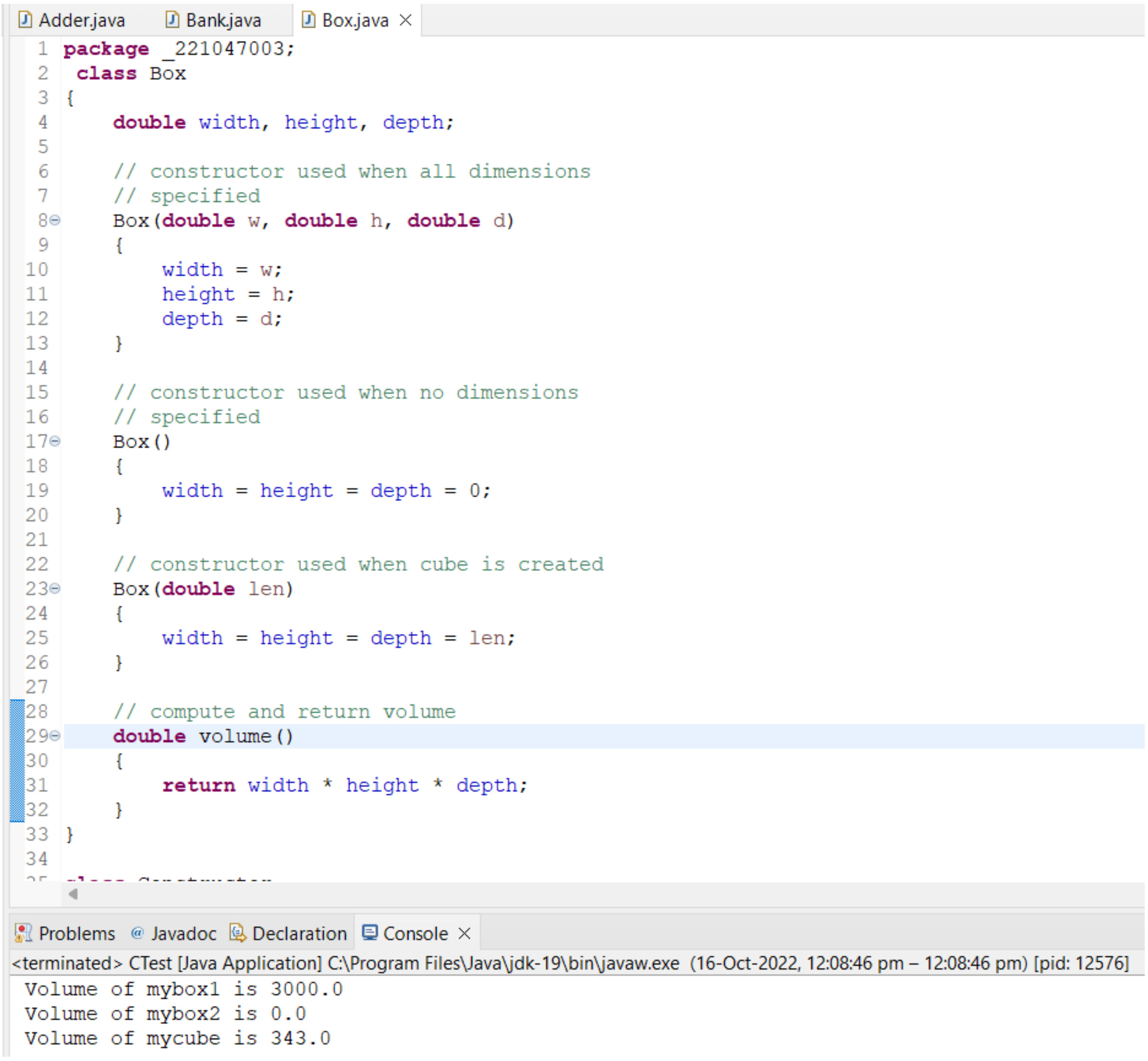
1.Illustrate the importance of Constructor Overloading with appropriate example

```java
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);
    }
```

}


Adder.java Bank.java Box.java

```
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;
    }
}

```

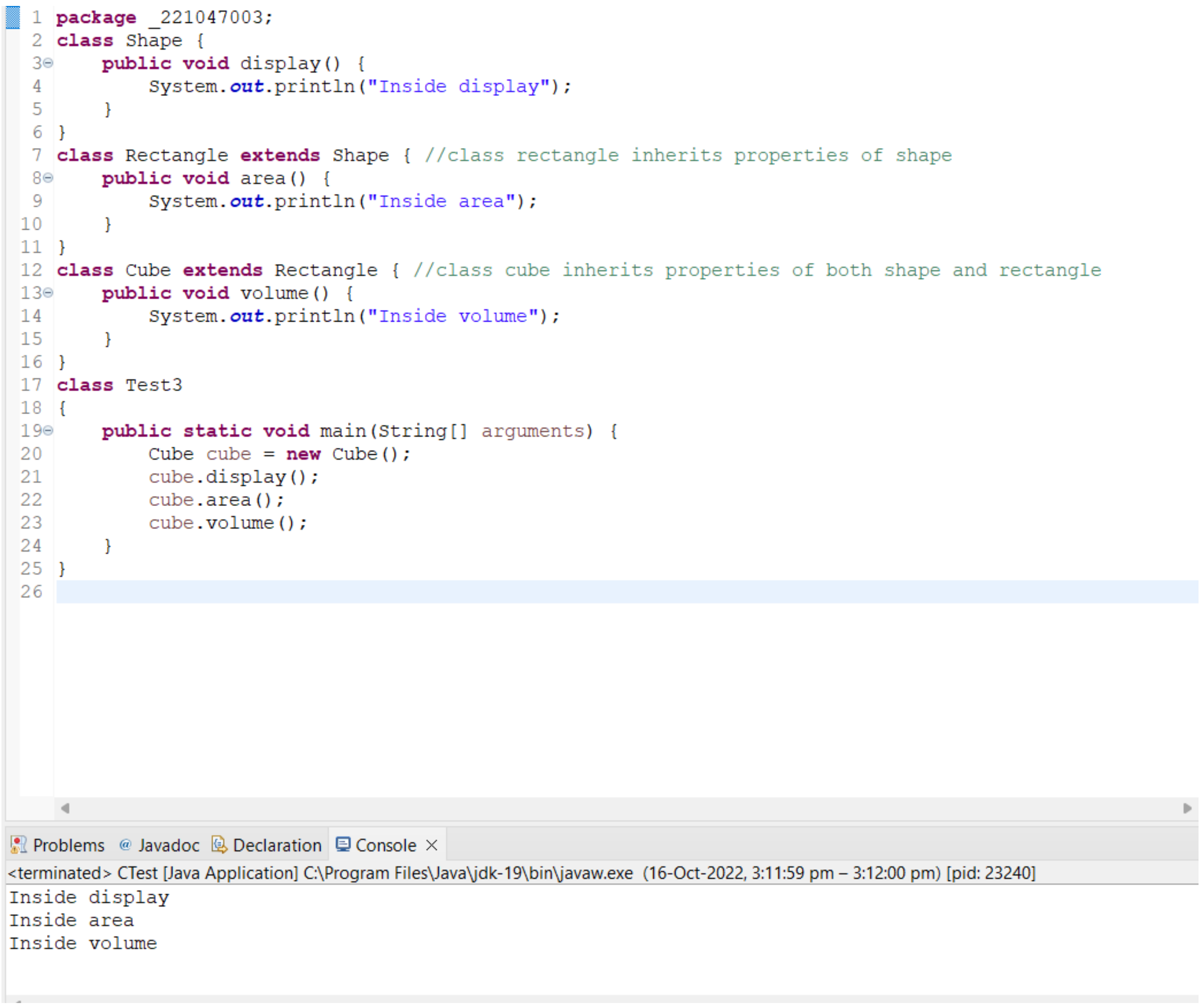
Problems Javadoc Declaration Console

```
<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();
    }
}
```

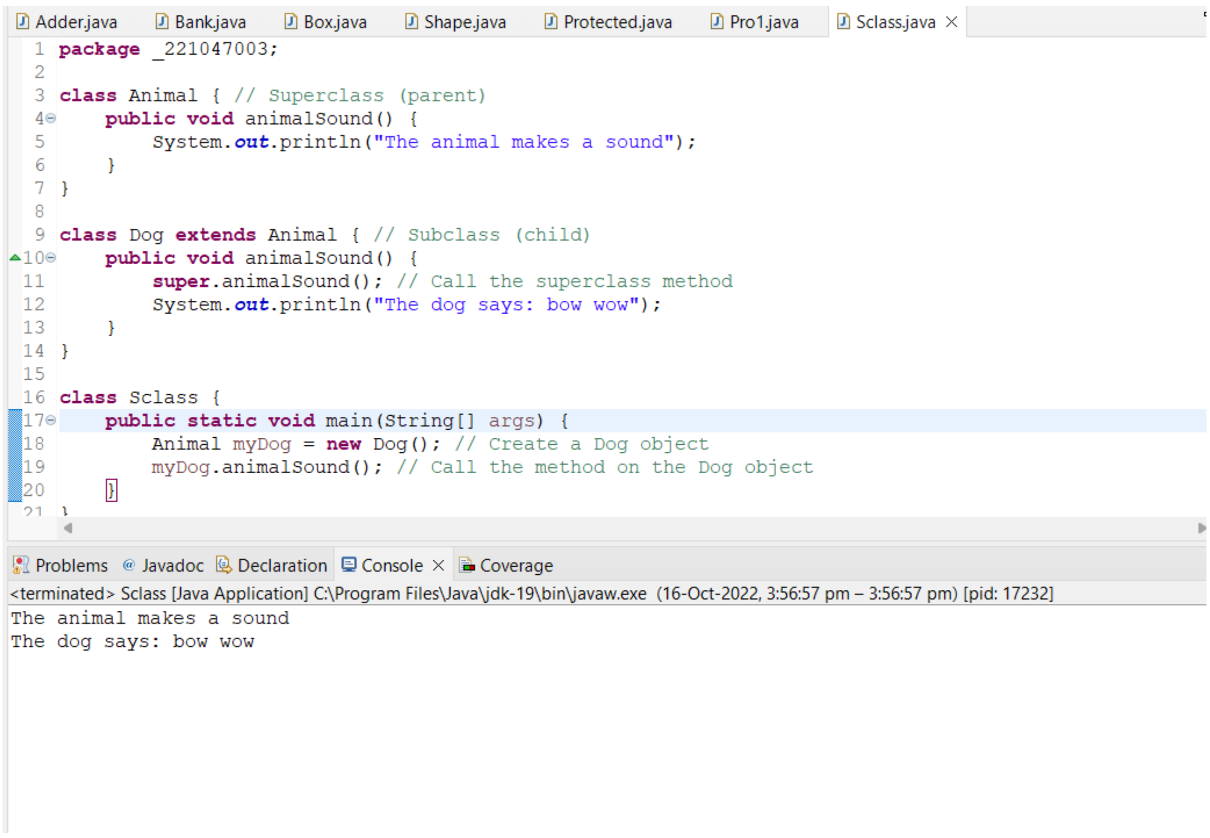


```java
package _221047003;
class Shape {
    public void display() {
        System.out.println("Inside display");
    }
}
class Rectangle extends Shape { //class rectangle inherits properties of shape
    public void area() {
        System.out.println("Inside area");
    }
}
class Cube extends Rectangle { //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();
    }
}
```

```
<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

```java
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
    }
}
```

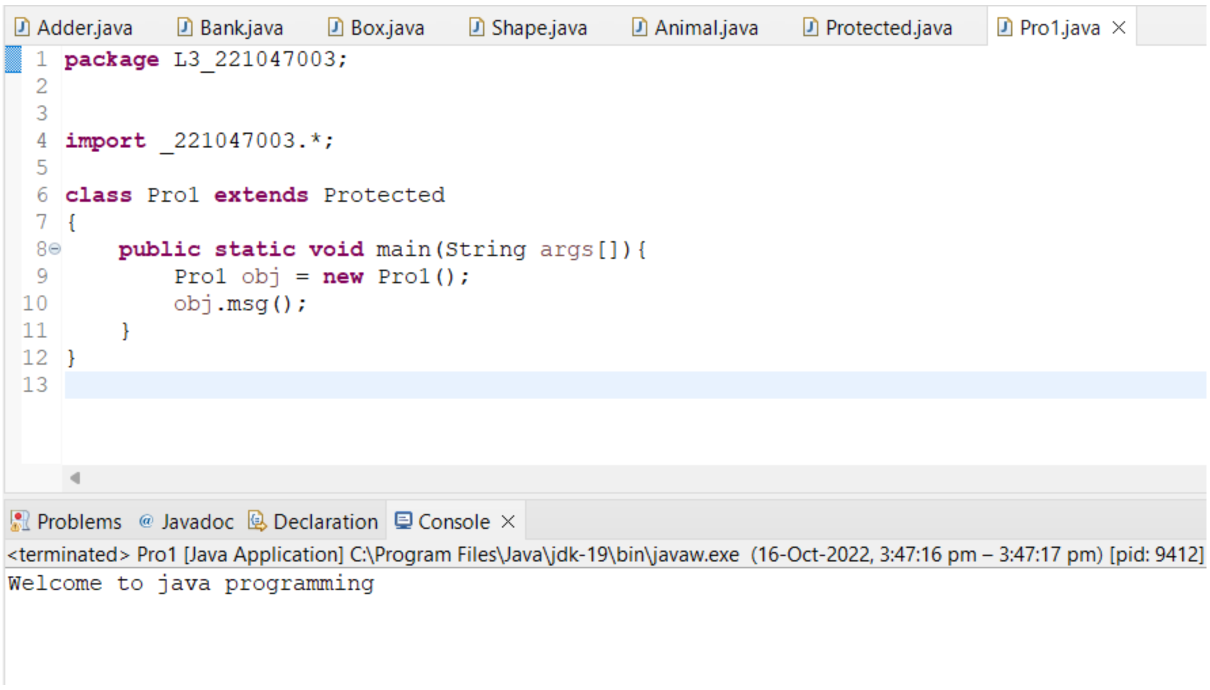


```
package _221047003;

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
    }
}
```

```
<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();
    }
}
```



```java
package L3_221047003;


import _221047003.*;

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

```
<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

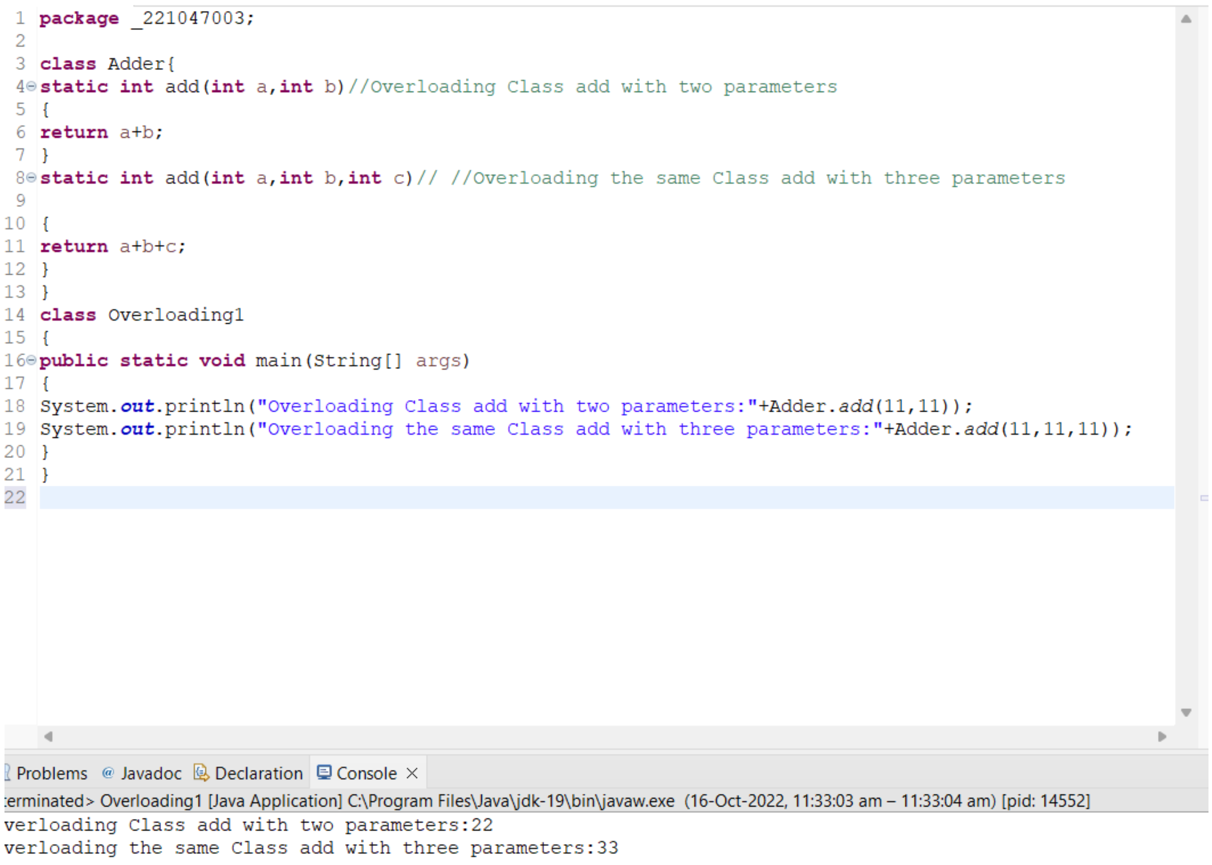
```java
package _221047003;
class Adder{
static int add(int a,int b)//Overloading Class add with two parameters
{
return a+b;
}
static int add(int a,int b,int c)// //Overloading the same Class add with three parameters


{
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));

}

}



```
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("Overloading Class add with two parameters:"+Adder.add(11,11));
System.out.println("Overloading the same Class add with three parameters:"+Adder.add(11,11,11));
}
}
```

```
erminated> 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{

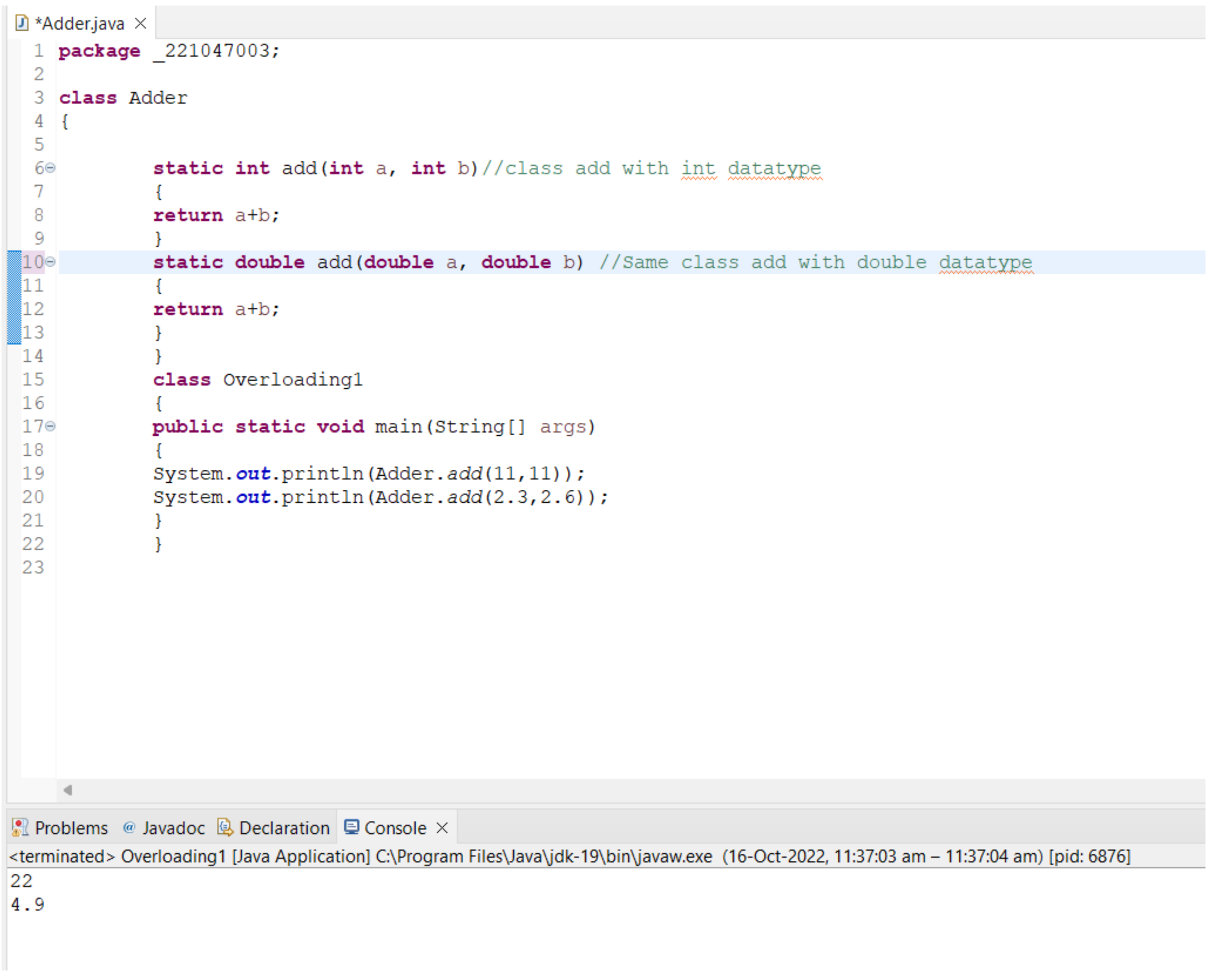
public static void main(String[] args){

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

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

}

}



*Adder.java

```
package _221047003;

class Adder
{

        static int add(int a, int b)//class add with int datatype
        {
        return a+b;
        }
        static double add(double a, double b) //Same class add with double datatype
        {
        return a+b;
        }
        }
        class Overloading1
        {
        public static void main(String[] args)
        {
        System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(2.3,2.6));
        }
        }

```

```
<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;

}

}

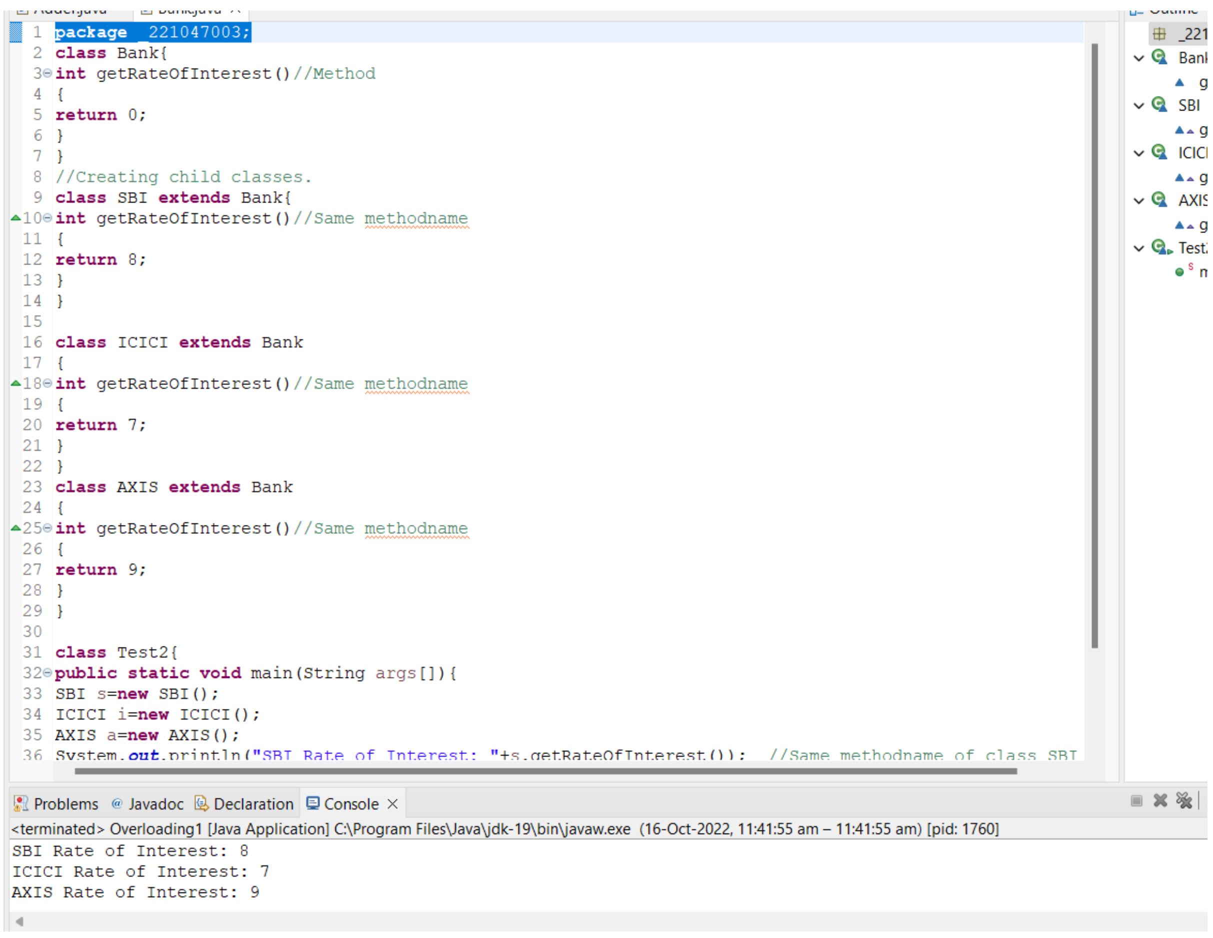
//Creating child classes.

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

}

}
```

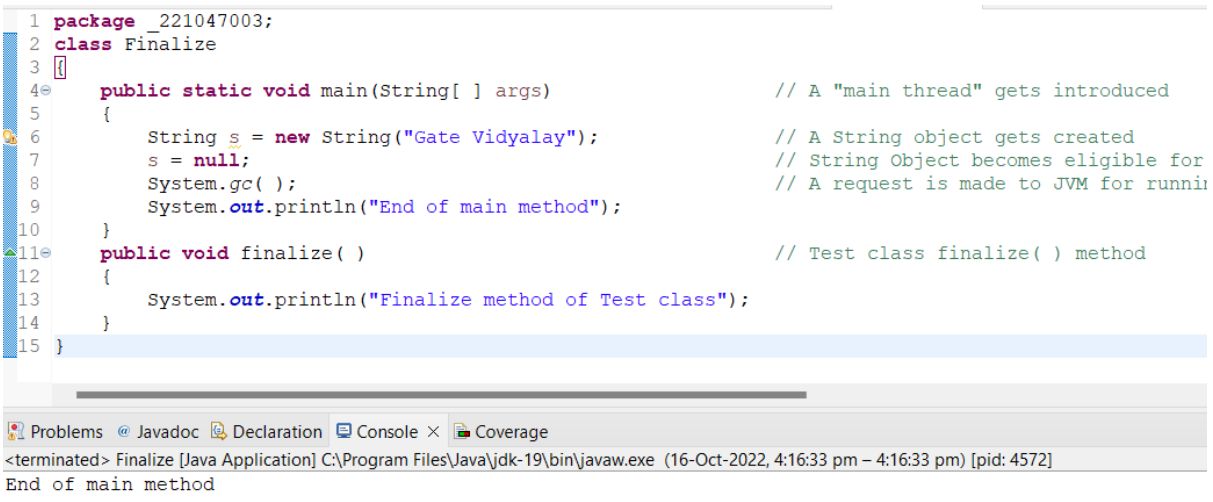


```
package _221047003;
class Bank{
int getRateOfInterest()//Method
{
return 0;
}
}
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest()//Same methodname
{
return 8;
}
}

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
```

```
<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");
    }
}
```



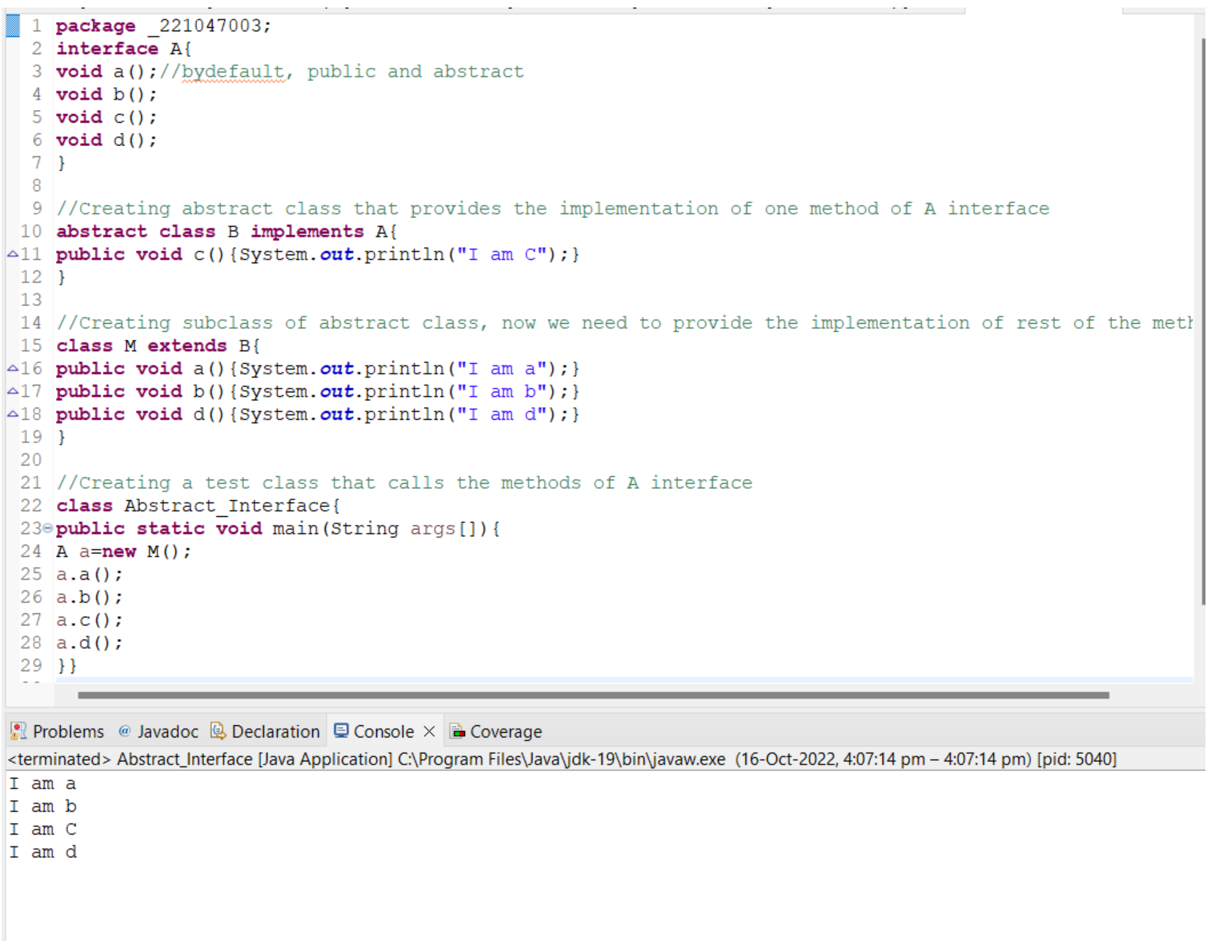
```
package _221047003;
class Finalize
{
    public static void main(String[ ] args)                    // A "main thread" gets introduced
    {
        String s = new String("Gate Vidyalay");               // A String object gets created
        s = null;                                            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

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

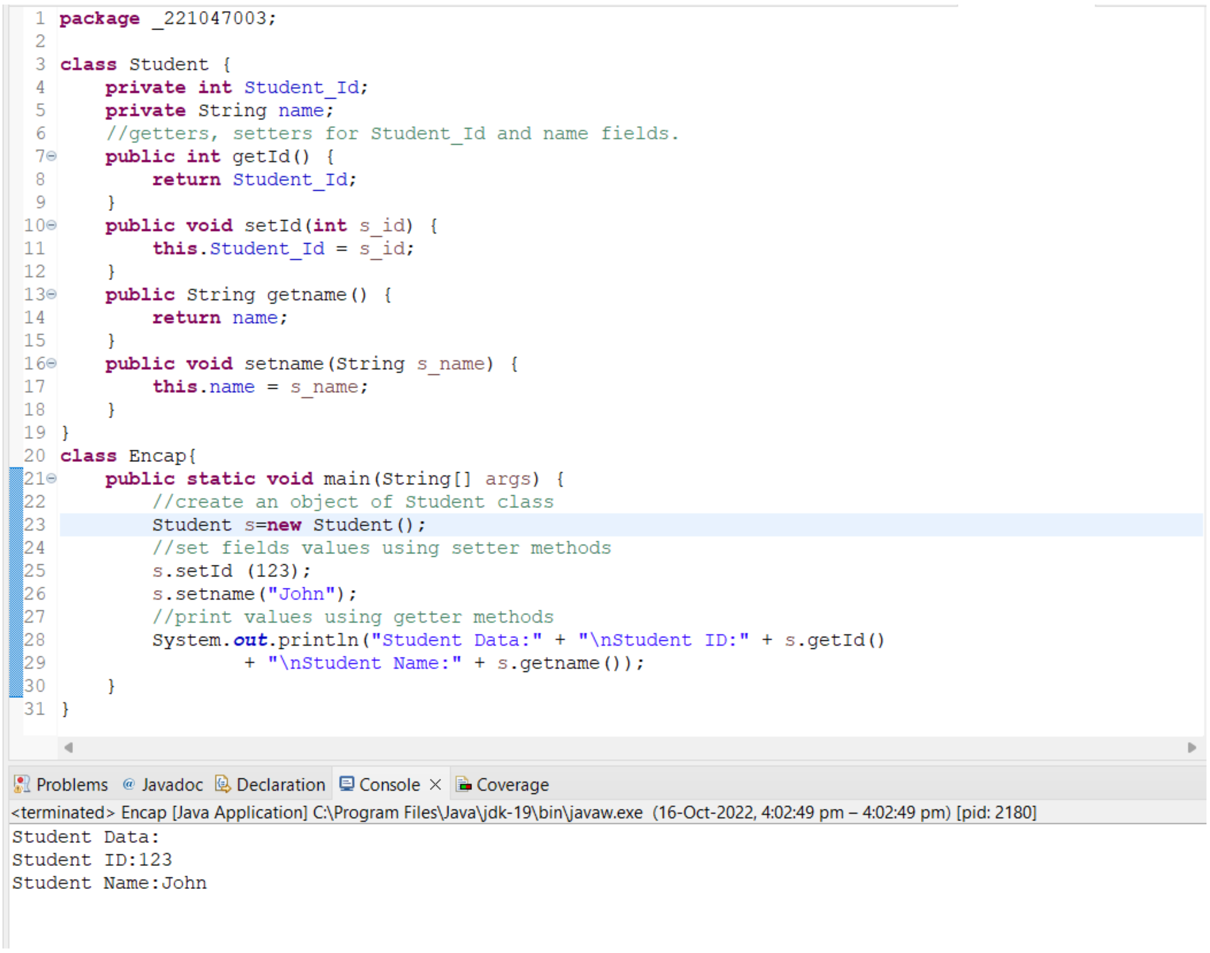
```
package _221047003;
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}

//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();
}}
```


```java
package _221047003;
interface A{
void a();//bydefault, 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 meth
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();
}}
```

```
<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.

```java
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());
    }
}
```



```
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());
    }
}
```

```
<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
```