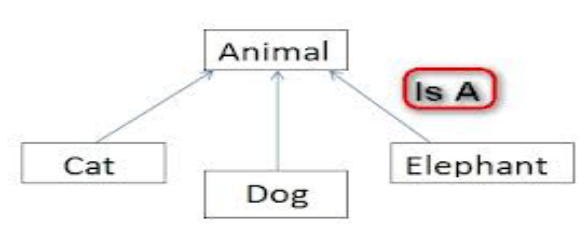
Super class vs. Sub class

• Super class:

**-** This is the class which contains common features of all subclasses

• Sub class:

- This is the class which inherits all the features of super class.



**Using Inheritance**

Inheritance relationship is created by using the keyword “extends”.

public class Animal {} // as superclass

• public class Dog extends Animal {} // Dog is subclass of Animal

• public class Cat extends Dog {} // Cat is subclass of Animal

public class Animal {

public void eat()

{

System.out.println("eating habit");

}

public void roam()

{

System.out.println("raoming habit");

}

}

public class Cat extends Animal{

public void sound()

{

System.out.println("meow meow!!!");

}

}

public class Dog extends Animal{

public void sound()

{

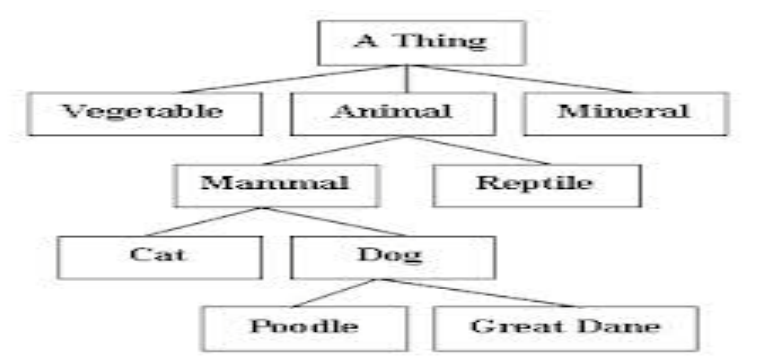
System.out.println("woof woof!!!");

}

}

**Inheritance Tree**

Inheritance Tree is basically a tree structure (upside down) that maps inheritance hierarchies of the classes.



**Method calling in Inheritance tree**

In inheritance tree the method calling is started from the lowest class in the tree.

e.g. if we call eat() method of “poddle” class then it starts searching from “Poddle” to “A Thing”.

**Method Overriding**

- Method Overriding is basically a ability to define a behavior which is specific to that sub class.

- Method Overriding RULES:

1. The arguments and the return type must be same as of superclass method.

2. The access level can’t be decreased.

Ex:

**The final Keyword**

If you don't want other classes to inherit from a class, use the final keyword:

If you try to access a final class, Java will generate an error:

final class Vehicle {

...

}

class Car extends Vehicle {

...

}

**Day 6**

**OOPS - Polymorphism - OverLoading**

**OOPS - Polymorphism - Overriding**

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.

java method overloading

Advantage of method overloading

Method overloading increases the readability of the program.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments

2. By changing the data type

**Note: In Java, Method Overloading is not possible by changing the return type of the method only.**

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

class Adder{

static int add(int a,int b){return a+b;}

static int add(int a,int b,int c){return a+b+c;}

}

class TestOverloading1{

public static void main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}}

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

class Adder{

static int add(int a, int b){return a+b;}

static double add(double a, double b){return a+b;}

}

class TestOverloading2{

public static void main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(12.3,12.6));

}}

Q) Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

class Adder{

static int add(int a,int b){return a+b;}

static double add(int a,int b){return a+b;}

}

class TestOverloading3{

public static void main(String[] args){

System.out.println(Adder.add(11,11));//ambiguity

}}

System.out.println(Adder.add(11,11)); //Here, how can java determine which sum() method should be called?

Note: Compile Time Error is better than Run Time Error. So, java compiler renders compiler time error if you declare the same method having same parameters.

**Can we overload java main() method?**

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

class TestOverloading4{

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

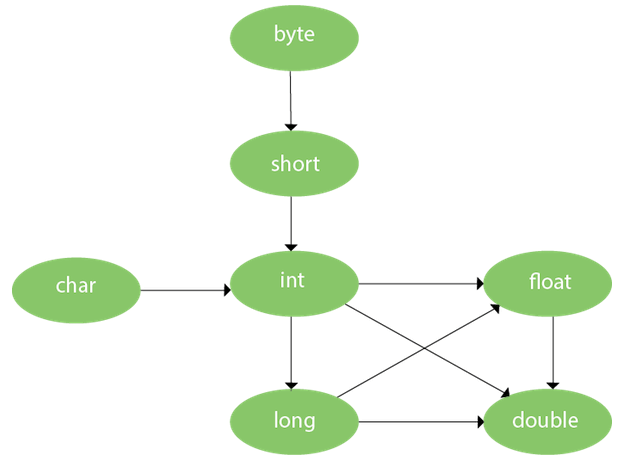
public static void main(String args){System.out.println("main with String");}

public static void main(){System.out.println("main without args");}

}

Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int,long,float or double and so on.

**Example of Method Overloading with TypePromotion**

class OverloadingCalculation1{

void sum(int a,long b){System.out.println(a+b);}

void sum(int a,int b,int c){System.out.println(a+b+c);}

public static void main(String args[]){

OverloadingCalculation1 obj=new OverloadingCalculation1();

obj.sum(20,20);//now second int literal will be promoted to long

obj.sum(20,20,20);

}

}

**Example of Method Overloading with Type Promotion if matching found**

If there are matching type arguments in the method, type promotion is not performed.

class OverloadingCalculation2{

void sum(int a,int b){System.out.println("int arg method invoked");}

void sum(long a,long b){System.out.println("long arg method invoked");}

public static void main(String args[]){

OverloadingCalculation2 obj=new OverloadingCalculation2();

obj.sum(20,20);//now int arg sum() method gets invoked

}

}

**Example of Method Overloading with Type Promotion in case of ambiguity**

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

class OverloadingCalculation3{

void sum(int a,long b){System.out.println("a method invoked");}

void sum(long a,int b){System.out.println("b method invoked");}

public static void main(String args[]){

OverloadingCalculation3 obj=new OverloadingCalculation3();

obj.sum(20,20);//now ambiguity

}

}

One type is not de-promoted implicitly for example double cannot be depromoted to any type implicitly.

# Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.

- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

- The method must have the same name as in the parent class

- The method must have the same parameter as in the parent class.

- There must be an IS-A relationship (inheritance).

Understanding the problem without method overriding

Let's understand the problem that we may face in the program if we don't use method overriding.

//Java Program to demonstrate why we need method overriding

//Here, we are calling the method of parent class with child

//class object.

//Creating a parent class

class Vehicle{

void run(){System.out.println("Vehicle is running");}

}

//Creating a child class

class Bike extends Vehicle{

public static void main(String args[]){

//creating an instance of child class

Bike obj = new Bike();

//calling the method with child class instance

obj.run();

}

}

Problem is that I have to provide a specific implementation of run() method in subclass that is why we use method overriding.

Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

//Java Program to illustrate the use of Java Method Overriding

//Creating a parent class.

class Vehicle{

//defining a method

void run(){System.out.println("Vehicle is running");}

}

//Creating a child class

class Bike2 extends Vehicle{

//defining the same method as in the parent class

void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){

Bike2 obj = new Bike2();//creating object

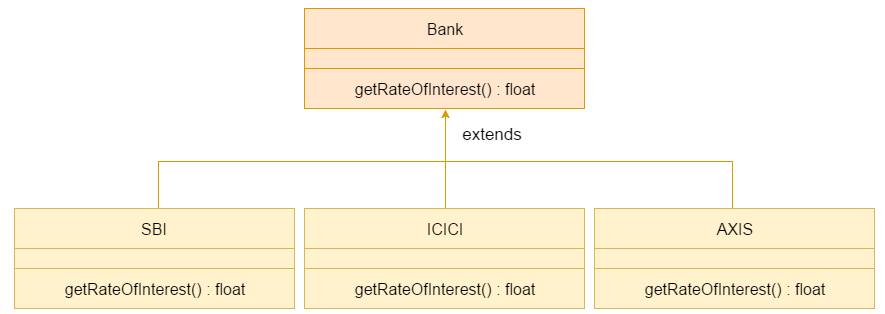
obj.run();//calling method

}

}

**A real example of Java Method Overriding**

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.



Java method overriding is mostly used in Runtime Polymorphism which we will learn in next pages.

//Java Program to demonstrate the real scenario of Java Method Overriding

//where three classes are overriding the method of a parent class.

//Creating a parent class.

class Bank{

int getRateOfInterest(){return 0;}

}

//Creating child classes.

class SBI extends Bank{

int getRateOfInterest(){return 8;}

}

class ICICI extends Bank{

int getRateOfInterest(){return 7;}

}

class AXIS extends Bank{

int getRateOfInterest(){return 9;}

}

//Test class to create objects and call the methods

class Test2{

public static void main(String args[]){

SBI s=new SBI();

ICICI i=new ICICI();

AXIS a=new AXIS();

System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());

System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());

System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());

}

}

### Can we override static method?

No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

### Why can we not override static method?

It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

### Can we override java main method?

No, because the main is a static method.

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter; Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a superclass called Animal that has a method called animalSound(). Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

**Example**

class Animal {

public void animalSound() {

System.out.println("The animal makes a sound");

}

}

class Pig extends Animal {

public void animalSound() {

System.out.println("The pig says: wee wee");

}

}

class Dog extends Animal {

public void animalSound() {

System.out.println("The dog says: bow wow");

}

}

Now we can create Pig and Dog objects and call the animalSound() method on both of them:

**Example**

class Animal {

public void animalSound() {

System.out.println("The animal makes a sound");

}

}

class Pig extends Animal {

public void animalSound() {

System.out.println("The pig says: wee wee");

}

}

class Dog extends Animal {

public void animalSound() {

System.out.println("The dog says: bow wow");

}

}

class MyMainClass {

public static void main(String[] args) {

Animal myAnimal = new Animal(); // Create a Animal object

Animal myPig = new Pig(); // Create a Pig object

Animal myDog = new Dog(); // Create a Dog object

myAnimal.animalSound();

myPig.animalSound();

myDog.animalSound();

}

}

**Day 7**

**OOPS - Abstract classes**

**OOPS - Interfaces**

• Abstraction is a design concept on which we only declare functionality but doesn't define it because we don't know about them at design point.

• Abstract Keyword :

This is a special keyword which is used as a non- access modifiers with classes and methods.

• Abstract Keyword with class:

If ‘abstract’ keyword is used with a class, then no one can instantiate that class and these classes are known as ‘abstract classes’.

• Abstract Keyword with method:

If ‘abstract’ keyword is used with method, then it must be overridden in first concrete class????

**Abstract Class Vs Concrete Class**

• The classes which cannot be instantiated are known as Abstract class.

• Concrete class are those classes which can be intantiated

**Abstract Methods and it’s properties**

**•** The abstract Method is that method which doesn’t contains any body and must be overridden in first concrete class.

**• Properties:**

- Abstract class should always use the keyword ‘abstract’ and in declaration it must be without

body.

- The class of abstract method must be an abstract class. Abstract method cannot be declare in a concrete class.

- The abstract method must be overridden by first

concrete class which extends the abstract class.

**Class-Object**

The class Object is mother of all classes in java.

• Any class which doesn’t extends any class extends class Object.

• Hence every class in java which extends other class is polymorphic, i.e. every class in java “IS-A type of Object”.

**e.g.**

Public class Test{}

Means

Public class Test extends Object{}

**Important Method of Class - Object** equals() method:

This method checks the equality of two objects, whether

they are equal or not

Animal a = new Cat();

Animal b = new Dog();  a.equals(b);

• getClass() method:

This method returns the real class name of it’s

corresponding object.

e.g. b.getClass();

• hashCode()

This method finds the hashCode of a object

e.g. b.hashCode();

• toString():

This method returns the string representation of an object

**Class of reference variable**

• During the call of method of an object, the

compiler searches the method in class of

reference variable and not in the actual class of

the object.

• public static void main(String[] args)

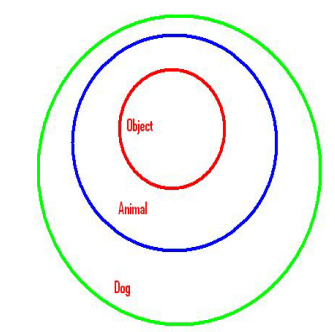
• {

• Object o = new Animal();

• o.eat();

• }

**Object creation in Java-heap**

****

**Interface and it’s declaration**

• Interface is nothing but the pure abstract class which contains only abstract method.

• Interface are always abstract whether we mention it or not.

• Declaration :

In order to declare an interface the keyword “interface” is used.

e.g. public interface MyInterface{ }

**Interface method properties**

• Interface methods are always “public” and “abstract” , so mentioning it as “public” and “abstract” is redundant.

• public abstract interface Test {

• public abstract void go();//public and abstract is redundant

• abstract void go2();//still don't need to use abstract

• void go3();//whether we mention or not...it is always public and abstract

• private abstract void go4(); //it will not compile}

• The return type of interface methods can be anything, just like normal methods, but same should be in overridden method and body of the method should be always empty.

•

• public abstract interface Test {

• String go();

• int first();

• boolean second();}

**Interface method properties continues** • No access modifier apart from public can be used with interface methods (using “public “ is redundant) and no non- access modifier apart from abstract can be used with interface methods (using “abstract” is also redundant).

• We are not allowed to use “final” and “static” as it will restrict the method to overridden.

• public abstract interface Test {

• String go();

• int first();

• static boolean second(); //only public and abstract is

permitted}

• The variables of the interface must be always “public”, “static”, and “final”, hence they are always constant.

• public abstract interface Test {

• String go();

• public static final float PI = 3.14f;}

**Super keyword and it’s use** The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

super can be used to refer immediate parent class instance variable.

super can be used to invoke immediate parent class method.

super() can be used to invoke immediate parent class constructor.

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

class Animal{

String color="white";

}

class Dog extends Animal{

String color="black";

void printColor(){

System.out.println(color);//prints color of Dog class

System.out.println(super.color);//prints color of Animal class

}

}

class TestSuper1{

public static void main(String args[]){

Dog d=new Dog();

d.printColor();

}}

In the above example, Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void eat(){System.out.println("eating bread...");}

void bark(){System.out.println("barking...");}

void work(){

super.eat();

bark();

}

}

class TestSuper2{

public static void main(String args[]){

Dog d=new Dog();

d.work();

}}

In the above example Animal and Dog both classes have eat() method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.

To call the parent class method, we need to use super keyword.

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

class Animal{

Animal(){System.out.println("animal is created");}

}

class Dog extends Animal{

Dog(){

super();

System.out.println("dog is created");

}

}

class TestSuper3{

public static void main(String args[]){

Dog d=new Dog();

}}

Note: super() is added in each class constructor automatically by compiler if there is no super() or this().

As we know well that default constructor is provided by compiler automatically if there is no constructor. But, it also adds super() as the first statement.

**Another example of super keyword where super() is provided by the compiler implicitly.**

class Animal{

Animal(){System.out.println("animal is created");}

}

class Dog extends Animal{

Dog(){

System.out.println("dog is created");

}

}

class TestSuper4{

public static void main(String args[]){

Dog d=new Dog();

}}

super example: real use

Let's see the real use of super keyword. Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default. To initialize all the property, we are using parent class constructor from child class. In such way, we are reusing the parent class constructor.

class Person{

int id;

String name;

Person(int id,String name){

this.id=id;

this.name=name;

}

}

class Emp extends Person{

float salary;

Emp(int id,String name,float salary){

super(id,name);//reusing parent constructor

this.salary=salary;

}

void display(){System.out.println(id+" "+name+" "+salary);}

}

class TestSuper5{

public static void main(String[] args){

Emp e1=new Emp(1,"ankit",45000f);

e1.display();

}}

**Day 8**

**Exception Handling - Part 1**

**Exception Handling - Part 2**

**Day 9**

File handling is an important part of any application.

Java has several methods for creating, reading, updating, and deleting files.

Java File Handling

The File class from the java.io package, allows us to work with files.

To use the File class, create an object of the class, and specify the filename or directory name:

import java.io.File; // Import the File class

File myObj = new File("filename.txt"); // Specify the filename

The File class has many useful methods for creating and getting information about files. For example:

|  |  |  |
| --- | --- | --- |
| Method | Type | Description |
| canRead() | Boolean | Tests whether the file is readable or not |
| canWrite() | Boolean | Tests whether the file is writable or not |
| createNewFile() | Boolean | Creates an empty file |
| delete() | Boolean | Deletes a file |
| exists() | Boolean | Tests whether the file exists |
| getName() | String | Returns the name of the file |
| getAbsolutePath() | String | Returns the absolute pathname of the file |
| length() | Long | Returns the size of the file in bytes |
| list() | String[] | Returns an array of the files in the directory |
| mkdir() | Boolean | Creates a directory |

To create a file in Java, you can use the createNewFile() method. This method returns a boolean value: true if the file was successfully created, and false if the file already exists. Note that the method is enclosed in a try...catch block. This is necessary because it throws an IOException if an error occurs (if the file cannot be created for some reason):

Example

import java.io.File; // Import the File class

import java.io.IOException; // Import the IOException class to handle errors

public class CreateFile {

public static void main(String[] args) {

try {

File myObj = new File("filename.txt");

if (myObj.createNewFile()) {

System.out.println("File created: " + myObj.getName());

} else {

System.out.println("File already exists.");

}

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

To create a file in a specific directory (requires permission), specify the path of the file and use double backslashes to escape the "\" character (for Windows). On Mac and Linux you can just write the path, like: /Users/name/filename.txt

File myObj = new File("C:\\Users\\MyName\\filename.txt");

Write To a File

In the following example, we use the FileWriter class together with its write() method to write some text to the file we created in the example above. Note that when you are done writing to the file, you should close it with the close() method:

Example

import java.io.FileWriter; // Import the FileWriter class

import java.io.IOException; // Import the IOException class to handle errors

public class WriteToFile {

public static void main(String[] args) {

try {

FileWriter myWriter = new FileWriter("filename.txt");

myWriter.write("Files in Java might be tricky, but it is fun enough!");

myWriter.close();

System.out.println("Successfully wrote to the file.");

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

Read a File

In the previous chapter, you learned how to create and write to a file.

In the following example, we use the Scanner class to read the contents of the text file we created in the previous chapter:

Example

import java.io.File; // Import the File class

import java.io.FileNotFoundException; // Import this class to handle errors

import java.util.Scanner; // Import the Scanner class to read text files

public class ReadFile {

public static void main(String[] args) {

try {

File myObj = new File("filename.txt");

Scanner myReader = new Scanner(myObj);

while (myReader.hasNextLine()) {

String data = myReader.nextLine();

System.out.println(data);

}

myReader.close();

} catch (FileNotFoundException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

**Get File Information**

To get more information about a file, use any of the File methods:

Example

import java.io.File; // Import the File class

public class GetFileInfo {

public static void main(String[] args) {

File myObj = new File("filename.txt");

if (myObj.exists()) {

System.out.println("File name: " + myObj.getName());

System.out.println("Absolute path: " + myObj.getAbsolutePath());

System.out.println("Writeable: " + myObj.canWrite());

System.out.println("Readable " + myObj.canRead());

System.out.println("File size in bytes " + myObj.length());

} else {

System.out.println("The file does not exist.");

}

}

}

Note: There are many available classes in the Java API that can be used to read and write files in Java: FileReader, BufferedReader, Files, Scanner, FileInputStream, FileWriter, BufferedWriter, FileOutputStream, etc. Which one to use depends on the Java version you're working with and whether you need to read bytes or characters, and the size of the file/lines etc.

**Reading Properties files**

**Java Streams - Writing in TXT and CSV files**

**Reading Notepad file**

**Reading and Writing Notepad and Properties file - code**

**Day 10**

**Java Stream - Apache POI Creating and Writing Excel Files**

**Java Stream - Apache POI Reading Excel Files, Extended Loops**

**Day 11**

**Day 12**

**Collections - Wrapper Classes, Autoboxing and Unboxing**

**Reflection API**

**Day 13**

**Collections - ArrayList**

**Collections - HashSet**

**Collections - HashMap**

**Day 14**

**Selenium**

**Day 16**

**Browser configurations**

**Day 17**

**Selenium WebDriver Aritecture**

**Day 18 - Day 26**

**TestNg: Configuring TestNg Plugin, Creating Test Cases, Adding Asserttions and Reporting Failures, SoftAssertions and Reporting Multiple Failures,**

**Adding Before and After Suite annotations, Creating Test Dependencies, Creating Test Suite (xml file), Managing Multiple Test Suites,**

**Test Groups - Include and Exclude, TestNg Listeners, Forcefully Skipping a Test Cases, TestNg Report and adding Screenshot to report when Test failure,**

**Parameterization through DataProviders, TestNg Parameterization Excel, TestNg Parameterization Excel With Hashtable, Handling Multiple Data Providers,**

**Reading Parameters from TestNg.xml, Running parallel test using testng.xml, Running parallel test using dataprovider, InvocationCount and ThreadPoolSize**

**Day 27,Day28**

**Locators**

**Day29**

**Selenium Grid, Docker**

**Day 30**

**Reading data from excel files**

**Day 31**

**Handling Synchronizations, Handling multiple windows**

**Day 32**

**Handling Iframes, Using javaScript, Handling alerts,**

**Day 33**

**Extent report, Capture screenshot**

**Day 34**

**Keyboard Events (Robot), Auto It, Sikuli**

**Day 35**

**Database connections, logger**

**Day 36**

**Jenkins, Maven**

**What is Maven:**

- Maven is a project Management tool that is based on POM. It is used for project build, dependancy and documentation.

- it simplifies the build process

Why Maven is required:

- Maven Project eliminates the adding of Jars in each project

- it creates the Project structure

- Building and deploying is very simple

What is Build Tool:

A Build tool takes care of everything for building a process. it does the following:

- Generates source the code

- compiles source code

- Packages compiled code into JAR of ZIP file

- Installs the packaged code in local repository, server or central repository

Maven Repository

there are 3 types of maven repository

1. Local repository: Means .m2 folder in the system

2. Central: Maven central repository is located on web. it has created by the apache maven community itself

3. Remote: Company specific library or custom library

In Pom xml

Project: It is the root element of pom.xml

Model Version: it is the sub element of project. it specifies the model version. it should be set maven version

Group ID: Will identify your project uniquely access all projects (com.test)

Artifact Id: Project Name

packaging defines packing type such as Jar/war...etc

dependencies: defines the dependencies for this project

dependency: defines a dependency. it is used inside dependencies

Scope: defines scope for this maven project. it can be compile, provided, runtime, test and system

<scope>test</scope>

**Maven Build lifecycle**

- it consists different phases lifecycle

1. validate: It checks everything in Order Like POM is Available and the configuration is proper..... these kind of checks are done

2. compile: It will validate and then will compile the java classes and downloads the dependency jars files from POM.XML

3. test: It execute the test cases through Surefire Plug-in by validating and compiling.

4. Package: It will generate the .JAR files. if the above 3 phases are successful, then .jar file will generates.

5. Install: After the package the runs, it installs package into local maven repository.

6. deploy: done in the build environment, copies the final package to the remote repository for sharing with other developers and projects

Add Maven Plug-ins

Add Maven Dependencies

Maven Commands

- Run from Eclipse

- Run from Command line.

MVN package -DskipTests to skip the test cases from command lline

mvn package -Dmaven.test.skip=true

mvn clean install

mvn test

from the POM - need to add in the properties - <maven.test.skip>true</maven.test.skip>

True - all the test cases will be ignores

False - All the test cases will executed

**Jenkins**

**What is Jenkins**

- Jenkins is an open source continuous integration tool which is written in Java

- Jenkins was originally developed as the Hudson project. So if you ever see Hudson then it will be known as Jenkins CI Tool. Hudson’s creation started in summer of 2004 at Sun Microsystems. It was first released in java.net in Feb. 2005.

[Know more about the Jenkins](http://en.wikipedia.org/wiki/Jenkins_(software))

Jenkins with Selenium

- When your test cases are ready and you want to handover to the client or manual testing team so that they can trigger these test cases using single click then Jenkins is the best choice.

- Using Jenkins we can create a build (Build – set of Test case combined together) and we can run easily using a batch file or Git or build.xml or SVN etc.

- In Jenkins, we can schedule the build periodically

Example- You want to run 100 Test case daily at 10 pm then Jenkins will take care of this based on our pattern it will trigger your build

- Email-Notification- Jenkins provide notification emails to once build passed or failed to respective recipients (Depends on t configuration)

Before starting if you have been having Eclipse , TestNG and if you are using Excel sheets or CSV then these jars should be ready.

**We will divide full tutorial into the following section.**

1- Download Jenkins (War file)

2- Configure Jenkins for Running Build (Configure Java,Email notification)

3- Execute Selenium build using Jenkins from git

4- Schedule Jobs in Jenkins to run periodically using cron pattern

Selenium integration with Jenkins

Part 1- Download Jenkins

Step 1- Open your web browser and then Navigate to Below URL

http://jenkins-ci.org this is the official website of Jenkins

Note- Below post will only work if you work with 1.6 version you can download from below url.

https://updates.jenkins-ci.org/download/war/

Step 2- Now download Jenkins.war file and save into desktop or any other location depends on your choice

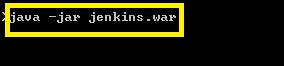
Step 3- Once download complete. You will get Jenkins.war file that we need to execute

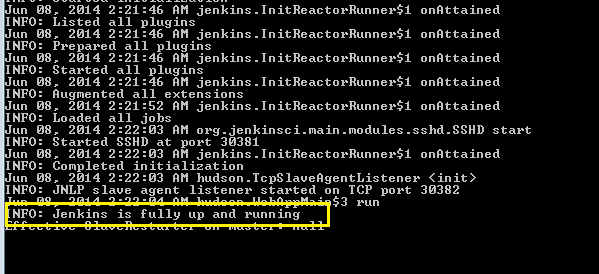
Part 2-How to configure Jenkins for Selenium

Step1: Go to the location where Jenkins.war is available.

Step 2- Open Command prompt knows as CMD and navigate to project home directory and Start Jenkins server

Start- cmd> Project\_home\_Directory> java -jar Jenkins.war





Step 3-

Once Jenkins server is up and running, you will get above success message.

By default Jenkins runs on 8080 port so you can navigate to below URL for Jenkins UI

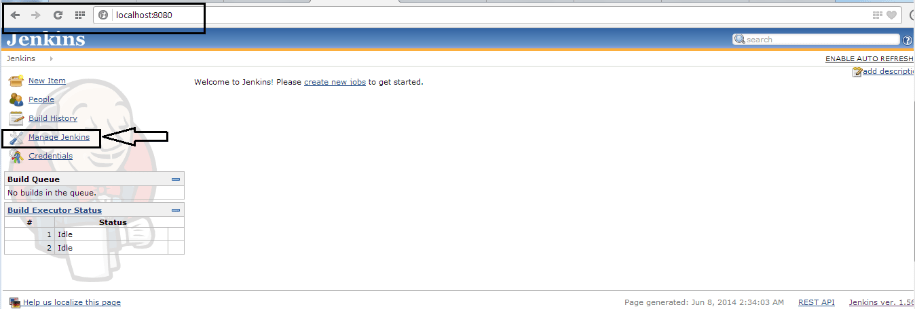
Open any browser and type the URL http://localhost:8080

Now Jenkins in up and running so now we have to configure Jenkins so that we can execute our test case via Jenkins.

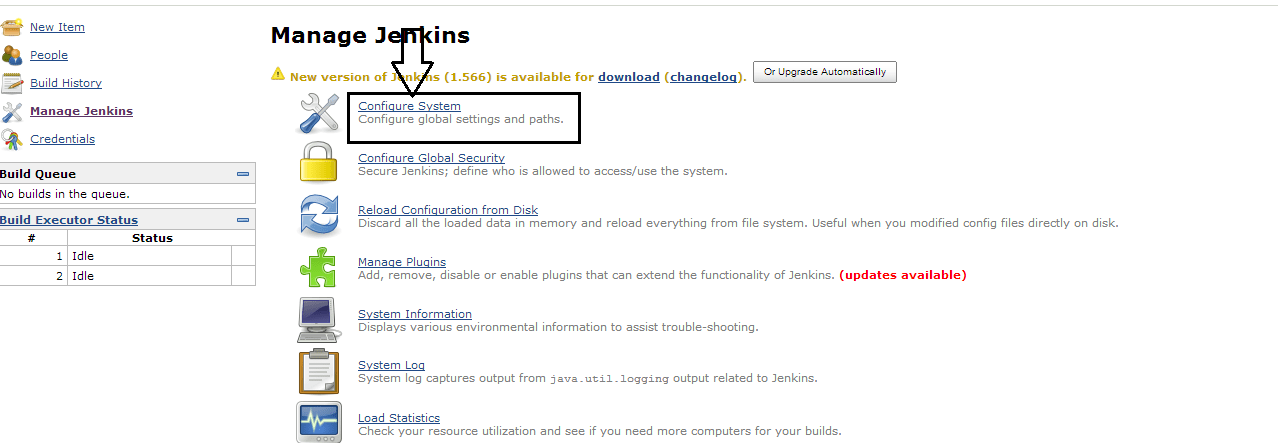
Step 4-

Once Jenkins is running so we are almost done but before moving to create build we need to configure Jenkins so that Jenkins can identify other tools as well like Java, Maven etc.

Click on > Manage Jenkins

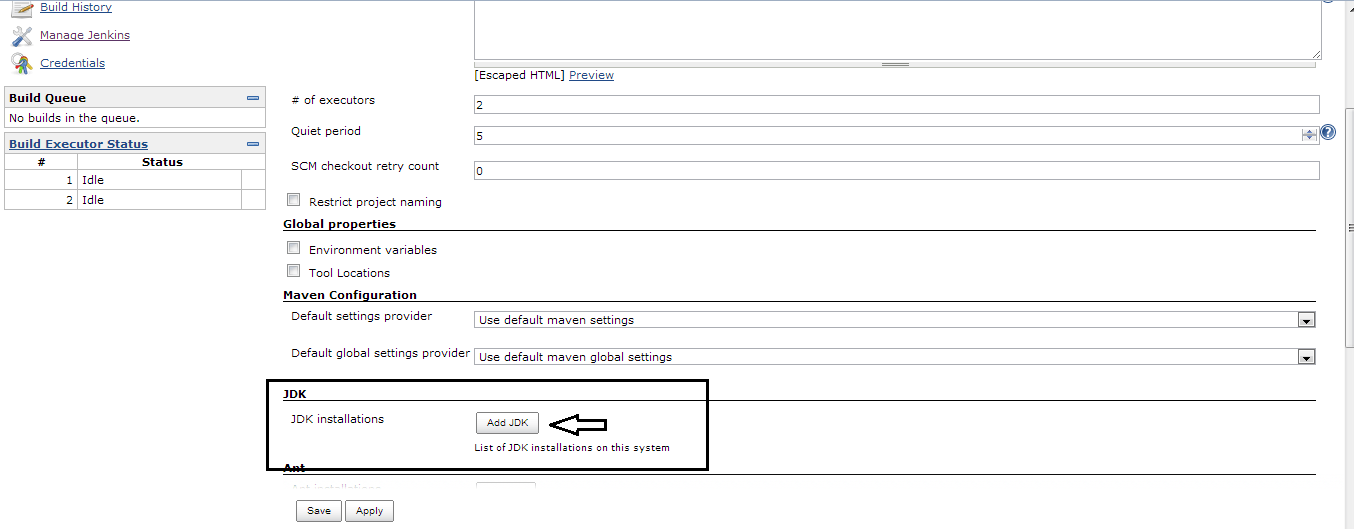


Click on Configure System

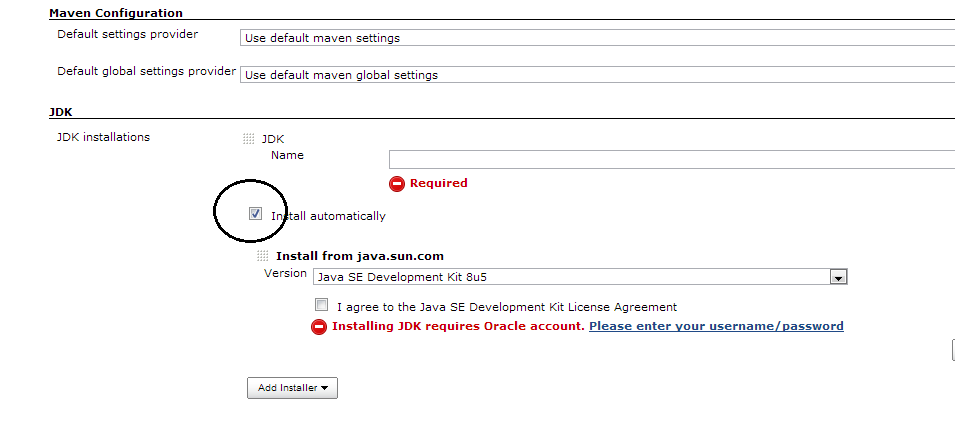


Here we are telling Jenkins that our java is located at this location so we don’t have to worry about the explicit path.

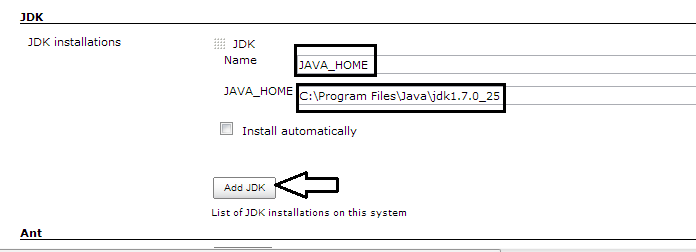
Navigate to JDK section and Click on Add JDK button



Uncheck Install automatically check box so Jenkins will only take java which we have mentioned above.



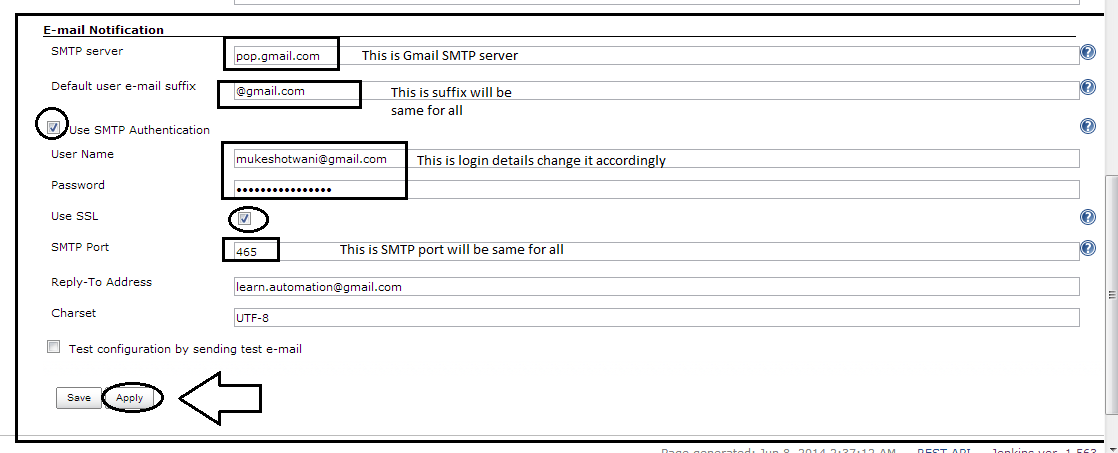
Give the name as JAVA\_HOME and Specify the JDK path



In Jenkins, we have a very good feature that you can configure email notification for the user.

This is optional but if you want to configure Email notification then you have to do little setting while configuring Jenkins

Refer below screenshot you can change login details and click on Apply.



**Once done click on save and apply.**

Now, your Jenkins is configured now.

**Part 3- Execute Selenium build using Jenkins by integrating with Git.**

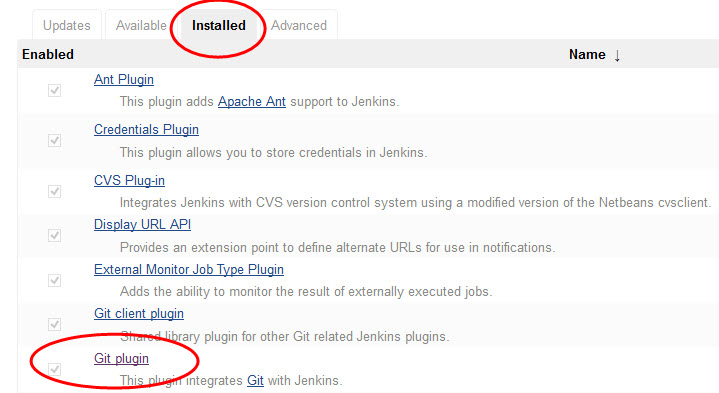
**Configure Git Plugin in Jenkins**

Step 1:- Manage Plugins -> Filter list of plug-ins availble with 'Git Plugin'. Find more details about Git Plug-in

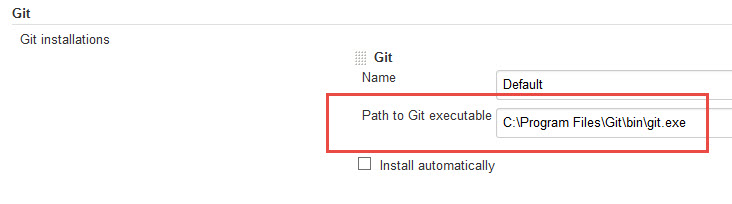
Step 2:- Check the Git Plug-in and click on the button ‘Install without restart’

Step 3:- After the installations are done , Please restart Jenkins by using the command in the browser. http://localhost:8080/jenkins/restart

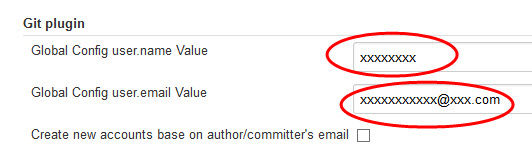
Once Jenkins is restarted, Git option should be available under source code management when configuring job.



Step 4:- From Manage Jenkins > Configure System, please provide the right Path to Git executable.



Step 5:- Under Git Plug-in, Set the global git user.name and user.email to match your global config options

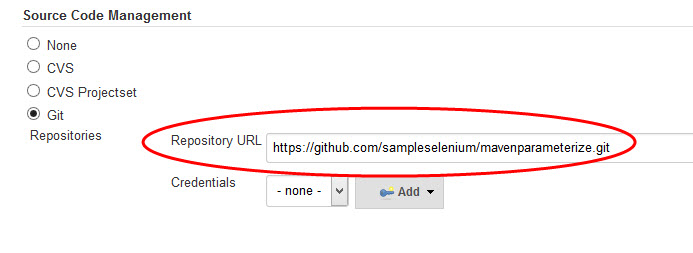


Now let us create a Maven Project and build a job from Git project

Step 1:- Click New Items -> Enter Project Name -> Select Maven Project -> Click OK

Step 2:- Provide the job description

Step 3:- In Source Code Management, Select 'Git' option. (This should be visible once you have successfully installed Git Plugin).



Step 4:- From Build Triggers, If you want Jenkins to monitor the repository and start a build whenever any changes have been committed, We can choose to pick the Poll SCM option and enter syntax of cron.

The other options include Build periodically (for example, once a day, ), Build whenever a SNAPSHOT dependency is built etc.

**Part 4-Schedule your build in Jenkins for periodic execution**

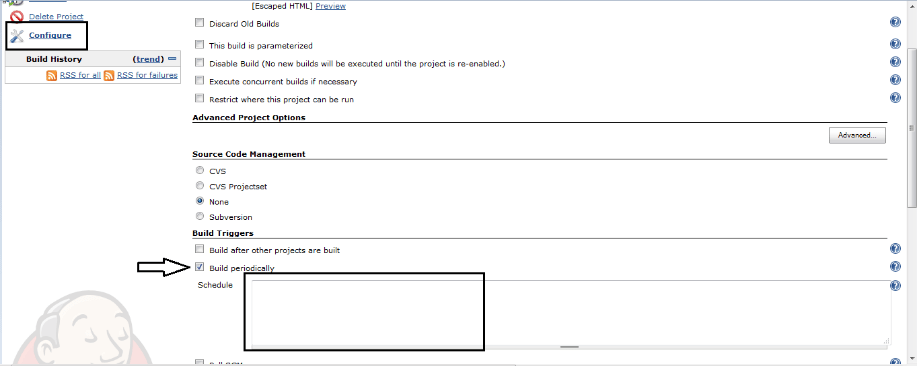
Jenkins comes with very good functionality in which we can schedule jobs which we created

You can schedule build for existing jobs which already created and while creating new project also we can specify the same.

Let’s schedule the job. Refer the below screenshot

Step 1-

Open job which we created now and Click on configure > select the check box build periodically



Step 2-

Specify the time here we need to careful about the syntax

Jenkins works on Cron pattern for more info about cron refer cron link [http://en.wikipedia.org/wiki/Cron](http://en.wikipedia.org/wiki/Cron%20)

enkins will accept 5 parameter lets discuss one by one

\* \* \* \* \*

Here the first parameter- specify minute and range will vary from 0-59

Here the second parameter- specify hours and range will vary from 0-11

Here the third parameter- specify day and range will vary from 0-7 here 0 and 7 will be Sunday

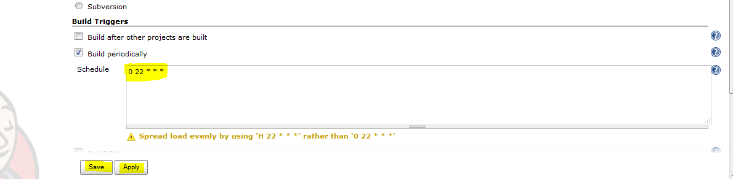
Here the fourth parameter- specify month and range will vary from 1-12

Here the fifth parameter- specify year so here you can specify \*

Example 1- if you specify 00 22 \* \* \* it means your build will run daily @ 10 PM

Example 2- if you specify 50 \* \* \* \* it means your build will run daily 50 min after every hour

Example 3- if you specify 00 22 1 \* \* it means your build will run every Monday @ 10 PM



Finally, we have executed our test case successfully.

**Day 37**

**Cucumber – a Behavior Driven Development (BDD) framework which is used with Selenium for performing acceptance testing.**

**Cucumber Introduction**

A cucumber is a tool based on Behavior Driven Development (BDD) framework which is used to write acceptance tests for the web application. It allows automation of functional validation in easily readable and understandable format (like plain English) to Business Analysts, Developers, Testers, etc.

Cucumber feature files can serve as a good document for all. There are many other tools like JBehave which also support BDD framework. Initially, Cucumber was implemented in Ruby and then extended to Java framework. Both the tools support native JUnit.

Behavior Driven Development is an extension of Test Driven Development and it is used to test the system rather than testing the particular piece of code. We will discuss more the BDD and style of writing BDD tests.

Cucumber can be used along with Selenium, Watir, and Capybara etc. Cucumber supports many other languages like Perl, PHP, Python, Net etc. In this tutorial, we will concentrate on Cucumber with Java as a language.

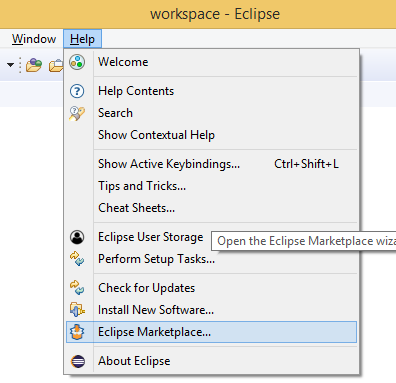
**Cucumber Basics:**

**Installation:**

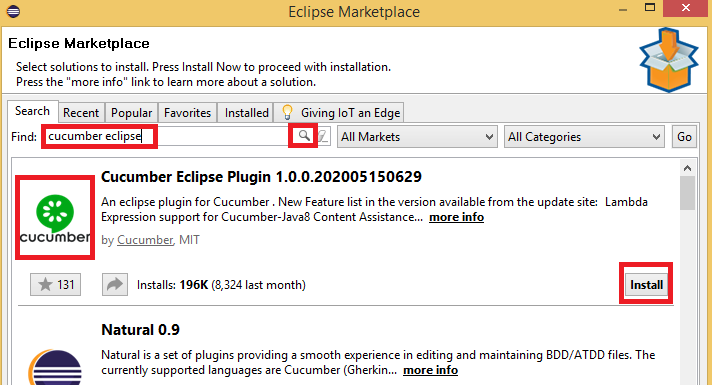
**Add Eclipse Cucumber Plugin**

Steps

Step1: Open Eclipse --> Help--> Eclipse Market Place

****

Step2: Enter Cucumber eclipse in the find filed and click on search icon as shown in the below image

****

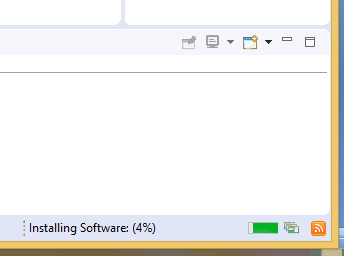
Step3: Click on Install for the Cucumber eclipse plugin

Step4: Click on Confirm

Step5: Select the terms & conditions and click on Finish

Step6: click on install install if it prompted

Step7: It will install the software as below

****

Step7: Restart the Eclipse

**#1) Feature Files:**

Feature files are the essential part of cucumber which is used to write test automation steps or acceptance tests. This can be used as the live document. The steps are the application specification. All the feature files end with .feature extension.

Sample feature file:

Feature: Login Functionality Feature

In order to ensure Login Functionality works,

I want to run the cucumber test to verify it is working

Scenario: Login Functionality

Given user navigates to SOFTWARETETINGHELP.COM

When user logs in using Username as “USER” and Password “PASSWORD”

Then login should be successful

Scenario: Login Functionality

Given user navigates to SOFTWARETETINGHELP.COM

When user logs in using Username as “USER1” and Password “PASSWORD1”

Then error message should be thrown

**#2) Feature:**

This gives information about the high-level business functionality (Refer to the previous example) and the purpose of Application under test. Everybody should be able to understand the intent of feature file by reading the first Feature step. This part is basically kept brief.

**#3) Scenario:**

Basically, a scenario represents a particular functionality which is under test. By seeing the scenario user should be able to understand the intent behind the scenario and what the test is all about. Each scenario should follow given, when and then format. This language is called as “gherkin”.

Cucumber Annotations

These are inbuilt to Cucumber. Normally tags are @Given, @When, @Then.

**Given:** As mentioned above, given specifies the pre-conditions. It is basically a known state.

**When:** This is used when some action is to be performed. As in above example, we have seen when the user tries to log in using username and password, it becomes an action.

**Then:** The expected outcome or result should be placed here. For Instance: verify the login is successful, successful page navigation.

**And:** And is used to combine two or more same type of action.

Example:

Feature: Login Functionality Feature

Scenario: Login Functionality

Given user navigates to SOFTWARETETINGHELP.COM

When user logs in using Username as “USER”

And password as “password”

Then login should be successful

And Home page should be displayed

**Background:** Whenever any step is required to perform in each scenario then those steps need to be placed in Background. i.e., These are steps or series of steps that are common to all the scenarios in the feature file.

It allows us to add some context to the scenarios for a feature where it is defined. It runs before every scenario for a feature in which it is defined.

Example of Background:

Background:

Given user logged in as databases administrator

And Open the Browser

**#4) Scenario Outline:**

Scenario outlines are used when the same test has to be performed with different data set. Let’s take the same example. We have to test login functionality with multiple different sets of username and password.

Feature: Login Functionality Feature

In order to ensure Login Functionality works,

I want to run the cucumber test to verify it is working

Scenario Outline: Login Functionality

Given user navigates to SOFTWARETESTINGHELP.COM

When user logs in using Username as <username> and Password <password>

Then login should be successful

Examples:

|username |password |

|Tom |password1 |

|Harry |password2 |

|Jerry |password3 |

**Note:**

As shown in above example column names are passed as a parameter to When statement.

In place of Scenario, you have to use Scenario Outline.

Examples are used to pass different arguments in the tabular format. Vertical pipes are used to separate two different columns. An example can contain many different columns.

**Creating a Feature File**

1. Go to src/test/resources and create a package as features

2. Right click on features>>new>> File>> give the feature name with extension as .feature (Ex: for login, give the name as login.feature)

3. Give the details in the feature file as below

**Creating Step definition and runner files,**

1. Go to src/main/java and Create a package as Stepdefinations

2. right click on Stepdefinations and create a StepDefinations of the pages(HomePageStepDefination, LoginPageStepDefinations)

3. Create the Steps in the respective pages

#1) Cucumber Hooks

These are the blocks of the code that runs before or after each scenario. So that we can define these, anywhere in our project. For Example, Step Definition.

As per the definition, these are just two annotation @After and @Before. In the console, we can see the blocks getting executed and giving clear output. We can also execute the hooks for specific Tags.

However, later if we need we can create our own annotation and then use it in our program. During execution, the matching glue code i.e. functions are written in a Stepdef file having @Given, @When, @Then will get executed.

**Runner file:**

1. Adding Feature to runner file - features="src/test/resources/features/fileName.feature", // runs your feature file

2. Glue - glue="com.projectName.cucumber.steps", This matches your steps in Stepdefinitions

3. Monochrome - monochrome = true, this is give the readable output

4. Tags:

Cucumber by default runs all scenarios in all the feature files. In real time projects, there could be hundreds of feature file which are not required to run at all times.

For instance: Feature files related to smoke test need not run all the time. So if you mention a tag as smokeless in each feature file which is related to smoke test and runs cucumber test with @SmokeTest tag. Cucumber will run only those feature files specific to given tags. Please follow the below example. You can specify multiple tags in one feature file**.**

Example of use of single tags:

@SmokeTest

Feature: Login Functionality Feature

In order to ensure Login Functionality works,

I want to run the cucumber test to verify it is working

Scenario Outline: Login Functionality

Given user navigates to zoho.com

When user logs in using Username as <username> and Password <password>

Then login should be successful

Examples:

|username |password |

|Tom |password1 |

|Harry |password2 |

|Jerry |password3 |

Example of use of multiple tags:

As shown in below example same feature file can be used for smoke test scenarios as well as for login test scenario. When you intend to run your script for a smoke test then use @SmokeTest. Similarly when you want your script to run for Login test use @LoginTest tag.

Any number of tags can be mentioned for a feature file as well as for scenario.

@SmokeTest @LoginTest

Feature: Login Functionality Feature

In order to ensure Login Functionality works,

I want to run the cucumber test to verify it is working

Scenario Outline: Login Functionality

Given user navigates to zoho.com

When user logs in using Username as <username> and Password <password>

Then login should be successful

Examples:

|username |password |

|Tom |password1 |

|Harry |password2 |

|Jerry |password3 |

Similarly, you can specify tags to run the specific scenario in a feature file. Please check below example to run specific scenario.

Feature: Login Functionality Feature

In order to ensure Login Functionality works,

I want to run the cucumber test to verify it is working

@positiveScenario

Scenario: Login Functionality

Given user navigates to zoho.com

When user logs in using Username as “USER” and Password “PASSWORD”

Then login should be successful

@negaviveScenario

Scenario: Login Functionality

Given user navigates to zoho.com

When user logs in using Username as “USER1” and Password “PASSWORD1”

Then error message should throw

Adding tagged Hooks

**Day 38**

**Datatables,**

**Day 39**

**Adding PICO containers, Adding TestNG Runner, Generating Cucumber Reports, Integrating Cucumber Extent Report Plugin,**

**Day 40**

**BDD Framework: POM with Page Factories and generating extent with Parallel execution on Local System, On Selenium Grid and by using Docker**