1. **Steps to install java**

JDK == JAVA Devlopment Kit The latest version of JDk is 1.8

Steps to Install JDK:

1. Go to Google , Type JDK Downloads

2. Select the First Link

3. JDK Download TAB

4. Check the Accept License Agrement Radio button

5. Click on the Windows 64 bit JAVA X SE downloads.

Steps of Installation:

1. Double Click on the File Downloaded, Right click and Run

2. Click Next, Next twice to finish the Installation.

3. Go to the Bin Folder and Copy the Bin Path from the Adress bar, C:/ Program Files/JAVA/JDK1.8/Bin

4. Right click on My Computer / Select Properties / Click on Advanced System Settings / Click on Environment Variables

5. In the System Variables Tab, go down and Select Path and Click on Edit/ Take the cursor to the end of the Path and type Semicolon ; and Paste the JAVA Bin Path copied from the Address bar

6. Click on OK OK.

7. To verify the Installation/ go to Command Prompt/ click on Start, type cmd in Search and u will get command prompt/

8. Type java -version and click Enter/

this must display the Version of JAVA -- "1.8.0\_45"

1. **Steps to install eclipse.**

Eclipse :

1. Go to google, Type Eclipse Downloads

2. Select the first Link, Select, Eclipse IDE for JAVA Development

3. Download the required OS bit version of Eclipse / 32 or 64 bit.

4. After the Download, there will be a Compressed ZIP folder of Eclipse

5. Extract the Compressed file, and copy the Setup Folder into a required Destination.

6. Inside an Extracted Folder, we will find an Eclipse Application Icon, Right click on that, Send to Desktop.

7. Shortcut of Eclipse gets created and we are good to use Eclipse.

Double click on the Eclipse Icon created/

It will ask for a Workspace: Give the workspace path and click on OK to get started.

1. **Steps to create workspace**

In Preferences->Workspaces, make sure that 'Prompt for Workspace on startup' is checked. Then you'll be prompted for a workspace to open. You can create a new workspace from that dialogue. You can createmultiple workspaces in Eclipse.

**4. Steps to create project**

1. Click on File - Select New - Select Project - Select JAVA -- Select JAVA Project

2. Click on NEXT -- Give the Project Name -- Click on Next and Finish.

3. We will see the Project Name in a form of a Folder under Project Explorer TAB.

**5. How to Create a Package Under a Project in Eclipse ?**

1. Right Click on the Project Created /-- Select New /-- Select Package.

2. It prompts for a Package Name.

General Naming Convention followed for the Package Name is

Eg : Google / google.com

Package name : com.google

**6. create .java file/class**

Right click on the package and select new class and give the name of the class in camel cases starting with capital letter. Make sure that the class starts with letter or any special character it should not start with number.

**7. how to create packages and what is best way to give name**

A package is a collection of related classes. It helps Organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve re-usability.

Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.

Although interfaces and classes with the same name cannot appear in the same package, they can appear in different packages. This is possible by assigning a separate namespace to each package.

Syntax:-

package;

**8. what is main method will do?**

When a program starts running, it has to start execution from somewhere. That somewhere is called

Main. You can compile any Java class without a main method, but a standalone application can't run without a main() method \*.

The main method is the method that's defined to be called at the start of an application. Without it, there is no place to start running.

**9. creating property/data members :**

Whatever the DATA or the INPUT being used in the JAVA Program are called as Data Members

We know that a class is a collection of data members and methods. In java programming we have two types of data members they are  
  
                     1. Instance/non-static data members  
                     2. Static data members

TO create a data member

**Syntax: -**  
     Data type v1, v2, v3 ………. Vn;

**10. what is data type and different data types?**

It tells us what type of data the variable is.

There are 8 primitve data types:

Boolean, char, short, int, long, float, double

**11. What is variable?**

A variable provides us with named storage that our programs can manipulate. Aka A variable is something which stores the value temporarily or throughout the program.

**12. creating method with void**

Creating a method with return type as void means the method doesn’t return anything after execution.

**void** insertRecord(**int** r, String n){

  rollno=r;

  name=n;

 }

**13. creating variable, we can create variables inside method**

Syntax for creating variable

**Syntax: datatype variable;**

The variables can be created inside the methods but it is restricted to that method only.

**14. creating method with return data type, int/string/double/float/date etc**

The method with return datatype with int/string/double/float/date etc would return that specific datatype to it’s super class.

**15. method that will return hard coded value**

The method will return the hard-coded value if the actual value is forced into the source code.

**16. Create default/paramterzied constructors**

Developing The Constructor with Argument is called as Argument or Parameterized Constructors

If The Programmer has defined the Constructor, then The compiler will not create Default Constructor

public class DemoConst2

{

int i = 25;

double d = 30000;

// Constructor with one double Argument and one integer Argument

DemoConst2(double p, int q)

{

this.i = q;

this.d = p;

System.out.println(" Invoking the Constructor" );

System.out.println(" the Value of q is " +q);

System.out.println(" the Value of p is " +p);

System.out.println();

}

public static void main(String[] args)

{

DemoConst2 a5 = new DemoConst2(37.8,5);

System.out.println(a5.i );

System.out.println(a5.d );

System.out.println(" The Program Starts" );

System.out.println();

}

}

**17. method that will return property value**

**18. creating method with return data type and parameter**

Java program that calls method in return statement

public class Program {

static int cube(int value) {

// Return number to the power of 3.

return (int) Math.pow(value, 3);

}

static int getVolume(int size) {

// Return cubed number.

return cube(size);

}

public static void main(String[] args) {

// Assign to the return value of getVolume.

int volume = getVolume(2);

System.out.println(volume);

}

}

**Output** 8

**18. creating static property:**

static variable (“static” Keyword = **Class Variables**) In Java Variables can be declared with the “ static ” keyword. Example: static int y = 0; When a variable is declared with the keyword static , its called a **class variable** .

**class** Counter2{

**static** **int** count=0;//will get memory only once and retain its value

Counter2(){

count++;

System.out.println(count);

}

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

Counter2 c1=**new** Counter2();

Counter2 c2=**new** Counter2();

Counter2 c3=**new** Counter2();

 }

}

**Output**:1

2

3

**19. creating static method**

If you apply static keyword with any method, it is known as static method.

A static method belongs to the class rather than object of a class.

A static method can be invoked without the need for creating an instance of a class.

static method can access static data member and can change the value of it.

//Program of changing the common property of all objects(static field).

**class** Student9{

**int** rollno;

     String name;

**static** String college = "ITS";

**static** **void** change(){

     college = "BBDIT";

     }

     Student9(**int** r, String n){

     rollno = r;

     name = n;

     }

**void** display (){System.out.println(rollno+" "+name+" "+college);}

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

    Student9.change();

    Student9 s1 = **new** Student9 (111,"Karan");

    Student9 s2 = **new** Student9 (222,"Aryan");

    Student9 s3 = **new** Student9 (333,"Sonoo");

    s1.display();

    s2.display();

    s3.display();

    }

}

**Output**: 111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

**20. create static block**

Is used to initialize the static data member. It is executed before main method at the time of classloading.

**class** A2{

**static**{System.out.println("static block is invoked");}

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

   System.out.println("Hello main");

  }

}

Output:static block is invoked

Hello main

**21. creating object**

An entity that has state and behavior is known as an object. An Object is an instance of a class.  Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

**22. calling method with void**

**class** Student{

**int** rollno;

 String name;

**void** insertRecord(**int** r, String n){

  rollno=r;

  name=n;

 }

**void** displayInformation(){System.out.println(rollno+" "+name);}

}

**class** TestStudent4{

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

  Student s1=**new** Student();

  Student s2=**new** Student();

  s1.insertRecord(111,"Karan");

  s2.insertRecord(222,"Aryan");

  s1.displayInformation();

  s2.displayInformation();

 }

}

Output: 111 Karan

222 Aryan

**23. calling method with no return and parameter**

public class ExampleVoid {

public static void main(String[] args) {

methodRankPoints(255.7);

}

public static void methodRankPoints(double points) {

if (points >= 202.5) {

System.out.println("Rank:A1");

}else if (points >= 122.4) {

System.out.println("Rank:A2");

}else {

System.out.println("Rank:A3");

}

}

}

**Output**

Rank:A1

**24. calling method with return and no parameter**

**public** **class** TestMethod {

**static** **int** *x*=19;

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

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

{

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

}

}

**static** **int** fill(){

*x*=*x*+1;

**return** *x*;

}

}

Output:

20

21

22

23

**25. calling method with return and parameter**

**public** **class** ReturnParameter {

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

// **TODO** Auto-generated method stub

System.***out***.print(*add*(1, 3));

}

**public** **static** **int** add(**int** value1, **int** value2) {

**return** value1 + value2;

}

}

**Output 4**

**26. calling method with return and storing the return data**

**package** my\_training;

**public** **class** StoreReturnValue {

**static** **int** *value*;

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

// **TODO** Auto-generated method stub

*value* = *add*(1, 3);

System.***out***.print(*value*);

}

**public** **static** **int** add(**int** value1, **int** value2) {

**return** value1 + value2;

}

}

**Output 4**

**27. calling static method**

**public** **class** StaticMethodCall {

**public** **static** **void** method(){

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

}

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

StaticMethodCall s = **null**;

s.*method*();

}

}

**Output:** Called

**28. using static property: it will maintain**

**29. create classes under multiple packages**

package package1;

public class Package1Class {

}

package package2;

public class Package2Class {

}

**30. calling classes under different packages**

package package1;

public class Package1Class {

}

package package2;

import package1.Package1Class;

public class Package2Class {

private Package1Class x;

public Package2Class (Package1Class x) {

this.x = x

}

}

**31. write code to handle exceptions with try/catch/finally**

**package** training;

**public** **class** Exception {

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

{

System.***out***.println(" The Program Starts" );

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

// Declaring An Array of Size 2

**int** [] arr1 = **new** **int** [2];

arr1[0] = 25;

arr1[1] = 55;

System.***out***.println("First Element is "+arr1[0] );

System.***out***.println("Second Element is "+arr1[1] );

**try**

{

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

System.***out***.println("Entering Try Block" );

System.***out***.println("First Element is "+arr1[2] );

}

**catch**(ArrayIndexOutOfBoundsException e1)

{

System.***out***.println("Array Index Out of bound Exception Caught in Catch block" );

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

}

**catch**(ArithmeticException e2)

{

System.***out***.println("Arithmetic Exception of dvision by Zero Caught in Catch block" );

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

}

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

System.***out***.println(" The Program Ends" );

}

}

**Output:**

**The Program Starts**

First Element is 25

Second Element is 55

Entering Try Block

Array Index Out of bound Exception Caught in Catch block

The Program Ends

**32. what is checked exception/unchecked exception**

**Checked Exception:**

are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws*keyword.

For example, consider the following Java program that opens file at locatiobn “C:\test\a.txt” and prints first three lines of it. The program doesn’t compile, because the function main() uses FileReader() and FileReader() throws a checked exception *FileNotFoundException*. It also uses readLine() and close() methods, and these methods also throw checked exception *IOException*

**Unchecked Exception:**

are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.  
In Java exceptions under *Error*and *RuntimeException*classes are unchecked exceptions, everything else under throwable is checked.

**33. what is final keyword, create final class, final method, final property**

the **final keyword** is used in several different contexts to define an entity that can only be assigned once. Once a **final** variable has been assigned, it always contains the same value.

**34. write code for interface and create class to implement that interface**

**public** **class** Interfaces {

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

shape circleshape=**new** circle();

circleshape.Draw();

}

}

**interface** shape

{

**public** String ***baseclass***="shape";

**public** **void** Draw();

}

**class** circle **implements** shape

{

**public** **void** Draw() {

System.***out***.println("Drawing Circle here");

}

}

**Output:** Drawing Circle here

**35. write code for creating abstract class**

A method that is declared as abstract and does not have implementation is known as abstract method.

**abstract** **class** AbstractClassBike {

**abstract** **void** run();

}

**class** Honda4 **extends** AbstractClassBike{

**void** run(){

System.***out***.println("running safely..");

}

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

AbstractClassBike obj = **new** Honda4();

obj.run();

}

}

**Output:**

running safely..

**36. implement method overloading**

**package** training;

**abstract** **class** Gmail

{

**abstract** **void** Display();

}

**class** Inbox **extends** Gmail

{

// Method OverRiding

**void** Display()

{

System.***out***.println("Displaying Inbox Details");

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

}

}

**class** Trash **extends** Gmail

{

// Method overloading

**void** Display(**int** a)

{

System.***out***.println("Method Overloaded with an Integer Argument");

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

}

**void** Display()

{

System.***out***.println("Displaying Trash Details");

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

}

}

**class** Spam **extends** Gmail

{

// Method OverRiding

**void** Display()

{

System.***out***.println("Displaying Spam Details");

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

}

}

**class** MyClass

{

**static** **void** Test(Gmail g1)

{

g1.Display();

}

**static** **void** Test(**int** a)

{

System.***out***.println("Method of Test Overloaded with an Integer Argument");

}

}

**public** **class** MethodOverload {

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

System.***out***.println("The Program Starts");

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

MyClass.*Test*(**new** Inbox());

MyClass.*Test*(**new** Trash());

MyClass.*Test*(**new** Spam());

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

System.***out***.println("The Program Ends");

}

}

**Output**: The Program Starts

Displaying Inbox Details

Displaying Trash Details

Displaying Spam Details

The Program Ends

**37. implement method overriding**

**package** training;

// Super Class

**class** X

{

**void** test1()

{

System.***out***.println("Running the Test1 Method of Class X ");

//System.out.println();

}

}

// Y is a Subclass

**class** Y **extends** X

{

// Test1 Method of Super class (Class X) is Overloaded in Sub Class y

**void** test1()

{

// Method Overriding because The test1() method s Signature is same..

System.***out***.println("OverRiding The Test1 Method of Class X in Y");

System.***out***.println("Changing the Implementation of Test1 Method of Class X in Y");

}

}

**public** **class** Override {

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

{

System.***out***.println("The program starts");

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

X x1 = **new** X();

Y y1 = **new** Y();

y1.test1();

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

x1.test1();

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

System.***out***.println("The program starts");

}

}

Output:

The program starts

OverRiding The Test1 Method of Class X in Y

Changing the Implementation of Test1 Method of Class X in Y

Running the Test1 Method of Class X

The program starts

**38. implementing polymorphism**

**package** training;

**abstract** **class** Gmail

{

**abstract** **void** Display();

}

**class** Inbox **extends** Gmail

{

// Method OverRiding

**void** Display()

{

System.***out***.println("Displaying Inbox Details");

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

}

}

**class** Trash **extends** Gmail

{

// Method overloading

**void** Display(**int** a)

{

System.***out***.println("Method Overloaded with an Integer Argument");

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

}

**void** Display()

{

System.***out***.println("Displaying Trash Details");

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

}

}

**class** Spam **extends** Gmail

{

// Method OverRiding

**void** Display()

{

System.***out***.println("Displaying Spam Details");

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

}

}

**class** MyClass

{

**static** **void** Test(Gmail g1)

{

g1.Display();

}

**static** **void** Test(**int** a)

{

System.***out***.println("Method of Test Overloaded with an Integer Argument");

}

}

**public** **class** MethodOverload {

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

System.***out***.println("The Program Starts");

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

MyClass.*Test*(**new** Inbox());

MyClass.*Test*(**new** Trash());

MyClass.*Test*(**new** Spam());

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

System.***out***.println("The Program Ends");

}

}

**Output:**

The Program Starts

Displaying Inbox Details

Displaying Trash Details

Displaying Spam Details

The Program Ends

**39. implementing interface**

public class RectanglePlus

implements Relatable {

public int width = 0;

public int height = 0;

public Point origin;

// four constructors

public RectanglePlus() {

origin = new Point(0, 0);

}

public RectanglePlus(Point p) {

origin = p;

}

public RectanglePlus(int w, int h) {

origin = new Point(0, 0);

width = w;

height = h;

}

public RectanglePlus(Point p, int w, int h) {

origin = p;

width = w;

height = h;

}

// a method for moving the rectangle

public void move(int x, int y) {

origin.x = x;

origin.y = y;

}

// a method for computing

// the area of the rectangle

public int getArea() {

return width \* height;

}

// a method required to implement

// the Relatable interface

public int isLargerThan(Relatable other) {

**RectanglePlus otherRect**

**= (RectanglePlus)other;**

if (this.getArea() < otherRect.getArea())

return -1;

else if (this.getArea() > otherRect.getArea())

return 1;

else

return 0;

}

}

**40. How to do inheritance in java (using extend keyword)**

public class AccessDemo1

{

// Public Access Level

public void testn1()

{

System.out.println("Invoking the public Testn1 Method");

System.out.println();

}

// Private Access Level

private void testn2()

{

System.out.println("Invoking the Private Testn2 Method");

System.out.println();

}

// Protected Access Level

protected void testn3()

{

System.out.println("Invoking the Protected Testn3 Method");

System.out.println();

}

// Default Access Level

void testn4()

{

System.out.println("Invoking the Default Testn4 Method");

System.out.println();

}

}

2. This Program in the Different Package

package com.JavaTraining2;

// importing the Class of the Other Package into the Current Package

import com.javaTraining.AccessDemo1;

// Achieving Inheritance for the Class present in other Package

public class AccessDemo3 extends AccessDemo1

{

public static void main(String[] args)

{

System.out.println("Program Starts");

System.out.println();

AccessDemo3 Ad1 = new AccessDemo3();

// Accessing the Method of Public access level present in other package

Ad1.testn1();

// Accessing the Method of Protected access level present in other package

Ad1.testn3();

System.out.println();

System.out.println("Program Ends");

}

}

**41. write code to add items to integer, string array**

**public** **class** AddIntIndex {

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

**int**[] series = **new** **int**[0];

**int** x = 5;

series = *addInt*(series, x);

//print out the array with commas as delimiters

System.***out***.print("New series: ");

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

**if** (i == series.length - 1){

System.***out***.println(series[i]);

}

**else**{

System.***out***.print(series[i] + ", ");

}

}

}

**public** **static** **int**[] addInt(**int** [] series, **int** newInt){

//create a new array with extra index

**int**[] newSeries = **new** **int**[series.length + 1];

//copy the integers from series to newSeries

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

newSeries[i] = series[i];

}

//add the new integer to the last index

newSeries[newSeries.length - 1] = newInt;

**return** newSeries;

}

}

Output: 5

**package** my\_training;

**public** **class** StringArray {

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

/\*

\* Java String array can be created in below given ways.

\*/

/\*

\* Declare and initialize String array in single statement as given below.

\* This method is particularly useful when we are dealing with very small size array.

\*/

String[] myFirstStringArray = **new** String[]{"String 1", "String 2", "String 3"};

/\*

\* Declaration and assignment can be done separately as given below.

\*/

//first declare String array

String[] mySecondStringArray = **new** String[3];

//Observe that giving size is mandatory here. While there was no size given in the first method.

//Now Assign individual String array elements

mySecondStringArray[0] = "String 1";

mySecondStringArray[1] = "String 2";

mySecondStringArray[2] = "String 3";

//Note that, like every other arrays, String array starts with index 0 and not index 1.

/\*

\* Retrieve values from String Array: \*

\* String array elements can be retrieved by directly accessing using index. You may also iterate

\* String array using loop.

\*/

//this will retrieve second element of first String array

System.***out***.println(myFirstStringArray[1]);

//iterate the String array using loop

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

System.***out***.println(mySecondStringArray[i]);

}

}

}

Output: String 2

String 1

String 2

String 3

**42. write code to add items to ArrayList collection**

**package** my\_training;

**import** java.util.ArrayList;

**class** A

{

}

**public** **class** ArrayListAdd

{

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

{

System.***out***.println(" The Program Starts" );

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

// Creating an Instance of Array List

ArrayList List1 = **new** ArrayList();

// Creating an Instance of Class A

A a1= **new** A();

// Using Add() Method to add the Elements to List

List1.add(10);

List1.add(500.5);

List1.add(**true**);

List1.add(**false**);

List1.add(**null**);

List1.add(10);

List1.add('A');

List1.add(a1);

List1.add(**new** A());

List1.add("Sunil");

List1.add("Aradhya");

System.***out***.println("The Size of the Given List List1 is "+List1.size());

System.***out***.println("Printing The reference variable of List");

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

System.***out***.println(List1);

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

System.***out***.println("Printing The Elements based on Index using Standard For Loop");

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

**for**(**int** i=0; i<List1.size();i++)

{

System.***out***.println(List1.get(i));

}

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

System.***out***.println(" The Program Ends" );

}

}

Output:

The Program Starts

The Size of the Given List List1 is 11

Printing The reference variable of List

-----------------------------------------------------

[10, 500.5, true, false, null, 10, A, my\_training.A@15db9742, my\_training.A@6d06d69c, Sunil, Aradhya]

Printing The Elements based on Index using Standard For Loop

-----------------------------------------------------

10

500.5

true

false

null

10

A

my\_training.A@15db9742

my\_training.A@6d06d69c

Sunil

Aradhya

The Program Ends

**43. write code to retrieve items from arraylist (using for each loop**

**package** my\_training;

**import** java.util.ArrayList;

**class** C

{

}

**public** **class** ArrayListRetrieve {

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

{

System.***out***.println(" The Program Starts" );

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

// Creating 3 Instances of Array List

ArrayList List1 = **new** ArrayList();

ArrayList List2 = **new** ArrayList();

ArrayList List3 = **new** ArrayList();

// Adding Elements for First Array List

List2.add("Sunil");

List2.add("10");

List2.add("45");

// Adding Elements for First Array List

List1.add("Hello");

List1.add("30");

List1.add("100");

// Adding Elements for First Array List

List3.add("Hi");

List3.add("75");

List3.add("200");

System.***out***.println(List1);

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

System.***out***.println(List2);

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

System.***out***.println(List3);

List1.addAll(List2);

List2.addAll(List3);

List3.addAll(List1);

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

System.***out***.println(List1);

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

System.***out***.println(List2);

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

System.***out***.println(List3);

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

System.***out***.println(" The Program Ends" );

}

}

Output:

The Program Starts

[Hello, 30, 100]

[Sunil, 10, 45]

[Hi, 75, 200]

[Hello, 30, 100, Sunil, 10, 45]

[Sunil, 10, 45, Hi, 75, 200]

[Hi, 75, 200, Hello, 30, 100, Sunil, 10, 45]

The Program Ends

**44. write code to add items HashMap**

**package** my\_training;

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

**public** **class** HashMapDemo {

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

// create hash map

HashMap newmap = **new** HashMap();

// populate hash map

newmap.put(1, "tutorials");

newmap.put(2, "point");

newmap.put(3, "is best");

System.***out***.println("Map value before change: "+ newmap);

// put new values at key 3

String prevvalue=(String)newmap.put(3,"is great");

// check returned previous value

System.***out***.println("Returned previous value: "+ prevvalue);

System.***out***.println("Map value after change: "+ newmap);

}

}

Output:

Map value before change: {1=tutorials, 2=point, 3=is best}

Returned previous value: is best

Map value after change: {1=tutorials, 2=point, 3=is great}

**45. Write code to retrieve items to hashset**

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

**public** **class** \_HashSet {

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

{

HashSet<String> set = **new** HashSet<String>();

//Adding values to the HashSet

set.add("test1");

set.add("test2");

set.add("test3");

System.***out***.println("Retrieving values from HashSet using Iterator");

*retrieveValuesFromListMethod1*(set);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

System.***out***.println("Retrieving values from HashSet using Enumeration");

*retrieveValuesFromListMethod2*(set);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

}

/\*This method retrieves values from HashSet using Iterator

\*/

**public** **static** **void** retrieveValuesFromListMethod1(Set set)

{

Iterator itr = set.iterator();

**while**(itr.hasNext())

{

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

}

}

/\*This method retrieves values from HashSet using Enumeration

\*/

**public** **static** **void** retrieveValuesFromListMethod2(Set set)

{

Enumeration e = Collections.*enumeration*(set);

**while**(e.hasMoreElements())

{

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

}

}

}

**Output:**

Retrieving values from HashSet using Iterator

test2

test3

test1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Retrieving values from HashSet using Enumeration

test2

test3

test1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**53. what is super and this keywords**

The Class from where the members are inherited is known as Super Class and the class to which the

members are iinherited is called as Sub Class. Super Class is also called as Base Class and Sub Class is also called as Derived Class. Super statement is used to invoke the specific constructor of the Super class in subclass.

This keyword is used to point the instance of the current class.

**54. can we call parent method from child method?**

Yes, using the inheritance concept. We can also use super keyword.

**55. can we create object for abstract class?**

If **we will create** an **object** of the **abstract class** and calls the method having no body(as the method is pure virtual) it **will** give an error. That is why **we** cant **create object** of **abstract class**. In short, it is legal to have a public constructor on an **abstract class**.

**56. can we over ride static methods, final methods?**

The answer is ‘Yes’. We can have two ore more static methods with same name, but differences in input parameters. For example, consider the following Java program.

|  |
| --- |
| // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public static void foo(int a) {          System.out.println("Test.foo(int) called ");      }      public static void main(String args[])      {          Test.foo();          Test.foo(10);      }  } |

Run on IDE

Output:

Test.foo() called

Test.foo(int) called