**JAVA**

**A1) Steps to install java**

**Answer:**

1. **Download JDK (Java Development Kit)**
   1. Download the JDK (Java Development Kit) from official website <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
   2. Click the **Download** button. It will open new page, which having several download options
   3. Click **Accept License Agreement** for accepting the license agreement.
   4. Click the download link, which is relates to your operating system. Save this software in specific location in your system.
2. **Install the JDK in our System**
   1. Double click on the **JDK** executable file to Install the **JDK** on your computer.
   2. It will show the prompt window to allow the software installed changes in your system. Click on yes
   3. Click on Next to process the installation. It will show the features and Installation location of the JDK software.
   4. Click on next. This will be install the JDK in our system. Wait for completing the installation.
   5. Next it will ask about the JRE (Java Runtime Environment) to install on our system. Click on Next. JRE will be installed on our system. Wait for completing the JRE installation.
   6. Click finish button.
3. **Now we should set the path of the JDK in environment variables**.
   1. Open the java installed location in our system. It will be in

**C 🡪 Program Files 🡪 Java 🡪 jdk1.8.0\_121 🡪 bin**

* 1. Copy the path from address bar **C:\Program Files\Java\jdk1.8.0\_121\bin**
  2. Click **Start** 🡪 **Control Panel 🡪 System and Security 🡪System**
  3. Click **Advanced system settings 🡪 Environment Variables**
  4. Add the path of the bin folder of JDK for the **PATH** in **System Variable.**

These are the steps to download, install and set the path of the Java

**A2) Steps to install eclipse**

**Answer:**

1. **Download and Install eclipse**
   1. Navigate to <http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/heliossr1>
   2. We can see the Download Links right side of the webpage. Click on the download link, which relates to your system.
   3. Eclipse zip file will be downloaded
   4. Copy that zip file in any location of your system and extract it.
   5. Eclipse folder will appear after extraction. Open it and double click on the eclipse executable file to run the eclipse.

These are the steps to install eclipse in our system.

**A3) Steps to create workspace**

1. Open eclipse
2. Click on **File** menu
3. Goto **Switch** **Workspace**
4. Click on Other
5. Workspace Launcher window will appear. Click on Browse button.
6. Create a new folder for workspace in any drive
7. Click on that folder then click **ok** button
8. Then click ok. Now eclipse will restart to open new workspace

These are the steps to create Workspace in eclipse

**A4) Steps to create project**

1. Open eclipse
2. Click **File** menu
3. Goto **New**
4. Click on **Project**
5. Now we can absorb many types of projects. We have to select specific project such as **Java Project.** Click on **Java Project.**
6. Click **Next**
7. Give the project Name ex: **Gmail\_Test**
8. Click **Next**
9. Click **Finish**

This way we can create project in eclipse

**A5) Create .java file / class**

1. After Creating the project we can see our project Gmail\_Test in the Project Explorer widown, which is in the left side of Eclipse IDE.
2. Extend **Gmail\_Test** then we can see **src** folder. **Right** **click** on **src** folder.
3. Goto **New**
4. Click **Class**
5. Give name for the class in Name field. Class name must start with the Capital letter. Class name first letter capital letter and next word first letter capital letter.

Ex: **GmailLoginTest**

1. Click on **finish**

These are the steps to create class file in eclipse.

**A6) How to create package and what is best way to give name**

**Steps to create pakage in eclipse:**

1. Right click on src folder in our project
2. Goto New then click on Package
3. Give the name to package
4. Click on Finish

These are the steps to create package in eclipse

**Best way to give name:**

1. Package name must write in all lowercase. This will help to avoid conflict with names of interfaces or classes.
2. First we should write domain name, then company name and after package name.

Ex: com.google.mypakagename

Domain Name: [www.google.com](http://www.google.com)

Package Name: com.google.packagename

**A7) What is main method will do**

**Answer:**

Execution of the program will starts from main method. JVM (Java Virtual Machine) looks for the main method in the class. So we need to write main method in one class. We can run the program without main method but we need to write static block to run the program. If we wont write static block and main method in the program then it can be compile but at run time it will show error.

**A8) Creating property / data members**

**Answer:**

Members of a class are called as data members, such as variables and methods in the class. Class is a collection of data members and methods. We have two types of data members in the Java class they are

1. Instance Data Members
2. Static Data Members

**Instance Data Members:**

Instance data members nothing but which are declared in the class without static keyword. Every time new memory space is created for these data members whenever the object of the class is created.

**Static Data Members:**

If the variable is having the static keyword inside the class then these members are called as static data members. Static data member’s memory will create only once. It will create whenever class is loaded in main memory. The static variable can be used to represent the common property of all objects.

**Code:**

**package** com.rithika;

**public** **class** DataMembers {

**int** empId; //Instance Data member

String empName;//Instance Data member

**static** String *designation*="Tester"; //Static data members

**public** DataMembers(**int** id,String n) {

empId=id;

empName=n;

}

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

System.***out***.println("EmpId="+empId+" EmpName="+empName+" Designation="+*designation*);

}

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

DataMembers d1=**new** DataMembers(123, "Rithika");

DataMembers d2=**new** DataMembers(124, "Vasu");

d1.display();

d2.display();

}

}

**Output:** EmpId=123 EmpName=Rithika Designation=Tester

EmpId=124 EmpName=Vasu Designation=Tester

**A9) What is data type and different data types**

Data type represent the different values to be stored in the variable. Such as int, double, float, etc. There are two types of data types in java, they are

1. Primitive Data types
2. Non – Primitive Data types

**Primitive Data Types:**

Boolean and Numeric types of data types are called as primitive data types.

**Boolean type** is Boolean

**Numeric Types are** char, byte, short, int, long, float and double.

**Non Primitive Data Types:**

String, Array etc. are called as Non Primitive Data Types.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Size** |
| Char | ‘\u0000’ | 2 bytes |
| Boolean | False | 1 bit |
| Short | 0 | 2 bytes |
| Byte | 0 | 1 byte |
| Int | 0 | 4 bytes |
| Long | 0L | 8 bytes |
| Float | 0.0f | 4 bytes |
| Double | 0.0d | 8 bytes |

**A10) What is variable**

The name of memory location is called as variable. The variable value can be changeable. Java having three types of variables they are local, instance and static.

1. Local variable:

The variable which is declared inside the method is called as local variable.

Ex:

class Demo{

void sum(){

int a,b,c; //Local variables

}

}

1. Instance variable:

The variable which is declared inside the class, out side the method but not declared as static is called as instance variable.

Ex:

class B{

int a,b,c; //Instance variable

}

1. Static variable:

The variable which is declared as static is called as static variable.

Ex:

class C{

static int a,b,c; //Static variables

}

**A11) Creating method with void**

Method is a collection of statements, which are grouped to perform an operation.

**Syntax for method in java:**

Modifier Return\_Type name\_Of\_Method(Parameter\_List){

}

Modifier: It is optional. It defines the scope of the method.

Return\_Type: It represents the type of value which is return from the method.

name\_Of\_Method: This is the method name. The name should be start with lowercase letter and next word first letter as uppercase letter.

Parameter\_List: It is oprional method may have the zero parameters.

**Creating method with void:**

void means nothing, that means there no return value from the method.

Example:

void sum(){

}

**Example code:**

public class Sum{

void sum(){

int a=20;

int b=30;

int c=a+b;

System.out.println(“Sum=”+c);

}

public static void main(String[] args){

Sum s=new Sum(); //Object Creation

s.sum(); // Calling Method

}

}

**A12) creating variable, we can create variables inside method**

Yes, we can create the variable inside the method. This variables are called as local variable. These variables can be accessed inside the method only.

**Example:**

class Demo{

void setValues(int x,int y){

int a,b,c; //local variables

a=x;

b=y;

c=a-b;

System.out.pritnln(“c=”+c);

}

public static void main(String[] args)

{

Demo d=new Demo();

d.setValues(10,20);

}

}

**A13) creating method with return data type, we can return int/string/double/float/date etc.**

**Creating method with return data type:**

Return data type indicates the value type which is return from the method.

**Syntax:**

Modifier return\_type methodName(Parameter\_List)

**Example:**

1. **Integer**

int getValue()

{

int x;

x=20;

return x;

}

Above method will return the integer value of x.

1. **String**

String getString()

{

return “This is example for string return”;

}

Above method will return the String value.

1. **double**

String getString()

{

double x;

x=2.07d;

return x;

}

Above method will return the double value.

Same as we can return different data types values from the method.

Note: value that we specify after return keyword should be of data type that is specified in method signature.

**A14) method that will return hard coded value**

**Answer:**

Hard Coding nothing but configuring the data directly into the source code of a program.

**Code:**

**package** com.rithika;

**public** **class** HardCodedValueRetrun {

**public** String testMethod(){

String test="Test"; //Hard Coded value

**return** test; //Hard Coded value return

}

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

HardCodedValueRetrun h=**new** HardCodedValueRetrun();

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

}

}

**Output:** Test

**A15) Create default/paramterzied constructors**

Constructor is a special method which is used to initialize the object. It will invoke at the time of object creation.

**Rules to create constructor in Java:**

1. Constructor name must be same as class name.
2. It doesn’t contain any explicit return type.

There are two types of constructors in java they are

1. Default constructor
2. Parameterized Constructor

**Code for default / parameterized constructor:**

class Student{

int id;

String name;

Student(){

//Default constructor for providing the default values to the object

}  
Student(int identification, String n){ //Parameterized Constructor

id=identification;

name=n;

}

void display(){

System.out.println(“Id=”+id);

System.out.println(“Name=”+name);

}

public static void main(String[] args){

Student student1=new Student(156,”Rithika”);

Student student2=new Student(157,”Vasu”);

student1.display();

student2.display();

}

}

**A16) method that will return property value**

**Answer:**

**package** com.rithika;

**public** **class** PropertyReturn {

**int** empId;

String name;

**public** **int** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**int** empId) {

**this**.empId = empId;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

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

PropertyReturn p=**new** PropertyReturn();

p.setEmpId(100);

p.setName("Rithika");

System.***out***.println("Name property in the class is: "+p.getName());

}

}

**Output:** Name property in the class is: Rithika

**A17) creating method with return data type and parameter**

**Syntax:**

**Return\_datatype methodName(parameter\_list)**

**Example:**

**public** **class** ReturnParameterMethod {

**int** a;

**int** b;

**int** sum(**int** x,**int** y){ //With return data type and parameters

a=x;

b=y;

**int** c=x+y;

**return** c;

}

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

ReturnParameterMethod r=**new** ReturnParameterMethod();

System.***out***.println("Sum="+r.sum(20, 30));

}

}

**A18) creating static property**

**Answer:**

**package** com.rithika;

**public** **class** PropertyReturn {

**static** **int** *empId*;

**static** String *name*;

**public** **static** **int** getEmpId() {

**return** *empId*;

}

**public** **static** **void** setEmpId(**int** empId) {

PropertyReturn.*empId* = empId;

}

**public** **static** String getName() {

**return** *name*;

}

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

PropertyReturn.*name* = name;

}

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

*setEmpId*(100);

*setName*("Rithika");

System.***out***.println("staic property in the class is: "+*getName*());

}

}

**Output:** staic property in the class is: Rithika

**A19) creating static method**

**Answer:**

Static method can be invoked without the instance of a class. We can invoke static method with using class name where static method is present. Static method can access the static data members and it can change the values of it.

**Code for static method:**

**public** **class** StaticMethodDemo {

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

StaticMethodDefination.*employeeDetails*(198, "Rithika");

}

}

**class** StaticMethodDefination{

**static** **int** *id*;

**static** String *name*;

**public** **static** **void** employeeDetails(**int** i,String n){

*id*=i;

*name*=n;

System.***out***.println("id="+*id*);

System.***out***.println("Name="+*name*);

}

}

**Output:** id=198

Name=Rithika

**A20) create static block**

**Answer:**

Static block is used to change the default values of static variables. It will executed automatically when the class is loaded in the memory.

**Code for static block:**

**public** **class** StaticBlockDemo {

**static** **int** *id*;

**static** String *name*;

**static**{

*id*=178;

*name*="Rithika";

}

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

System.***out***.println("id="+*id*);

System.***out***.println("Name="+*name*);

}

}

**Output:**

id=178

Name=Rithika

**A21) creating object**

**Answer:**

Java allows to create object in four different ways they are

1. Using **new** keyword

This is the common way to create object in java.

ClassName object=new ClassName();

**Example:**

**public** **class** ObjectCreation {

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

ObjectCreation oc=**new** ObjectCreation();

}

}

1. Using Class.forName()

We should know the class name and it should having the public default constructor then we can create object for that class by using Class.forName();

**Example:**

**package** com.rithika;

**public** **class** ObjectCreation {

**public** ObjectCreation(){

}

**public** **static** **void** main(String[] args) **throws** InstantiationException, IllegalAccessException, ClassNotFoundException{

ObjectCreation oc=(ObjectCreation) Class.*forName*("com.rithika.ObjectCreation").newInstance();

}

}

1. Using clone()

It is used to create copy of the existing object.

**Example**:

**public** **class** ObjectCreation {

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

ObjectCreation obj=**new** ObjectCreation();

ObjectCreation obj1=(ObjectCreation) obj.clone();

}

}

1. Using object deserialization

Creating object from its serialized form then it is called as object deserialization.

**Example:**

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**public** **class** ObjectCreation {

**public** **static** **void** main(String[] args) **throws** CloneNotSupportedException, IOException, ClassNotFoundException{

FileInputStream file=**new** FileInputStream("D:\\temp.txt");

ObjectInputStream ois=**new** ObjectInputStream(file);

ObjectCreation obj=(ObjectCreation) ois.readObject();

ois.close();

file.close();

}

}

**A22) calling method with void**

Method cannot do anything until we call them in to action. Calling method with void means there is no return values from the method.

**Calling Method with void:**

**package** com.rithika;

**public** **class** VoidMethod {

**public** **void** getValue(){

System.***out***.println("This method which is no return value");

}

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

VoidMethod v=**new** VoidMethod();

v.getValue(); //Callingg method with void

}

}

**A23) calling method with no return and parameter**

**Answer:**

Syntax:

Method Signature:

void methodName(parameterList)

**Code:**

**package** com.rithika;

**public** **class** VoidParameterMethod {

**int** a;

**int** b;

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

a=x;

b=y;

}

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

System.***out***.println("a="+a);

System.***out***.println("b="+b);

}

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

VoidParameterMethod v=**new** VoidParameterMethod();

v.setValue(10, 20); //Calling method with no return and parameters

v.display();

}

}

**Output:** This method which is no return value

**A24) calling method with return and no parameter**

**Answer:**

**Method signature:**

Return\_type methodName()

**Code**:

**package** com.rithika;

**public** **class** ReturnNoParameter {

String getValue(){

**return** "This is with return no parameter method";

}

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

ReturnNoParameter r=**new** ReturnNoParameter();

System.***out***.println(r.getValue()); //With return no parameter

}

}

**Output:** This is with return no parameter method

**A25) calling method with return and parameter**

**Method signature:**

Return\_type methodName(parameter\_list)

**Code**:

**package** com.rithika;

**public** **class** Calculator {

**int** a,b;

**int** sum(**int** x,**int** y){

a=x;

b=y;

**int** c=a+b;

**return** c;

}

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

Calculator c=**new** Calculator();

System.***out***.println(c.sum(20, 30));

}

}

**Output:** 50

**A26) calling method with return and storing the return data**

**Answer:**

**Code:**

**package** com.rithika;

**public** **class** Calculator {

**int** a,b;

**int** sum(**int** x,**int** y){

a=x;

b=y;

**int** c=a+b;

**return** c;

}

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

Calculator c=**new** Calculator();

**int** result=c.sum(50, 80); //Storing return value in result

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

}

}

**Output:** 130

**A27) calling static method**

**Answer:**

1. Static method can be called by the other static method in the same class without using instance of a class.
2. Static method can be called by using its class name.

**Code 1:**

**package** com.rithika;

**public** **class** Calculator {

**static** **void** display(){

System.***out***.println("Static mehtod");

}

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

*display*();

}

}

**Output:** Static method

**Code 2:**

**package** com.rithika;

**public** **class** Calculator {

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

Result.*display*();

}

}

**class** Result{

**static** **void** display(){

System.***out***.println("Static mehtod");

}

}

**Output:** Static method

**A28) using static property: it will maintain**

**Ans: public static int empid;**

1. It is used to represent the common property of all objects
2. It will make our program memory efficient.

**A29) create classes under multiple packages**

**Answer:**

We can create classes under the multiple packages in java.

**Code:**

**Package: com.rithika.initialize**

**package** com.rithika.initialize;

**public** **class** Initialize {

**public** **int** empId;

**public** String empName;

**public** **void** setValues(**int** id,String name){

empId=id;

empName=name;

}

}

**Package: com.rithika.action:**

**package** com.rithika.action;

**import** com.rithika.initialize.Initialize;

**public** **class** EmployeeDetailsDisplay {

**public** **void** display(Initialize i){

System.***out***.println("EmpId="+i.empId+" EmpName="+i.empName);

}

}

**Package: com.rithika.execute**

**package** com.rithika.execute;

**import** com.rithika.action.EmployeeDetailsDisplay;

**import** com.rithika.initialize.Initialize;

**public** **class** EmployeeDetails {

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

Initialize i=**new** Initialize();

EmployeeDetailsDisplay emp=**new** EmployeeDetailsDisplay();

i.setValues(111, "Rithika");

emp.display(i);

}

}

**Output:** EmpId=111 EmpName=Rithika

**A30) calling classes under different packages**

**Answer:**

**Code:**

**Package: com.rithika.initvalues**

**package** com.rithika.initvalues;

**public** **class** Initialization {

**public** **int** empId;

**public** String empName;

**public** **int** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**int** empId) {

**this**.empId = empId;

}

**public** String getEmpName() {

**return** empName;

}

**public** **void** setEmpName(String empName) {

**this**.empName = empName;

}

}

**Package: com.rithika.display**

**package** com.rithika.display;

**import** com.rithika.initvalues.Initialization;

**public** **class** Display {

**public** **void** disp(Initialization i){

System.***out***.println("EmpId:"+i.getEmpId());

System.***out***.println("EmpName:"+i.getEmpName());

}

}

**Package: com.rithika.app**

**package** com.rithika.app;

**import** com.rithika.display.Display; //Calling classes from different packages

**import** com.rithika.initvalues.Initialization; //Calling classes from different packages

**public** **class** Execute {

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

Initialization i=**new** Initialization();

Display d=**new** Display();

i.setEmpId(100);

i.setEmpName("Rithika");

d.disp(i);

}

}

**Output:**

EmpId:100

EmpName:Rithika

**A31) write code to handle exceptions with try/catch/finally**

**Answer:**

1. Try block is used to enclose the code which may throw exception. It must followed by the catch or finally block.
2. Catch block is used to handle the exception. It must used after try block. It will handle the exception throws from try block.
3. Finally block will execute the important code such as closing connections and stream. It will always executed irrespective of exception.

**Code:**

**package** com.rithika;

**public** **class** TryCatchFinally {

**public** **static** **void** tryCatchFinallyExample(){

**int** a=20;

**try**{

a=a/0; //It will throws the ArithmeticException

}

**catch**(ArithmeticException e){

e.printStackTrace();

}

**finally**{

System.***out***.println("Finally Block");

}

}

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

*tryCatchFinallyExample*();

}

}

**Output:**

java.lang.ArithmeticException: / by zero

at com.rithika.TryCatchFinally.tryCatchFinallyExample(TryCatchFinally.java:7)

at com.rithika.TryCatchFinally.main(TryCatchFinally.java:17)

Finally Block

**A32) what is checked exception/unchecked exception**

**Answer:**

**Checked Exceptions:**

Throwable exception of a class except RuntimeException is known as checked exceptions. These are checked at compile time.

Example: SQLException, IOException etc.

**Unchecked Exceptions:**

RuntimeException are known as Unchecked Exceptions. These exception can checked at runtime.

Example: ArrayIndexOutOfBoundsException, ArithmeticException, NullPointerException etc.

**A33) what is final keyword, create final class, final method, final property**

**Answer:**

final keyword is used to represent the constant behavior to variables, methods and class. We can apply final keyword to variables, methods and class.

1. **Final Class**

Final class cannot be extended.

**Code:**

**package** com.rithika;

**final** **class** Animal{

**void** display(){

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

}

}

//Final class cannot be extends. it will show compile time exception

**public** **class** FinalMethod **extends** Animal {

**void** display(){

System.***out***.println("Method Overriding");

}

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

FinalMethod f=**new** FinalMethod();

f.display();

}

}

**Output:** Exception in thread "main" java.lang.Error: Unresolved compilation problem:

at com.rithika.FinalMethod.main(FinalMethod.java:13)

1. **Final method**

We cannot override the final method.

**Code:**

**package** com.rithika;

**class** Animal{

**final** **void** display(){

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

}

}

**public** **class** FinalMethod **extends** Animal {

//Overriding Final Method will show error at compilation

**void** display(){

System.***out***.println("Method Overriding");

}

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

FinalMethod f=**new** FinalMethod();

f.display();

}

}

**Output:** Exception in thread "main" java.lang.VerifyError: class com.rithika.FinalMethod overrides final method display.

1. **Final Variable:**

If we make variable as final then we cannot change the value of final variable.

**package** com.rithika;

**public** **class** FinalVariable {

**final** **double** pi=3.141;

**void** changePi(){

//We are changing the value of final variable it is compile time error

pi=3.141567;

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

}

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

FinalVariable f=**new** FinalVariable();

f.changePi();

}

}

**Output:** Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The final field FinalVariable.pi cannot be assigned

**A34) write code for interface and create class to implement that interface**

**Answer:**

**Code:**

**Interface:**

**package** com.rithika;

**public** **interface** EmployeeInterface {

**public** **void** setEmployee(**int** empid,String empName);

**public** **void** displayEmployee();

}

**Implementation:**

**package** com.rithika;

**public** **class** EmployeeImpl **implements** EmployeeInterface {

**int** id;

String name;

@Override

**public** **void** setEmployee(**int** empid, String empName) {

id=empid;

name=empName;

}

@Override

**public** **void** displayEmployee() {

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

}

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

EmployeeImpl emp=**new** EmployeeImpl();

emp.setEmployee(100, "Rithika");

emp.displayEmployee();

}

}

**Output:** 100 Rithika

**A35) write code for creating abstract class**

**Answer:**

**Abstract Class:**

Abstract class is nothing but declaring the class with abstract keyword. It can have two types of methods that are abstract and non-abstract methods.

**Abstraction:**

Abstraction is nothing but hiding the implementation of method and showing only functionality to the client. We can achieve abstraction through Abstract class and Interface.

**Code:**

**package** com.rithika;

**public** **class** AbstarctClassDemo **extends** AbstractDemo{

@Override

**void** dispaly() {

System.***out***.println(calcSpeed(100));

}

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

AbstractDemo a=**new** AbstarctClassDemo();

a.dispaly();

}

}

**abstract** **class** AbstractDemo{

**abstract** **void** dispaly();

String calcSpeed(**int** speed){

**if**(speed==40){

**return** "Limit";

}

**else** **if**(speed==70){

**return** "Speed";

}

**else** **if**(speed==100){

**return** "High Speed";

}

**return** **null**;

}

}

**Output:** High Speed

**A36) implement method overloading**

**Answer:**

**Code:**

**package** com.rithika;

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

**void** sum(**int** x){

System.***out***.println("x="+x);

}

**void** sum(**int** x,**int** y){

**int** z=x+y;

System.***out***.println("z="+z);

}

**void** sum(**int** x,**double** y){

**double** z=x+y;

System.***out***.println("sum="+z);

}

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

MethodOverloading m=**new** MethodOverloading();

m.sum(10);

m.sum(20,30.56d);

m.sum(10, 20);

}

}

**Output:**

x=10

sum=50.56

z=30

**A37) implement method overriding**

**Answer:**

**Method overriding:**

If the child class is having the same method as declared in the parent or base class then it is called as method overriding. It is used for runtime polymorphism.

**Rules for method overriding:**

1. Method should be the same as in the parent class.
2. Method parameters and return type should be the same as in the parent class.
3. It must be inheritance (IS-A relationship).

**Code:**

**package** com.rithika;

**class** BankInterest{

**int** getInterest(){

**return** 0;

}

}

//Bank of America interest rate 8%

**class** BOA **extends** BankInterest{

**int** getInterest(){

**return** 8;

}

}

//WellsFargo interest rate 7%

**class** WellsFargo **extends** BankInterest{

**int** getInterest(){

**return** 7;

}

}

//Citigroup interest rate 9%

**class** Citigroup **extends** BankInterest{

**int** getInterest(){

**return** 9;

}

}

**public** **class** BankApp {

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

BOA b=**new** BOA();

WellsFargo w=**new** WellsFargo();

Citigroup c=**new** Citigroup();

System.***out***.println("BOA Rate of Interest:"+b.getInterest());

System.***out***.println("WellsFargo Rate of Interest:"+w.getInterest());

System.***out***.println("Citigroup Rate of Interest:"+c.getInterest());

}

}

**Output:**

BOA Rate of Interest:8

WellsFargo Rate of Interest:7

Citigroup Rate of Interest:9

**A38) implementing polymorphism**

**Answer:**

Polymorphism is nothing but many forms. The ability of take more than one form. There are two type of polymorphism in java they are

1. **Static polymorphism**

It is also called as compile time polymorphism. It can achieve by the method overloading. At compile time java knows which method is needed to invoke based on the parameters of the method. So, it is also called as static binding.

**Code:**

**package** com.rithika.calc;

**public** **class** Calculator {

**void** multiplication(**int** x){

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

}

**void** multiplication(**int** x,**double** y){

System.***out***.println("Two parameters multiplication result: "+(x\*y));

}

**void** multiplication(**int** x,**double** y,**long** z){

System.***out***.println("Three parameters multiplication result: "+(x\*y\*z));

}

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

Calculator c=**new** Calculator();

c.multiplication(10);

c.multiplication(20, 30.89d);

c.multiplication(20, 40.87d, 200);

}

}

**Output:**

One parameter: 10

Two parameters multiplication result: 617.8

Three parameters multiplication result: 163480.0

1. **Dynamic polymorphism**

Dynamic polymorphism is nothing but runtime polymorphism. It can be achieve by method overriding. In this scenario java compiler doesn’t know which method is invoked at compilation time. JVM will decide the which method should invoked at run time.

**Code:**

**package** com.rithika.erpapplication;

**class** Laptop{

**int** acerPrice(){

**return** 0;

}

}

**class** WalMart **extends** Laptop{

**int** acerPrice(){

**return** 719;

}

}

**class** Amazon **extends** Laptop{

**int** acerPrice(){

**return** 650;

}

}

**public** **class** ErpApp {

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

WalMart w=**new** WalMart();

Amazon a=**new** Amazon();

System.***out***.println("Acer Laptop Price in WalMart: "+w.acerPrice());

System.***out***.println("Acer Laptop Price in Amazon: "+a.acerPrice());

}

}

**Output:**

Acer Laptop Price in WalMart: 719

Acer Laptop Price in Amazon: 650

**A39) implementing interface**

**Answer:**

Interface is a reference type in java. It is similar to class. It is having abstract methods. We need to implement these abstract methods in other class. It is also having constants, static methods, default methods, and nested types. Default and static methods are having method body in the interface.

**Code:**

**Interface:**

**package** com.rithika.bank;

**public** **interface** BankInterface {

**float** interestRate();

}

**Class for implementing Interface:**

**package** com.rithika.bank;

//Bank of America interest rate 8%

**class** BOA **implements** BankInterface{

**public** **float** interestRate(){

**return** 8.18f;

}

}

//WellsFargo interest rate 7%

**class** WellsFargo **implements** BankInterface{

**public** **float** interestRate(){

**return** 7.50f;

}

}

//Citigroup interest rate 9%

**class** Citigroup **implements** BankInterface{

**public** **float** interestRate(){

**return** 9.00f;

}

}

**public** **class** BankImpl {

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

BOA b=**new** BOA();

WellsFargo w=**new** WellsFargo();

Citigroup c=**new** Citigroup();

System.***out***.println("BOA Rate of Interest:"+b.interestRate());

System.***out***.println("WellsFargo Rate of Interest:"+w.interestRate());

System.***out***.println("Citigroup Rate of Interest:"+c.interestRate());

}

}

**Output:**

BOA Rate of Interest:8.18

WellsFargo Rate of Interest:7.5

Citigroup Rate of Interest:9.0

**A40) How to do inheritance in java (using extend keyword)**

**Answer:**

One object is acquires the all properties and behavior of parent or base object is called as inheritance. Benefit of inheritance is reusing the methods and fields of parent class.

Inheritance in java:

1. **Single Inheritance:**

Single inheritance is having only one base class and one child class.

**Class A**

**Class B**

**Code:**

**package** com.rithika;

**class** Vehicle{

**void** travelling(){

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

}

}

**class** Car **extends** Vehicle{

**void** wheels(){

System.***out***.println("Car having four wheels");

}

}

**public** **class** SingleInheritance {

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

Car c=**new** Car();

c.travelling();

c.wheels();

}

}

**Output:**

Travelling

Car having four wheels

1. **Multilevel Inheritance:**

**Grand Parent**

**Class A**

**Class B**

**Parent**

**Class C**

**Child**

**Code:**

**package** com.rithika.multi;

**class** Vehicle{

**void** travelling(){

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

}

}

**class** Car **extends** Vehicle{

**void** wheels(){

System.***out***.println("Car having four wheels");

}

}

**class** C6W **extends** Car{

**void** noOfWheels(){

System.***out***.println("C6W is a sports car having 6 wheels");

}

}

**public** **class** MultiLevel {

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

C6W c=**new** C6W();

c.travelling();

c.wheels();

c.noOfWheels();

}

}

**Output:**

Travelling

Car having four wheels

C6W is a sports car having 6 wheels

1. **Hierarchical Inheritance:**

It is having one base class and many derived classes.

**Class A**

**Class C**

**Class B**

**Code:**

**package** com.rithika.hira;

**class** Vehicle{

**void** travelling(){

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

}

}

**class** BMW **extends** Vehicle{

**void** bmwCar(){

System.***out***.println("BMW CAR");

}

}

**class** Ferrary **extends** Vehicle{

**void** ferraryCar(){

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

}

}

**public** **class** Hirarchical {

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

BMW b=**new** BMW();

Ferrary f=**new** Ferrary();

b.travelling();

b.bmwCar();

f.travelling();

f.ferraryCar();

}

}

**Output:**

Travelling

BMW CAR

Travelling

Ferrary

1. **Multiple Inheritance:**

Java doesn’t support the multiple inheritances. Because of reducing the complexity and simplify the language.

**A41) write code to add items to integer, string array**

**Answer:**

**Code:**

**Add items to integer array:**

**package** com.rithika;

**public** **interface** AddIntegerToArray {

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

//**int**[] a=**new** **int**[10];

Integer[] a=**new** Integer[10];

a[0]=20; //Adding integer items to array

a[1]=30;

a[2]=40;

a[3]=50;

a[4]=60;

**for**(**int** i=0;i<5;i++){

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

}

}

}

**Output:** 20 30 40 50 60

**Add items to String array:**

**Code:**

**package** com.rithika;

**public** **interface** AddStringToArray {

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

String[] a=**new** String[10];

a[0]="Rhithika"; //Adding integer items to array

a[1]="Pinky";

a[2]="String1";

a[3]="String2";

a[4]="String3";

**for**(**int** i=0;i<5;i++){

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

}

}

}

**Output:** Rhithika Pinky String1 String2 String3

**Note:** Array is a fixed size. We cannot extend the array. If we want to add more items in array it is better to use ArrayList.

**A42) write code to retrieve items from integer, string array**

**Answer:**

**Retrieving items from integer array:**

**Code:**

**package** com.rithika;

**class** ArrayRetrive{

**int**[] newArray=**new** **int**[10];

**int** length;

**void** retriveArray(**int**[] intArray){

length=intArray.length; //Storing intArray Length

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

newArray[i]=intArray[i]; //Retriving array items

}

}

**void** display(){

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

System.***out***.print(newArray[i]+" "); //display Array items

}

}

}

**public** **class** IntegerArray {

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

ArrayRetrive a=**new** ArrayRetrive();

**int**[] intArray={20,30,40,50,25};

a.retriveArray(intArray);

a.display();

}

}

**Output:** 20 30 40 50 25

**Retrieving items from String array:**

**Code:**

**package** com.rithika;

**class** ArrayRetrive{

String[] newArray=**new** String[10];

**int** length;

**void** retriveArray(String[] stringArray){

length=stringArray.length; //Storing stringArray Length

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

newArray[i]=stringArray[i]; //Retriving array items

}

}

**void** display(){

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

System.***out***.print(newArray[i]+" "); //display Array items

}

}

}

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

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

ArrayRetrive a=**new** ArrayRetrive();

String[] stringArray={"Rithika","Pinky","String1","String2","String2"};

a.retriveArray(stringArray);

a.display();

}

}

**Output:** Rithika Pinky String1 String2 String2

**A43) write code to add items to ArrayList collection**

**Answer:**

**Code:**

**package** com.rithika.arraylistexample;

**import** java.util.ArrayList;

**public** **class** ArrayListDemo {

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

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

al.add("Rithika");

al.add("Siddala");

al.add("Vasu");

al.add("Pinky");

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

}

}

**Output:** [Rithika, Siddala, Vasu, Pinky]

**A44) write code to retrieve items from arraylist (using for each loop\_**

**Answer:**

**Code:**

**package** com.rithika.arraylistexample;

**import** java.util.ArrayList;

**public** **class** ArrayListDemo {

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

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

al.add("Rithika");

al.add("Siddala");

al.add("Vasu");

al.add("Pinky");

**for**(String name:al){

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

}

}

}

**Output:**Rithika  
SiddalaVasuPinky

**A45) write code to add items HashMap**

**Answer:**

**Code:**

**package** com.rithika.hashmapexample;

**import** java.util.HashMap;

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

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

HashMap<Integer, String> emp=**new** HashMap<Integer,String>();

emp.put(156, "Rithika");

emp.put(157, "Pinky");

emp.put(158, "Vasu");

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

}

}

**Output:** {156=Rithika, 157=Pinky, 158=Vasu}

**A46) write code to retrieve items HashMap**

**Answer:**

**Code:**

**package** com.rithika.hashmapexample;

**import** java.util.HashMap;

**import** java.util.Map;

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

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

HashMap<Integer, String> emp=**new** HashMap<Integer,String>();

emp.put(156, "Rithika");

emp.put(157, "Pinky");

emp.put(158, "Vasu");

**for**(Map.Entry e:emp.entrySet()){

System.***out***.println(e.getKey()+" "+e.getValue());

}

}

}

**Output:**

156 Rithika

157 Pinky

158 Vasu

**A47) Write code to add items to hashset**

**Answer:**

**Code:**

**package** com.rithika.hashsetexample;

**import** java.util.HashSet;

**public** **class** HashSetDemo {

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

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

empName.add("Rithika");

empName.add("Pinky");

empName.add("Vasu");

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

}

}

**Output:** [Vasu, Rithika, Pinky]

**A48) Write code to retrieve items to hashset**

**Answer:**

**Code:**

**package** com.rithika.hashsetexample;

**import** java.util.HashSet;

**import** java.util.Iterator;

**public** **class** HashSetDemo {

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

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

empName.add("Rithika");

empName.add("Pinky");

empName.add("Vasu");

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

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

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

}

}

}

**Output:** Vasu

Rithika

Pinky

**A49) write code to connect to JDBC to get rows from employee table**

**Answer:**

**Create database in mysql**

**Query:** CREATE DATABASE emp;

USE emp;

**Create Table employee in mysql:**

**Query:**

CREATE TABLE employee(empid int not null,empname varchar(50) not null,empsal double not null,primary key(empid));

**Insert Data into employee Table:**

**Query:** INSERT INTO employee VALUES(100,”Rithika”,2000.50);

INSERT INTO employee VALUES(101,”Pinky”,5000.00);

INSERT INTO employee VALUES(102,”Vasu”,4000.00);

**Code:**

**Database Connection Code:**

**package** com.rithika.dbconnection;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**public** **class** DatabaseConnection {

**public** Connection getDbConnection(){

Connection con=**null**;

String url="jdbc:mysql://localhost:3306/emp";

String userName="root";

String password="root";

**try**{

Class.*forName*("com.mysql.jdbc.Driver");

con=DriverManager.*getConnection*(url,userName,password);

}

**catch**(Exception ex){

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

}

**return** con;

}

}

**Result Code:**

**package** com.rithika.employee;

**import** java.sql.Connection;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** com.rithika.dbconnection.DatabaseConnection;

**public** **class** GetEmployeeDetails {

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

DatabaseConnection db=**new** DatabaseConnection();

Statement st=**null**;

Connection con=db.getDbConnection();

String query="SELECT \* FROM employee";

**try** {

st=con.createStatement();

ResultSet rs=st.executeQuery(query);

**while**(rs.next()){

System.***out***.print("EmpId:"+rs.getInt(1));

System.***out***.print(" EmpName:"+rs.getString(2));

System.***out***.println(" EmpSal:"+rs.getDouble(3));

}

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

**Output:**

EmpId:100 EmpName:Rithika EmpSal:2000.5

EmpId:101 EmpName:Pinky EmpSal:5000.0

EmpId:102 EmpName:Vasu EmpSal:4000.0

**A50) Write method to return list of rows code to loop throughs**

**Answer:**

**Code:**

**Database Connection Code:**

**package** listreturn;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**public** **class** DatabaseConnection {

**public** Connection getDbConnection(){

Connection con=**null**;

String url="jdbc:mysql://localhost:3306/emp";

String userName="root";

String password="root";

**try**{

Class.*forName*("com.mysql.jdbc.Driver");

con=DriverManager.*getConnection*(url,userName,password);

}

**catch**(Exception ex){

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

}

**return** con;

}

}

**Return List of rows:**

**Code:**

**package** listreturn;

**import** java.sql.Connection;

**import** java.sql.ResultSet;

**import** java.sql.ResultSetMetaData;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**public** **class** ReturnListofRows {

//Return list of all rows from the method

**public** ArrayList<ArrayList<String>> listOfRows(){

ArrayList<String> al=**null**;

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

DatabaseConnection db=**new** DatabaseConnection();

Connection con=db.getDbConnection();

Statement st;

**try** {

st = con.createStatement();

ResultSet rs=st.executeQuery("SELECT \* FROM employee");

ResultSetMetaData rsm=rs.getMetaData();

**int** columnCount=rsm.getColumnCount();

**while**(rs.next()){

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

**for**(**int** c=1;c<=columnCount;c++){

al.add(rs.getString(c));

}

outer.add(al);

}

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** outer;

}

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

ReturnListofRows r=**new** ReturnListofRows();

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

al=r.listOfRows();

**for**(ArrayList<String> data:al){

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

}

}

}

**Output:**

[100, Rithika, 2000.5]

[101, Pinky, 5000]

[102, Vasu, 4000]

**A51) create Employee class**

**Answer:**

**Code:**

**package** com.rithika;

**public** **class** Employee {

**private** **int** id;

**private** String empName;

**private** String designation;

**private** **double** salary;

**public** Employee(**int** id, String empName, String designation, **double** salary) {

**this**.id = id;

**this**.empName = empName;

**this**.designation = designation;

**this**.salary = salary;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getEmpName() {

**return** empName;

}

**public** **void** setEmpName(String empName) {

**this**.empName = empName;

}

**public** String getDesignation() {

**return** designation;

}

**public** **void** setDesignation(String designation) {

**this**.designation = designation;

}

**public** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) {

**this**.salary = salary;

}

}

**A52) Add employee class to list collection**

**Answer:**

**Code: Adding employee class to ArrayList:**

**package** com.rithika;

**import** java.util.ArrayList;

**public** **class** AddEmployeeToListCollection {

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

Employee e1=**new** Employee(100, "Rithika", "Tester", 1000.00d);

Employee e2=**new** Employee(101, "Pinky", "Developer", 2000.00d);

Employee e3=**new** Employee(102, "Vasu", "Tester", 2500.00d);

ArrayList<Employee> al=**new** ArrayList<Employee>();

al.add(e1);

al.add(e2);

al.add(e3);

}

}

**A53) create method that return list of employee collection**

**Answer:**

**Code: Method for return list of employee collection:**

**package** com.rithika;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**class** ReturnEmployeeList{

//Method returns the arraylist of employee

**public** ArrayList<Employee> employeeDetails(){

Employee e1=**new** Employee(100, "Rithika", "Tester", 1000.00d);

Employee e2=**new** Employee(101, "Pinky", "Developer", 2000.00d);

Employee e3=**new** Employee(102, "Vasu", "Tester", 2500.00d);

ArrayList<Employee> al=**new** ArrayList<Employee>();

al.add(e1);

al.add(e2);

al.add(e3);

**return** al;

}

}

**public** **class** AddEmployeeToListCollection {

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

ReturnEmployeeList r=**new** ReturnEmployeeList();

ArrayList al1=**new** ArrayList();

al1=r.employeeDetails();

Iterator i=al1.iterator();

**while**(i.hasNext()){

Employee e=(Employee)i.next();

System.***out***.println(e.getId()+" "+e.getEmpName()+" "+e.getDesignation()+" "+e.getSalary());

}

}

}

**Output:**

100 Rithika Tester 1000.0

101 Pinky Developer 2000.0

102 Vasu Tester 2500.0

**A54) Difference between string, string buffer, string builder with example**

**Answer:**

**Difference Between String, StringBuffer and StringBuilder**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **String** | **StringBuffer** | **StringBuilder** |
| 1 | It is immutable | It is mutable | It is mutable |
| 2 | If we alter values of String then another object will created. | It can change its value because it is mutable. | It can change its value because it is mutable. |
| 3 |  | It is thread-safe. | It is not thread-safe. |
| 4 | If our string is not going to change then we should use String. | If our string can change and it will be accessed from multiple threads then it is better to use StringBuffer. | If our string can change and it can be accessed from a single thread then it is better to use StringBuilder. |

**Code for StringBuffer:**

**public** **class** StringBufferDemo {

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

StringBuffer sb=**new** StringBuffer("Example code for");

sb.append(" String Buffer");

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

}

}

**Output**: Example code for String Buffer

**Code for StringBuilder:**

**public** **class** StringBuilderDemo {

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

StringBuilder sbr=**new** StringBuilder("This is example");

sbr.append(" for StringBuilder");

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

}

}

**Output:** This is example for StringBuilder

**Code for String:**

**public** **class** StringDemo {

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

String s=**new** String("This is example code for");

s+=" String";

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

}

}

**Output:** This is example code for String

**A55) write a code to save data into excel file and read from excel file (POI and jexcel API)**

**Answer:**

**Excel file creation:**

1. Open Excel file
2. Give data in the excel file like this



1. Save file as ExcelOperation.xlsx

**pom.xml:**

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.rithika.excelactions</groupId>

<artifactId>savedatatoexcel</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>savedatatoexcel</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi</artifactId>

<version>3.16</version>

</dependency>

</dependencies>

</project>

**Code for Save data into excel:**

**package** com.rithika.excelactions.savedatatoexcel;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** org.apache.poi.hssf.usermodel.HSSFWorkbook;

**import** org.apache.poi.ss.usermodel.Cell;

**import** org.apache.poi.ss.usermodel.Row;

**import** org.apache.poi.ss.usermodel.Sheet;

**import** org.apache.poi.ss.usermodel.Workbook;

**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;

**class** SaveDataToExcel{

**public** **void** writeDataIntoExcel(String fPath,String fName,String sheetName,String[] data) **throws** IOException{

//Creating file object to open excel file

File f=**new** File(fPath+"\\"+fName);

//Read excel file

FileInputStream fInStream=**new** FileInputStream(f);

Workbook wb=**null**;

//Finding extension of file

String extension=fName.substring(fName.indexOf("."));

//If extension is xlsx, then we should create object of XSSFWorkbook class

**if**(extension.equals(".xlsx")){

wb=**new** XSSFWorkbook(fInStream);

}

//If extension is xlsx, then we should create object of HSSFWorkbook class

**else** **if**(extension.equals(".xls")){

wb=**new** HSSFWorkbook(fInStream);

}

Sheet sh=wb.getSheet(sheetName);

**int** rowsCount=sh.getLastRowNum()-sh.getFirstRowNum();

//First row from sheet

Row row=sh.getRow(0);

//Creating New row

Row newRow=sh.createRow(rowsCount+1);

//Loop for over the cell of created row

**for**(**int** i=0;i<row.getLastCellNum();i++){

Cell cell=newRow.createCell(i);

cell.setCellValue(data[i]);

}

//Close the file input stream

fInStream.close();

//Save data into excel file

FileOutputStream fOutStream=**new** FileOutputStream(f);

wb.write(fOutStream);

//Close Output Stream

fOutStream.close();

}

}

**public** **class** App

{

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

{

String[] data={"Rithika Siddala","Address3"};

SaveDataToExcel s=**new** SaveDataToExcel();

s.writeDataIntoExcel("D:\\excel", "ExcelOperation.xlsx", "Sheet1", data);

}

}

**Output in excel file:**

****

**Read data from excel file:**

**package** com.rithika.excelactions.savedatatoexcel;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** org.apache.poi.hssf.usermodel.HSSFWorkbook;

**import** org.apache.poi.ss.usermodel.Row;

**import** org.apache.poi.ss.usermodel.Sheet;

**import** org.apache.poi.ss.usermodel.Workbook;

**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;

**public** **class** ReadDataFromExcel {

**public** **void** readFromExcel(String fPath,String fName,String sheetName) **throws** IOException{

File f=**new** File(fPath+"\\"+fName);

//Reading Excel file

FileInputStream fInStream=**new** FileInputStream(f);

Workbook wb=**null**;

String extension=fName.substring(fName.indexOf("."));

**if**(extension.equals(".xlsx")){

wb=**new** XSSFWorkbook(fInStream);

}

**else** **if**(extension.equals(".xls")){

wb=**new** HSSFWorkbook(fInStream);

}

Sheet s=wb.getSheet(sheetName);

**int** rowsCount=s.getLastRowNum()-s.getFirstRowNum();

**for**(**int** r=0;r<rowsCount+1;r++){

Row row=s.getRow(r);

**for**(**int** c=0;c<row.getLastCellNum();c++){

System.***out***.print(row.getCell(c).getStringCellValue()+"\t");

}

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

}

}

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

ReadDataFromExcel read=**new** ReadDataFromExcel();

read.readFromExcel("D:\\excel", "ExcelOperation.xlsx", "Sheet1");

}

}

**OUTPUT:**

Name Address

Rithika Address1

Pinky Address2

Rithika Siddala Address3

**A56) how to update the data into XML file and read data from XML file**

**Answer:**

**Create XML File in D:\xml folder**

**org.xml:**

<?xml version="1.0" encoding="UTF-8"?>

<Organization>

<employee>

<id>120</id>

<name>Rithika</name>

<salary>5000</salary>

</employee>

</Organization>

**Code to update data into xml:**

**package** updatexml;

**import** java.io.File;

**import** java.io.IOException;

**import** javax.xml.parsers.DocumentBuilder;

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** javax.xml.parsers.ParserConfigurationException;

**import** javax.xml.transform.Transformer;

**import** javax.xml.transform.TransformerConfigurationException;

**import** javax.xml.transform.TransformerException;

**import** javax.xml.transform.TransformerFactory;

**import** javax.xml.transform.dom.DOMSource;

**import** javax.xml.transform.stream.StreamResult;

**import** org.w3c.dom.Document;

**import** org.w3c.dom.Element;

**import** org.w3c.dom.NamedNodeMap;

**import** org.w3c.dom.Node;

**import** org.xml.sax.SAXException;

**class** XmlFileUpdate{

**public** **void** modifyXmlFile() **throws** ParserConfigurationException, SAXException, IOException, TransformerException{

String fPath="D:\\xml\\org.xml";

DocumentBuilderFactory factory=DocumentBuilderFactory.*newInstance*();

DocumentBuilder builder=factory.newDocumentBuilder();

Document docFile=builder.parse(fPath);

//Retriving the root element from xml

Node organization=docFile.getFirstChild();

Node employee=docFile.getElementsByTagName("employee").item(0);

//Update Employee attribute

NamedNodeMap attribute=employee.getAttributes();

Node nodeAttribute=attribute.getNamedItem("id");

nodeAttribute.setTextContent("1");

//Add new node to employee

Element designation=docFile.createElement("designation");

designation.appendChild(docFile.createTextNode("Tester"));

employee.appendChild(designation);

//Write the data into xml file

TransformerFactory transFactory=TransformerFactory.*newInstance*();

Transformer trans=transFactory.newTransformer();

DOMSource ds=**new** DOMSource(docFile);

StreamResult rs=**new** StreamResult(**new** File(fPath));

trans.transform(ds, rs);

System.***out***.println("Updated XML File");

}

}

**public** **class** UpdateXml {

**public** **static** **void** main(String[] args) **throws** ParserConfigurationException, SAXException, IOException, TransformerException{

XmlFileUpdate x=**new** XmlFileUpdate();

x.modifyXmlFile();

}

}

**OUTPUT MODIFIED XML FILE:**

**<?xml version="1.0" encoding="UTF-8" standalone="no"?>**

**<Organization>**

**<employee id="1">**

**<name>Rithika</name>**

**<salary>5000</salary>**

**<designation>Tester</designation>**

**</employee>**

**</Organization>**

**Code to Read data into xml:**

**package** updatexml;

**import** java.io.File;

**import** java.io.IOException;

**import** javax.xml.parsers.DocumentBuilder;

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** javax.xml.parsers.ParserConfigurationException;

**import** javax.xml.transform.Transformer;

**import** javax.xml.transform.TransformerConfigurationException;

**import** javax.xml.transform.TransformerException;

**import** javax.xml.transform.TransformerFactory;

**import** javax.xml.transform.dom.DOMSource;

**import** javax.xml.transform.stream.StreamResult;

**import** org.w3c.dom.Document;

**import** org.w3c.dom.Element;

**import** org.w3c.dom.NamedNodeMap;

**import** org.w3c.dom.Node;

**import** org.w3c.dom.NodeList;

**import** org.xml.sax.SAXException;

**class** XmlFileRead{

**public** **void** readXmlFile() **throws** ParserConfigurationException, SAXException, IOException, TransformerException{

String fPath="D:\\xml\\org.xml";

DocumentBuilderFactory factory=DocumentBuilderFactory.*newInstance*();

DocumentBuilder builder=factory.newDocumentBuilder();

Document docFile=builder.parse(fPath);

//Retriving the root element from xml

docFile.getDocumentElement().normalize();

System.***out***.println("Root Node:"+docFile.getDocumentElement().getNodeName());

//Create List for employee node

NodeList nodeList=docFile.getElementsByTagName("employee");

**for**(**int** tmp=0;tmp<nodeList.getLength();tmp++){

Node nameNode=nodeList.item(tmp);

System.***out***.println("Present Element:"+nameNode.getNodeName());

**if**(nameNode.getNodeType()==Node.***ELEMENT\_NODE***){

Element ele=(Element)nameNode;

System.***out***.println("Emp Id: "+ele.getAttribute("id"));

System.***out***.println("Name: "+ele.getElementsByTagName("name").item(0).getTextContent());

System.***out***.println("Salary: "+ele.getElementsByTagName("salary").item(0).getTextContent());

System.***out***.println("Designation: "+ele.getElementsByTagName("designation").item(0).getTextContent());

}

}

}

}

**public** **class** ReadXml {

**public** **static** **void** main(String[] args) **throws** ParserConfigurationException, SAXException, IOException, TransformerException{

XmlFileRead x=**new** XmlFileRead();

x.readXmlFile();

}

}

**Output:**

Root Node:Organization

Present Element:employee

Emp Id: 1

Name: Rithika

Salary: 5000

Designation: Tester