

## Lesson 04 Demo 05

### Creating Packages and Access Modifiers

**Objective** - To implement packages and access modifiers in Java

**Tools required** - Eclipse IDE

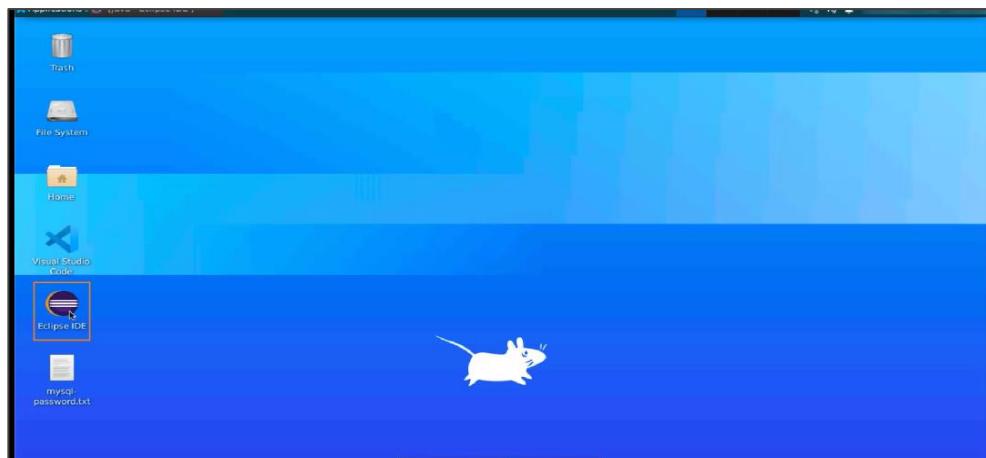
**Prerequisite** - None

#### Steps to be followed:

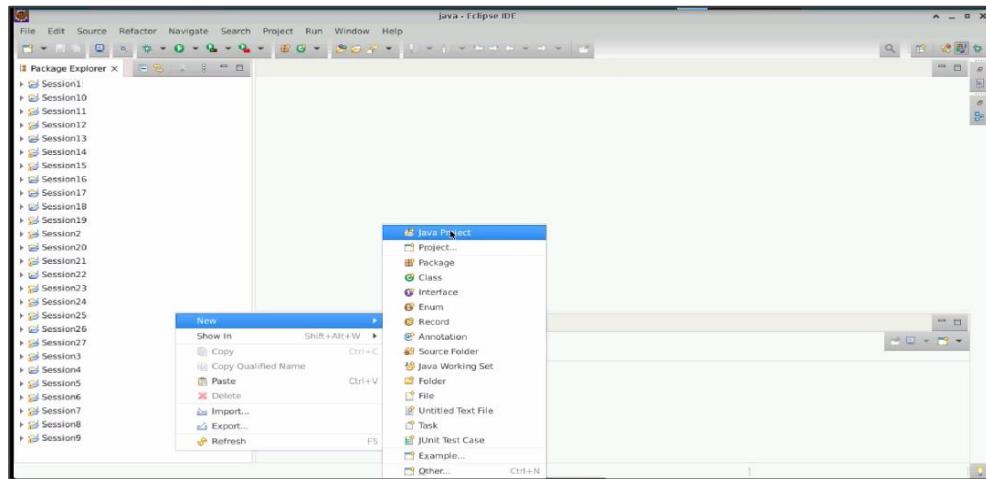
1. Create java package and class
2. Implement the use of access modifiers in class declaration
3. Implement the use of access modifiers inside the class
4. Implement the use of access modifiers outside the class
5. Implement the difference between protected and default access modifier

#### Step 1: Create java package and class

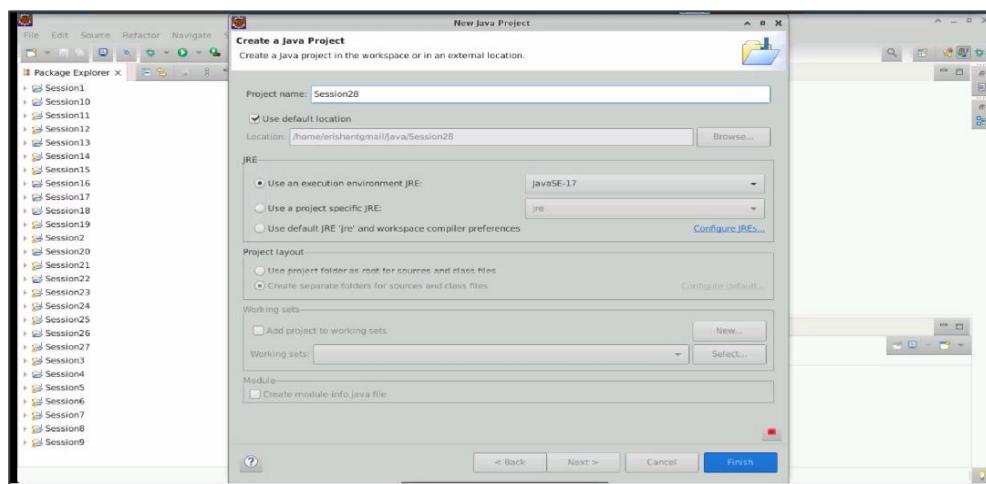
##### 1.1 Open the **Eclipse IDE**.



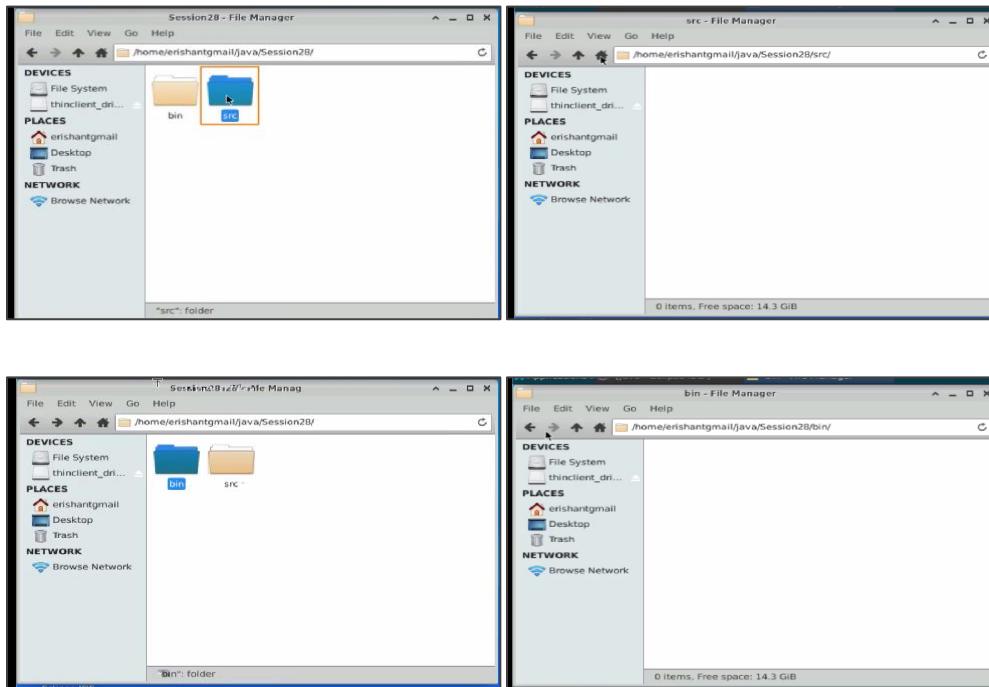
## 1.2 Select File, then New, and then Java Project.



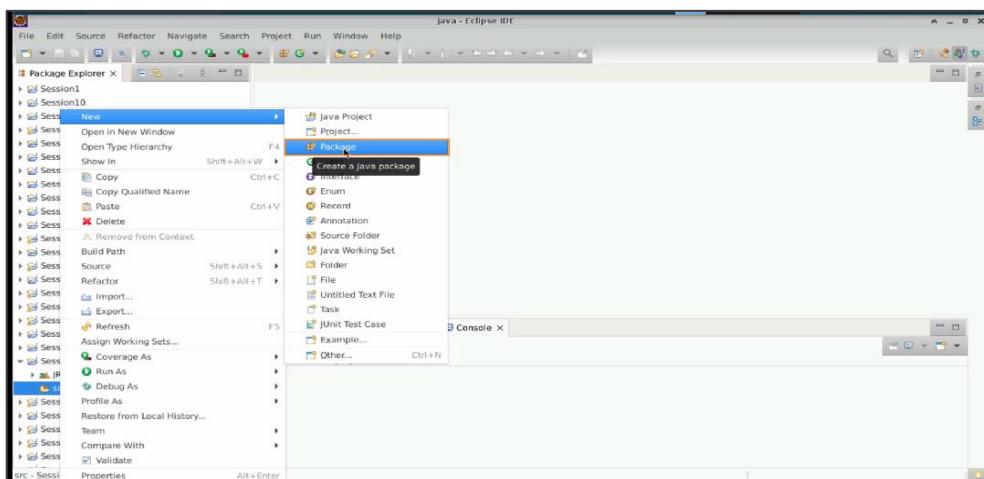
## 1.3 Name the project as “Session28” and select Finish.



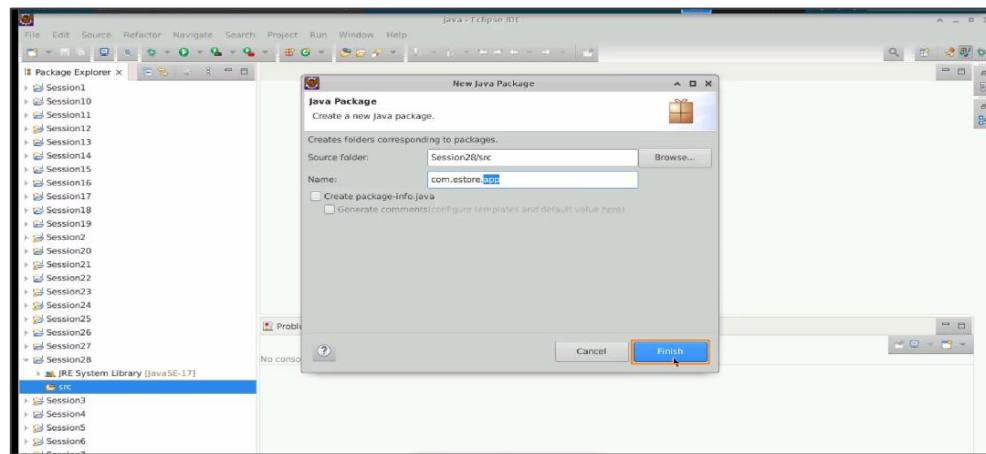
- 1.4 In your **File Manager**, click on the java workspace and open **Session28** folder. Two folders are placed inside, a **source folder** and a **bin folder**. As of now, both the folders are empty.



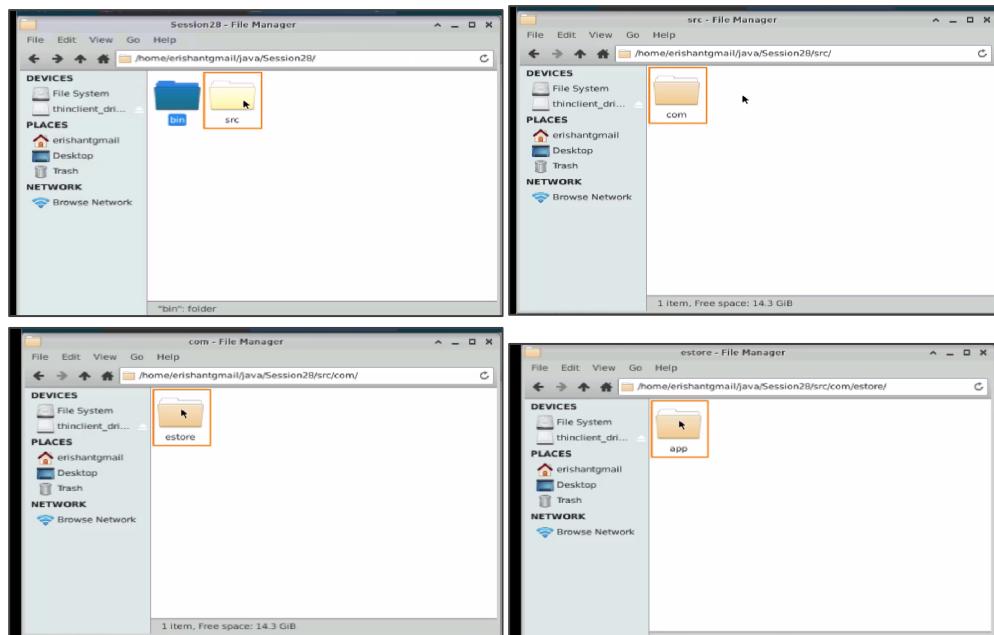
- 1.5 In the left panel, inside the project **Session28**, right click on the **src** folder, select **New**, and then **Package** in order to create a java package. A java package is a directory in which the source files are placed.



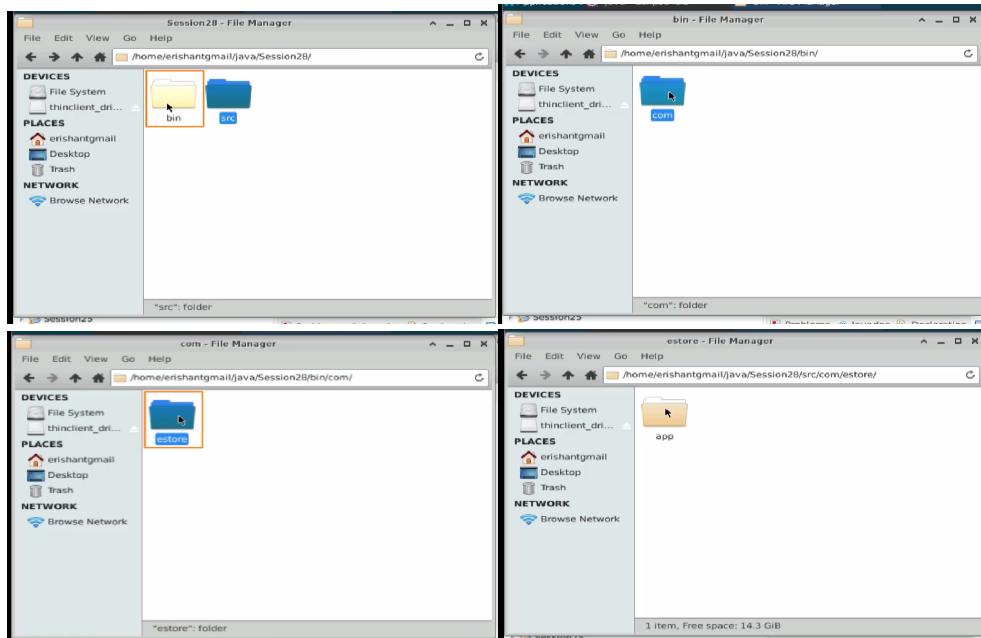
- 1.6 Name the java package as **com.estore.app**. The structure of the package name is the company's domain name in the reverse order. Select **Finish** and hence a package is created.



- 1.7 Open the **File Manager** and in the **src** folder of **Session28**, we will have a **com** folder, then the **estore** and then the **app** folder. Only the directory structure has been created in the app.

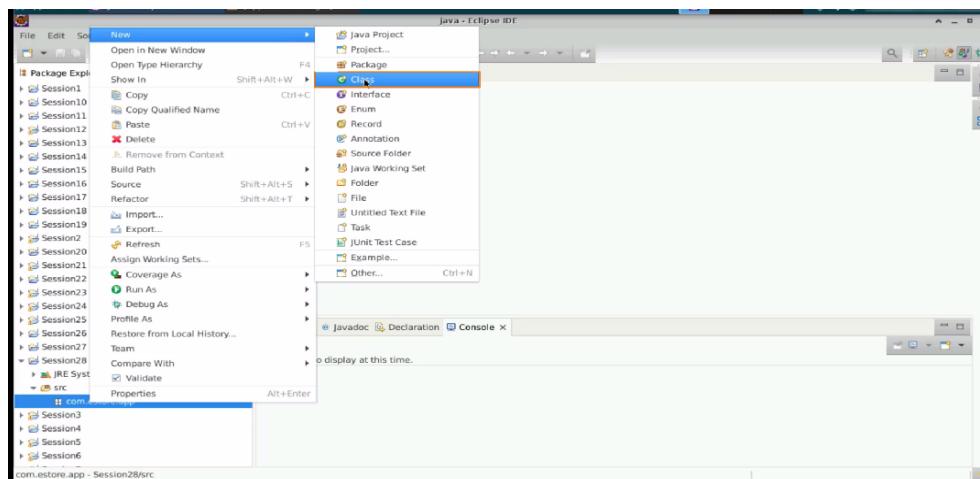


1.8 In the **bin** folder of **Session28** also, we will have a **com** folder, then the **estore** and then the **app** folder.

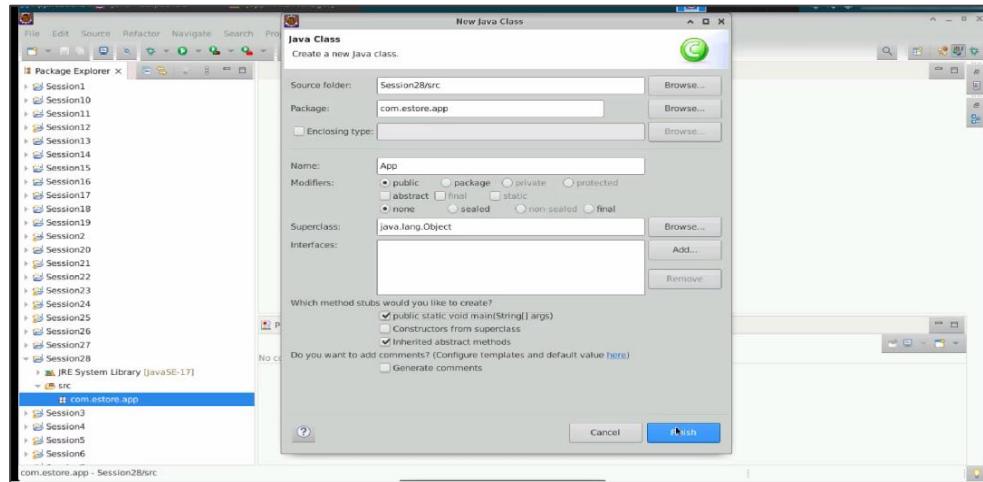


Note:- When you build a package, you are basically creating folders, and when you use the dot operator, you are creating nested folders.

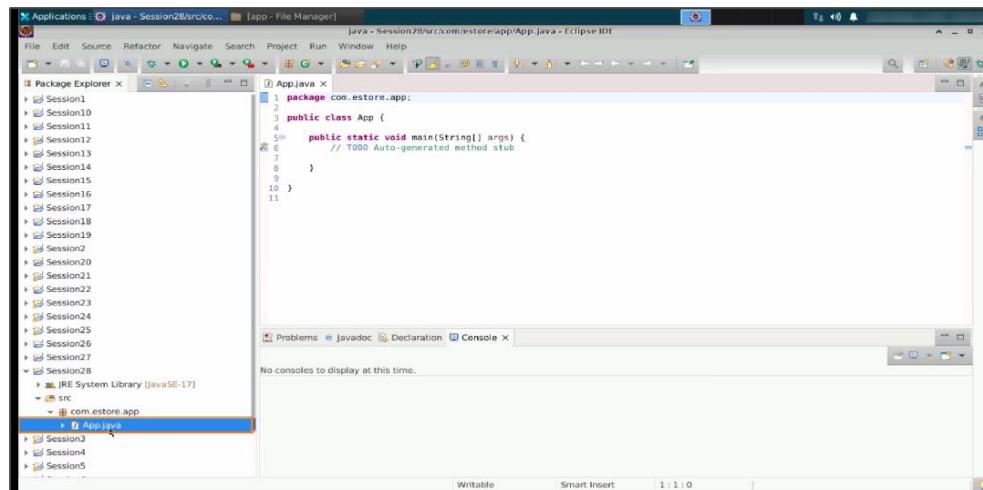
1.9 Right click on the package, select **New**, and then select **Class**.

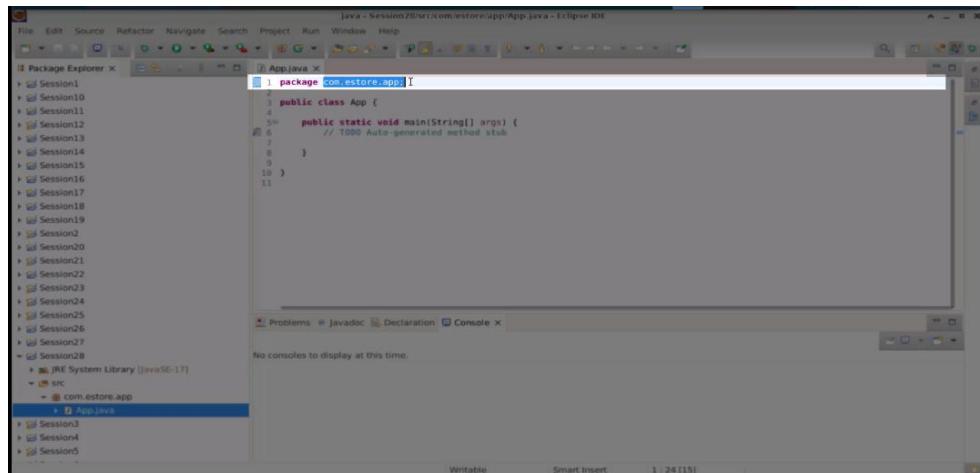


- 1.10 Name the class as **App** and select the option **public static void main(String[] args)**.  
 Select **Finish**. This will create a java class with a **main()** method.

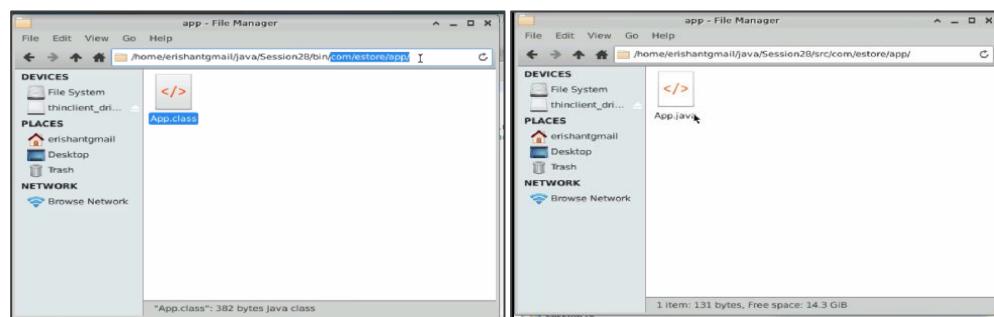


- 1.11 Inside the package, an **App.java** has been created and when you open this class, you can see a declaration on the top as **package com.estore.app** which means that when this class will be compiled, it will be created inside this package.





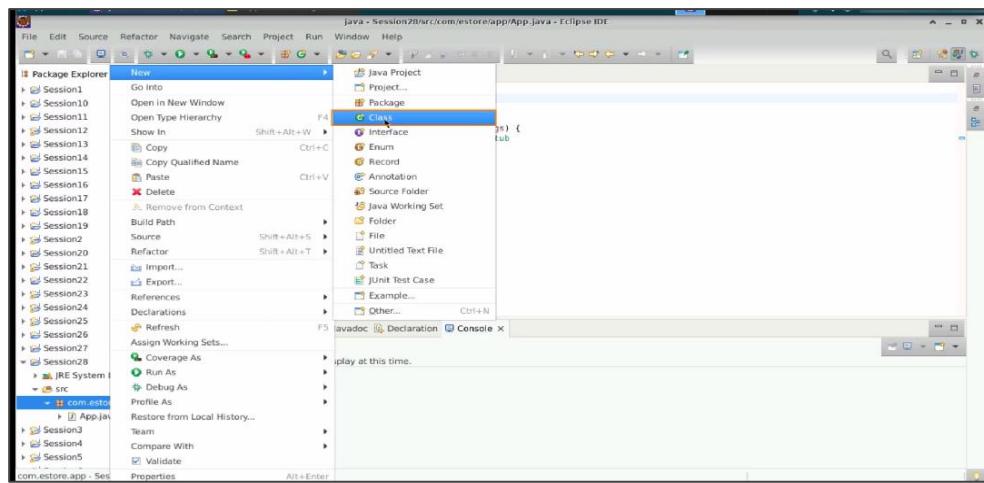
1.12 In the **File manager**, inside the **bin** folder, we will have **App.class** and inside the **src** folder, we will have **App.java**.



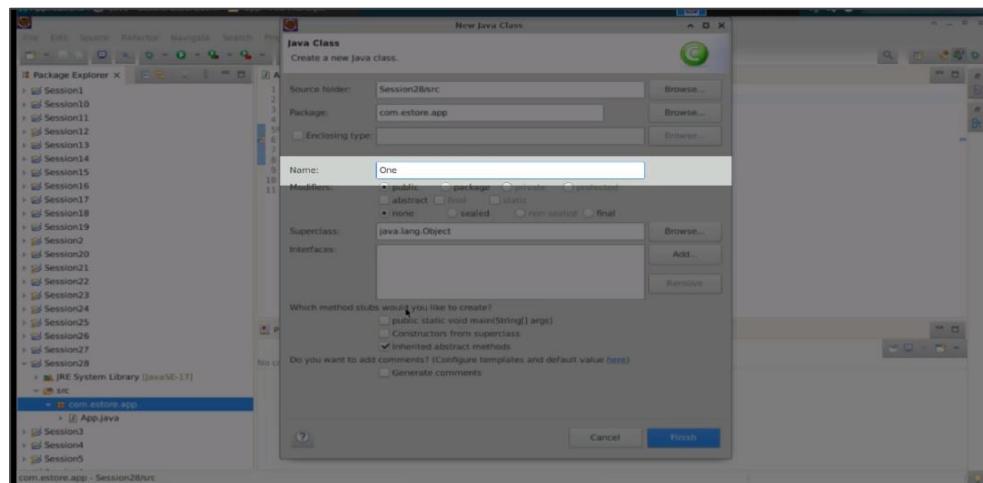
Note:- Classes and interfaces are categorised using Java packages to make maintenance easier. Java packages make the code organized.

## Step 2 Implement the use of access modifiers in class declaration

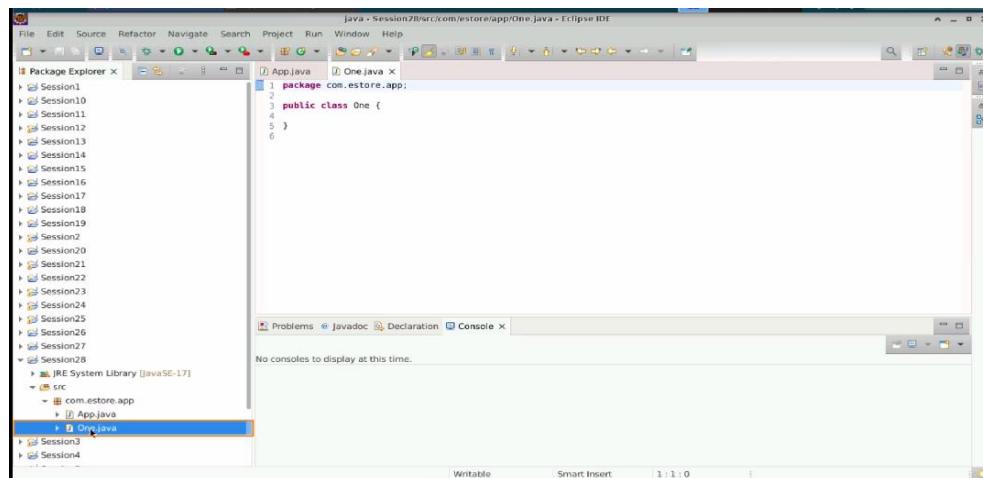
2.1 In the same package, right click, select **New** and then select **Class**.



2.2 Name the class as **One** to create a class without a main method.



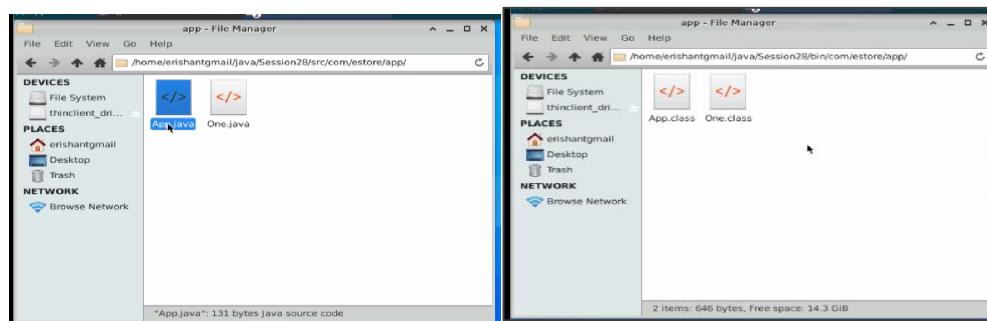
2.3 Thus there is a separate file called **One.java** in the same package.



2.4 You can view the bytecodes by visiting the bin **com.estore.app**.

The bytecodes are in the other window, while the java file is in the first.

This is a straightforward description of what packages are and how to manage bytecodes.



2.5 Class **One** is marked as public. Create another class called **Two**. This class will be a default class or a package level class and not a public class. If you try to put the default keyword in front of the class **Two** declaration, you will get an error because the **default keyword** is meant for interface methods.

```

Java - Session28/src/com/estore/app/One.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X App.java One.java X
Session1 Session10 Session11 Session12 Session13 Session14 Session15 Session16 Session17 Session18 Session19 Session2 Session20 Session21 Session22 Session23 Session24 Session25 Session26 Session27 Session28 JRE System Library [JavaSE-17]
src com.estore.app App.java One.java
Session3 Session4

1 package com.estore.app;
2
3 // public class
4 public class One {
5
6 }
7
8 // default class or package level class
9 default class Two{
10
11 }
12
13

Problems Javadoc Declaration Console X
No consoles to display at this time.

Writable Smart Insert 10 : 1 [8]

```

2.6 If you try to make a class **public or protected** by writing a public or protected keyword in front of the class declaration, in both the cases an error will be raised. Hover over the error and it says that **illegal modifiers and only public, abstract and final are permitted**. This means that class can either be **default or public** but not **private or protected**.

```

Java - Session28/src/com/estore/app/One.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X App.java One.java X
Session1 Session10 Session11 Session12 Session13 Session14 Session15 Session16 Session17 Session18 Session19 Session2 Session20 Session21 Session22 Session23 Session24 Session25 Session26 Session27 Session28 JRE System Library [JavaSE-17]
src com.estore.app App.java One.java
Session3 Session4

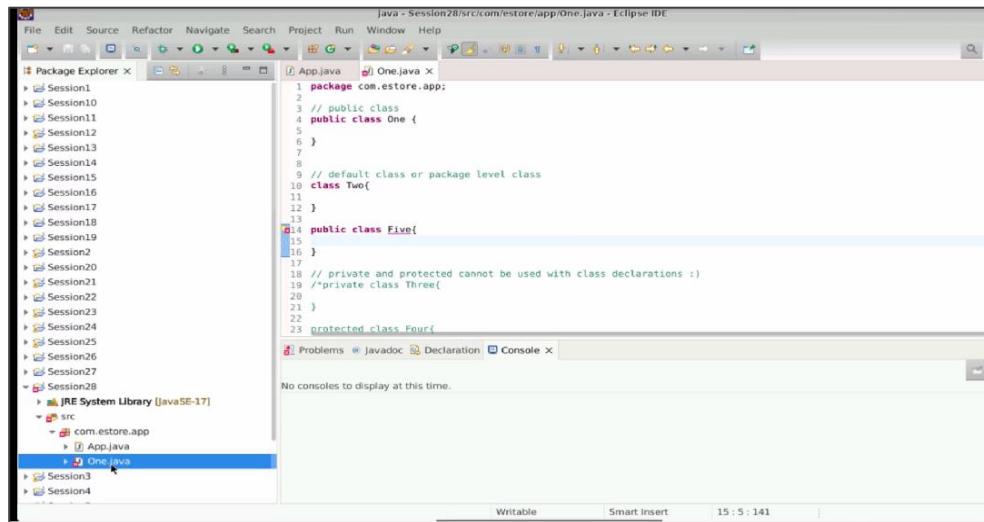
1 package com.estore.app;
2
3 // public class
4 public class One {
5
6 }
7
8 // default class or package level class
9 class Two{
10
11 }
12
13
14 illegal modifier for the class Three; only public, abstract & final are permitted
15
16
17
18 protected class Four{
19
20 }
21

Problems Javadoc Declaration Console X
No consoles to display at this time.

Writable Smart Insert 17 : 1 [146]

```

2.7 Create another public class as **public class five** in the same source file and an error will be raised which means **public class five** must be in its own file.



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the menu is a toolbar with various icons. The central workspace displays two tabs: App.java and One.java. The App.java tab contains a single line of code: "package com.estore.app;". The One.java tab contains the following Java code:

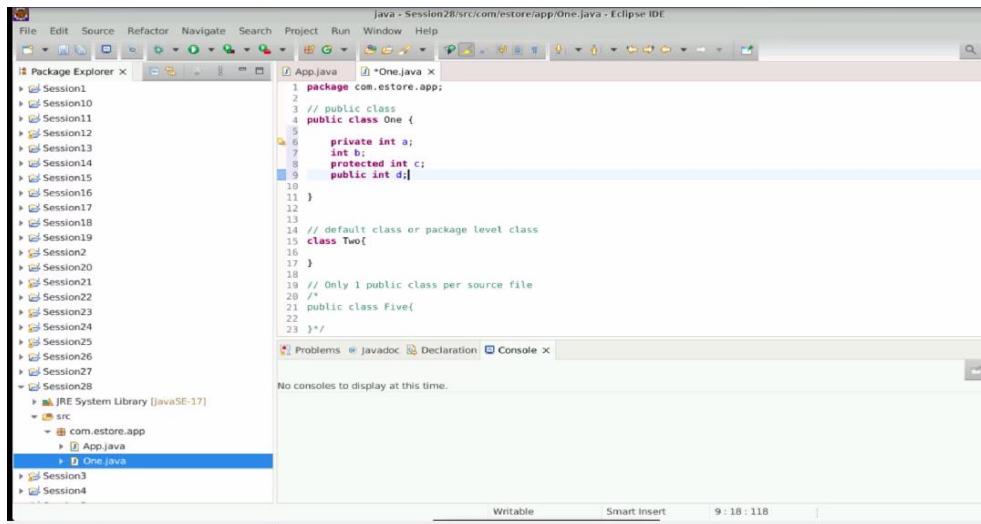
```
1 package com.estore.app;
2
3 // public class
4 public class One {
5
6 }
7
8
9 // default class or package level class
10 class Two{
11
12 }
13
14 public class Five{
15
16 }
17
18 // private and protected cannot be used with class declarations :
19 /*private class Three(
20
21 )
22 */
23 protected class Four{
```

The line "14 public class Five{" is highlighted with a red box, indicating a syntax error. The status bar at the bottom right shows "15 : 5 : 141".

Note:- If a class has been created as public, then the name of the source file should be the same name as that of the class name. In a single source file, there could only be one public class. However, there can be multiple default classes.

## Step 3 Implement the use of access modifiers inside the class

3.1 Inside the class **One**, first define the attributes as **private int A, int B, protected int C and public int D.**



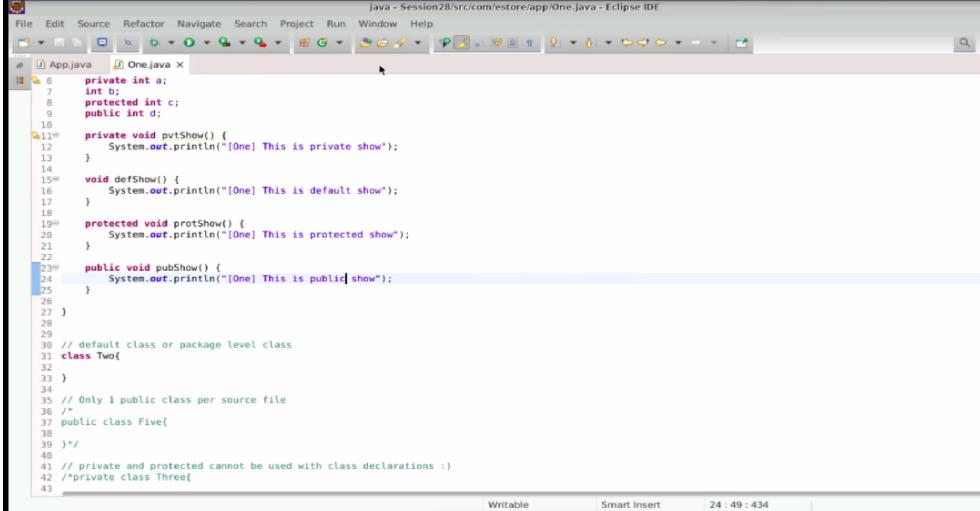
The screenshot shows the Eclipse IDE interface with the following details:

- File Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Package Explorer:** Shows multiple session files (Session1, Session10, Session11, Session12, Session13, Session14, Session15, Session16, Session17, Session18, Session19, Session2, Session20, Session21, Session22, Session23, Session24, Session25, Session26, Session27, Session28) and a project structure under "com.estore.app". The file "One.java" is selected in the Package Explorer.
- Editor Area:** Displays the Java code for class One:

```
1 package com.estore.app;
2
3 // public class
4 public class One {
5
6     private int a;
7     int b;
8     protected int c;
9     public int d;
10
11 }
12
13 // default class or package level class
14 class Two{
15
16 }
17 }
18
19 // Only 1 public class per source file
20 /*
21 public class Five(
22
23 )*/
24
```
- Bottom Status Bar:** Writable, Smart Insert, 9:18:118.

3.2 Now define the methods in class **One**. The methods are:

- **private void pvtShow()** - a method which prints **[One] this is a private method.**
- **void defShow()** - a method which prints **[One] this is a default method.**
- **protected void protShow()** - a method which prints **[One] this is a protected method.**
- **public void pubShow()** - a method which prints **[One] this is a public method.**



The screenshot shows the Eclipse IDE interface with the title bar "java - Session28/src/com/estore/app/One.java - Eclipse IDE". The code editor displays the following Java code:

```
File Edit Source Refactor Navigate Search Project Run Window Help
App.java One.java X
6  private int a;
7  int b;
8  protected int c;
9  public int d;
10
11  private void pvtShow() {
12      System.out.println("[One] This is private show");
13  }
14
15  void defShow() {
16      System.out.println("[One] This is default show");
17  }
18
19  protected void protShow() {
20      System.out.println("[One] This is protected show");
21  }
22
23  public void pubShow() {
24      System.out.println("[One] This is public show");
25  }
26
27 }
28
29 // default class or package level class
30 class Two{
31 }
32
33 }
34
35 // Only 1 public class per source file
36 /**
37 public class Five{
38 }
39 */
40
41 // private and protected cannot be used with class declarations :)
42 /*private class Three{
43 }
```

The code defines a class named **One** with four methods: **pvtShow()**, **defShow()**, **protShow()**, and **pubShow()**. Each method prints a specific message to the console. The code also includes a note about class declarations and a comment for a class named **Three**.

3.3 Similarly, define the methods in class **Two**. The methods are:

- **private void pvtShow()** - a method which prints **[Two]** this is a **private method**.
- **void defShow()** - a method which prints **[Two]** this is a **default method**.
- **protected void protShow()** - a method which prints **[Two]** this is a **protected method**.
- **public void pubShow()** - a method which prints **[Two]** this is a **public method**.

The screenshot shows the Eclipse IDE interface with two open files: **One.java** and **App.java**. The **One.java** file contains the following code:

```

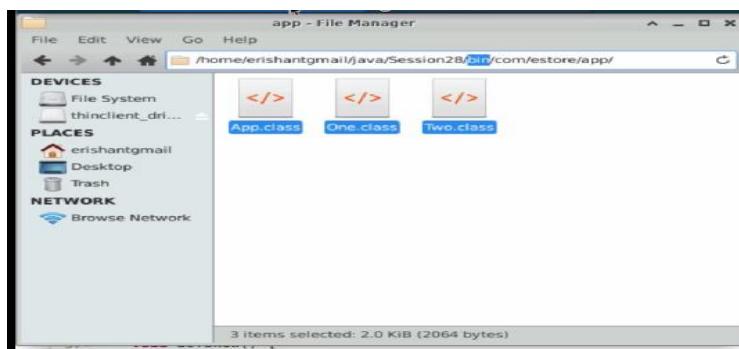
14
15    void defShow() {
16        System.out.println("[One] This is default show");
17    }
18
19    protected void protShow() {
20        System.out.println("[One] This is protected show");
21    }
22
23    public void pubShow() {
24        System.out.println("[One] This is public show");
25    }
26
27 }
28
29
30 // default class or package level class
31 class Two{
32
33    private void pvtShow() {
34        System.out.println("[Two] This is private show");
35    }
36
37    void defShow() {
38        System.out.println("[Two] This is default show");
39    }
40
41    protected void protShow() {
42        System.out.println("[Two] This is protected show");
43    }
44
45    public void pubShow() {
46        System.out.println("[Two] This is public show");
47    }
48
49 }
50
51 // Only 1 public class per source file

```

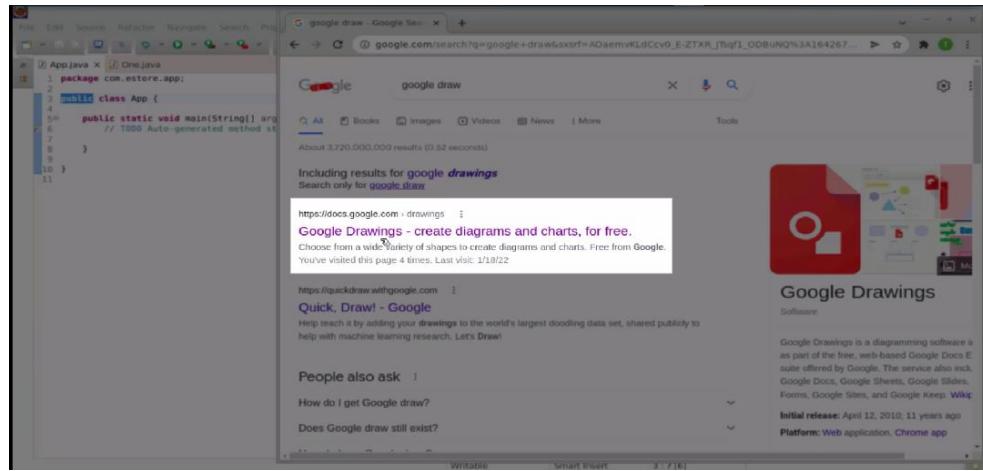
The **App.java** file is currently empty.

3.4 The package bin has three classes.

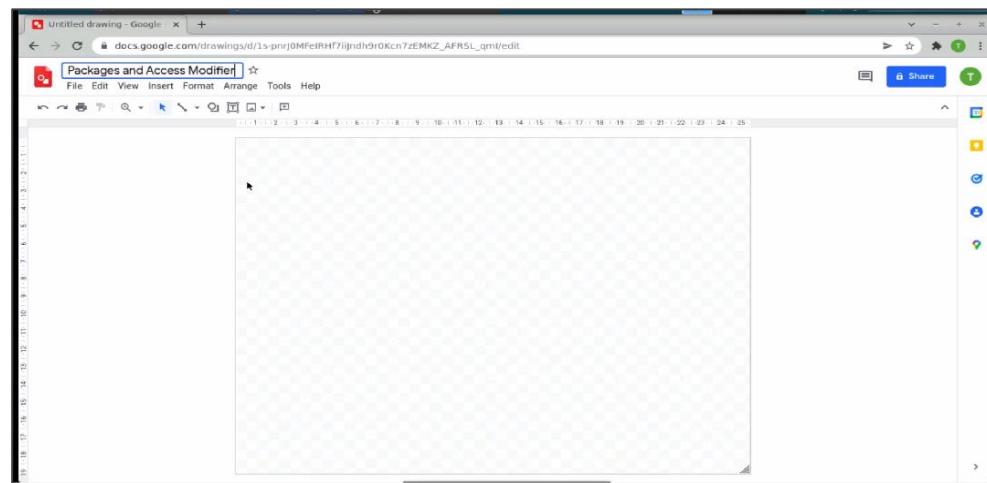
The **app** is **public**, class **one** is **public**, and class **Two** is **default** in the file manager. You can see the same thing in the eclipse IDE; class two is the default, class one is public, and since a file can only contain one public class, this App is also public in the other file under the name **App.Java**.



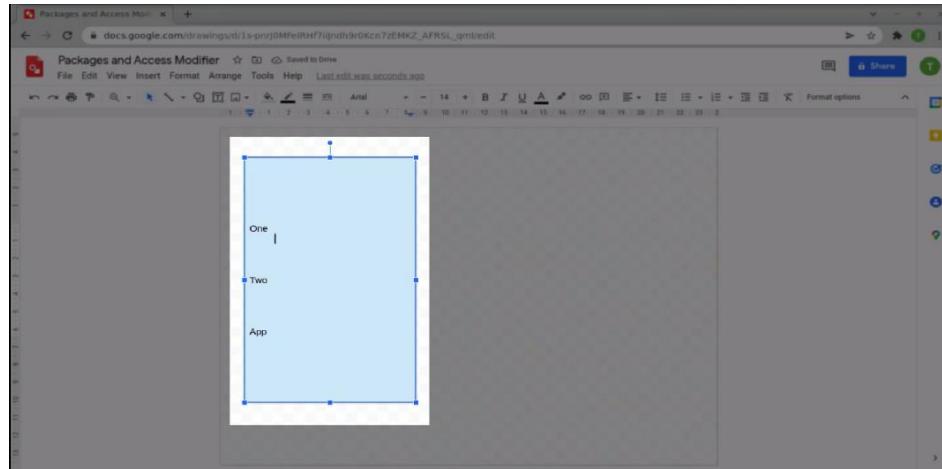
3.5 In your web browser, search **Google Draw** in the search bar and open it.



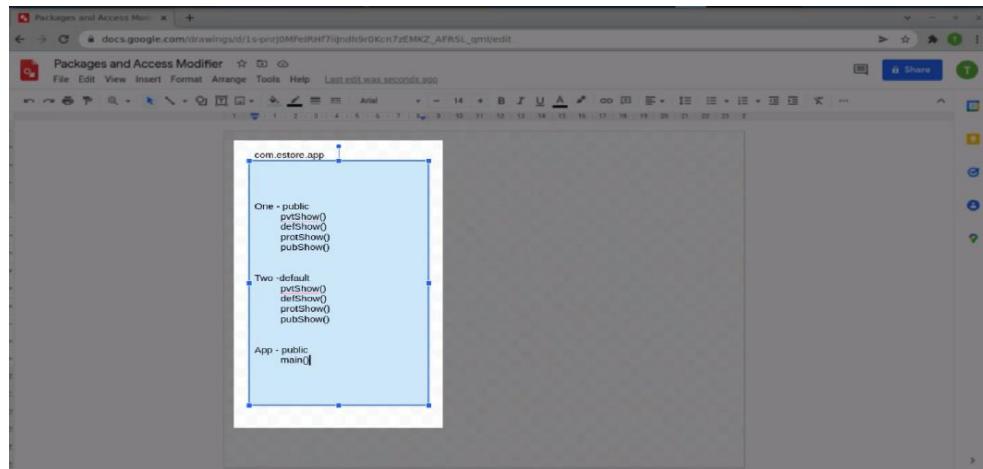
3.6 Name the file as **Packages and Access Modifiers**.



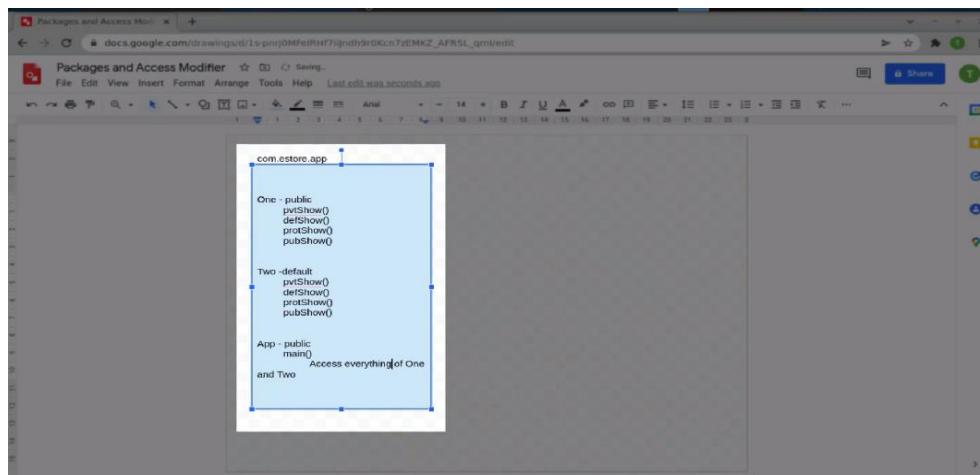
3.8 Draw a square box that will represent the package **com.estore.app**. Inside the package, we have class **One (public)**, class **Two (default)** and class **App (public)**.



3.9 In class One, four methods are there : **pvtShow()**, **defShow()**, **protShow()** and **pubShow()**. The same methods are there in class Two. Class App has a main method.



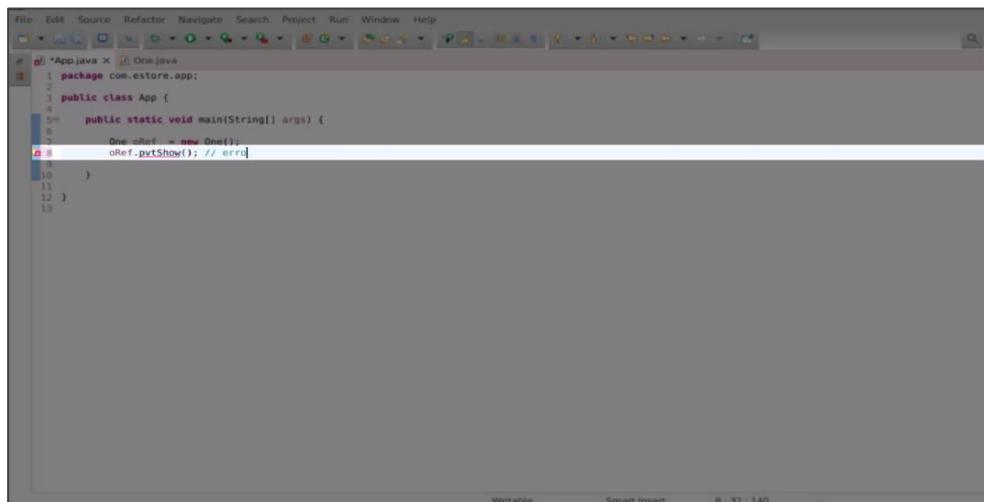
3.10 Inside the main(), everything of **One** and **Two** can be accessed other than **private attributes or methods**.



3.11 In the main() of **App.java**, create an object of **One** as **One oRef = new One();**. This statement creates an object of class One.

```
File Edit Source Refactor Navigate Search Project Run Window Help
*App.java x One.java
1 package com.estore.app;
2
3 public class App {
4
5     public static void main(String[] args) {
6
7         One oRef = new One();
8
9     }
10
11 }
12
```

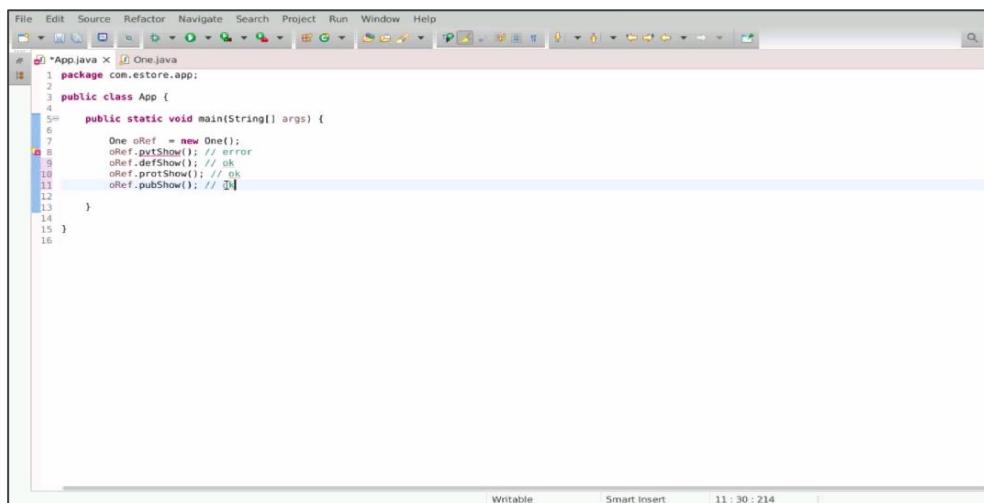
3.12 As soon as we write **oRef.pvtShow()**, an error is raised as we are trying to access a private method. The default show, protected show, and public show can all be accessed via the reference variable respectively. Therefore everything here is acceptable.



A screenshot of an IDE showing a Java code editor. The code is as follows:

```
File Edit Source Refactor Navigate Search Project Run Window Help
# App.java X One.java
1 package com.estore.app;
2
3 public class App {
4
5     public static void main(String[] args) {
6
7         One oRef = new One();
8         oRef.pvtShow(); // error
9
10    }
11
12 }
13
```

The line `oRef.pvtShow();` is highlighted in red, indicating a syntax error. The status bar at the bottom right shows "8 - 32 : 140".



A screenshot of the same IDE showing the Java code after the error was fixed. The code is now:

```
File Edit Source Refactor Navigate Search Project Run Window Help
# App.java X One.java
1 package com.estore.app;
2
3 public class App {
4
5     public static void main(String[] args) {
6
7         One oRef = new One();
8         oRef.pvtShow(); // error
9         oRef.defShow(); // ok
10        oRef.protShow(); // ok
11        oRef.pubShow(); // ok
12
13    }
14
15 }
16
```

The status bar at the bottom right shows "11 : 30 : 214".

3.13 Comment down the private method and run the code. The following output will be obtained.

```

File Edit Source Refactor Navigate Search Project Run Window Help
App.java X Run App (already running)
1 package com.estore.app;
2
3 public class App {
4
5     public static void main(String[] args) {
6
7         One oRef = new One();
8         oRef.defShow(); // error
9         oRef.dfrShow(); // ok
10        oRef.protShow(); // ok
11        oRef.pubShow(); // ok
12
13    }
14
15 }
16

```

[One] This is default show  
[One] This is protected show  
[One] This is public show

3.14 Create an object of class Two as **Two tRef = new Two()** and start accessing the methods. The default show, protected show, and public show can all be accessed via the reference variable but the private show will raise an error.

```

File Edit Source Refactor Navigate Search Project Run Window Help
App.java X One.java
1 package com.estore.app;
2
3 public class App {
4
5     public static void main(String[] args) {
6
7         One oRef = new One();
8         oRef.defShow(); // error
9         oRef.dfrShow(); // ok
10        oRef.protShow(); // ok
11        oRef.pubShow(); // ok
12
13        System.out.println();
14
15        Two tRef = new Two();
16        tRef.pvtShow(); // error
17        tRef.defShow(); // ok
18        tRef.protShow(); // ok
19        tRef.pubShow(); // ok
20
21    }
22
23 }
24

```

Writable Smart Insert 19 - 24 / 346

- 3.15 Comment down the private method and run the code. The following output will be obtained.

```

File Edit Source Refactor Navigate Search Project Run Window Help
App.java X One.java
1 package com.estore.app;
2
3 public class App {
4     public static void main(String[] args) {
5         One oRef = new One();
6         //oRef.pvtShow(); // error
7         oRef.defShow(); // ok
8         oRef.protShow(); // ok
9         oRef.pubShow(); // ok
10
11         System.out.println();
12
13         Two tRef = new Two();
14         //tRef.pvtShow(); // error
15         tRef.defShow(); // ok
16         tRef.protShow(); // ok
17         tRef.pubShow(); // ok
18
19         System.out.println();
20
21     }
22 }
23
24

```

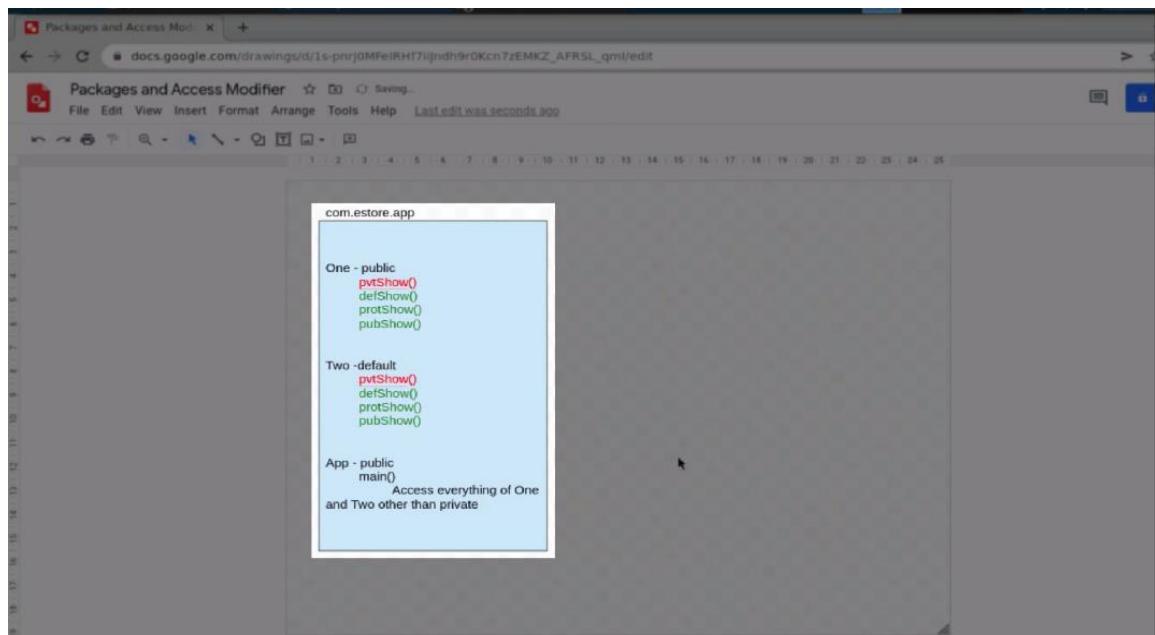
Output in Console:

```

[One] This is default show
[One] This is protected show
[One] This is public show
[Two] This is default show
[Two] This is protected show
[Two] This is public show

```

- 3.16 Private methods and attributes cannot be accessed outside their class. Apart from private, the other three access modifiers (public, protected, and default) allow access outside the class if they are in the same package. Marking an attribute as private restricts its access to within the class, thereby achieving encapsulation. You can then access these private members using methods, or through getters and setters.



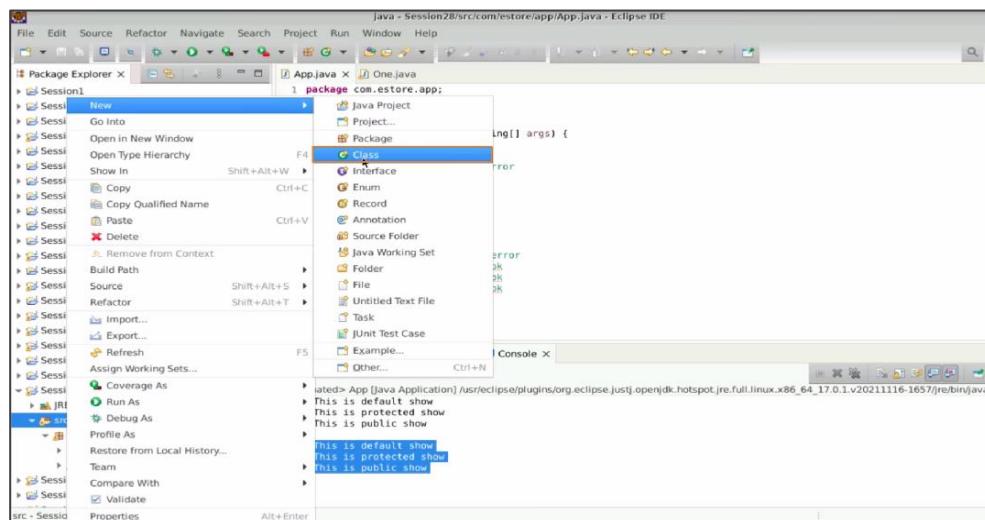
```

1 *App.java X One.java
2 package com.estore.app;
3
4 public class App {
5     public static void main(String[] args) {
6         One oRef = new One();
7         oRef.protShow(); // error
8         oRef.defShow(); // ok
9         oRef.protShow(); // ok
10        oRef.pubShow(); // ok
11
12        System.out.println();
13
14        Two tRef = new Two();
15        tRef.protShow(); // error
16        tRef.defShow(); // ok
17        tRef.protShow(); // ok
18        tRef.pubShow(); // ok
19
20
21    }
22
23 }
24
25 // private: which cannot be accessed outside the class
26

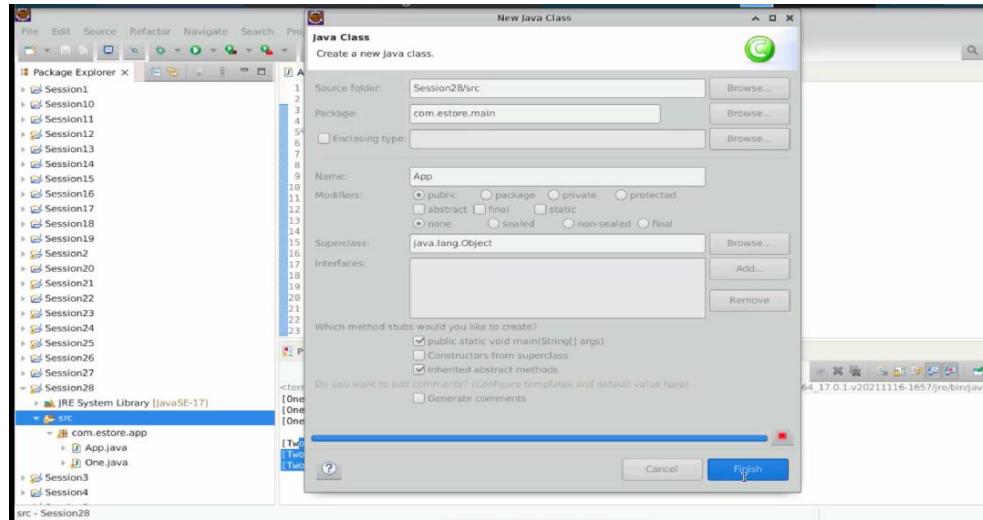
```

## Step 4 : Implement the use of access modifiers outside the class

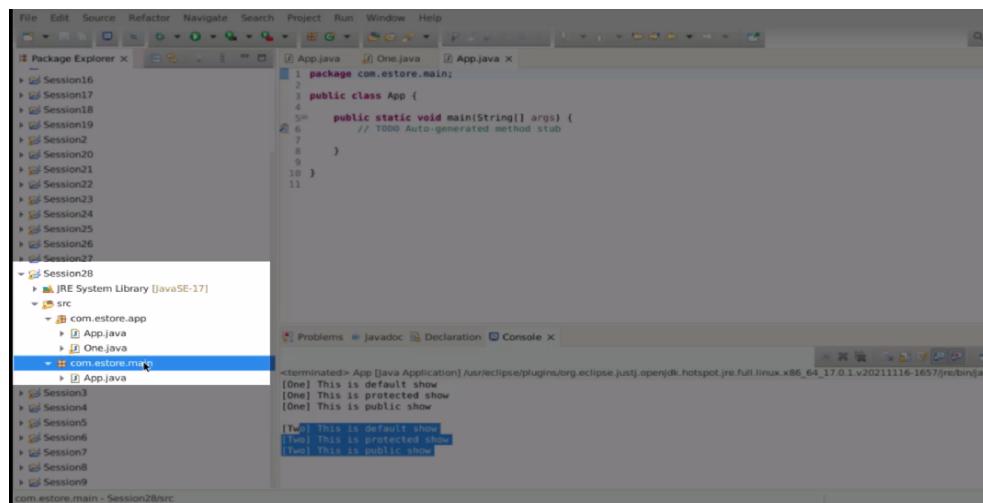
- In the same package, right click, select **New** and then select **Class**.



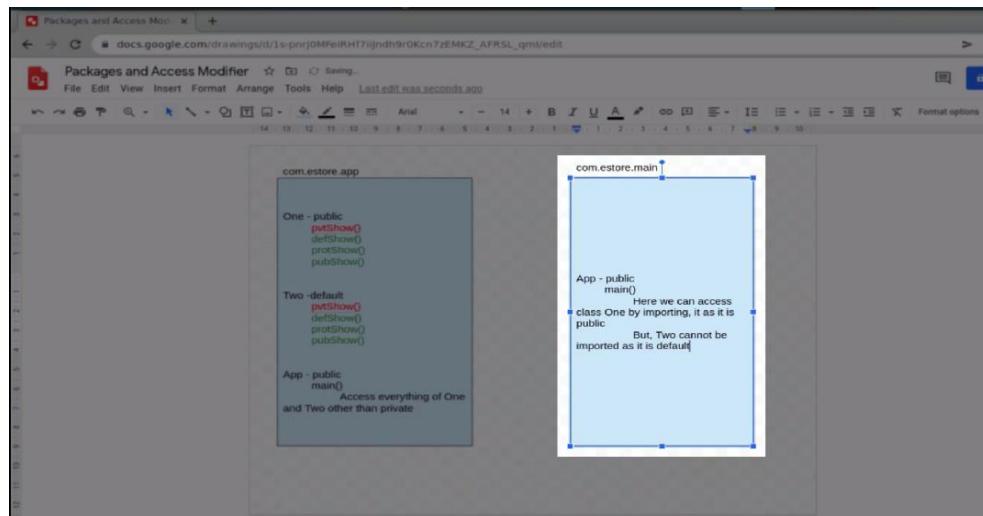
4.2 Name the class as **App** but change the package name to **com.estore.main**. Select **Finish**.



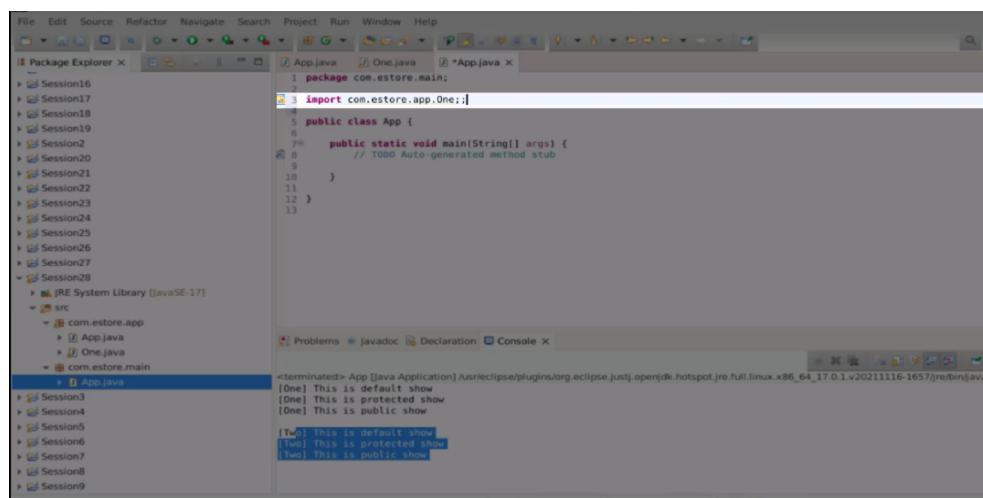
4.3 There can be different packages with the same class name.



- 4.4 In the package **com.estore.main**, a class named **App (public)** has been created with a **main()** method. In the class **App** of this package, class **One** of the other package can be imported as it is **public**. But class **Two** cannot be imported as it is **default**.



- 4.5 Inside the **App** class of **com.estore.main** package, import class **One** by writing **import com.estore.app.One**. This works fine and class **One** has been imported. But as you try to import class **Two** by writing **com.estore.app.Two**, an error is raised as default methods are accessible inside the same package.



The screenshot shows the Eclipse IDE interface. On the left is the Package Explorer view, which lists multiple session files (Session16 through Session28) and a JRE System Library entry for JavaSE-17. In the center is the Java Editor view containing the following code:

```

1 package com.estore.main;
2
3 import com.estore.app.One;
4 import com.estore.app.Two;
5
6 public class App {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10    }
11 }
12
13
14

```

Below the editor is the Problems view, which is currently empty. To the right of the editor is the Console view, which displays the following output:

```

<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java
[One] This is default show
[One] This is protected show
[One] This is public show
[Two] This is default show
[Two] This is protected show
[Two] This is public show

```

4.6 **default** is accessible only within the same package and not outside the package.

The screenshot shows the Eclipse IDE interface. On the left is the Package Explorer view, which lists multiple session files (Session1 through Session21) and a JRE System Library entry for JavaSE-17. In the center is the Java Editor view containing the following code:

```

1 package com.estore.app;
2
3 public class App {
4
5     public static void main(String[] args) {
6
7         One oRef = new One();
8         //oRef.pvtShow(); // error
9         oRef.defShow(); // ok
10        oRef.protShow(); // ok
11        oRef.pubShow(); // ok
12
13        System.out.println();
14
15        Two tRef = new Two();
16        //tRef.pvtShow(); // error
17        tRef.defShow(); // ok
18        tRef.protShow(); // ok
19        tRef.pubShow(); // ok
20
21    }
22
23 }
24
25 // private: which cannot be accessed outside the class. Hence it is visible only inside the class
26 // default: which is accessible only within the same package and not outside the package
27

```

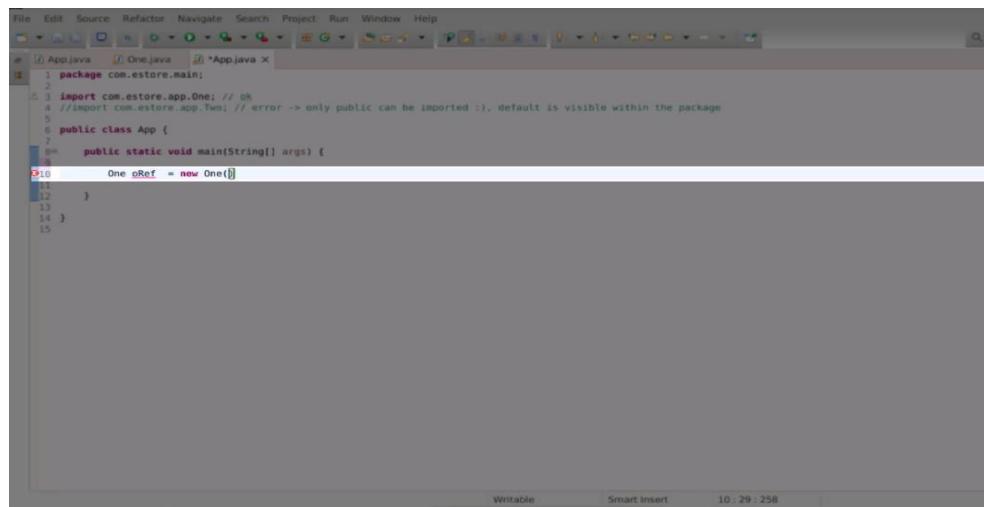
Below the editor is the Problems view, which is currently empty. To the right of the editor is the Console view, which displays the following output:

```

[One] This is default show
[One] This is protected show
[One] This is public show
[Two] This is default show
[Two] This is protected show
[Two] This is public show

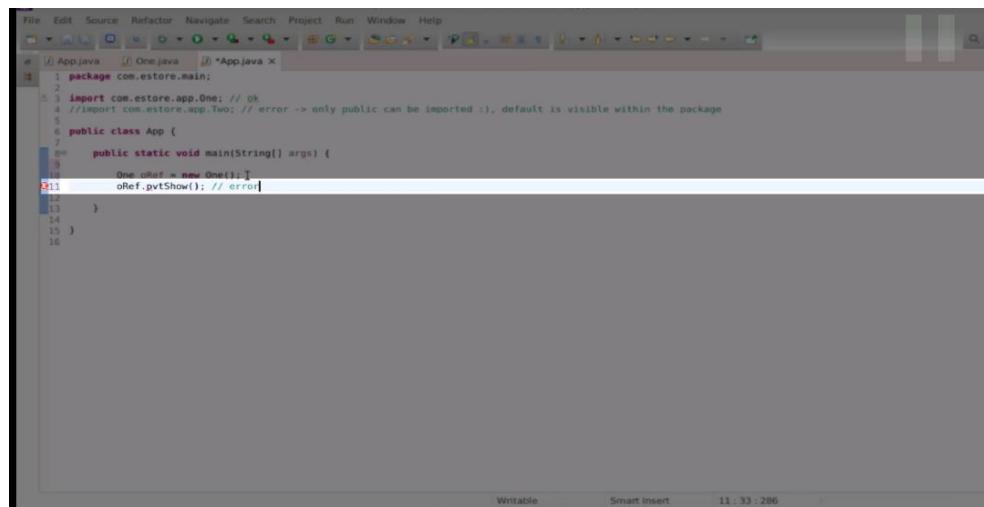
```

4.7 Inside the package **com.estore.main**, create an object of class One as **One oRef = new One()**.



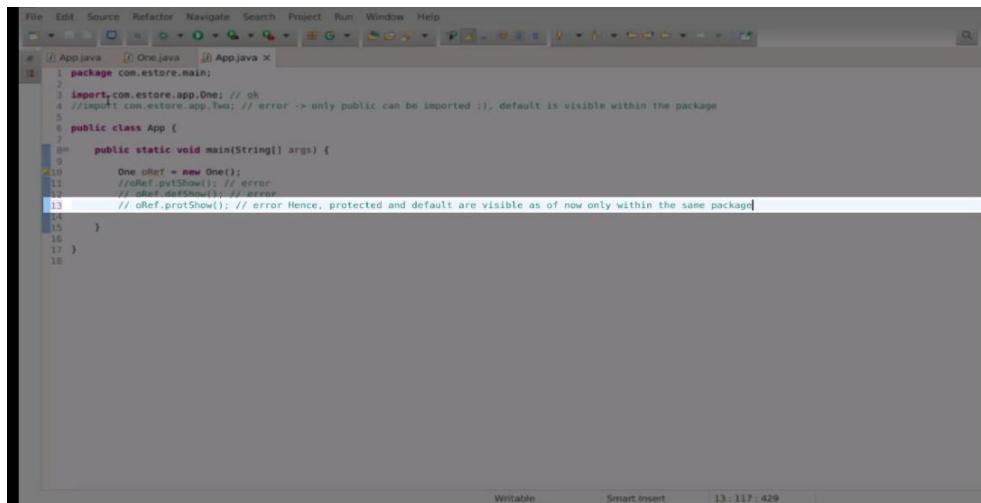
```
File Edit Source Refactor Navigate Search Project Run Window Help
App.java One.java *App.java X
1 package com.estore.main;
2
3 import com.estore.app.One; // ok
4 //import com.estore.app.Two; // error -> only public can be imported :, default is visible within the package
5
6 public class App {
7
8     public static void main(String[] args) {
9
10        One oRef = new One();
11
12    }
13
14 }
15
```

4.8 Write **oRef.pvtShow()** and try to access the private show, it raises an error. This is because private is accessible inside the same class.



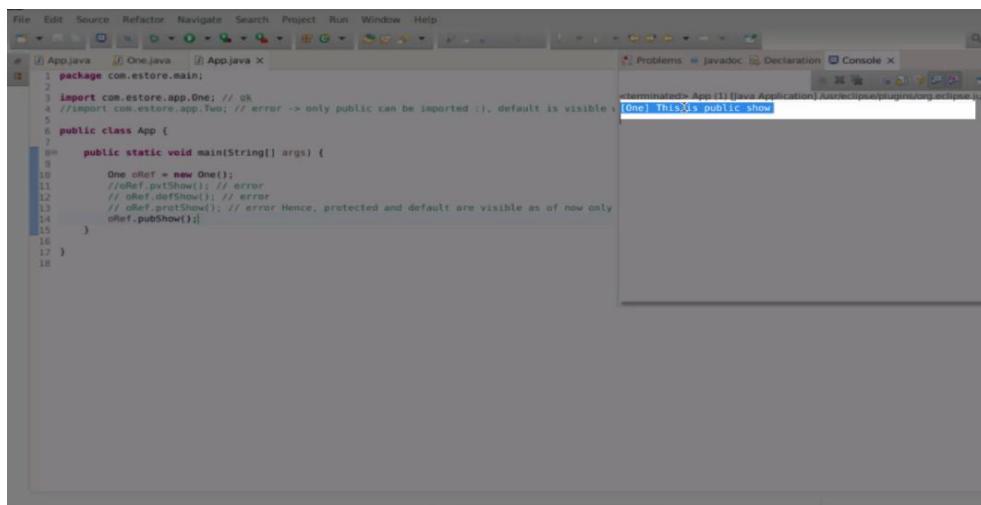
```
File Edit Source Refactor Navigate Search Project Run Window Help
App.java One.java *App.java X
1 package com.estore.main;
2
3 import com.estore.app.One; // ok
4 //import com.estore.app.Two; // error -> only public can be imported :, default is visible within the package
5
6 public class App {
7
8     public static void main(String[] args) {
9
10        One oRef = new One();
11        oRef.pvtShow(); // error
12
13    }
14
15 }
16
```

4.9 Write **oRef.defShow()** and **oRef.protShow()** and try to access the default show and protected show, but it also raises an error. This is because default and protected are accessible inside the same package.



```
File Edit Source Refactor Navigate Search Project Run Window Help
App.java One.java App.java X
1 package com.estore.main;
2
3 import com.estore.app.One; // ok
4 //import com.estore.app.Two; // error -> only public can be imported ;), default is visible within the package
5
6 public class App {
7
8     public static void main(String[] args) {
9         One oRef = new One();
10        //oRef.protShow(); // error
11        // oRef.defShow(); // error
12        // oRef.protShow(); // error Hence, protected and default are visible as of now only within the same package
13        oRef.pubShow(); // ok
14    }
15
16
17 }
18
```

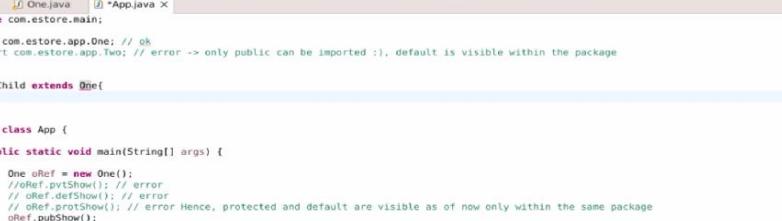
4.10 Write **oRef.pubShow()** and try to access the public show, it works fine. This is because public is accessible everywhere i.e. inside the class or outside the class.



```
File Edit Source Refactor Navigate Search Project Run Window Help
App.java One.java App.java X
1 package com.estore.main;
2
3 import com.estore.app.One; // ok
4 //import com.estore.app.Two; // error -> only public can be imported ;), default is visible within the package
5
6 public class App {
7
8     public static void main(String[] args) {
9         One oRef = new One();
10        //oRef.protShow(); // error
11        // oRef.defShow(); // error
12        // oRef.protShow(); // error Hence, protected and default are visible as of now only
13        oRef.pubShow(); // ok
14    }
15
16
17 }
18
```

## **Step 5 : Implement the difference between protected and default access modifier**

5.1 Inside the package, **com.estore.main**, create a child class of class **One** by writing class **Child extends One**, which means One is the parent class and Child is the child class. This is called **package level inheritance**, where parent and child are in separate packages.



The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Applications - Session28/src/com/... Packages and Access M... Thunar
- Project Explorer:** Shows three open files: App.java, One.java, and App.java.
- Editor:** Displays the contents of App.java. The code includes imports for com.estore.app.One and com.estore.app.Two, a class Child extending One, and a public class App with a main method. It also contains several commented-out lines involving protected and default methods from the One class.
- Toolbars and Status Bar:** Standard Eclipse toolbars and status bar at the bottom indicating Writable, Smart Insert, and file statistics.

```
1 package com.estore.main;
2
3 import com.estore.app.One; // ok
4 //import com.estore.app.Two; // error -> only public can be imported :, default is visible within the package
5
6
7 class Child extends One{
8 }
9
10
11 public class App {
12
13     public static void main(String[] args) {
14
15         One oRef = new One();
16         //oRef.protShow(); // error
17         // oRef.defShow(); // error
18         // oRef protShow(); // error Hence, protected and default are visible as of now only within the same package
19         oRef.pubShow();
20     }
21
22 }
23
```

5.2 Inside the child class, create a method **void show()**, Inside this method, try to access the four methods of class **One**. **pvtShow()** and **defShow()** will raise an error as they cannot be accessed outside the class. But **protShow()** and **pubShow()** will work fine. This implies that **protected** is accessible from the child class outside the package.

The screenshot shows an IDE interface with a toolbar at the top and a code editor below. The code editor displays Java code with several annotations:

```
File Edit Source Refactor Navigate Search Project Run Window Help
App.java One.java App.java x
1 package com.estore.main;
2
3 import com.estore.app.One; // ok
4 //import com.estore.app.Two; // error -> only public can be imported :, default is visible within the package
5
6
7 //Package Level Inheritance
8 // Parent and Child are in separate packages
9 class Child extends One{
10
11    void show() {
12        defShow();
13        protShow();
14        pubShow();
15    }
16 }
17
18 }
19
20 public class App {
21
22     public static void main(String[] args) {
23
24         One oRef = new One();
25         //oRef.pvtShow(); // error
26         //oRef.defShow(); // error
27         //oRef.protShow(); // error Hence, protected and default are visible as of now only within the same package
28         oRef.pubShow();
29     }
30
31 }
32
```

The annotations are placed near specific code elements:

- Annotations 1-4:** Located above the first four lines of code. Annotations 1 and 2 are green, while 3 and 4 are red.
- Annotation 11:** Located above the first line of the `show()` method.
- Annotations 12-15:** Located inside the `show()` method body.
- Annotations 24-28:** Located inside the `main()` method body.

### 5.3 Accessibility from low to high is as follows:

```

29
30 // Accessibility|
31 // LEAST MOST
32 // private > default > protected > public
33

```

### 5.4 Create the object of the child class as **Child ch = new Child()** and call the show() method of the child class.

```

File Edit Source Refactor Navigate Search Project Run Window Help
App.java One.java *App.java X
1 package com.estore.main;
2
3 import com.estore.app.One; // ok
4 //import com.estore.app.Two; // error -> only public can be imported ;), default is visible within the package
5
6
7 //Package Level Inheritance
8 // Parent and Child are in separate packages
9 class Child extends One{
10
11    void show() {
12        //privShow(); // error
13        //defShow(); // error
14        protShow(); // ok -> accessible form the child class and not outside
15        pubShow(); // ok
16    }
17
18 }
19
20 public class App {
21
22    public static void main(String[] args) {
23
24        //One oRef = new One();
25        //oRef.privShow(); // error
26        //oRef.defShow(); // error
27        //oRef.protShow(); // error Hence, protected and default are visible as of now only within the same package
28        //oRef.pubShow(); // ok
29
30        Child ch = new Child();
31
32    }
33
34 }
35

```

Writable Smart Insert 30 : 31 : 750

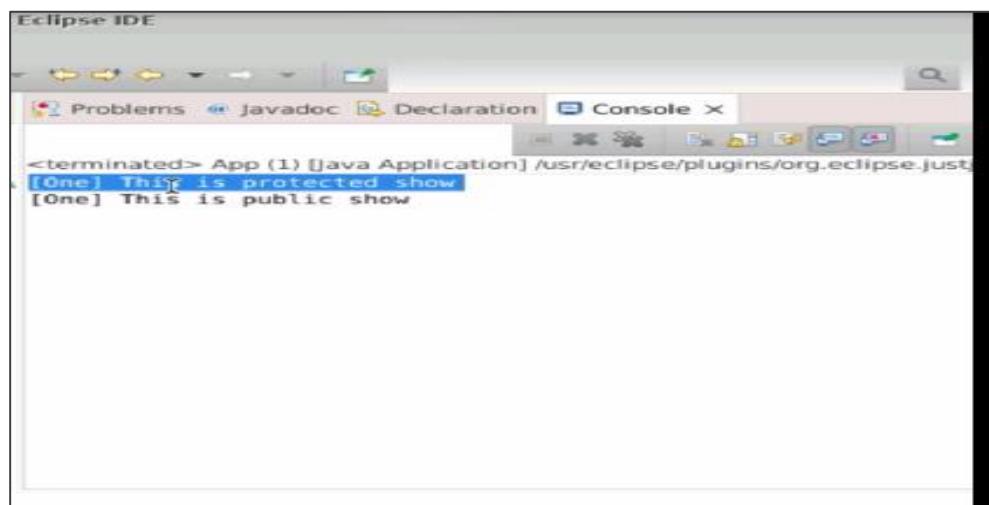
```

File Edit Source Refactor Navigate Search Project Run Window Help
App.java One.java *App.java X
1 package com.estore.main;
2
3 import com.estore.app.One; // ok
4 //import com.estore.app.Two; // error -> only public can be imported ;), default is visible within the package
5
6
7 //Package Level Inheritance
8 // Parent and Child are in separate packages
9 class Child extends One{
10
11    void show() {
12        //privShow(); // error
13        //defShow(); // error
14        protShow(); // ok -> accessible form the child class and not outside
15        pubShow(); // ok
16    }
17
18 }
19
20 public class App {
21
22    public static void main(String[] args) {
23
24        //One oRef = new One();
25        //oRef.privShow(); // error
26        //oRef.defShow(); // error
27        //oRef.protShow(); // error Hence, protected and default are visible as of now only within the same package
28        //oRef.pubShow(); // ok
29
30        Child ch = new Child();
31        ch.show();
32
33    }
34
35 }
36

```

Writable Smart Insert 31 : 19 : 764

5.5 Run the code and the following output will be obtained:



The screenshot shows the Eclipse IDE interface with the title bar "Eclipse IDE". Below the title bar is a toolbar with various icons. The main window contains a tab bar with "Problems", "Javadoc", "Declaration", and "Console X". The "Console X" tab is selected, showing the output of a Java application. The output text is:  
<terminated> App (1) [Java Application] /usr/eclipse/plugins/org.eclipse.justj.runtime/jre/lib/jce.jar  
[One] This is protected show  
[One] This is public show

By following the above steps, you have successfully implemented packages and access modifiers in Java, ensuring organized code structure and controlled accessibility.