Chapter 6

Advanced features of OOP

JAVA is a fully object oriented programming language. In previous chapter-5, some features as Class, Object, reusing of Class, Inheritance etc. were discussed. Some advanced features of OOP like: Interface, Multiple Inheritance, Constructor, default constructor, Overloaded constructor, Package, Using the user defined Package, development of package, Package development environment, Integrated Development Environment, Eclipse etc. are discussed.

Interface:

JAVA does not support multiple inheritances. A child class can't inherit from multiple parents at a time but it is possible in C++. As example, **class C** can't inherit properties of **Class A** and **Class B** directly. But, **Class C** can inherit from **Class B** and **Class B** can inherit from **Class A**. It is known as single inheritance. But, through interfaces concept of multiple inheritances are possible to achieve and discussed below:
Interfaces are like Class declaration and class constructions. But, interfaces don't contain and constructors. All variables of interfaces must be public, static and final by default; methods are public, abstract and non-static. Within interfaces different methods are declared but not constructed. According to the requirements, methods are constructed within the operating class which is created from the interface class. Multiple-inheritance is not provided by Java but, **interface** gives that facility in Java. Interface looks like a class and is declared as:

```
public interface WildAnimal
{
    public abstract void food();
}
```

Rule: in JAVA, a class can't be inherited from multiple classes directly, but from multiple interfaces.

Problem: wap to create two interfaces and then create another class from those two interfaces. Declare two methods within two interfaces and then call those methods from main() method.

```
//Population.java: two interfaces are declared, a main class 'Population' is inherited from two interfaces 'man' and 'home'.
interface man
{
	int count_man=5;
	void product();
}
interface home
{
	int count_home=8;
	void show();
}
//new class is created from two interfaces
class Population implements home, man
{
	public void product()
	{
		System.out.println("Total population : "+count_man*count_home);
	}
	public void show()
	{
		System.out.println(" Total home : "+count_home);
		System.out.println(" No of man per home: "+count_man);
	}
	public static void main(String arr[])
	{
		Population pop= new Population();
		pop.show();
		pop.product();
	}
}
OUTPUT:
Total home : 8
 No of man per home: 5
Total population : 40
```

Discussion: in this program, two interfaces by names '*man*' and '*home*' are created within the same program. The main class '*Population.java*' is created from the two interfaces as:

class Population implements home, man
{…}

The advantage is that the new class '*Population*' will inherit all properties of the both classes '*man*' and '*home*' which is the art of multiple inheritances of Java. All properties, as per declaration, will be inherited within the child class '*Population*' and through the object all methods and variables can be accessed as it is done:

```
pop.show();
pop.product();
```

Accordingly, the *product()* method has achieved the rights of accessing variables of both interfaces as:

```
count_man*count_home
```

Problem: wap to create an Interface with two method declaration, then write another program where create a class from that interface, construct both the methods within this class and access both methods.

```
//Hotel.java: this interface is used in Account.java program
public interface Hotel {
   public int nsf=10;
   public void sweep(int noOfFloor);
   public void cook(int noOfCheff);
}

//Account.java: this program calls interface Hotel.class
//Hotel.java file should be created before and compiled.
class Account implements Hotel
{
    int fl = 0;
    int sf = 0;
     //constructing the method
   public void sweep(int noOfFloor) {
       fl = noOfFloor;
       System.out.println("Number of floors in Hotel: " +fl);
    }
  //constructing another method
  public void cook(int noOfSheff) {
       sf = noOfSheff;
       System.out.println("Number of Sheff in Hotel: " +sf);
   }
   public static void main(String[] arg){
            Account ac = new Account();
            //implementation of methods
            ac.sweep(5);
            ac.cook(nsf); //nsf is a public member of interface Hotel
           System.out.println("Number of Cheff per floor: " +(ac.sf/ac.fl));
    }
}
OUTPUT:
Number of floors in Hotel: 5
Number of Sheff in Hotel: 10
Number of Cheff per floor: 2
```

Discussion: in this example, interface Hotel is an external class. It is a special class called Interface and have three properties of public data type, one is variable *public int nsf=10;* and other two are methods. Observe that methods are declared but not built. The working class 'Account' is created making interface with 'Hotel'. Like

reuses of class file, interface file is also reused in this program. Also, advantages of interfacing are demonstrated in the way that methods could be constructed as requirements within the working class. So, within different classes these methods can be overloaded with different functioning but, names will remain the same.

Multiple Inheritance:

When a class is created from another class and another interface or from two or more interfaces, properties of super class and the interface(s) are inherited within the new class. In previous examples, interfacing with single file are discussed, now, in this example interfacing with two interfaces are demonstrated.

Problem: wap to create two interfaces by name *man* and *home*, and then create another class *Population* from these two interfaces. Demonstrate that properties of the two interfaces are accessible from the working class *Population*.

```
//Population.java: two interfaces are created, another main class 'Population' is created from
//these two interfaces.
interface man
{
    int count_man=5;
    void product();
}
interface home
{
    int count_home=8;
    void show();
}
//a new class from two interfaces
class Population implements home, man
{
    public void product()
   {
            System.out.println("Total population: "+count_man*count_home);
   }
   public void show()
   {
            System.out.println(" Total home : "+count_home);
            System.out.println(" No of man per home: "+count_man);
   }
  public static void main(String arrg[])
  {
            Population pop= new Population();
            pop.show();
            pop.product();
  }
}
```

```
OUTPUT:
Total home: 8
 No of man per home: 5
Total population: 40
```

Discussions: multiple inheritances is achieved from two interfaces as an alternative way. In this example, two interfaces are created as 'man' and 'home' and a new class 'population' is derived from these two interfaces. The keyword is 'implements' used as:

```
class Population implements home, man
{
….
….
}
```

In this way, multiple inheritances is possible in JAVA, but, it is not possible if 'home' and 'man' are like Classes.

Lab Test1: in this program, add 'private' identifier here " private int count_man=5;" and you will get error message,

```
Population.java:4: error: modifier private not allowed here
private int count_man=5;
```

Lab Test2: again put protected as "*protected int count_home=8*;" , compile and you will see error message:

```
Population.java:9: error: modifier protected not allowed here
protected int count_home=8;
```

Constructor:

Constructor has the same name as the class and is invoked automatically when an object(instance of class) is initiated. Constructor has no datatype or return value even void. It is like a method and can be overloaded with different arguments to create different versions of constructors. Sometimes it may be confused as a method. If you don't create a new overloaded constructor, it will be invoked automatically- no need to call it. But if you create duplicate versions of constructors with arguments that to be invoked externally. Within the class, constructor is created to initialize some jobs which would be common for all the program.

There are two types of constructors:

1. Default constructor: it has no arguments
2. Overloaded constructors: it has one or more arguments.

*An example of a Default constructor:

```
//Home.java: demonstrate default constructor
class Home
{
	Home()
	{
		System.out.printf("Class instance has called me");
	}
	public static void main(String[] args)
	{
		Home hm = new Home();
	}
}
```

*Another Example of argument-based constructor:

```
class Volume{
	int l, w, h;
	Volume(int a, int b, int c){
		l = a; w=b, h=c;}
	int vol(){
		int v=l*w*h;
		return v;}
}
```

*Calling the Constructor:

Two ways to calling the constructors are shown below:

```
Home hm = new Home(); //calling default constructor having no arguments
Volume vol=new Volume(4, 7, 12);//calling constructor having three arguments
```

Problem: wap to create a class with a constructor having two arguments-one is id and the other is the name of a student; create a method and from the main() call the constructor providing two data as arguments.

```
//StdConstructor.java: constructor is a method and invoked automatically.
//here, StdConsturctor() has the same name with the class name and has two
//parameters of integer and string data types.
class StdConstructor{
	int stdRoll;
	String stdName;
	StdConstructor(int id, String name)  {
		this.stdRoll = id;
		this.stdName = name;
	}
	void info(){
	    System.out.println("Roll No: "+stdRoll+" Name: "+stdName);
	}
	public static void main(String args[]){
		StdConstructor object1 = new StdConstructor(102,"Ramesh Goyel");
		StdConstructor object2 = new StdConstructor(103,"Marta Zurek");
		object1.info();
```

```
                object2.info();
        }
}
OUTPUT:
Roll No: 102 Name: Ramesh Goyel
Roll No: 103 Name: Marta Zurek
```

Discussion: here, in main() method at first two objects are created by names object1 and object2. These objects have all access to their methods() created within the class. The class StdConstructor has a constructor by same name and initializes two variables. From the main() method, two objects call the constructors with two data as shown in the program. These two data go into two parameters of the two objects as shown in the figure.

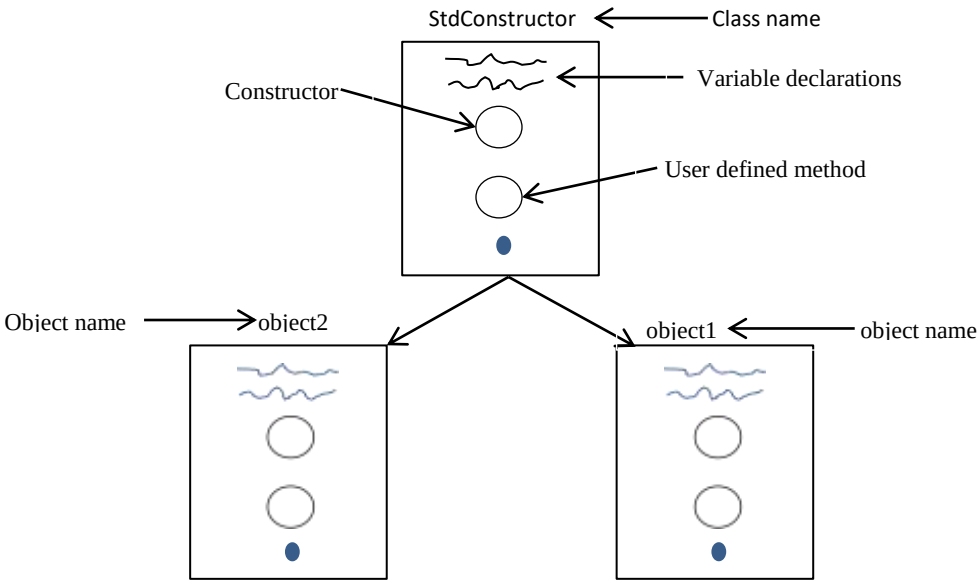

Fig. 6.1. Figure shows how to create object from a Class. Here, two objects are created by names object1 and object1.

Problem: wap to construct two overloaded constructors, two methods by name *add()* with different arguments, create another main class by name *ConstOverload* and from the *main()* module call constructors and methods to display some data.

```
// ConstOverload.java:Write a program in JAVA to demonstrate the method and constructor
overloading.
class Construct
{
        int p, q;
        public Construct(){}
        public Construct(int x, int y){
                p=x;
                q=y;
        }
        public int add(int i, int j){
                return (i+j);
```

```
        }
        public int add(int i, int j, int k){
                return (i+j+k);
        }
        public float add(float f1, float f2){
                return (f1+f2);
        }
        public void printData(){
                System.out.print("First variable = "+p);
                System.out.println("  : Second variable = "+q); }
}
class ConstOverload
{
        public static void main(String args[]){
        int var1=101, var2=201, var3=301;
        Construct c0=new Construct();
        Construct c1=new Construct(var1, var3 );
        c1.printData(); // constructor having two argument will perform
        float f1=7.2F, f2=5.2F;
        int k2=c0.add(var1,var2); // perform on add() having two int arguments
        System.out.println("Value of k2 = "+k2);
        int tot3=c0.add(var1,var2,var3);
        System.out.println("total of 3 var= "+tot3);
        float ft2=c0.add(f1, f2);
        System.out.println("Value of ft2 = "+ft2);
        }
}
```

OUTPUT:

```
First variable = 101  : Second variable = 301
Value of k2 = 302
total of 3 var= 603
Value of ft2 = 12.4
```

Discussion: in this example concept of constructor overloading is shown. In java, when a class is created, a constructor by same name is also created- if the constructor is not created separately. Always, the name of constructor and the class are the same. Constructor has no data type. If constructor is not created, an empty constructor will be executed. Here, in this example, within the class 'Construct' two constructors are created, the first one is empty and having done nothing within braces {}, but, the second constructor's name is same and has two arguments as:

```
public Construct(int x, int y){
                        p=x;
                        q=y;}
```

When, by same name to constructor any method is created it is called overloading of constructor. In this constructor two arguments are assigned as '*int x, int y*'. It will be called through two arguments as:

```
Construct c1=new Construct(var1, var3 );
```

In this example, also function overloading is demonstrated. If by same name many methods are created, it is known as method overloading. Here, three methods are created by name '*add()*', but, careful observation will show that arguments are different for each method of 'add'; the first method has two integer types of arguments as '*add(int i, int j)*' ; the second method has three arguments as '*add(int i, int j, int k)*' and the third method has float type of argument as '*public float add(float f1, float f2)*'. In these ways, methods are identified as different to the compiler. When any method is invoked with two integer values the corresponding method will be called and executed as '*int k2=c0.add(var1,var2)*;' . Here, c0 is an object created from the class '*Construct*', so, this object has access to all variables and to all methods of the class. So, from the object, *c0*, the method *add()* has invoked or called in this way '*c0.add(var1,var2)*'; the return value is integer of this method as declaration. A parameter is required to hold the return value of that method and written in this way:'*k2=c0.add(var1,var2)*;'. Variable '*k2*' was not declared before, to match the return data type of the method, it is declared as integer as ' *int k2*'.

In the same way, when the method is called as '*int tot3=c0.add(var1,var2,var3)*;' the corresponding method will be invoked. In all cases, the name of the method is same but the arguments are different in numbers or as data type. This concept of using the same name of methods is known as method or function overloading.

N.B. in Java, the term 'method' is used instead of 'function'.

Package:

Till now, we have used many inbuilt packages of java such as: java, javax, lang, awt, swing, io, net, util etc. These packages contain some Classes, methods and constants or variables and importing a particular package we can use components of that package as:

```
Import awt.*;
import javax.swing.*;
```

All inbuilt packages and Classes are reusable; users can import and use components present inside it. We have demonstrated in many programs how to create user defined class. Now we shall demonstrate how to create a user defined package and to create some classes inside that package. A package may contain classes, interfaces, enumerations, and annotations within it. **The rule is that the 'package' statement should be the first line of the new program of creating a package**. In a file only one package can be created and it is unique.

An example is shown below:

```
//Datareadwrite.java: to create a package
package packDB;
public class Datareadwrite{
public void fileCon()
```

```
{
  System.out.println("IP,Username,password to be given");
}
public static void main(String[] args){
     System.out.println(" to develop data reading..");
  }
}
```

How to run this program:

```
F:\example\javac –d . Datareadwrite.java
F:\example\java packDB.Datareadwrite
```

Discussions: Here, '-d' parameter is mentioned for telling the compiler about the directory where the package will be created. A new folder will be created as 'packDB' and the compiled class file will be stored within that folder. After '-d' there is a full stop mark '.' which will indicate the directory to follow at the time of execution of the package. This full stop must be given after –d parameter.

(N.B. when we tested the program by putting a comma ',' instead of full stop marks, it did not give any error in compilation, but, did not created either package directory nor the class file.)

Rule: package name will be the same to the directory name where the package will be stored and the same directory name will be aerated automatically.

An example of my computer is shown below:
Directory of C:\example\advancedJAVA**packDB**

```
11-09-2021  11:01    <DIR>          .
11-09-2021  11:01    <DIR>          ..
11-09-2021  11:01               453 Datareadwrite.class
               1 File(s)            453 bytes
               2 Dir(s)  63,980,519,424 bytes free
```

This example creates a new package by name 'packDB' in the default directory where the program in compiled. It also creates a class 'Datareadwrite.class' in the default directory after compilation.

Using the user defined Package:

In above example, we have created a package by name 'packDB' and it creates a folder for the package automatically by the same name of the package 'packDB'. Now, we shall create a program which will import this package and use the class and method available under the package.

```
 //ExPackage.java: this program uses package packDB and its method.
import packDB.*;
public class ExPackage{
public static void main(String[] args){
          packDB.Datareadwrite rd=new packDB.Datareadwrite();
     rd.fileCon();// a method of the packDB package.
```

```
    //System.out.println(" ..to develop data reading..");
  }
}
```

Discussion: in this example previously created package 'packDB' is imported, we have loaded all components present within the package as '*import packDB.*;*'. Another point is that path of the class should be mentioned here as '*packDB.Datareadwrite*', because the class file is not available in the default directory but into the subdirectory of 'packDB' as we have mentioned before when the package was created.

Development of Package:

A package is an integrated, interlinked and assembled thing of many components of programs. Everyday, we work with different software. Like: msword, msexel, windows, linux, unix, myjio, facebook, whatsapp etc. which all are examples of packages. If you check the folder of that software, you will see there are different subfolders and files which word in an integrated environment name as package. As general idea, we can say that they have compiled all these components under any development IDE and have made the package. Here, in this section, some preliminary concept of developing package will be given with practical examples.

Eclipse:

Eclipse is known as Integrated Development Environment (IDE). For developing java based large projects, it is impossible to manage large number of files from command line arguments. Eclipse is a software tool to manage projects under visual environments. It has all necessary menus and icons to manage the project. At opening it may ask about the workspace or directory of the project where it will be created as shown below:

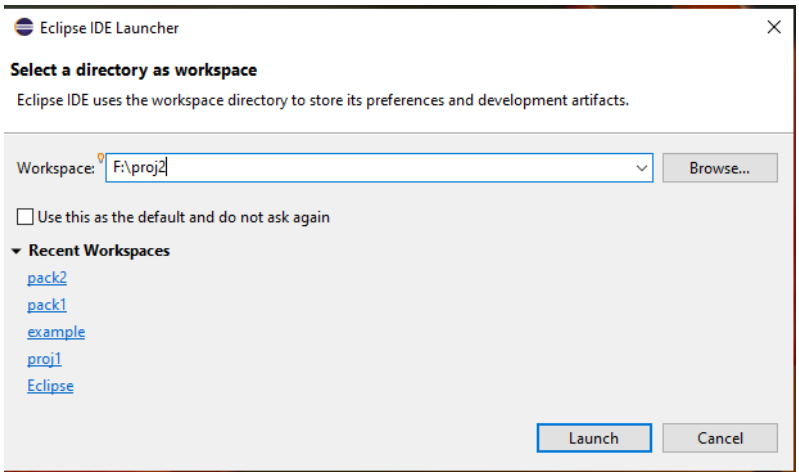

Fig. 6.2. opening window of Eclipse; it shows how to start a project under Eclipse.

By clicking the Launch button, Eclipse will open a screen as shown below,

Users can do necessary jobs selecting particular options. In this case, if we selected 'Create a new Java project', it will ask to give name of project and then follows next steps.

Creating a Project:

A Project is an integrated form of many packages, classes, interfaces, constants etc. As example, JAVA is a project having many packages and other components inside it.

In Eclipse, from the menu bar a new project can be initiated as:
File > new > Java Project, (give a name, say proj2)

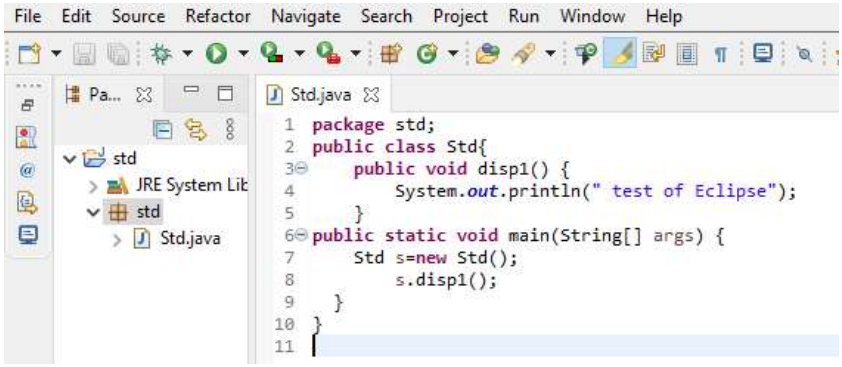

Fig. 6.3. figure shows editor of development of project programs. Here, package name is 'std' and the main class name is 'Std.java'; differences are that first letter is small in 'std' package but, it is capital letter for 'Std.java' class name.

The main class name in 'Std.java' which is created as: from the package explorer pane of left side, select package name 'student' then press right click> new > class> say 'Std', if this class will contain main() module then select the check box as shown in figure, then click 'Finish' button. Now, you have to write some code of programming as example, we have written as shown in figure. Once, let us check what we have done, work space-it is the project name; package name is 'std' – first letter is in small letter; main class is 'Std.java' which contain main() module. Now, we have to execute/run the program; select 'Std.java' from left pane >right click on mouse> Run as > Java Application; you will see the output in the downmost pane of the screen.

Another example with reusable class:

In this example, using Eclipse, we have created a package 'student' under the main class 'Std.class' is created. That screen is shown below; we have run it as a project and output is shown below also.

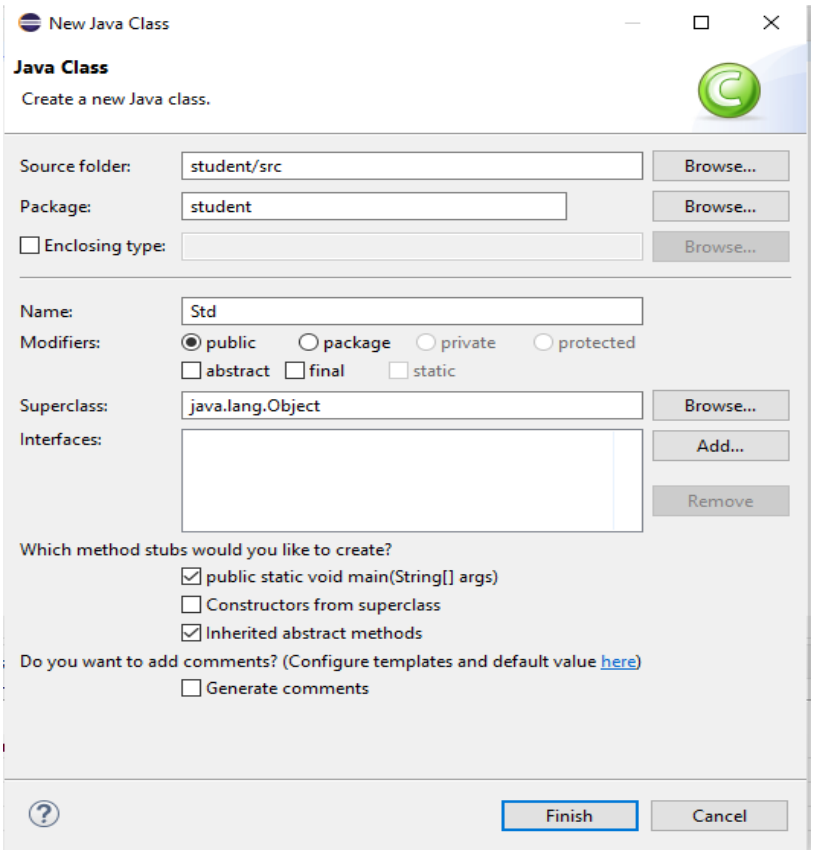

Fig. 6.4. figure shows setup environments

All works are done.

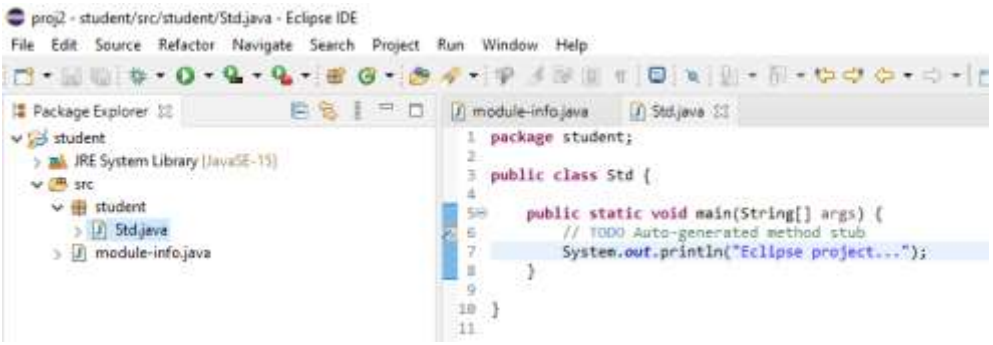

Fig. 6.5. figure shows the project 'student' in left panel and the 'Std.java' class

After that, we have created another class 'Routine.java' as shown below; this class does not contain main() module but contains two variables 'class_date' and 'teacher' with fixed data. This class is invoked by the main class 'Std.java' and printed values of two variables.

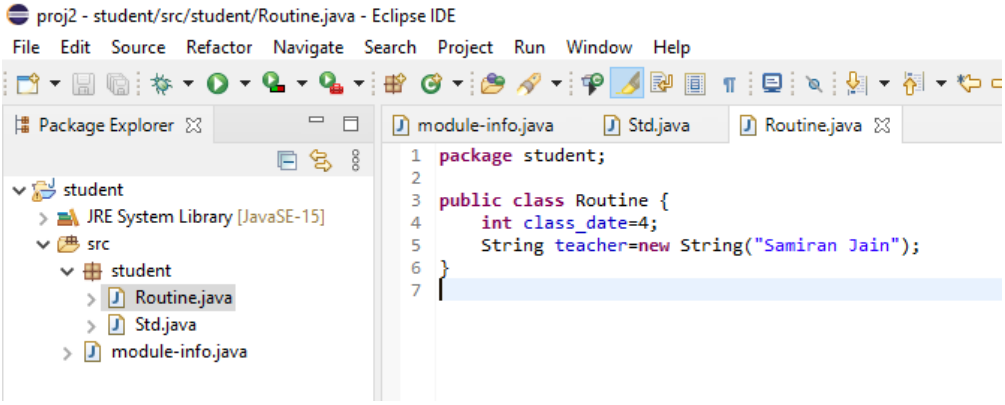

Fig. 6.6. figure shows the project 'student' in left panel and another class 'Routine.java' in right panel.

All of these things are now under package '*student*' and project '*proj2*'. In Eclipse, separate compilation is not required, it is done internally, we need to run the package by clicking right button of mouse over 'Std.java'>Run as> Java Application. We got result in down pane and shown in figure 6.7 below. The main class is 'Std.java' which contain main(){…} module and when this class is compiled, all other components are compiled also.

A project can be edited later also. Other modules can be added, deleted or edited as per requirements. All options are available under different menus of Eclipse IDE.

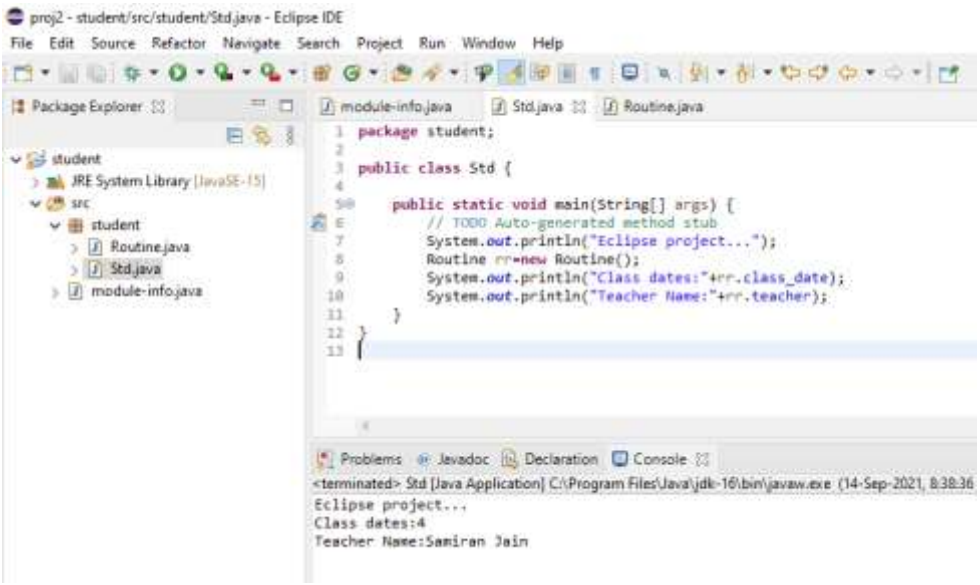

Fig. 6.7. figure shows the project 'student', class 'Routine.java' and class 'Std.java' in left panel. To execute, the main class, Std.java, is executed. Output of the program is shown at down most panels.

Below, both the programs are shown which are written within the Eclipse IDE for creating a package by name '*student*'; it was created as a project '*proj2*'. Within *Routine.java* only a simple class '*Routine*' is developed which was invoked within another program '*Std.java*'. Now it has become a user defined package by name '*student*'.

```
//Routine.java
package student;
public class Routine {
        int class_date=4;
        String teacher=new String("Samiran Jain");
}
//Std.java :
package student;
public class Std {
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                System.out.println("Eclipse project...");
                Routine rr=new Routine();
                System.out.println("Class dates:"+rr.class_date);
                System.out.println("Teacher Name:"+rr.teacher);
        }
}
```

N.B. these programs were written and executed under Eclipse IDE, from that editor program is copied, so, different colors are shown in different keywords.
Some concepts of developing package or project are discussed but not sufficient for developing commercial projects. For that it is suggested to develop small projects with many class files, interlined with main class. Then, compile the main class and run the project. Results will be displayed within the result-panel of Eclipse.

Select True / False:

1. In Java Multiple Inheritances are possible through multiple interfaces.(T/F)

2. Protected type of variable will be inherited within the derived class.(T/F)

3. Private elements will be derived as private in derived class(T/F)

4. Constructor name and Class name can be dissimilar(T/F).

5. Default constructor has no arguments(T/F).

6. Void keyword must be used at beginning of a constructor(T/F).

7. In a class, many constructors can be there (T/F).

8. Overloaded constructors can contain multiple arguments(T/F)

9. Default constructor is called/executed by mentioning its name in main method(T/F)

10. The keyword package creates a interface (T/F).

Answer:
1. True , 2. False , 3. False 4. False 5. True

6. False , 7. True 8. True 9. False 10. False

Short Questions:

(a) What do you understand by OOP ?
(b) Mention some features of OOP?
(c) What is the advantages of inheritance?
(d) Why it is required to create object to write program?
(e) What is multiple inheritances?
(f) Define constructor of JAVA.
(g) What do you understand by constructor overloading?
(h) How a package is can be created in JAVA?
(i) Describe the Eclipse IDE.
(j) What are the differences of a Project and a Package?

Link of PPT of this book and all source codes:

https://drive.google.com/drive/folders/1bMxMCaqPe0W35COAdn_-BrnV_uzQ-tNO?usp=sharing

(Note: to get access, copy and paste this link to your browser)