1. **Steps to install java?**
2. Choose the Operating System. (Windows, Linux, Mac, Solaris)
3. Open the java website and download the installer JDK.
4. Run the installer by clicking “Run” and start installation process by double clicking the saved file.
5. Click on “Install” button to accept license terms.
6. Click “Next” and click “Close” on last dialogue box. This will install java. Set environmental variables.
7. Verify whether java is installed and working properly on your computer by opening the command prompt and typing “java -version” in it.
8. If it shows the version number then java is installed successfully.
9. **Steps to install eclipse?**
10. Choose the Operating System and Download the latest Eclipse Installer from Eclipse website.
11. Start the eclipse installer executable which is available in download directory. Run the installer by clicking “Run” and start installation process.
12. Select the package you want to install (Eclipse IDE for Java Developers).
13. Select your installation folder and click on “Install” button.
14. Launch eclipse after completion of installation.
15. **Steps to create workspace?**
16. Double click on the eclipse icon which you have installed.
17. It will open a dialogue box named “Workspace Launcher”.
18. Eclipse stores your projects in a folder called Workspace.
19. Choose a Workspace folder to use by clicking on “Browse” button or by clicking on “Ok” to use the default folder which is shown.
20. This creates Workspace and you are ready to create project.
21. **Steps to create project?**
22. Click on File---New---Java Project.
23. It will open a dialogue box named “New Java Project”.
24. Create a java project by entering Project Name in the box and leave other fields as default.
25. Click on “Finish” button.
26. You can see the created project in “Package Explorer”.

1. **Steps to create class/.java file?**
2. Right Click on the project which you have created.
3. Click on New---Class.
4. It will open a dialogue box named “New Java Class”.
5. Create a java class by entering class Name in the box (first letter of the word should be capital) and leave other fields as default by ensuring that source folder and package are correct.
6. If you want public static void main(String args[]) method in your class click the checkbox.
7. Click on “Finish” button.
8. This will create a class/.java file which appears in editor area.
9. **How to create packages and what is best way to give name?**
10. Click on File---New---Package.
11. It will open a dialogue box named “New Java Package”.
12. Create a java package by entering package Name in the box and leave other fields as default by ensuring whether the source folder is correct.
13. Click on “Finish” button.
14. This will create a package which appears in package Explorer.

The best way to name package is to start naming package with domain names like com, edu, gov, org, net etc. Example:- com.example.myPackage

1. **what main method will do?**

Main is a function/method name. It calls all other functions to run a program. A standalone application cannot run without main method. Start of an application takes place at main method.

1. **How to create** **property/data members?**

We define data members and properties of a class in the main block of the class. Class data members and properties have access mode (public, private, protected, default) which determines where and how we can access property/data members. We can define data members as data elements like variables, objects etc. and property as data type.

1. **What is data type and what are different data types?**

Data type specifies the size and type of values that can be stored in an identifier. In java, data types are classified into 2 types.

1. Primitive data types (include 8 data types shown in below table)
2. Non-primitive data types (includes classes, interfaces, arrays)

|  |  |
| --- | --- |
| **Type** | **Size** |
| Byte | 1 byte |
| Short | 2 bytes |
| Char | 2 bytes |
| Int | 4 bytes |
| Float | 4 bytes |
| Long | 8 bytes |
| double | 8 bytes |
| boolean | Not defined |

1. **what is a variable?**

Variable is a name of memory location which can store the values of specified data type. There are 3 types of variables in java. They are: Local, Instance and Static.

int a=10, b=20;

Here a and b are called variables.

10 and 20 are values.

int is a data type.

Local- A variable which is declared inside a method is called local variable.

Instance- A variable which is declared inside the class but outside method is called instance variable.

Static- A variable which is declared with static keyword is called static variable.

***Example:***

class name{

int a=10; *//instance variable*

static int b=20; *//static variable*

void method(){

int c=30; *//local variable*

}

}

1. **Creating method with void?**

Syntax: Access specifier | return type | method name | Arguments

Public static void sum(int a, int b){

//body;

}

1. **Creating variable, we can create variables inside method?**

Syntax: Access specifier | return type | method name | Arguments

Public static void sum(int a, int b){

int c = a+b;

System.out.println(“sum is: ”+c);

}

1. **Creating method with return data type, int/string/double/float/date etc?**

Syntax: Access specifier | return type | method name | Arguments

public int sum(int a, int b) {

return (a+b);

}

1. **Method that will return hard coded value?**

This happens when data is directly embedded into the source code of a program instead of obtaining the data from the external sources.

1. **Create default/parametrized constructors (overloaded constructor)?**

Constructor overloading is a concept of having more than one constructor with different parameter list.

**package** overloading;

**public** **class** ConstructorOverloading {

String name="Brunda";

**int** rank;

String university;

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

ConstructorOverloading object=**new** ConstructorOverloading();

System.***out***.println("These are parameterized constructors:");

System.***out***.println("1. Given String is "+object.name);

**new** ConstructorOverloading(19964,3475879);

**new** ConstructorOverloading(20,10.8764235);

**new** ConstructorOverloading(40.268832,94.37186531);

**new** ConstructorOverloading(85.0674379942,5);

**new** ConstructorOverloading(1,"Rivier");

}

ConstructorOverloading(){

System.***out***.println("This is no argument constuctor");

}

ConstructorOverloading(String name){

}

ConstructorOverloading(**int** a,**int** b){

System.***out***.println("2. Addition is "+(a+b));

}

ConstructorOverloading(**int** a,**double** b){

System.***out***.println("3. Subtraction is "+(a-b));

}

ConstructorOverloading(**double** a,**double** b){

System.***out***.println("4. Multiplication is "+(a\*b));

}

ConstructorOverloading(**double** a,**int** b){

System.***out***.println("5. Division is "+(a/b));

}

ConstructorOverloading(**int** rank,String university){

**this**.rank=rank;

**this**.university=university;

//Here usage of this keyword is optional. Anyways it will result same output.

System.***out***.println("6. Rank: "+rank+", university: "+university);

}

}//end of class

1. **Method that will return property value?**

Property value means data type like int, float, long, double. Method can return any type of property value.

1. **Creating method with return data type and parameter?**

Syntax: Access specifier | return type | method name | Arguments

Example-

public int sum(int a, int b) {

return (a+b);

}

1. **Creating static property?**

Static property is declared outside the main method and can be called directly in the program with reference class name. These are global.

1. **Creating static method?**

A method with the “static” keyword is known as static method.

1. Static method belongs to the class (rather than object of a class).
2. A static method can be invoked without the need for creating an instance of a class.
3. Static method can access static data member and can change the value of it.

**package** method;

**public** **class** Sum {

**int** x=10,y=20,c=30;

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

System.***out***.println("Types of Methods-");

*HelloWorld*();

*everyone*("Brunda Chennapragada");

**int** x=*brunda*();

System.***out***.print("3. Output of the method with return type and without arguments is: ");

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

**int** result=*exited*(5,5);

System.***out***.print("4. Output of the method with return type and with arguments is: ");

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

Sum obj=**new** Sum();

**int** z=obj.add(obj.x,obj.y,obj.c);

System.***out***.print("5. Output of the method without using static with using object is: ");;

System.***out***.println("The sum of instance variables x, y and c is "+z);

} //end of main method

**public** **static** **void** HelloWorld(){

System.***out***.print("1. Output of the method without return type and without arguments is: ");

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

}

**public** **static** **void** everyone(String name){

System.***out***.print("2. Output of the method without return type and with arguments is: ");

System.***out***.println("This is " +name);

}

**public** **static** **int** brunda(){

**return** 5;

}

**public** **static** **int** exited(**int** a,**int** b){

**return** (a+b);

}

**public** **int** add(**int** x, **int** y, **int** c){

**return**(x+y+c);

} //end of method

} //end of class

1. **Creating static block?**

Static block is used to initialize the static data member. At the time of loading, static block is executed before executing main method.

Example-

Class block{

Static{System.out.println(“static block is used”);}

  public static void main(String args[]){

   System.out.println("Hello world");

  }

}

1. **Creating object?**

Object is an instance of class which have states and behaviors. Using “new” keyword we can create an object. Whenever we want to use class methods/properties we create object.

Creating an object involves 3 steps: Declaration, Instantiation, Initialization. In the first step, variable is declared. Second step, new keyword is used. Third step, new keyword follow call to a constructor which initializes the new object.

**public** **class** objectCreation {

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

Arithmetic obj=**new** Arithmetic();

obj.operation(19964,3475879, 233866);

obj.operation("Everyone");

} //end of main method

}//end of class objectCreation

**class** Arithmetic{

**public** **void** operation(**int** a,**int** b,**int** c){

System.***out***.println("1. Addition is "+(a+b+c));

}

**public** **void** operation(String S){

System.***out***.println("Hello "+S);

}

}//end of class Arithmetic

1. **Calling method with void?**

**package** method;

**public** **class** Sum {

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

*HelloWorld*();

} //end of main method

**public** **static** **void** HelloWorld(){

System.***out***.print("1. Output of the method without return type and without arguments is: ");

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

}

1. **Calling method with no return and with parameter?**

**package** method;

**public** **class** Sum {

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

*everyone*("Brunda Chennapragada");

} //end of main method

**public** **static** **void** everyone(String name){

System.***out***.print("2. Output of the method without return type and with arguments is: ");

System.***out***.println("This is " +name);

} //end of method

} //end of class

1. **Calling method with return and no parameter?**

**package** method;

**public** **class** Sum {

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

**int** x=*brunda*();

System.***out***.print("3. Output of the method with return type and without arguments is: ");

} //end of main method

**public** **static** **int** brunda(){

**return** 5;

} //end of method

} //end of class

1. **calling method with return and parameter?**

**package** method;

**public** **class** Sum {

**int** x=10,y=20,c=30;

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

Sum obj=**new** Sum();

**int** z=obj.add(obj.x,obj.y,obj.c);

System.***out***.print("5. Output of the method without using static with using object is: ");

System.***out***.println("The sum of instance variables x, y and c is "+z);

} //end of main method

**public** **int** add(**int** x, **int** y, **int** c){

**return**(x+y+c);

} //end of method

} //end of class

1. **Calling method with return and storing the return data?**

Pending

1. **Calling static method?**

Static method is called using class name.

**package** methods;

**public** **class** Calling{

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

Arithmetic obj=**new** Arithmetic();

obj.operation(19964,3475879, 233866);

obj.operation(20,10.8764235);

} //end of main method

}//end of class Calling

**class** Arithmetic{

**public static void** operation(**int** a,**int** b,**int** c){

System.***out***.println("1. Addition is "+(a+b+c));

}

**public** **void** operation(**int** a,**double** b){

System.***out***.println("2. Subtraction is "+(a-b));

}

}//end of class Arithmetic

1. **Using static property: it will maintain?**

Pending

1. **Create classes under multiple packages?**

*First Package*

**package** creatingClass;

**public** **class** newpackage {

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

newpackage obj = **new** newpackage();

System.***out***.println(obj.add(10, 20));

}

**public** **int** add(**int** a,**int** b){

**return** (a+b);}

}

*Second Package*

**package** testing;

**import** creatingClass.newpackage;

**public** **class** myPackages {

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

newpackage obj = **new** newpackage();

System.***out***.println(obj.add(10, 20));

}

}

1. **Calling classes under different packages?**

*First Package*

**package** creatingClass;

**public** **class** newpackage {

**public** **int** add(**int** a,**int** b){

**return** (a+b);}

}

*Second Package*

**package** testing;

**import** creatingClass.newpackage;

**public** **class** myPackages {

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

newpackage obj = **new** newpackage();

System.***out***.println(obj.add(10, 20));

}

}

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

**package** finallyException;

**public** **class** HandlingFinallyException {

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

*handlingNullPointer*();

//implemented nested try loop

**try**{

**int** x=477;

System.***out***.println("x value: "+x);

**try**{

**int** a[]=**new** **int**[5];

a[1]=5;

a[10]=20;

a[4]=2;

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

System.***out***.println("array elements: "+a[i]);}

}

**catch**(ArrayIndexOutOfBoundsException e){

System.***out***.println("Exception occured");

e.printStackTrace();

}}

**catch**(Exception y){

System.***out***.println("This will not be executed as there is no exception");

y.printStackTrace();

}

**finally**{

String s="This will execute even if above code fails";

System.***out***.println("Execute Finally Block: "+s);

}

}

**public** **static** **void** handlingNullPointer(){

**try**{

**int** x[] = **null**;

**int** l= x[2];

System.***out***.println("length is: "+l);

}

**catch**(NullPointerException y){

System.***out***.println("Handling null pointer exception in different method");

y.printStackTrace();

}

}

}

1. **What is checked exception/unchecked exception?**

The main difference between checked and unchecked exception is that the checked exceptions are checked at compile-time while unchecked exceptions are checked at runtime.

*Checked Exceptions –*

SQLException

IOException

DataAccessException

ClassNotFoundException

InvocationTargetException

*Unchecked exceptions –*

NullPointerException

ArrayIndexOutOfBoundsException

ArithmeticException

IllegalArgumentException

1. **What is final keyword, create final class, final method, final property?**

Final is immutable. This keyword is used when we don’t want to perform overridden operation. Word itself stands for it is fixed.

If a class is declared with final keyword then it is called final class.

A final method is created when we don’t want to override or hide by subclasses.

A final property means making any variable as final.

*Final Program:*

public final class ImmutableStudent { //final class

    private final int id;

    private final String name;

private final String address=”USA”; //final variable

  public ImmutableStudent(int id, String name) {

        this.name = name;

        this.id = id;

    }

    public int getId() {

        return id;

    }

    public String getName() {

        return name;

    }

Public static final void(int age){ //final method

this.age = age;

}

public int getAge(){

return age;

}

}

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

**package** interfaceProgram;

**interface** Interface {

//in interface only declaration

//no definition

**public** **void** display();

**public** **void** frontcam();

**interface** Interface1 {

**public** **void** camera();

**public** **abstract** **void** backcam();

**class** InterfaceClass **implements** Interface1,Interface{

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

InterfaceClass obj=**new** InterfaceClass();

obj.backcam();

obj.camera();

obj.display();

obj.frontcam();

**int** a=10,b=20;

System.***out***.println("sum is: "+(a+b));

}

@Override

**public** **void** display() {

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

}

@Override

**public** **void** frontcam() {

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

}

@Override

**public** **void** camera() {

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

}

@Override

**public** **void** backcam() {

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

}

}

}

}

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

**package** abstraction;

**public** **abstract** **class** Abstract {

**public** **void** display(){

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

}

**public** **void** frontcam(){

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

}

**public** **abstract** **void** backcam();

**public** **abstract** **class** abstractClass{

**public** **void** cam()

{

System.***out***.println("New method");

}

**public** **abstract** **void** backcam2();

}

**public** **class** DummyClass **extends** Abstract{

@Override

**public** **void** backcam() {

System.***out***.println("Defined BackCam");

//In java multiple inheritance is not allowed.

//so to avoid this problem interface comes into existence.

}

}

}

**36. Implement method overloading?**

**package** overloading;

**public** **class** MethodOverloading {

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

Arithmetic obj=**new** Arithmetic();

obj.operation(19964,3475879, 233866);

obj.operation(20,10.8764235);

obj.operation(40.268832,94.37186531);

obj.operation(85.0674379942,5);

obj.operation(82,5);

obj.operation("Everyone");

} //end of main method

}//end of class MethodOverloading

**class** Arithmetic{

**public** **void** operation(**int** a,**int** b,**int** c){

System.***out***.println("1. Addition is "+(a+b+c));

}

**public** **void** operation(**int** a,**double** b){

System.***out***.println("2. Subtraction is "+(a-b));

}

**public** **void** operation(**double** a,**double** b){

System.***out***.println("3. Multiplication is "+(a\*b));

}

**public** **void** operation(**double** a,**int** b){

System.***out***.println("4. Division is "+(a/b));

}

**public** **void** operation(**int** a,**int** b){

System.***out***.println("5. Modulas is "+(a%b));

}

**public** **void** operation(String S){

System.***out***.println("Hello "+S);

}

}//end of class Arithmetic

**37. Implement method overriding?**

If a child class and parent class has same method declaration then it is called method overriding.

**package** overriding;

**public** **class** MethodOverriding {

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

object obj = **new** object();

obj.name();

}

**void** name(){

System.***out***.println("Parent class method");}

}

**class** object **extends** MethodOverriding{

**void** name(){

System.***out***.println("Parent class method is overridden and");

System.***out***.println("Child class method is displayed");

}

}

**38. Implement polymorphism?**

Polymorphism in java is a concept by which we can perform a single action by different ways. There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism.

**public** **class** Runtime {

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

Runtime b = **new** Splender();

b.method1();

}

**void** method1(){

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

}

}

**class** Splender **extends** Runtime{

**void** method1(){

System.***out***.println("polymorphism concept");

}

}

1. **Implementing Interface?**

**package** interfaceProgram;

**interface** Interface {

//in interface only declaration

//no definition

**public** **void** display();

**public** **void** frontcam();

**interface** Interface1 {

**public** **void** camera();

**public** **abstract** **void** backcam();

**class** InterfaceClass **implements** Interface1,Interface{

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

InterfaceClass obj=**new** InterfaceClass();

obj.backcam();

obj.camera();

obj.display();

obj.frontcam();

**int** a=10,b=20;

System.***out***.println("sum is: "+(a+b));

}

@Override

**public** **void** display() {

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

}

@Override

**public** **void** frontcam() {

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

}

@Override

**public** **void** camera() {

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

}

@Override

**public** **void** backcam() {

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

}

}

}

}

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

**package** inherit;

**public** **class** InheritanceProgram {

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

brunda obj=**new** brunda();

obj.intro("Chennapragada");

obj.sis(20,30);

obj.mom(obj.x,obj.y);

}

}

**class** brunda **extends** karuna{

**int** x=10,y=100;

**public** **void** intro(String name){

System.***out***.println("Hello, this is Brunda "+name);

}

}

**class** karuna **extends** sandhya{

**public** **void** sis(**int** a,**int** b){

System.***out***.println("Hello, this is Karuna "+(a+b));

}

}

**class** sandhya{

**public** **void** mom(**int** x,**int** y){

System.***out***.println("Hello, this is Sandhya "+(y-x));

}

}

1. **Write code to add items to integer, string array?**

**package** array;

**public** **class** ArrayProgram {

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

**int** a[] = **new** **int**[5];

a[0]=10;

a[1]=20;

a[2]=30;

a[3]=40;

a[4]=50;

System.***out***.println("values of a: "+a[0]+ ", "+a[1]+", "+a[2]+", "+a[3]+", "+a[4]);

String s[] = {"hello","world"};

System.***out***.println("values of s: "+s[0]+ ", "+s[1]);

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

b[0]="Hello";

b[1]="World";

b[2]="java";

System.***out***.println("values of b: ");

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

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

}

}

}

1. **Write code to retrieve items to integer, string array?**

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

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

}

1. **write code to add items to ArrayList collection?**

**package** arrayList;

**import** java.util.ArrayList;

**public** **class** ArrayListProgram {

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

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

ArrayList<String>a = **new** ArrayList<String>();

a.add("veggies");

a.add("potato");

a.add("spinach");

a.add("carrot");

a.add("lettuce");

a.add("onion");

a.add("cabbage");

a.add("cauliflower");

System.***out***.println("Read value of index 6- "+a.get(6));

System.***out***.println("Read value of index 1- "+a.get(1));

System.***out***.println("Read value of index 3- "+a.get(3));

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

System.***out***.println("Array list elements are: "+a.get(i));

}

}

}

1. **write code to retrieve items from arraylist (using for each loop)?**

**string** str = **string**.Empty;

**foreach** (**string** strName **in** arrayList)

{

str += strName + "\n";

}

MessageBox.Show(str);

1. **write code to add items HashMap?**

package hashMap;

import java.util.HashMap;

import java.util.Map;

public class HashMapList {

public static void main(String[] args) {

// HashMap will not allow duplicate keys but

//will allow duplicate values

HashMap<Integer,String>a = new HashMap<Integer,String>();

a.put(1, "Brunda");

a.put(6, "orange");

a.put(9, "apple");

a.put(5, "banana");

a.put(8, "pineapple");

a.put(2, "green grape");

a.put(4, "citrus");

a.put(3, "green grape");

a.put(7, "fruits");

}

}

//HashSet will not allow duplicate elements

1. **write code to retrieve items HashMap?**

package hashMap;

import java.util.HashMap;

import java.util.Map;

public class HashMapList {

public static void main(String[] args) {

// HashMap will not allow duplicate keys but

//will allow duplicate values

HashMap<Integer,String>a = new HashMap<Integer,String>();

a.put(1, "Brunda");

a.put(6, "orange");

a.put(9, "apple");

a.put(5, "banana");

a.put(8, "pineapple");

a.put(2, "green grape");

a.put(4, "citrus");

a.put(3, "green grape");

a.put(7, "fruits");

for(Map.Entry m:a.entrySet()){

System.out.println(m.getKey()+" "+m.getValue());

}

System.out.println("Value of key 7 is: "+a.get(7));

System.out.println("Get default value of key 5: "+a.getOrDefault(5, "citrus"));

System.out.println("Does the HashMap list contains key 10? "+a.containsKey(10));

System.out.println("Does the HashMap list contains value apple? "+a.containsValue("apple"));

a.remove(7, "fruits");

a.remove(8);

a.replace(3, "green grape", "grape");

a.replace(5, "tomato");

System.out.println("Values of modified hashMap list: "+a.values());

System.out.println("The new HashMap list is: "+a.clone());

**}**

**}**

1. **write code to add items to HashSet?**

**package** hashSet;

**import** java.util.HashSet;

**public** **class** HashSetList {

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

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

// HashSet declaration

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

// Adding elements to the HashSet

a.add("veggies");

a.add("potato");

a.add("spinach");

a.add("carrot");

a.add("lettuce");

a.add("onion");

a.add("cabbage");

a.add("cauliflower");

//Addition of duplicate elements

a.add("cabbage");

a.add("cauliflower");

//Addition of null values

a.add(**null**);

a.add(**null**);

//hashSet will not allow duplicate elements and will only one null.

//Displaying HashSet elements

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

}

}

1. **write code to retrieve items from HashSet?**

**package** hashSet;

**import** java.util.HashSet;

**import** java.util.Iterator;

**public** **class** RetrieveHashsetElements {

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

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

set.add("Hello");

set.add("this");

set.add("is");

set.add("sai");

//Traversing elements

Iterator<String> itr=set.iterator();

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

{

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

}

}

}

1. **write code to connect to JDBC to get rows from employee table?**

**Pending**

1. **Write method to return list of rows code to loop throughs?**

Answer is in the above code.

1. **Create Employee class?**

public class Employee

{

    // create data fields

    private String firstName;

    private String lastName;

    private int phoneNumber;

    private String address;

    private int id;

    private double Salary;

}

1. **Add Employee class to list collection?**

import java.util.ArrayList;

public class NewClass {

public static void main(String[] a) {

ArrayList<Employee> a = new ArrayList<Employee>();

a.add(new Employee("giu", "gyujh"));

a.add(new Employee("gyuj", "fth"));

a.add(new Employee("fuyjh", "gvjh"));

System.out.println(a);

Employee e =a.get(1);

e.setLastName("new");

System.out.println(a);

}

}

1. **Create method that return list of employee collection?**

package sqlProgram;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Iterator;

import java.util.List;

public class helloEmploye {

static void main(String[] args) {

List<Employee> list=new ArrayList<Employee>();

list.add(new Employee(1,"one",234));

list.add(new Employee(2,"two",2366));

list.add(new Employee(3,"three",987));

list.add(new Employee(4,"four",8744));

Collections.sort(list,new Employee\_comparator());

for(int i=0;i<list.size();i++){

System.out.print("age:"+list.get(i).getAge());

System.out.print("empid:"+list.get(i).getEmpid());

System.out.println("name:"+list.get(i).getname());

}

Iterator<Employee> itr=list.iterator();

while(itr.hasNext())

{

Employee employee = itr.next();

System.out.print("age:"+employee.getAge());

System.out.print("empid:"+employee.getEmpid());

System.out.println("name:"+employee.getname());

}

}}

1. **Difference between string, string buffer, string builder with example?**

String is immutable, if you try to alter their values, another object gets created, whereas StringBuffer and StringBuilder are mutable so they can change their values. Thread-Safety Difference: The difference between StringBuffer andStringBuilder is that StringBuffer is thread-safe.

1. **write a code to save data into excel file and read from excel file (POI and jexcel API)?**

Pending

1. **What is super and this keywords?**

Super keyword is used to refer immediate parent class whereas, this keyword is used to refer current class.

1. **Can we call parent method from child method?**

By using *“Super keyword”* inside the body of child method, we can call parent method.

1. **Can we create object for abstract class?**

An abstract class is an incomplete class like Interfaces. They have abstract functions which are not completely defined. So, an object of abstract class could not be created.

1. **Can we over ride static methods, final methods?**

No. We cannot override static methods and final methods. The word itself says that it is final, so we cannot override final method. In case of static method, they come under class which are global and can be accessed by objects which cannot be overridden.

1. **How to read data from properties file?**

**package** test1;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.util.Properties;

**class** hello{

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

**try** (FileReader reader = **new** FileReader("file.properties")) {

Properties properties = **new** Properties();

properties.load(reader);

String url = properties.getProperty("file.url");

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

} **catch** (IOException e) {

e.printStackTrace();

}

}}

1. **How to manipulate json file (reading key and adding key)?**

We will write data to json

pending

1. **Create sample program to create thread?**

Thread can be created by extending thread class or by implementing runnable interface.

1. *By extending thread:*

Thread **class** Multi **extends** Thread{

**public** **void** run(){

System.out.println("By extending thread");

}

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

Multi t1=**new** Multi();

t1.start();

 }

}

1. *By implementing runnable interface:*

**class** Multi3 **implements** Runnable{

**public** **void** run(){

System.out.println("thread is running...");

}

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

Multi3 m1=**new** Multi3();

Thread t1 =**new** Thread(m1);

t1.start();

 }

}

1. **How to read text, attributes from XML files?**

Pending

1. **List of interfaces/classes in collection package?**

Collections is a framework that provides an architecture to store and manipulate group of objects. Collection framework provides Interfaces and classes.

1. *Interfaces:*

Set, List, Queue, Deque

1. *Classes:*

LinkedList, ArrayList, HashSet, LinkedHashSet, TreeSet, Vector, priorityQueue.

1. **Difference between throw, throws and throwable?**

Throw: It is a keyword used to throw an exception manually from any method or block.

Throws: Throws keyword is used in a method to indicate that this method may throw mentioned exceptions. Those mentioned exceptions can be handled by either using try-catch block or throws keyword.

Throwable: It is a super class for all types of errors and exceptions in java.