Differentiate the following

1. Method and Constructor.

In Java, methods and constructors are essential components of classes, serving different purposes. Here’s an overview of both:

Methods

- **Definition**: A method is a block of code that performs a specific task and can be called upon to execute that task.
- **Syntax**:

```
returnType methodName(parameterType parameterName) {

    // method body

    // return statement (if applicable)

}
```

- **Characteristics**:
    - **Return Type**: Specifies the type of value the method will return. If no value is returned, the return type is void.
    - **Name**: Should be descriptive of the task it performs.
    - **Parameters**: Methods can take parameters (inputs), which are defined in parentheses.

    - **Access Modifiers**: Methods can have access modifiers (e.g., public, private, protected), which control visibility.
- **Example**:

```
public int add(int a, int b) {
    return a + b;
}
```

Constructors

- **Definition**: A constructor is a special method that is called when an object of a class is created. It initializes the object.
- **Syntax**:

```
ClassName(parameters) {
    // constructor body
}
```

- **Characteristics**:
    - **Same Name as Class**: Constructors have the same name as the class they belong to.
    - **No Return Type**: Constructors do not have a return type, not even void.
    - **Overloading**: You can have multiple constructors in a class with different parameter lists (constructor overloading).

   - **Default Constructor**: If no constructor is defined, Java provides a default constructor (no-argument constructor).
- **Example**:

```
public class MyClass {
    private int value;


    // Constructor
    public MyClass(int value) {
        this.value = value;
    }
}
```

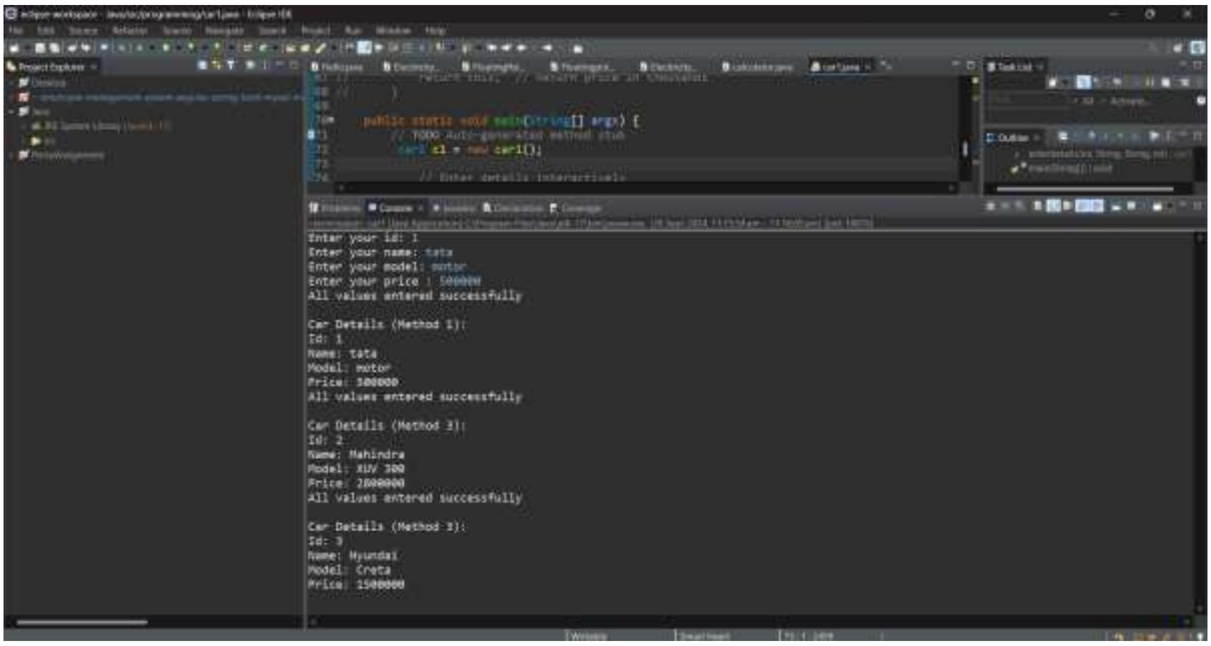
Key Differences

- **Purpose**: Methods perform actions; constructors initialize objects.
- **Return Type**: Methods have a return type; constructors do not.
- **Invocation**: Methods can be called anytime; constructors are called when an object is instantiated.

```
package programming;

import java.util.Scanner;

public class car1 {
     int id;
     String name;
     String model;
     long price;
     void enterdetails() {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter your id: ");
        id = input.nextInt();

        System.out.print("Enter your name: ");
        name = input.next();

        System.out.print("Enter your model: ");
        model = input.next();

        System.out.print("Enter your price : ");
        price = input.nextLong();
        System.out.println("All values entered successfully");
        System.out.println("\nCar Details (Method 1):");
        System.out.println("Id: " + id);
        System.out.println("Name: " + name);
        System.out.println("Model: " + model);
        System.out.println("Price: " + price);
    }
      void enterdetails(int x, String y, String z, long p) {
              id = x;
              name = y;
              model = z;
              price = p;
              System.out.println("All values entered successfully");
              System.out.println("\nCar Details (Method 2):");
              System.out.println("Id: " + id);
              System.out.println("Name: " + name);
              System.out.println("Model: " + model);
              System.out.println("Price: " + price);
          }

          car1 enterdetails(int id, String name, String model, int
price) {
              this.id = id;
              this.name = name;
              this.model = model;
              this.price = price;
              System.out.println("All values entered successfully");
              System.out.println("\nCar Details (Method 3):");
              System.out.println("Id: " + this.id);
              System.out.println("Name: " + this.name);
              System.out.println("Model: " + this.model);
```

```
                System.out.println("Price: " + this.price);
                return this;
            }
//          car1 enterdetails() {
//              this.id = 4;
//              this.name = "Maruti";
//              this.model = "Suzuki";
//              this.price = 3000000;
//              System.out.println("\nCar Details (Method 4):");
//              System.out.println("Id: " + this.id);
//              System.out.println("Name: " + this.name);
//              System.out.println("Model: " + this.model);
//              System.out.println("Price: " + this.price);
//              return this;  // Return price in thousands
//          }

        public static void main(String[] args) {
              // TODO Auto-generated method stub
               car1 c1 = new car1();

                // Enter details interactively
                c1.enterdetails();

                // Enter details via method
                car1 c2 = new car1();
                c2.enterdetails(2, "Mahindra", "XUV 300", 2800000);

                car1 c3 = new car1();
                // Calculate and return result from enterdetails3
                car1 resultMult = c3.enterdetails(3, "Hyundai",
"Creta", 1500000);
//              car1 c4 = new car1();
//              c4.enterdetails();
        }

}
```

Output:-



```
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        car1 c1 = new car1();

            // Enter details interactively
```

```
Enter your id: 1
Enter your name: tata
Enter your model: motor
Enter your price : 500000
All values entered successfully

Car Details (Method 1):
Id: 1
Name: tata
Model: motor
Price: 300000
All values entered successfully

Car Details (Method 2):
Id: 2
Name: Mahindra
Model: XUV 300
Price: 2000000
All values entered successfully

Car Details (Method 3):
Id: 3
Name: Hyundai
Model: Creta
Price: 1500000
```

2. Pass by Value and Pass by Reference.

1. Pass by Value:

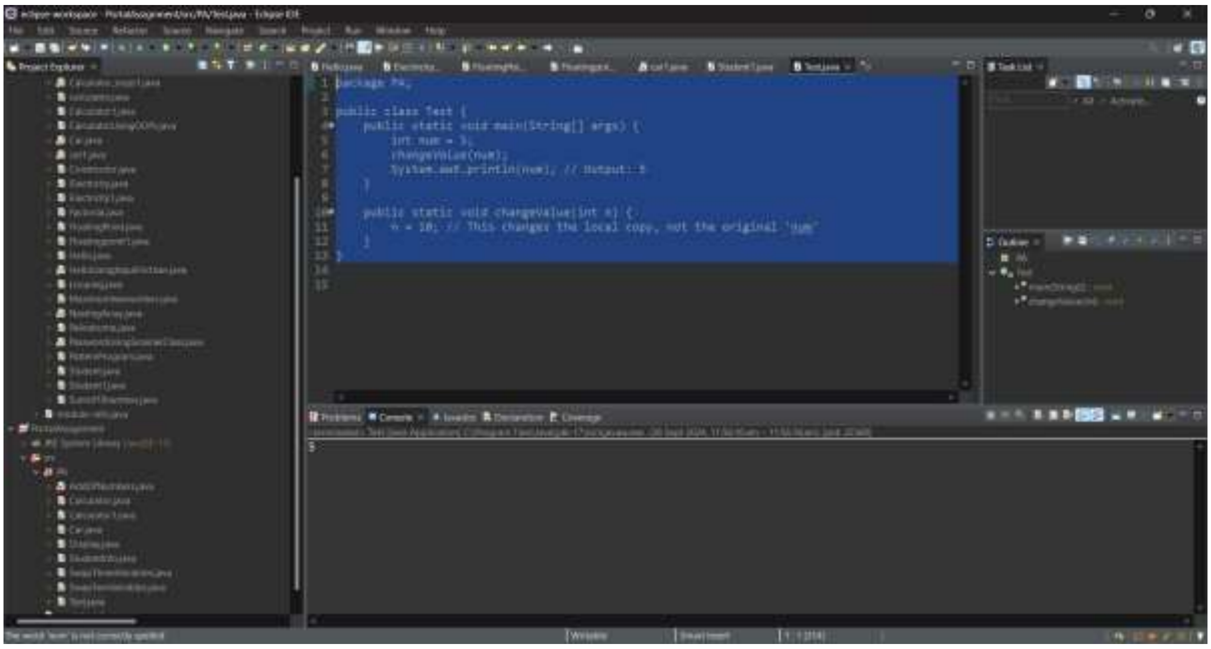
In Java, all arguments are passed by value, meaning that **a copy of the argument's value** is passed to the method. This means changes to the parameter within the method do not affect the original argument outside the method.

- **Primitive data types** (e.g., int, float, char) are passed by value. When you pass a primitive type, a copy of the actual value is passed.

Example:

```java
package PA;

public class Test {
    public static void main(String[] args) {
        int num = 5;
        changeValue(num);
        System.out.println(num); // Output: 5
    }

    public static void changeValue(int n) {
        n = 10; // This changes the local copy, not the original
'num'
    }
}
```

Output:-



```
package PA;

public class Test {
    public static void main(String[] args) {
        int num = 5;
        changeValue(num);
        System.out.println(num); // Output: 5
    }

    public static void changeValue(int n) {
        n = 10; // This changes the local copy, not the original 'num'
    }
}
```

```
5
```

2. Pass by Reference (Not directly supported in Java):

Java **does not use pass by reference** in the traditional sense. However, **objects** (non-primitive types) are passed by value as well, but the value passed is the reference to the object, not the object itself. This means changes to the object's state within the method affect the original object because both the method parameter and the original argument refer to the same object.

Example:

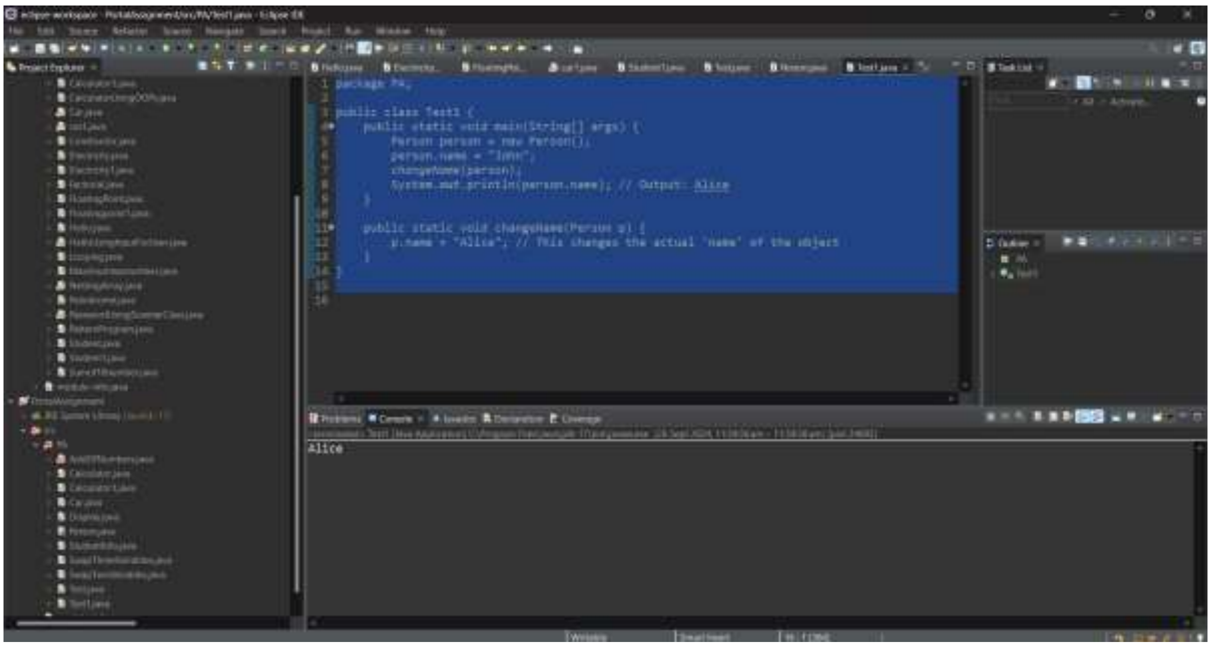
Save this as Person.java.

```
package PA;

class Person {
    String name;
}
```

Save this as Test.java.

```java
package PA;

public class Test1 {
    public static void main(String[] args) {
        Person person = new Person();
        person.name = "John";
        changeName(person);
        System.out.println(person.name); // Output: Alice
    }

    public static void changeName(Person p) {
        p.name = "Alice"; // This changes the actual 'name' of the
object
    }
}
```

Output:-



```java
package PA;

public class Test1 {
    public static void main(String[] args) {
        Person person = new Person();
        person.name = "John";
        changeName(person);
        System.out.println(person.name); // Output: Alice
    }

    public static void changeName(Person p) {
        p.name = "Alice"; // This changes the actual 'name' of the object
    }
}
```

```
Alice
```

3. Public,Private and Protected.

In Java, the access modifiers **public**, **private**, and **protected** control the visibility of classes, methods, and variables. These modifiers define the level of access other classes have to a class or its members.

1. Public

- **Access Level**: **Accessible everywhere**.
- A public class, method, or variable can be accessed from any other class, regardless of whether the classes are in the same package or not.

Example:

```
public class Person {
    public String name;
    public void display() {
        System.out.println("Name: " + name);
    }
}
```

- The Person class, its name field, and display() method can be accessed from any other class.

2. Private

- **Access Level**: **Accessible only within the same class**.
- A private method or variable is not visible outside the class in which it is declared. This ensures **encapsulation**, where internal details of a class are hidden from other classes.

Example:

```
public class Person {

    private String name;


    private void display() {

        System.out.println("Name: " + name);

    }

}
```

- The name field and display() method are not accessible from outside the Person class. Only the Person class itself can use these members.

3. Protected

- **Access Level**: **Accessible within the same package** and **by subclasses** (even if they are in different packages).
- A protected member can be accessed by any class in the same package and by subclasses, even if those subclasses are in a different package. This is commonly used when you want to allow some level of access to subclasses but still maintain a degree of encapsulation.

Example:

```
public class Person {

    protected String name;
    protected void display() {
        System.out.println("Name: " + name);
    }
}
```

- The name field and display() method are accessible from any subclass or any class in the same package, but not from classes in other packages unless they inherit from Person.

Summary of Access Levels:

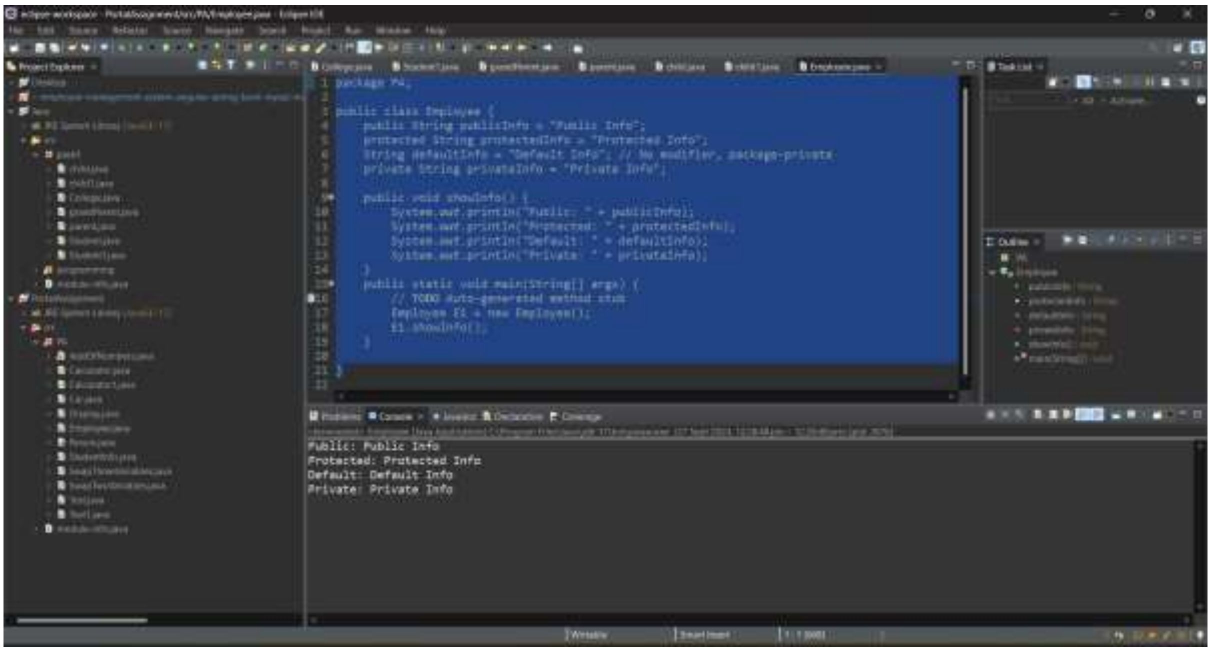
Modifier	**Class**	**Package**	**Subclass**	**World (Other Packages)**
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
no modifier (default)	Yes	Yes	No	No
private	Yes	No	No	No

Example:

// Class in same package

```
package PA;

public class Employee {
      public String publicInfo = "Public Info";
    protected String protectedInfo = "Protected Info";
    String defaultInfo = "Default Info"; // No modifier, package-
private
    private String privateInfo = "Private Info";

    public void showInfo() {
        System.out.println("Public: " + publicInfo);
        System.out.println("Protected: " + protectedInfo);
        System.out.println("Default: " + defaultInfo);
        System.out.println("Private: " + privateInfo);
    }
      public static void main(String[] args) {
            // TODO Auto-generated method stub
            Employee E1 = new Employee();
            E1.showInfo();
      }

}
```

Output:-



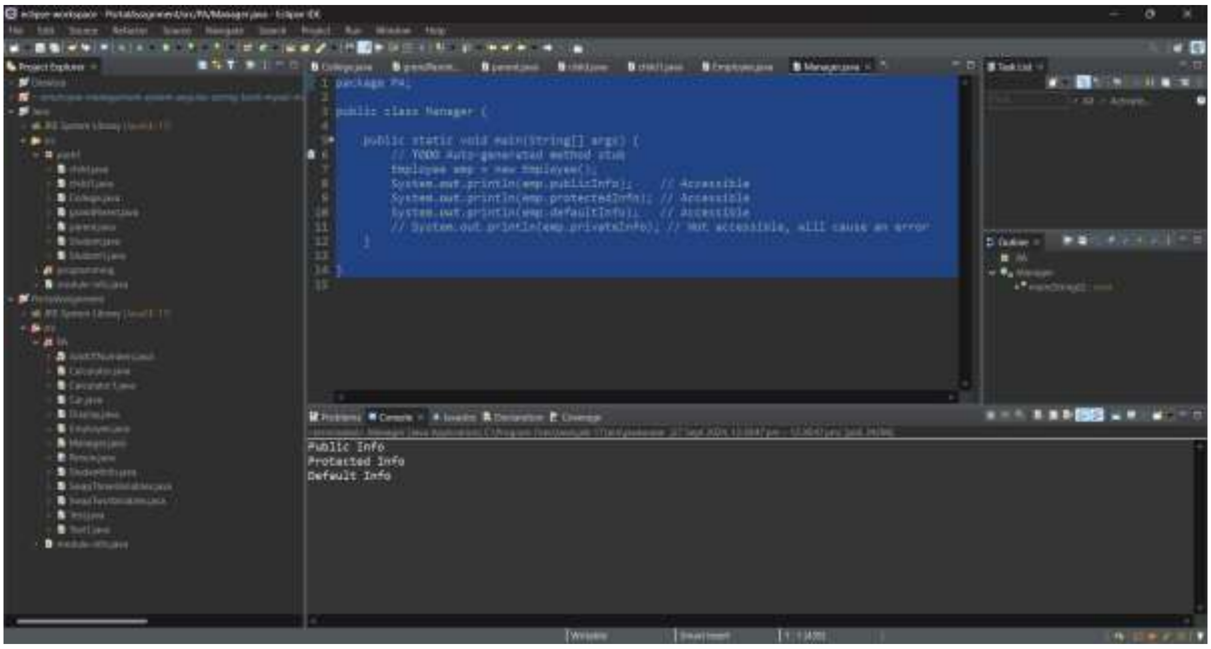
```
package PA;

public class Employee {
    public String publicInfo = "Public Info";
    protected String protectedInfo = "Protected Info";
    String defaultInfo = "Default Info"; // No modifier, package-private
    private String privateInfo = "Private Info";

    public void showInfo() {
        System.out.println("Public: " + publicInfo);
        System.out.println("Protected: " + protectedInfo);
        System.out.println("Default: " + defaultInfo);
        System.out.println("Private: " + privateInfo);
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Employee E1 = new Employee();
        E1.showInfo();
    }

}
```

```
Public: Public Info
Protected: Protected Info
Default: Default Info
Private: Private Info
```

In another class from the same package:

```
package PA;

public class Manager {

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            Employee emp = new Employee();
        System.out.println(emp.publicInfo);     // Accessible
        System.out.println(emp.protectedInfo); // Accessible
        System.out.println(emp.defaultInfo);   // Accessible
        // System.out.println(emp.privateInfo); // Not accessible,
will cause an error
     }

}
```

Output:-



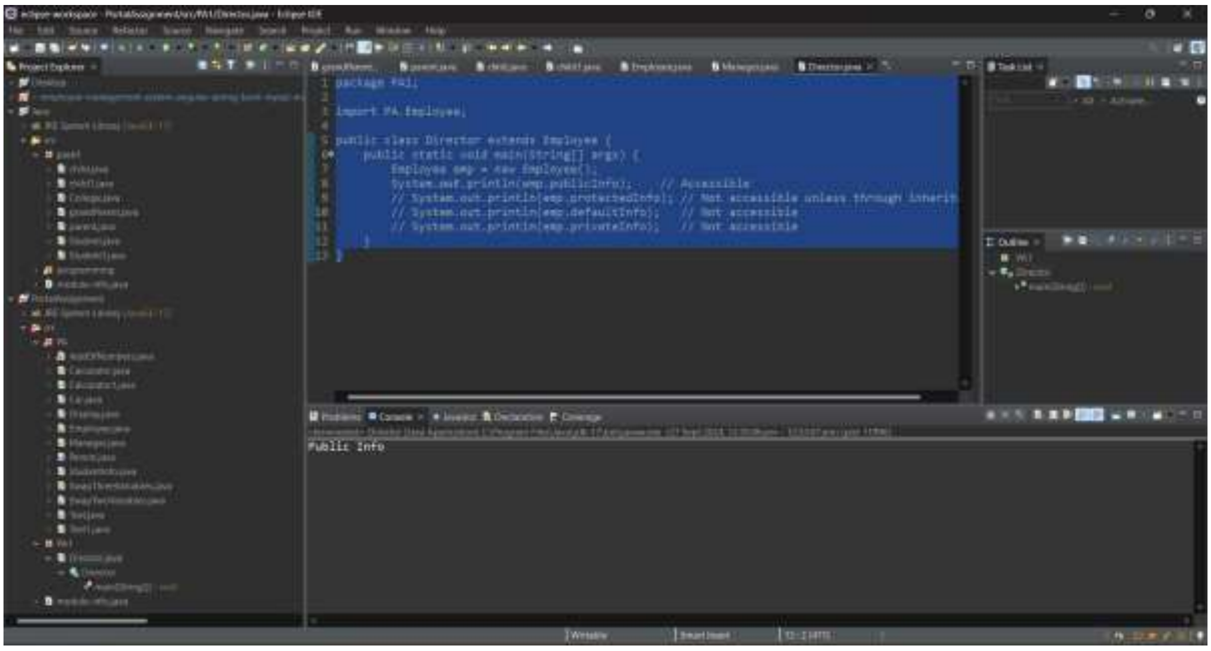
```
package PA;

public class Manager {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Employee emp = new Employee();
        System.out.println(emp.publicInfo);    // Accessible
        System.out.println(emp.protectedInfo); // Accessible
        System.out.println(emp.defaultInfo);   // Accessible
        // System.out.println(emp.privateInfo); // Not accessible, will cause an error
    }

}
```

```
Public Info
Protected Info
Default Info
```

In another class from a different package:

```
package PA1;

import PA.Employee;

public class Director extends Employee {
    public static void main(String[] args) {
        Employee emp = new Employee();
        System.out.println(emp.publicInfo);    // Accessible
        // System.out.println(emp.protectedInfo); // Not accessible
unless through inheritance
        // System.out.println(emp.defaultInfo);   // Not accessible
        // System.out.println(emp.privateInfo);   // Not accessible
    }
}
```

Output:-



```java
package PA1;

import PA.Employee;

public class Director extends Employee {
    public static void main(String[] args) {
        Employee emp = new Employee();
        System.out.println(emp.publicInfo);    // Accessible
        // System.out.println(emp.protectedInfo); // Not accessible unless through inherit
        // System.out.println(emp.defaultInfo);   // Not accessible
        // System.out.println(emp.privateInfo);   // Not accessible
    }
}
```

```
Public Info
```

4. Instance variable and Static Variable.

In Java, **instance variables** and **static variables** (also known as class variables) differ in how they are associated with instances of a class and how they are stored and accessed.

1. Instance Variables:

- **Definition**: These variables are defined inside a class but outside any method, constructor, or block. Each instance (or object) of the class has its own copy of the instance variable.
- **Scope**: Instance variables are unique to each instance of the class. If you create multiple objects, each will have its own separate copy of the instance variable.
- **Access**: They are accessed via object references.

2. Static Variables:

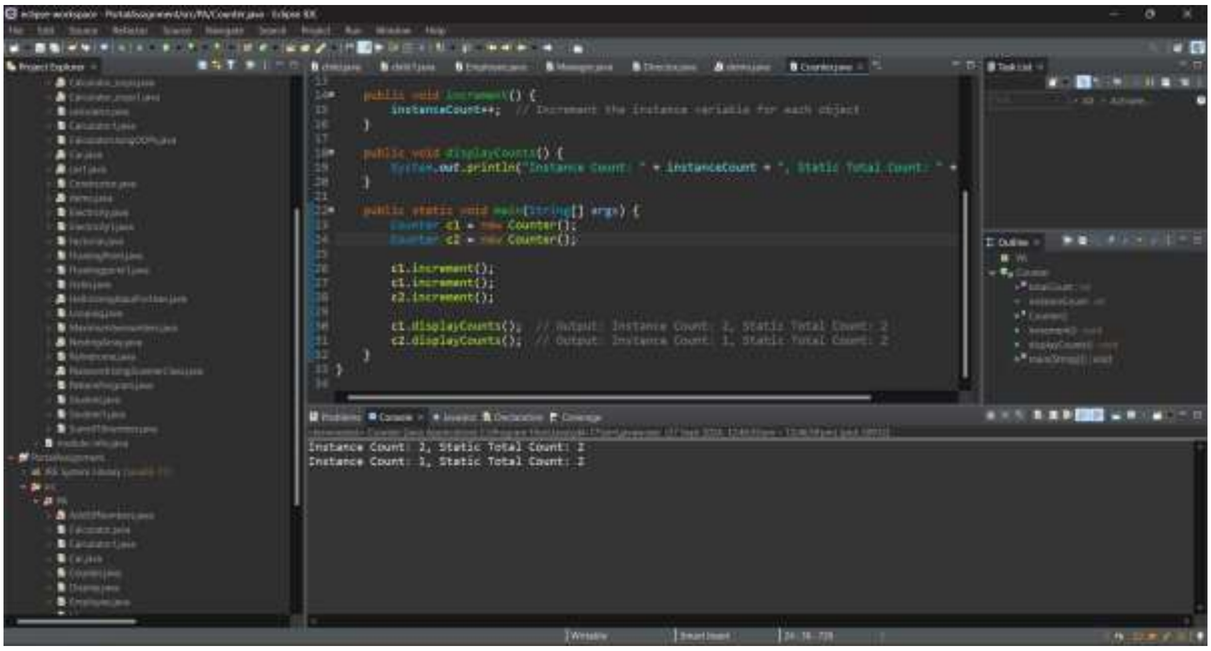
- **Definition**: These variables are defined inside a class with the static keyword and are shared among all instances of the class. There is only one copy of the static variable, which belongs to the class itself rather than any specific instance.
- **Scope**: Static variables are common to all objects of the class. Any changes made to the static variable by one object are reflected across all instances.
- **Access**: Static variables can be accessed via the class name (without creating an instance) or through object references, but the preferred way is via the class name.

Code for both variables in one program:-

```
package PA;

public class Counter {
    // Static variable to count all instances of the class
    static int totalCount = 0;

    // Instance variable to count individual object's method calls
    int instanceCount = 0;

    public Counter() {
        totalCount++;  // Increment the static variable for each new
object
    }

    public void increment() {
        instanceCount++;  // Increment the instance variable for
each object
    }

    public void displayCounts() {
        System.out.println("Instance Count: " + instanceCount + ",
Static Total Count: " + totalCount);
```

```
    }

    public static void main(String[] args) {
        Counter c1 = new Counter();
        Counter c2 = new Counter();

        c1.increment();
        c1.increment();
        c2.increment();

        c1.displayCounts();  // Output: Instance Count: 2, Static 
Total Count: 2
        c2.displayCounts();  // Output: Instance Count: 1, Static 
Total Count: 2
    }
}
```

Output:-



```
    public void increment() {
        instanceCount++;  // Increment the instance variable for each object
    }

    public void displayCounts() {
        System.out.println("Instance Count: " + instanceCount + ", Static Total Count: " +
    }

    public static void main(String[] args) {
        Counter c1 = new Counter();
        Counter c2 = new Counter();

        c1.increment();
        c1.increment();
        c2.increment();

        c1.displayCounts();  // Output: Instance Count: 2, Static Total Count: 2
        c2.displayCounts();  // Output: Instance Count: 1, Static Total Count: 2
    }
}
```

```
Instance Count: 2, Static Total Count: 2
Instance Count: 1, Static Total Count: 2
```