

1. Illustrate the importance of Constructor Overloading with appropriate example

```
package _221047003;
class Box
{
    double width, height, depth;

    // constructor used when all dimensions
    // specified
    Box(double w, double h, double d)
    {
        width = w;
        height = h;
        depth = d;
    }

    // constructor used when no dimensions
    // specified
    Box()
    {
        width = height = depth = 0;
    }

    // constructor used when cube is created
    Box(double len)
    {
        width = height = depth = len;
    }

    // compute and return volume
    double volume()
    {
        return width * height * depth;
    }
}

class Constructor
{
    public static void main(String args[])
    {
        // create boxes using the various
        // constructors
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);

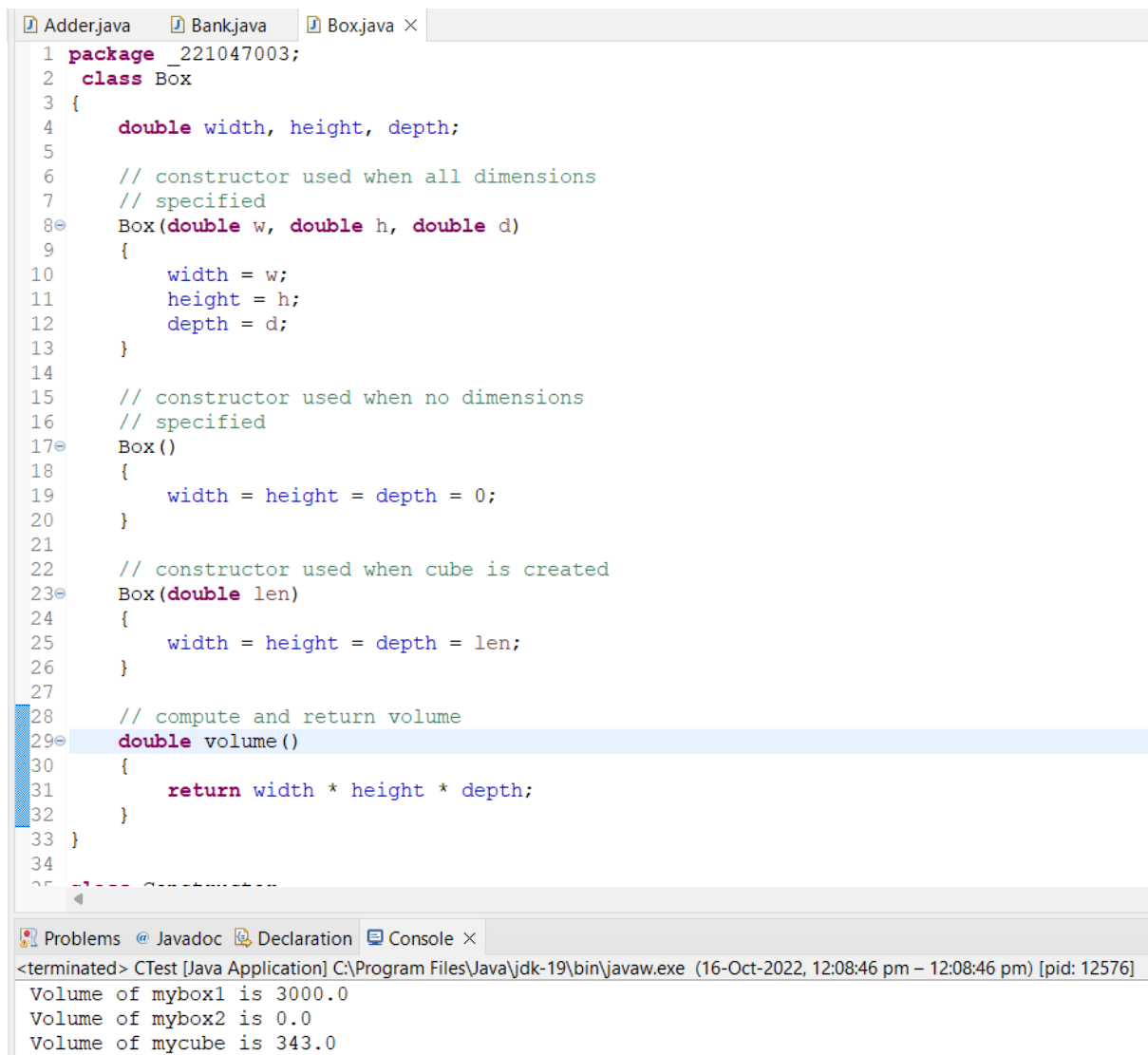
        double vol;

        // get volume of first box
        vol = mybox1.volume();
        System.out.println(" Volume of mybox1 is " + vol);

        // get volume of second box
        vol = mybox2.volume();
        System.out.println(" Volume of mybox2 is " + vol);

        // get volume of cube
        vol = mycube.volume();
        System.out.println(" Volume of mycube is " + vol);
    }
}
```

```
}
```



The screenshot shows an IDE with three tabs: 'Adder.java', 'Bank.java', and 'Box.java'. The 'Box.java' tab is active, displaying the following code:

```
1 package _221047003;
2 class Box
3 {
4     double width, height, depth;
5
6     // constructor used when all dimensions
7     // specified
8     Box(double w, double h, double d)
9     {
10         width = w;
11         height = h;
12         depth = d;
13     }
14
15     // constructor used when no dimensions
16     // specified
17     Box()
18     {
19         width = height = depth = 0;
20     }
21
22     // constructor used when cube is created
23     Box(double len)
24     {
25         width = height = depth = len;
26     }
27
28     // compute and return volume
29     double volume()
30     {
31         return width * height * depth;
32     }
33 }
34
35 ----- Constructor
```

Below the code editor, the 'Console' tab is active, showing the output of a Java application:

```
<terminated> CTest [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (16-Oct-2022, 12:08:46 pm – 12:08:46 pm) [pid: 12576]
Volume of mybox1 is 3000.0
Volume of mybox2 is 0.0
Volume of mycube is 343.0
```

2a. Java's support to multi-level inheritance

```
package _221047003;
```

```
class Shape {
```

```
    public void display() {
```

```
        System.out.println("Inside display");
```

```
    }
```

```
}
```

```
class Rectangle extends Shape { //class rectangle inherits properties of shape
```

```
    public void area() {
```

```
        System.out.println("Inside area");
```

```
    }  
}  
class Cube extends Rectangle { //class cube inherits properties of both shape and rectangle  
    public void volume() {  
        System.out.println("Inside volume");  
    }  
}  
class Test3  
{  
    public static void main(String[] arguments) {  
        Cube cube = new Cube();  
        cube.display();  
        cube.area();  
        cube.volume();  
    }  
}
```

```

1 package _221047003;
2 class Shape {
3     public void display() {
4         System.out.println("Inside display");
5     }
6 }
7 class Rectangle extends Shape { //class rectangle inherits properties of shape
8     public void area() {
9         System.out.println("Inside area");
10    }
11 }
12 class Cube extends Rectangle { //class cube inherits properties of both shape and rectangle
13     public void volume() {
14         System.out.println("Inside volume");
15     }
16 }
17 class Test3
18 {
19     public static void main(String[] arguments) {
20         Cube cube = new Cube();
21         cube.display();
22         cube.area();
23         cube.volume();
24     }
25 }
26

```

Problems @ Javadoc Declaration Console ×

<terminated> CTest [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (16-Oct-2022, 3:11:59 pm – 3:12:00 pm) [pid: 23240]

Inside display
Inside area
Inside volume

2.b Usage of Super from at method level and constructor level

```

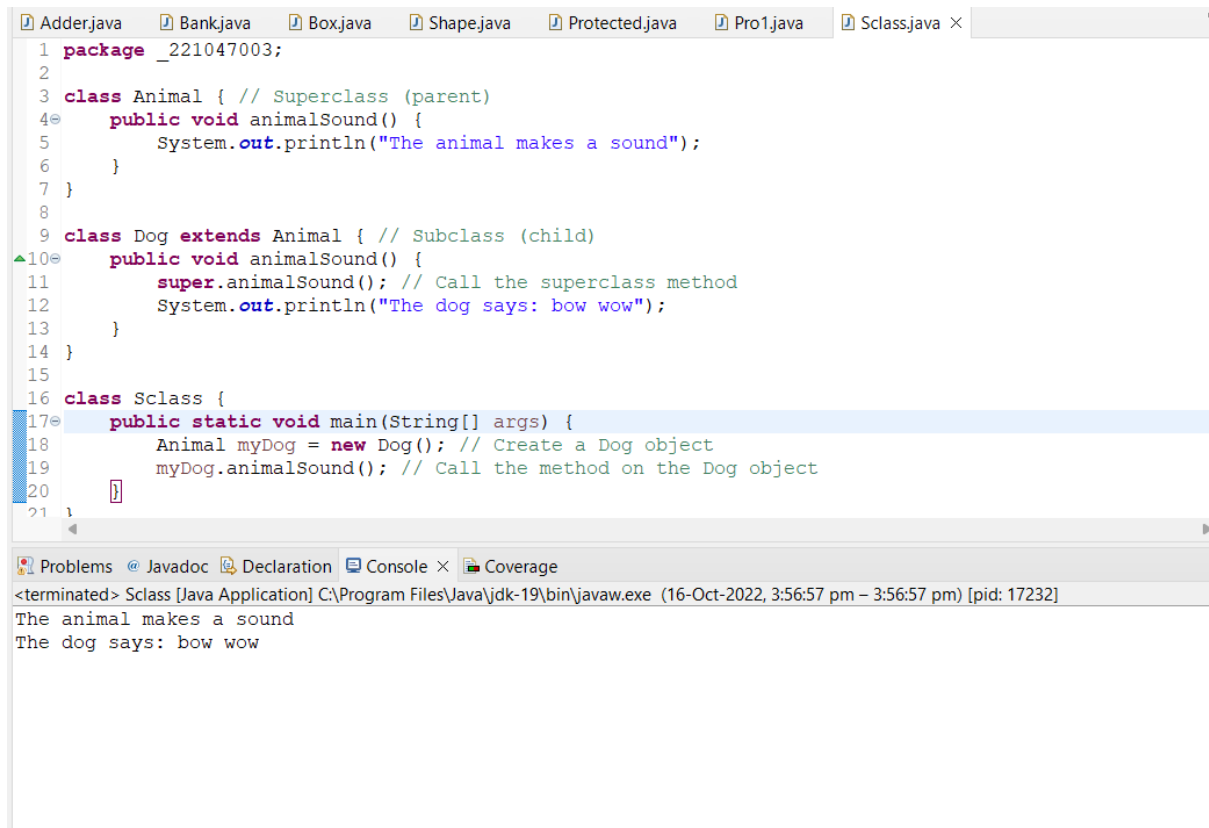
package _221047003;

class Animal { // Superclass (parent)
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

class Dog extends Animal { // Subclass (child)
    public void animalSound() {
        super.animalSound(); // Call the superclass method
        System.out.println("The dog says: bow wow");
    }
}

class Sclass {
    public static void main(String[] args) {
        Animal myDog = new Dog(); // Create a Dog object
        myDog.animalSound(); // Call the method on the Dog object
    }
}

```



The screenshot shows an IDE with several tabs: Adder.java, Bank.java, Box.java, Shape.java, Protected.java, Pro1.java, and Sclass.java. The Sclass.java tab is active, displaying the following code:

```
1 package _221047003;
2
3 class Animal { // Superclass (parent)
4     public void animalSound() {
5         System.out.println("The animal makes a sound");
6     }
7 }
8
9 class Dog extends Animal { // Subclass (child)
10    public void animalSound() {
11        super.animalSound(); // Call the superclass method
12        System.out.println("The dog says: bow wow");
13    }
14 }
15
16 class Sclass {
17    public static void main(String[] args) {
18        Animal myDog = new Dog(); // Create a Dog object
19        myDog.animalSound(); // Call the method on the Dog object
20    }
21 }
```

Below the code editor, the console window shows the output of the program:

```
<terminated> Sclass [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (16-Oct-2022, 3:56:57 pm - 3:56:57 pm) [pid: 17232]
The animal makes a sound
The dog says: bow wow
```

2.c Working of Protected access

```
package _221047003;

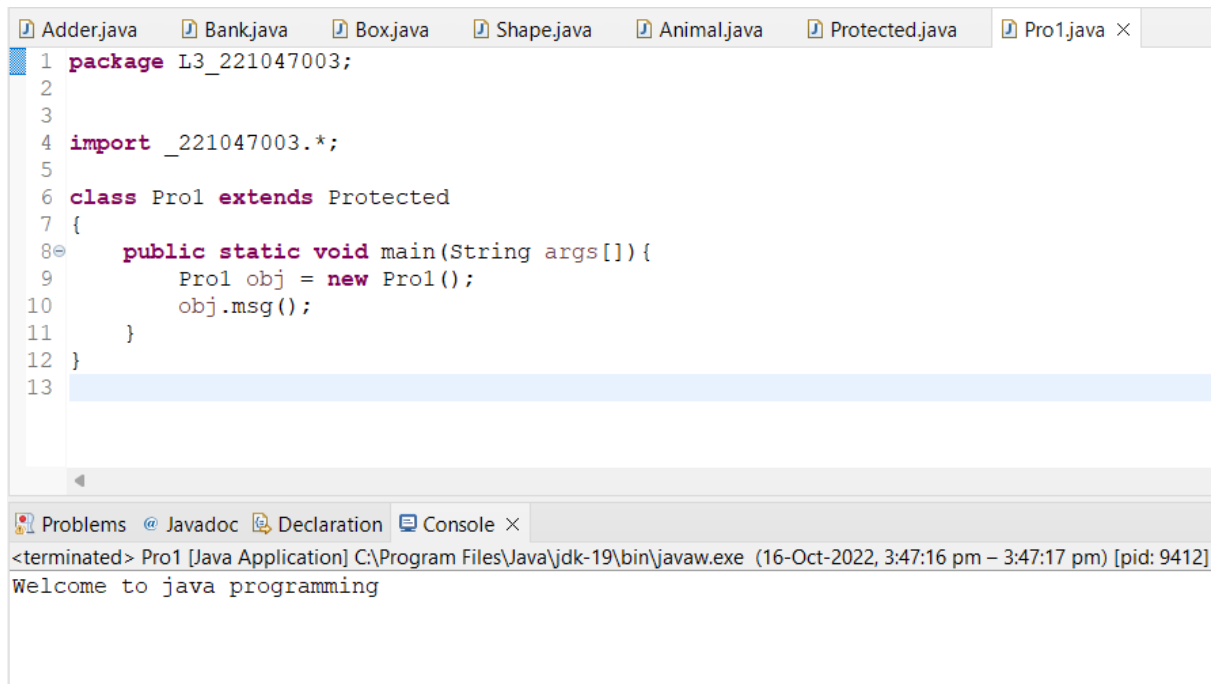
public class Protected{

    protected void msg()
    {
        System.out.println("Welcome to java programming");
    }
}

package L3_221047003;

import _221047003.*;

class Pro1 extends Protected
{
    public static void main(String args[]){
        Pro1 obj = new Pro1();
        obj.msg();
    }
}
```



The screenshot shows an IDE with several tabs: Adder.java, Bank.java, Box.java, Shape.java, Animal.java, Protected.java, and Pro1.java. The Pro1.java tab is active, displaying the following code:

```
1 package L3_221047003;
2
3
4 import _221047003.*;
5
6 class Pro1 extends Protected
7 {
8     public static void main(String args[]) {
9         Pro1 obj = new Pro1();
10        obj.msg();
11    }
12 }
13
```

Below the code editor, the console window shows the output of the program:

```
<terminated> Pro1 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (16-Oct-2022, 3:47:16 pm - 3:47:17 pm) [pid: 9412]
Welcome to java programming
```

3. Differentiate between method overloading and overriding with appropriate example

// Method overloading by different Number of parameters in argument list

```
package _221047003;

class Adder{

    static int add(int a,int b)//Overloading Class add with two parameters
    {
        return a+b;
    }

    static int add(int a,int b,int c)//Overloading the same Class add with three parameters
    {
        return a+b+c;
    }
}

class Overloading1
{
    public static void main(String[] args)
    {
```

```

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}

}

```

```

1 package _221047003;
2
3 class Adder{
4     static int add(int a,int b)//Overloading Class add with two parameters
5     {
6         return a+b;
7     }
8     static int add(int a,int b,int c)// //Overloading the same Class add with three parameters
9
10    {
11        return a+b+c;
12    }
13 }
14 class Overloading1
15 {
16     public static void main(String[] args)
17     {
18         System.out.println("Overloading Class add with two parameters:"+Adder.add(11,11));
19         System.out.println("Overloading the same Class add with three parameters:"+Adder.add(11,11,11));
20     }
21 }
22

```

Problems @ Javadoc Declaration Console ×

terminated> Overloading1 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (16-Oct-2022, 11:33:03 am – 11:33:04 am) [pid: 14552]

verloading Class add with two parameters:22

verloading the same Class add with three parameters:33

Example 2: Method Overloading: changing data type of arguments

```

package _221047003;

class Adder{

    static int add(int a, int b){return a+b;

}

    static double add(double a, double b){return a+b;}

}

class TestOverloading2{

    public static void main(String[] args){

        System.out.println(Adder.add(11,11));

        System.out.println(Adder.add(12.3,12.6));

    }

}

```

}

}

```
*Adder.java ×
1 package _221047003;
2
3 class Adder
4 {
5
6     static int add(int a, int b) //class add with int datatype
7     {
8         return a+b;
9     }
10    static double add(double a, double b) //Same class add with double datatype
11    {
12        return a+b;
13    }
14 }
15 class Overloading1
16 {
17     public static void main(String[] args)
18     {
19         System.out.println(Adder.add(11,11));
20         System.out.println(Adder.add(2.3,2.6));
21     }
22 }
23
```

Problems @ Javadoc Declaration Console ×

<terminated> Overloading1 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (16-Oct-2022, 11:37:03 am – 11:37:04 am) [pid: 6876]

22
4.9

//Method overriding

```
package _221047003;
```

```
class Bank{
```

```
int getRateOfInterest()//Method
```

```
{
```

```
return 0;
```

```
}
```

```
}
```

//Creating child classes.

```
class SBI extends Bank{
```

```
int getRateOfInterest()//Same methodname
```

```
{
```



```
return 8;
```

```
}
```

```
}
```

```
class ICICI extends Bank
```

```
{
```

```
int getRateOfInterest();//Same methodname
```

```
{
```

```
return 7;
```

```
}
```

```
}
```

```
class AXIS extends Bank
```

```
{
```

```
int getRateOfInterest();//Same methodname
```

```
{
```

```
return 9;
```

```
}
```

```
}
```

```
class Test2{
```

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

```
SBI s=new SBI();
```

```
ICICI i=new ICICI();
```

```
AXIS a=new AXIS();
```

```
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest()); //Same methodname of class SBI
```

```
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest()); // Same methodname of class  
ICICI
```

```
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest()); Same methodname of class AXIX
```

```
}
```

```
}
```

```

1 package _221047003;
2 class Bank{
3     int getRateOfInterest()//Method
4     {
5         return 0;
6     }
7 }
8 //Creating child classes.
9 class SBI extends Bank{
10    int getRateOfInterest()//Same methodname
11    {
12        return 8;
13    }
14 }
15
16 class ICICI extends Bank
17 {
18    int getRateOfInterest()//Same methodname
19    {
20        return 7;
21    }
22 }
23 class AXIS extends Bank
24 {
25    int getRateOfInterest()//Same methodname
26    {
27        return 9;
28    }
29 }
30
31 class Test2{
32    public static void main(String args[]){
33        SBI s=new SBI();
34        ICICI i=new ICICI();
35        AXIS a=new AXIS();
36        System.out.println("SBI Rate of Interest: "+s.getRateOfInterest()); //Same methodname of class SBI
    }
}

```

Problems @ Javadoc Declaration Console ×

```

<terminated> Overloading1 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (16-Oct-2022, 11:41:55 am – 11:41:55 am) [pid: 1760]
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9

```

4.Demonstrate the usefulness of finalize() method

```

package _221047003;
class Finalize
{
    public static void main(String[ ] args) // A "main
thread" gets introduced
    {
        String s = new String("Gate Vidyalay"); // A String
object gets created
        s = null; // String
Object becomes eligible for garbage collection
        System.gc( ); // A request
is made to JVM for running garbage collector ; A "gc thread" gets
introduced
        System.out.println("End of main method");
    }
    public void finalize( ) // Test class
finalize( ) method
    {
        System.out.println("Finalize method of Test class");
    }
}

```

```
1 package _221047003;
2 class Finalize
3 {
4     public static void main(String[] args)           // A "main thread" gets introduced
5     {
6         String s = new String("Gate Vidyalay");      // A String object gets created
7         s = null;                                     // String Object becomes eligible for
8         System.gc( );                                // A request is made to JVM for runni
9         System.out.println("End of main method");
10    }
11    public void finalize( )                           // Test class finalize( ) method
12    {
13        System.out.println("Finalize method of Test class");
14    }
15 }
```

Problems @ Javadoc Declaration Console × Coverage

<terminated> Finalize [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (16-Oct-2022, 4:16:33 pm – 4:16:33 pm) [pid: 4572]

End of main method

5. Illustrate the concepts of Abstract class and Interface with appropriate example

```
package _221047003;
interface A{
void a();//by default, public and abstract
void b();
void c();
void d();
}
```

//Creating abstract class that provides the implementation of one method of A interface

```
abstract class B implements A{
public void c(){System.out.println("I am C");}
}
```

//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods

```
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}
```

//Creating a test class that calls the methods of A interface

```
class Abstract_Interface{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

```
1 package _221047003;
2 interface A{
3     void a();//bydefault, public and abstract
4     void b();
5     void c();
6     void d();
7 }
8
9 //Creating abstract class that provides the implementation of one method of A interface
10 abstract class B implements A{
11     public void c(){System.out.println("I am C");}
12 }
13
14 //Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
15 class M extends B{
16     public void a(){System.out.println("I am a");}
17     public void b(){System.out.println("I am b");}
18     public void d(){System.out.println("I am d");}
19 }
20
21 //Creating a test class that calls the methods of A interface
22 class Abstract_Interface{
23     public static void main(String args[]){
24         A a=new M();
25         a.a();
26         a.b();
27         a.c();
28         a.d();
29     }}
--
```

Problems @ Javadoc Declaration Console × Coverage

<terminated> Abstract_Interface [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (16-Oct-2022, 4:07:14 pm – 4:07:14 pm) [pid: 5040]

I am a
I am b
I am C
I am d

6. Illustrate the significance of Encapsulation – namely the control the concept provides in your application through appropriate examples.

```
package _221047003;

class Student {
    private int Student_Id;
    private String name;
    //getters, setters for Student_Id and name fields.
    public int getId() {
        return Student_Id;
    }
    public void setId(int s_id) {
        this.Student_Id = s_id;
    }
    public String getname() {
        return name;
    }
    public void setname(String s_name) {
        this.name = s_name;
    }
}

class Encap{
    public static void main(String[] args) {
```

```

        //create an object of Student class
        Student s=new Student();
        //set fields values using setter methods
        s.setId (123);
        s.setName("John");
        //print values using getter methods
        System.out.println("Student Data:" + "\nStudent ID:" + s.getId()
            + "\nStudent Name:" + s.getName());
    }
}

```

```

1 package _221047003;
2
3 class Student {
4     private int Student_Id;
5     private String name;
6     //getters, setters for Student_Id and name fields.
7     public int getId() {
8         return Student_Id;
9     }
10    public void setId(int s_id) {
11        this.Student_Id = s_id;
12    }
13    public String getName() {
14        return name;
15    }
16    public void setName(String s_name) {
17        this.name = s_name;
18    }
19 }
20 class Encap{
21    public static void main(String[] args) {
22        //create an object of Student class
23        Student s=new Student();
24        //set fields values using setter methods
25        s.setId (123);
26        s.setName("John");
27        //print values using getter methods
28        System.out.println("Student Data:" + "\nStudent ID:" + s.getId()
29            + "\nStudent Name:" + s.getName());
30    }
31 }

```

Problems @ Javadoc Declaration Console × Coverage

<terminated> Encap [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (16-Oct-2022, 4:02:49 pm – 4:02:49 pm) [pid: 2180]

Student Data:
Student ID:123
Student Name:John