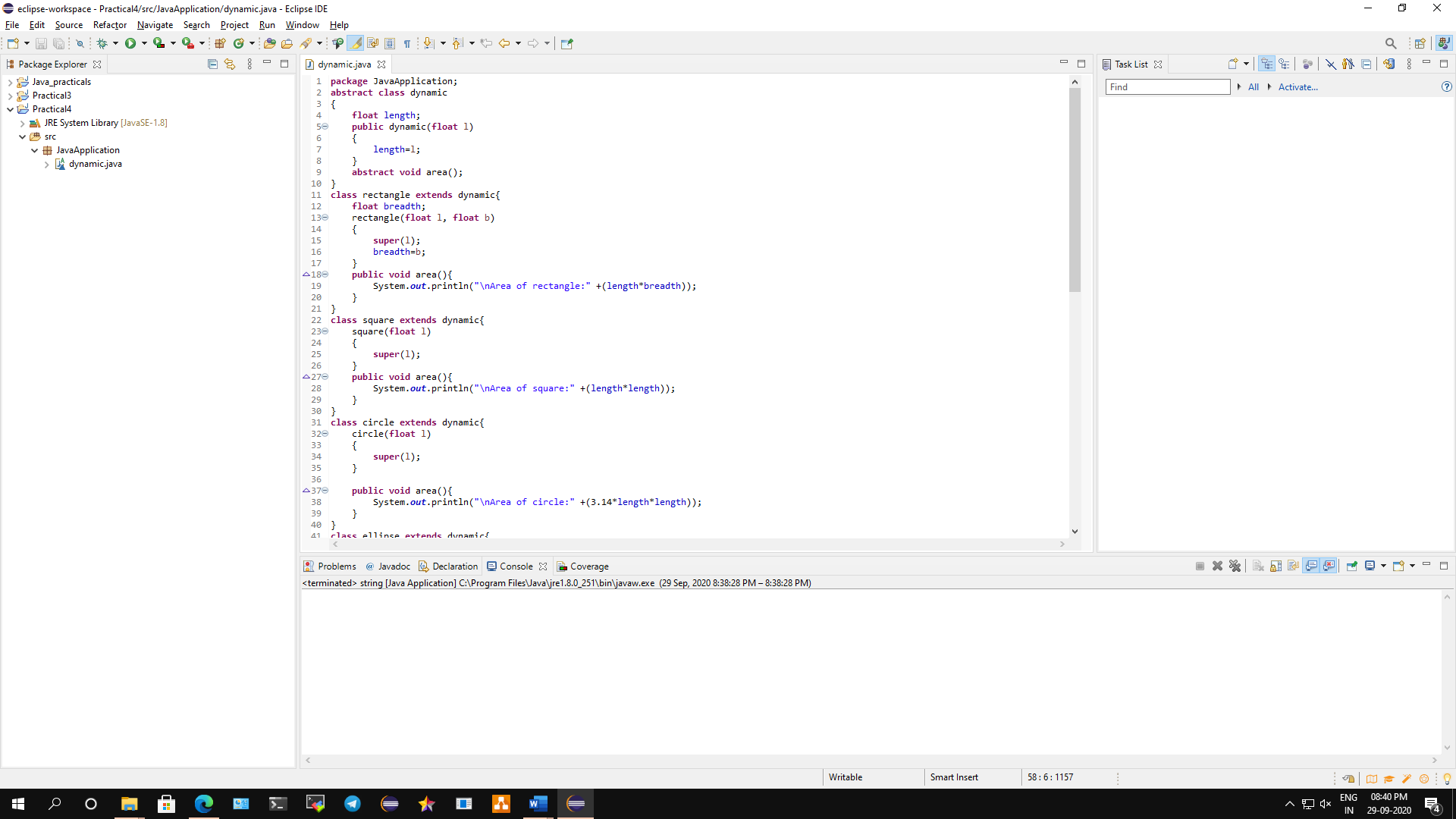
**Practical No 4**

**A) Program to implement Dynamic Method Dispatch in Java using Abstract Class To find the area of various shape: Rectangle, Circle, Ellipse, Square and Triangle.**

**Aim: Write a program to implement Dynamic Method Dispatch in Java using Abstract Class.**

**Description:**

Method overriding is one of the ways in which Java supports Runtime Polymorphism. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time. In this program we have defined package as JavaApplication and abstract class as dynamic. Then we use float value to find the various shape value. We use length as l to define in class. Class rectangle is extended with dynamic class, then we used void area to get the value of rectangle same goes for all other shape value. After all the values we have defined abstract class in which the main function start to get the desired output.



**Conclusion: We have written a program to implement Dynamic Method Dispatch in Java using Abstract Class.**

**Code:**

**package** JavaApplication;

**abstract** **class** dynamic

{

**float** length;

**public** dynamic(**float** l)

{

length=l;

}

**abstract** **void** area();

}

**class** rectangle **extends** dynamic{

**float** breadth;

rectangle(**float** l, **float** b)

{

**super**(l);

breadth=b;

}

**public** **void** area(){

System.***out***.println("\nArea of rectangle:" +(length\*breadth));

}

}

**class** square **extends** dynamic{

square(**float** l)

{

**super**(l);

}

**public** **void** area(){

System.***out***.println("\nArea of square:" +(length\*length));

}

}

**class** circle **extends** dynamic{

circle(**float** l)

{

**super**(l);

}

**public** **void** area(){

System.***out***.println("\nArea of circle:" +(3.14\*length\*length));

}

}

**class** ellipse **extends** dynamic{

**float** b;

ellipse(**float** l, **float** b)

{

**super**(l);

**this**.b=b;

}

**public** **void** area(){

System.***out***.println("\nArea of ellipse:" +(3.14\*length\*b));

}

}

**class** triangle **extends** dynamic{

**float** h;

triangle(**float** l, **float** h)

{

**super**(l);

**this**.h=h;

}

**public** **void** area()

{

System.***out***.println("\nArea of triangle:" +(length\*h/2));

}

}

**class** abstract\_dispatch

{

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

{ rectangle obj=**new** rectangle(10.1F,20.2F);

square objsquare=**new** square(40.5F);

circle objcircle=**new** circle(15.5F);

ellipse objellipse=**new** ellipse(55F,16.4F);

triangle objtriangle=**new** triangle(80,44.4F);

dynamic ref;

ref=obj;

ref.area();

ref=objsquare;

ref.area();

ref=objcircle;

ref.area();

ref=objellipse;

ref.area();

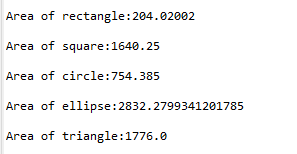
ref=objtriangle;

ref.area();

}

}

**Output:**

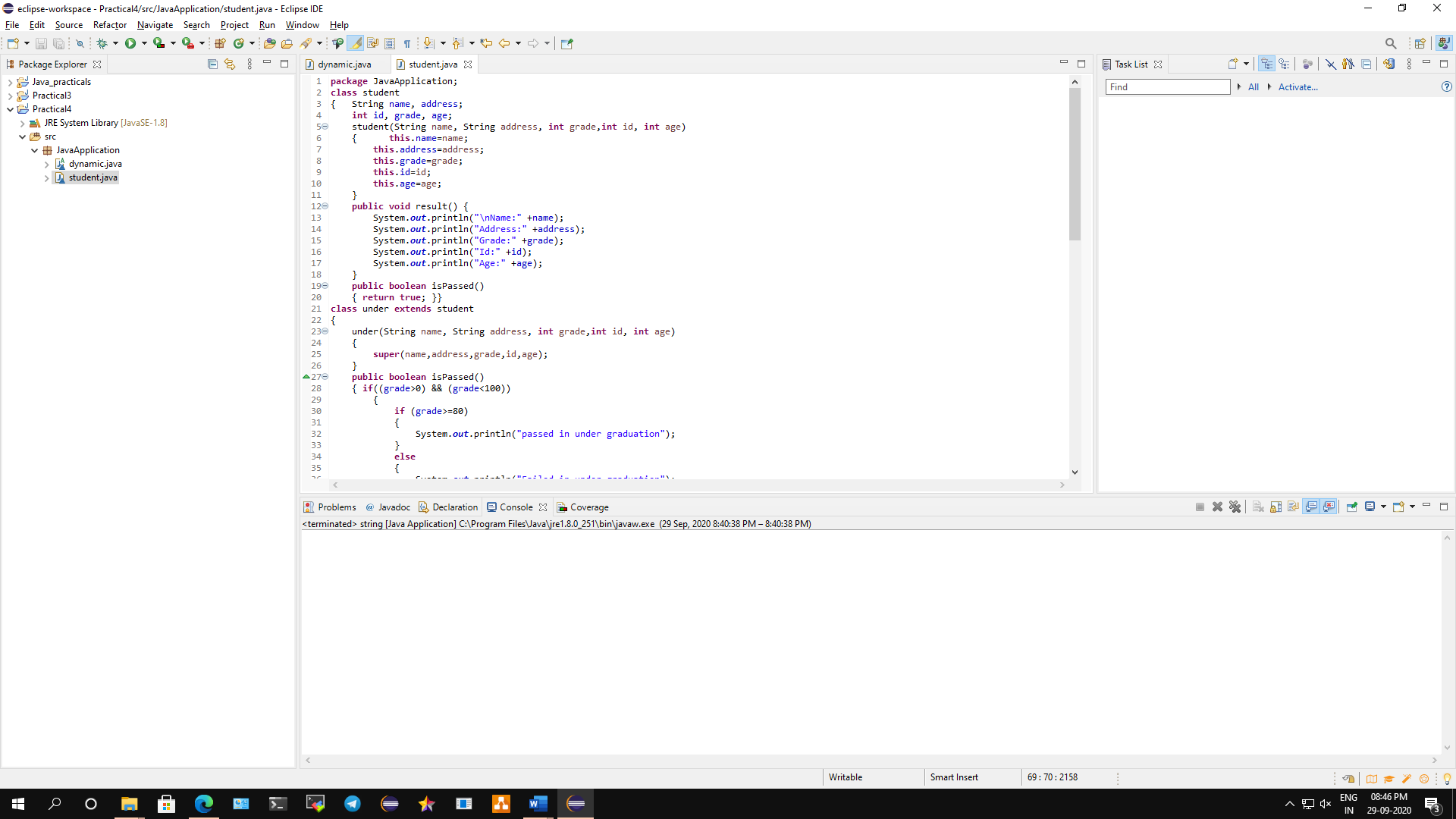


**B) Create Super Class Student and two subclasses of it, Graduate and Under Graduate. The members of the Student are name, id, grade, age and address and at least one method: boolean method IsPassed which takes in the parameter integer grade (0-100). The two sub classes override the method, for UG its 70% for passing and for G its 80% as passing grade.**

**Aim: Write a java program to Create Super Class Student and two subclasses of it, Graduate and Under Graduate.**

**Description:**

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable. Super can be used to refer immediate parent class instance variable. Super can be used to invoke immediate parent class method. super() can be used to invoke immediate parent class constructor. In this program we have to create the super class of student with two subclasses for graduate and under graduate. We have defined class student followed by string name and address. Then we used the integer int which defines id, grade and age. Then void result will display the result output followed by extended class which defined as passing grade and other extended class has failed grade. The output will display pass if student score more than 75% else it will fail if less than 75%.



**Conclusion: We have written a program to Create Super Class Student and two subclasses of it, Graduate and Under Graduate.**

**Code:**

**package** JavaApplication;

**class** student

{ String name, address;

**int** id, grade, age;

student(String name, String address, **int** grade,**int** id, **int** age)

{ **this**.name=name;

**this**.address=address;

**this**.grade=grade;

**this**.id=id;

**this**.age=age;

}

**public** **void** result() {

System.***out***.println("\nName:" +name);

System.***out***.println("Address:" +address);

System.***out***.println("Grade:" +grade);

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

System.***out***.println("Age:" +age);

}

**public** **boolean** isPassed()

{ **return** **true**; }}

**class** under **extends** student

{

under(String name, String address, **int** grade,**int** id, **int** age)

{

**super**(name,address,grade,id,age);

}

**public** **boolean** isPassed()

{ **if**((grade>0) && (grade<100))

{

**if** (grade>=80)

{

System.***out***.println("passed in under graduation");

}

**else**

{

System.***out***.println("Failed in under graduation");

System.*exit*(0);

}

}

**else**

{

System.***out***.println("Grade should be between 0-100");

}**return** **true**; }

**public** **void** result()

{ System.***out***.println("\n\nName:" +name);

System.***out***.println("Address:" +address);

System.***out***.println("Grade:" +grade);

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

System.***out***.println("Age:" +age); }}

**class** graduate **extends** under

{

graduate(String name, String address, **int** grade,**int** id, **int** age)

{

**super**(name,address,grade,id,age);

}

**public** **boolean** isPassed() {

**if**((grade>0) && (grade<100))

{ **if** (grade>=70){

System.***out***.println("passed in under graduation");

}

**else**

{

System.***out***.println("Failed in under graduation");

} }

**else**

{

System.***out***.println("Grade should be between 0-100");

} **return** **true**; }

**public** **void** result() { System.***out***.println("\n\nName:" +name);

System.***out***.println("Address:" +address);

System.***out***.println("Grade:" +grade);

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

System.***out***.println("Age:" +age);}}

**class** student1

{

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

{

student objstud=**new** student("Jon","Chembur",90,33,22);

student s2= **new** under("Kiev","Kurla",84,12,23);

student s1= **new** graduate("Mukul","Bandra",85,56,22);

objstud.result();

s2.result();

s2.isPassed();

s1.result();

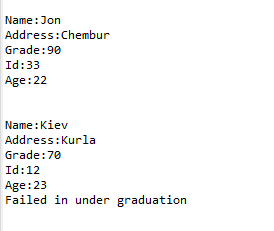
s1.isPassed();

}

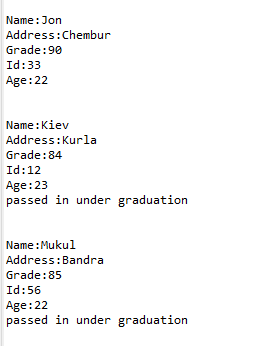
}

**Output:**

**Failed in graduate**



**Passed in under graduate**



**C) Sample Program to demonstrate Access Specifier in Packages.**

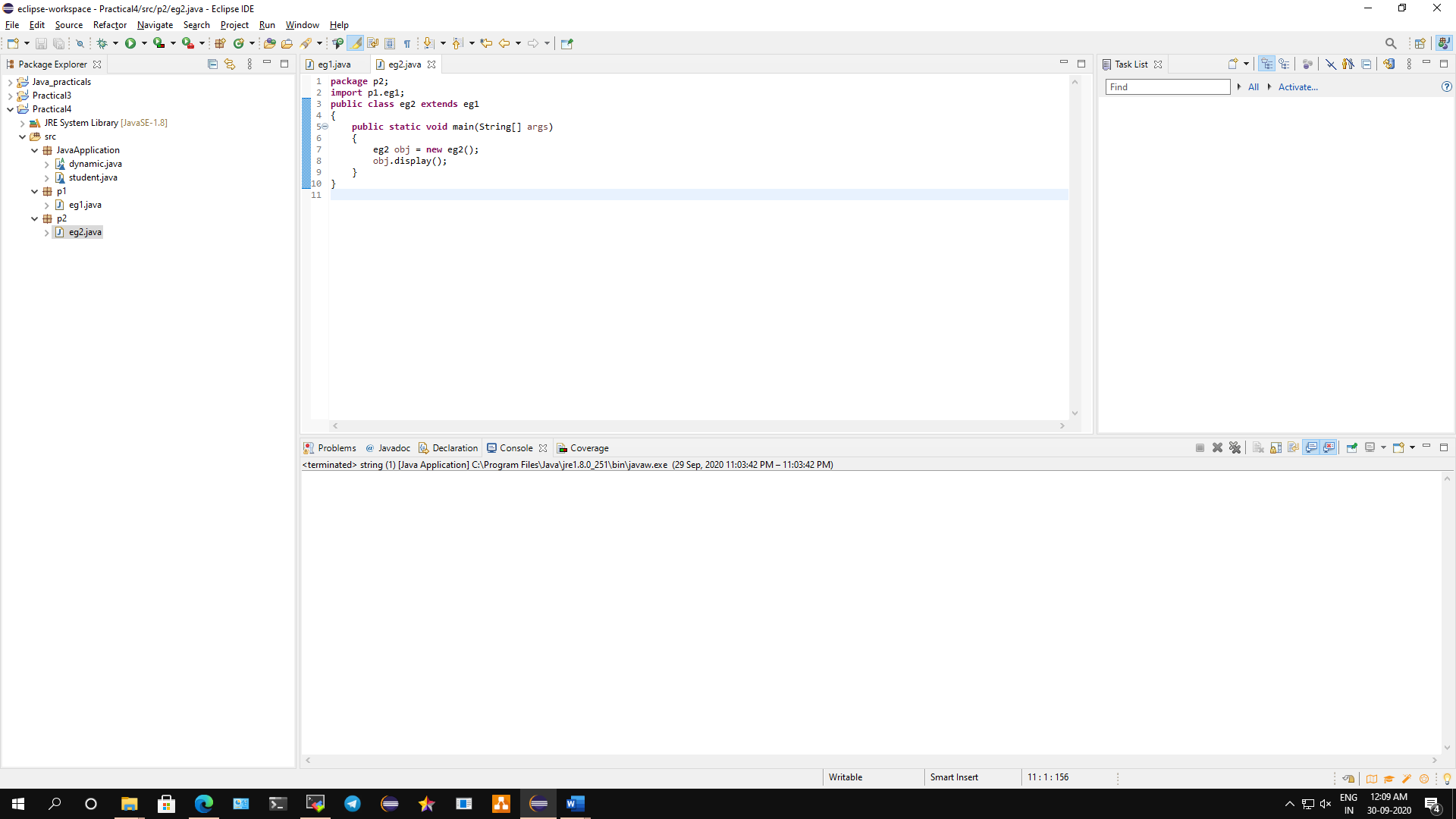
**Aim: Write a sample java program to demonstrate Access Specifier in Packages.**

**Description:**

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.



**Conclusion: We have demonstrated sample program on Access Specifier in Packages.**

**A) Protected Modifier**

**Code:**

**EG1.java**

**package** p1;

**public** **class** eg1

{

**protected** **void** display()

{

System.***out***.println("Welcome to java programming");

}

}

**EG2.java**

**package** p2;

**import** p1.eg1;

**public** **class** eg2 **extends** eg1

{

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

{

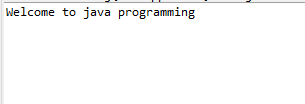
eg2 obj = **new** eg2();

obj.display();

}

}

**Output:**



**B) Public Modifier**

**Code:**

**EG1.java**

**package** p1;

**public** **class** eg1

{

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

{

System.***out***.println("Public access modifier is used");

}

}

**EG2.java**

**package** p2;

**import** p1.eg1;

**public** **class** eg2 **extends** eg1

{

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

{

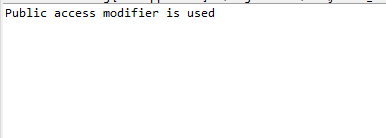
eg2 obj = **new** eg2();

obj.display();

}

}

**Output:**



**C) Private Modifier**

**Code:**

**EG1.java**

**package** p1;

**public** **class** eg1

{

**private** **void** display()

{

System.***out***.println("Private access modifier is used");

}

}

**EG2.java**

**package** p2;

**import** p1.eg1;

**public** **class** eg2 **extends** eg1

{

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

{

eg2 obj = **new** eg2();

obj.display();

}

}

**Output:**

