**Q1. Write a singleton class. Confirm that singleton class cannot be inherited.**

**Singleton Class:-**

The singleton design pattern is used to restrict the instantiation of a class and ensures that only one instance of the class exists in the JVM. In other words, a singleton class is a class that can have only one object (an instance of the class) at a time per JVM instance.

**Specifications :**

class SingletonInheritanceCheck{ }  
public class Assignment2Q1 {}

**Theory:**

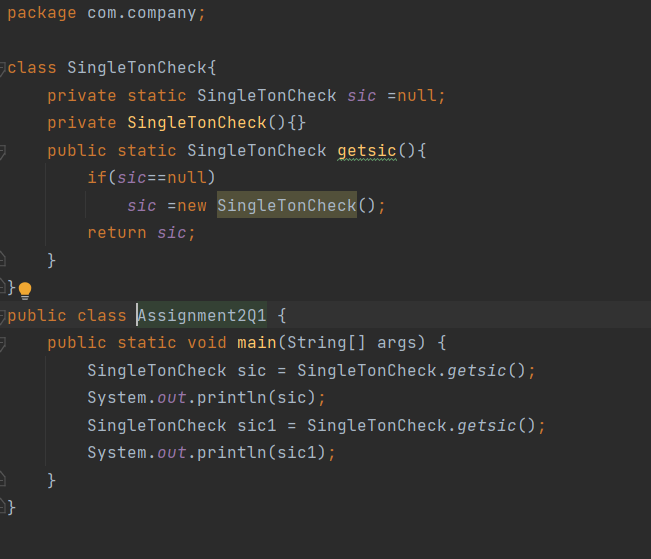
**Singleton Class in Java**

In object-oriented programming, a singleton class is a class that can have only one object (an instance of the class) at a time.

After first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created. So whatever modifications we do to any variable inside the class through any instance, it affects the variable of the single instance created and is visible if we access that variable through any variable of that class type defined.

Remember the key points while defining class as singleton class that is while designing a singleton class:

* Make constructor private.
* Write a static method that has return type object of this singleton class. Here, the concept of Lazy initialization is used to write this static method.



**Q2. Write a program that describes the hierarchy of an organization. Here we need to write 3 classes Employee, Manager & Labour where Manager & Labour are the sub classes of the Employee. Manager has incentive & Labour has over time. Add the functionality to calculate total salary of all the employees. Use polymorphism i.e. method overriding.**

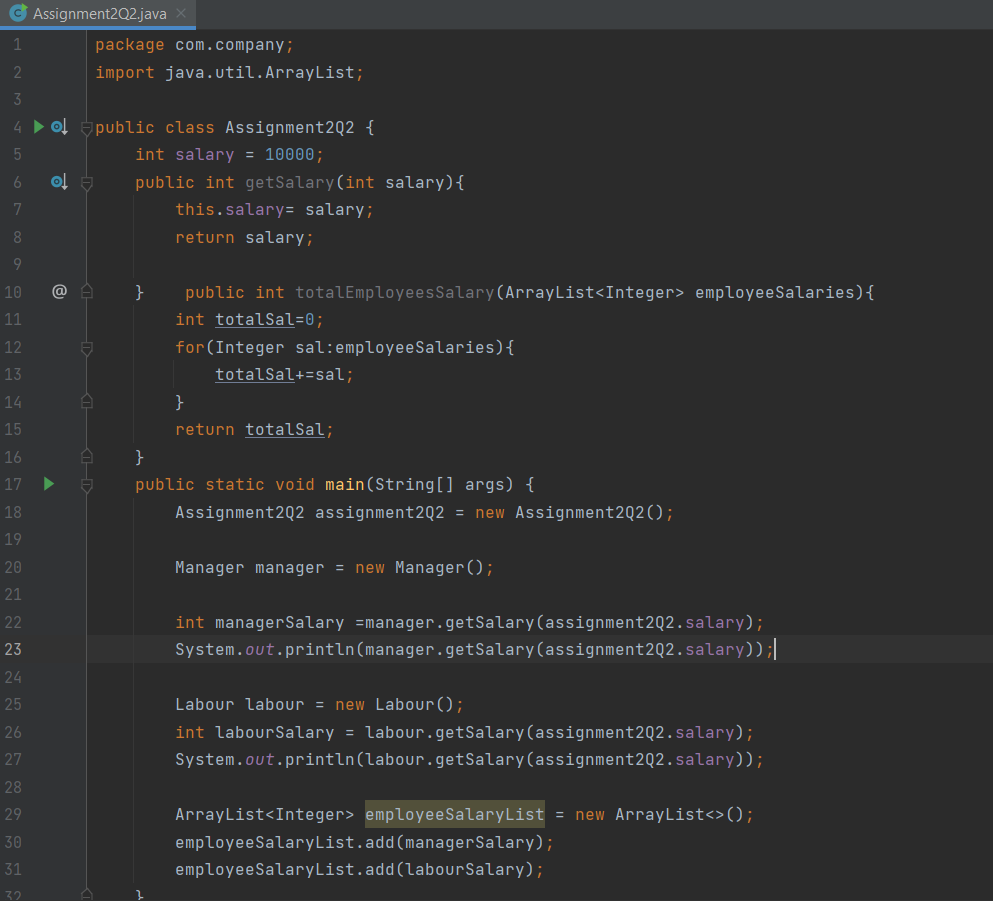
**Description:-**

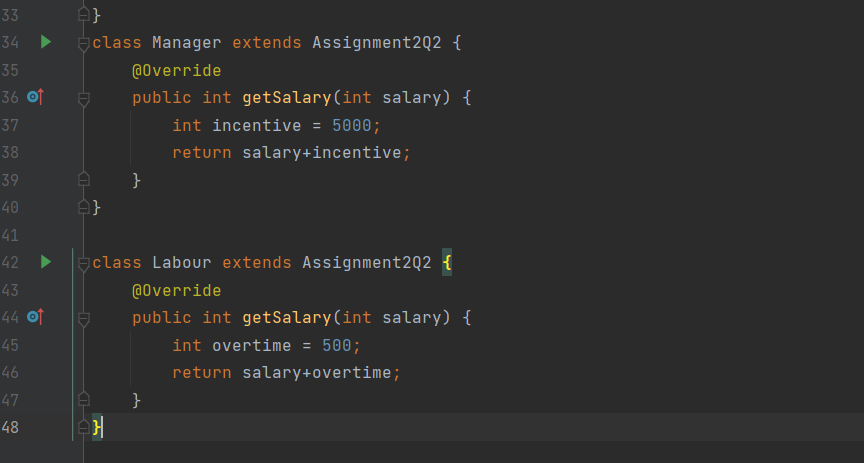
Write a program to calculate the total salaries given to all the employees by an organization. In this organization there are employees category under which Manager and Labour are working where manager will receive incentive and labourer will receive overtime benefits.

**Specifications:**

class Manager extends Assignment2Q2 {  
    @Override  
    public int getSalary(int salary) {  
        int incentive = 5000;  
    }  
}  
  
class Labour extends Assignment2Q2 {  
    @Override  
    public int getSalary(int salary) {  
        int overtime = 500;  
    }  
}  
  
public class Assignment2Q2 {  
    int salary = 10000;  
    public int getSalary(int salary){}  
    public int totalEmployeesSalary(ArrayList<Integer> employeeSalaries){}  
    public static void main(String[] args) {}  
}

**Code:**





**Q3. Write a program to consider saving & current account in the bank. Saving account holder has ‘Fixed Deposits’ whereas Current account holder has cash credit. Apply polymorphism to find out total cash in the bank.**

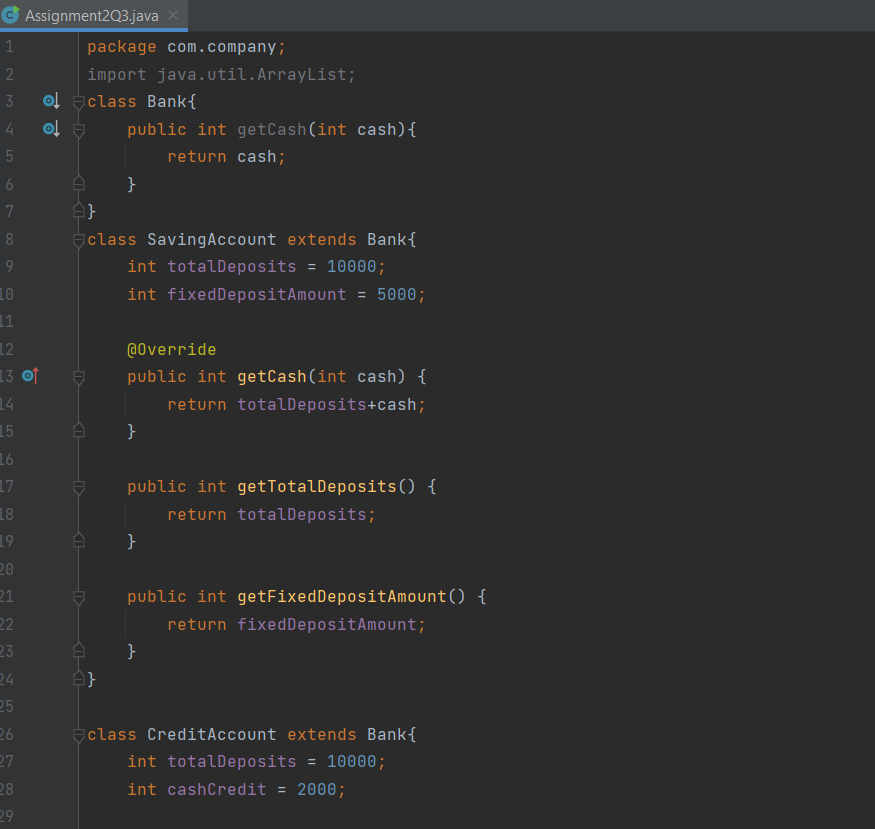
**Description: -**

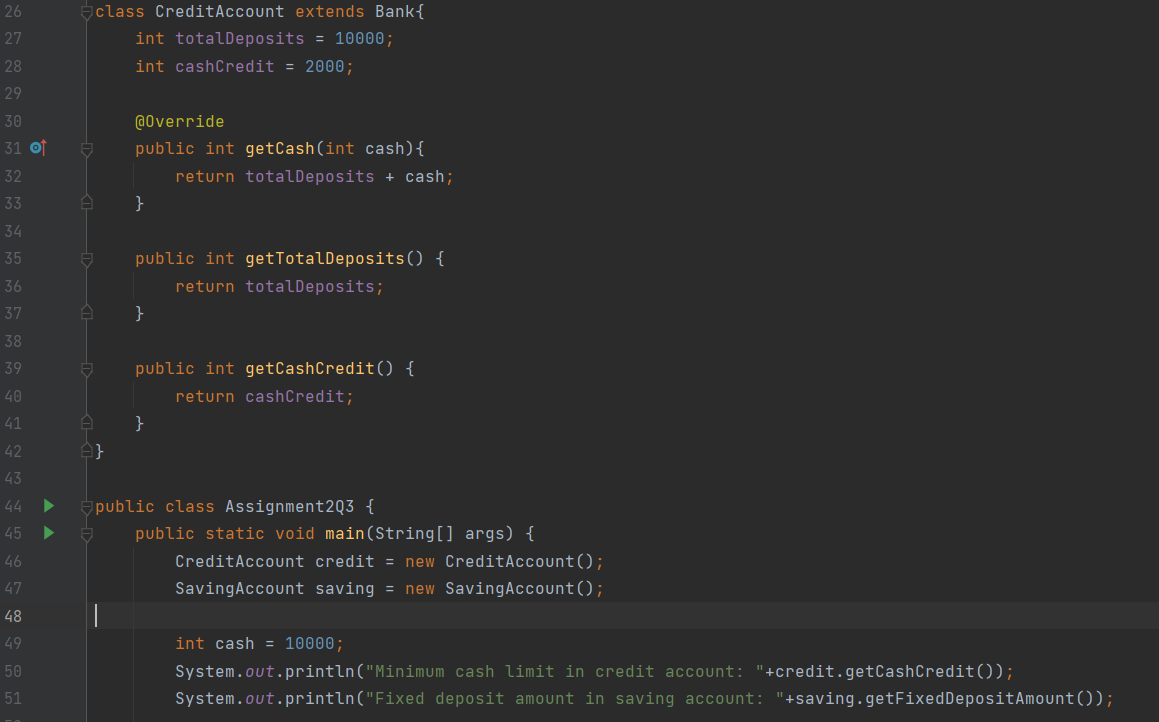
Write a program to calculate the total cash available in the bank. In this bank, bank will provide two types of accounts one is savings accounts and another is current account where current account has cash credits and savings account has fixed deposit options.

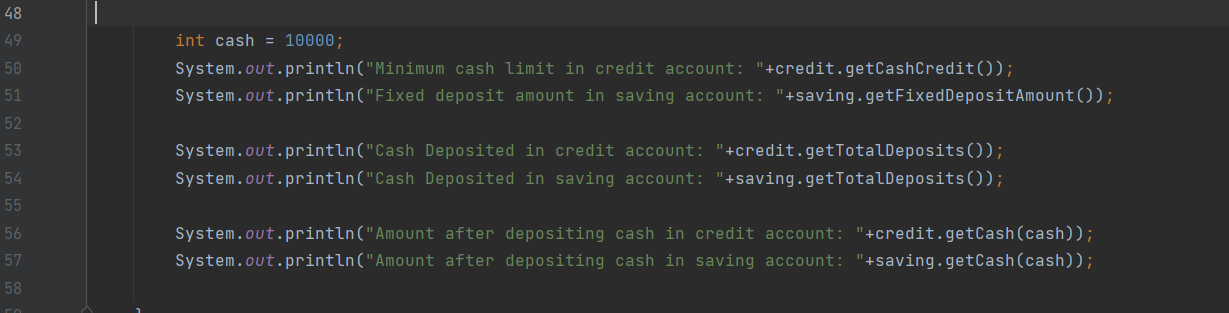
**Specifications:**

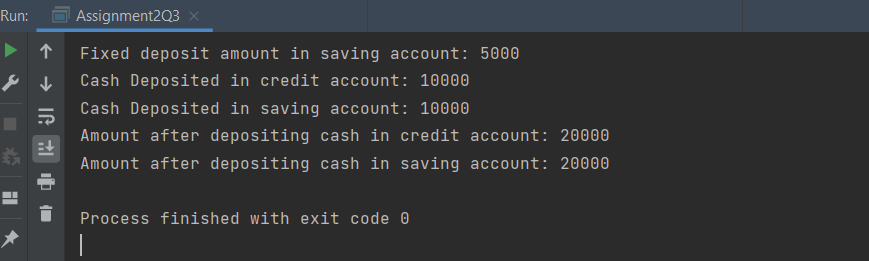
class CurrentAccount extends Assignment2Q3 {  
    int totalDeposits = 10000;  
    int creditLimit = 2000;  
    @Override  
    public int getCash() {}  
}  
class SavingsAccount extends Assignment2Q3 {  
    int totalDeposits = 10000;  
    int fixedDepositAmount = 5000;  
    @Override  
    public int getCash() {}  
}  
public class Assignment2Q3 {  
    public int totalCashInBank(ArrayList<Integer> cash){}  
    public int getCash(){}  
    public static void main(String[] args) {}  
}

**Code:**









**4. Test the following principles of an abstract class:**

**• If any class has any of its method abstract then you must declare entire class abstract.**

**• Abstract class cannot be instantiated.**

**• When we extend an abstract class, we must either override all the abstract methods in sub class or declare subclass as abstract.**

**• Abstract class cannot be private.**

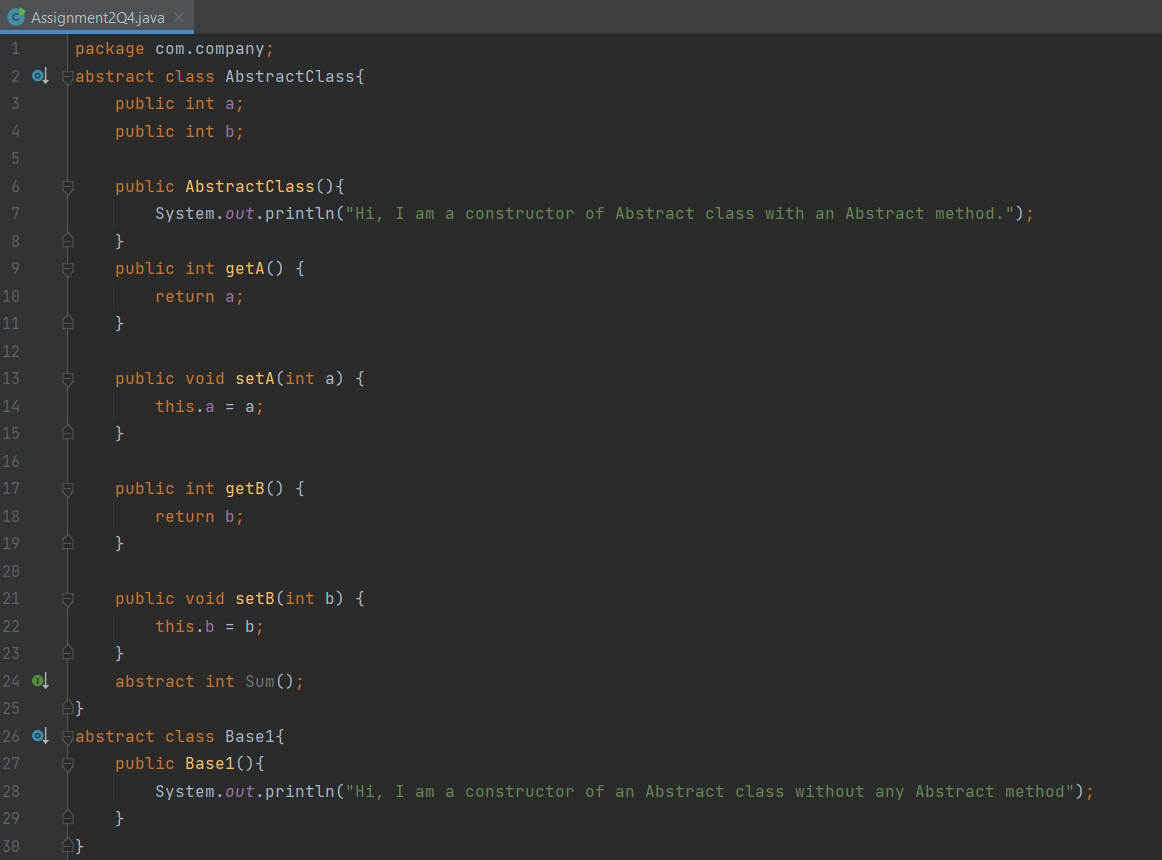
**• Abstract class cannot be final.**

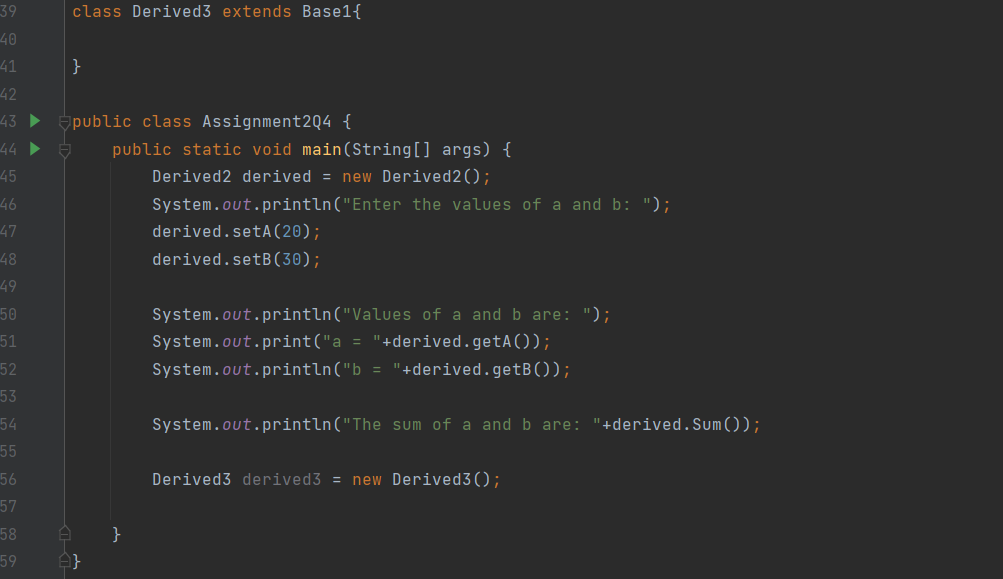
**• You can declare a class abstract without having any abstract method.**

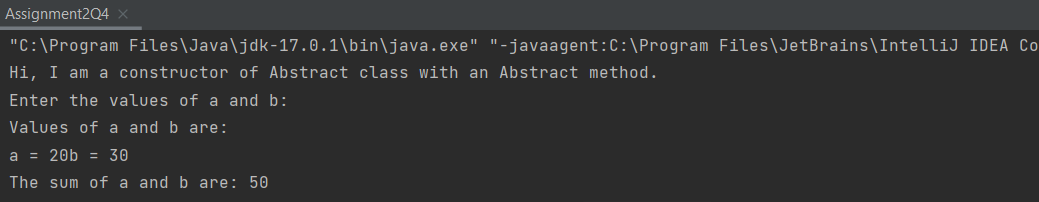
**Description:-**

Write a program in such a way that all the conditions above for abstract class should satisfy.

**Code:**







**Q5. Write the classes Line, Rectangle, Cube etc. & make the Shape as their base class. Add an abstract draw() method in the class Shape & draw all shapes.**

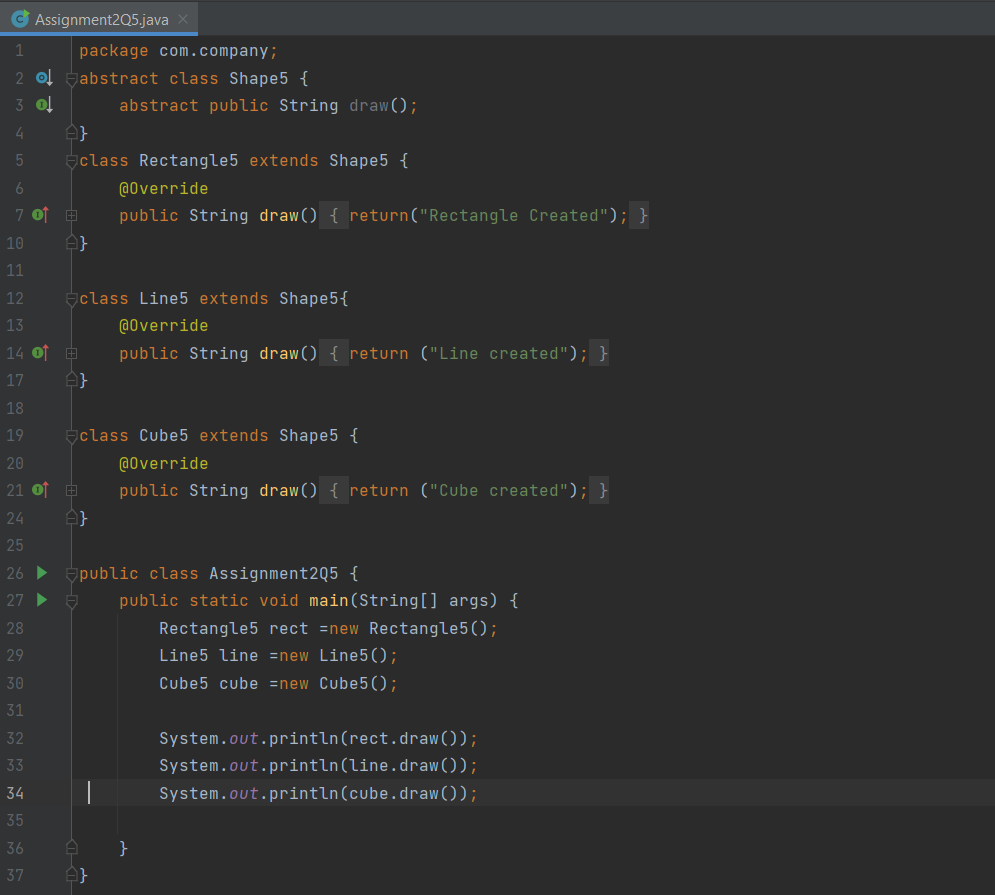
**Description:-**

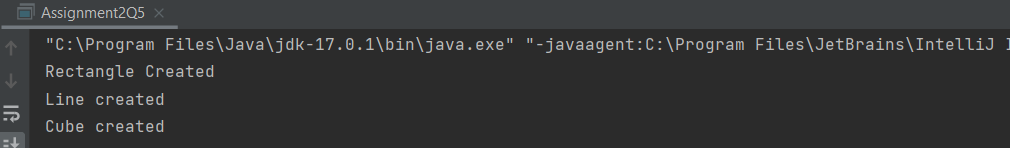
Write a java class named Shape which has abstract draw() and also classes Line, Rectangle, Cube etc. extends Shape to implement the draw method.

**Specifications:-**

class Rectangle5 extends Shape5 {  
    @Override  
    public String draw() {}  
}  
  
class Line5 extends Shape5{  
    @Override  
    public String draw() {}  
}  
  
class Cube5 extends Shape5 {  
    @Override  
    public String draw() {}  
}  
abstract class Shape5 {  
    abstract public String draw();  
}  
public class Assignment2Q5{  
    public static void main(String[] args) {}  
}

**Code:**





**Q6. Write an abstract class ‘Persistence’ along with two sub classes ‘FilePersistence’ & ‘DatabasePersistence’. The base class with have an abstract method persist() which will be overridden by its sub classes. Write a client who gets the Persistence object at runtime & invokes persist() method on it without knowing whether data is being saved in File or in Database.**

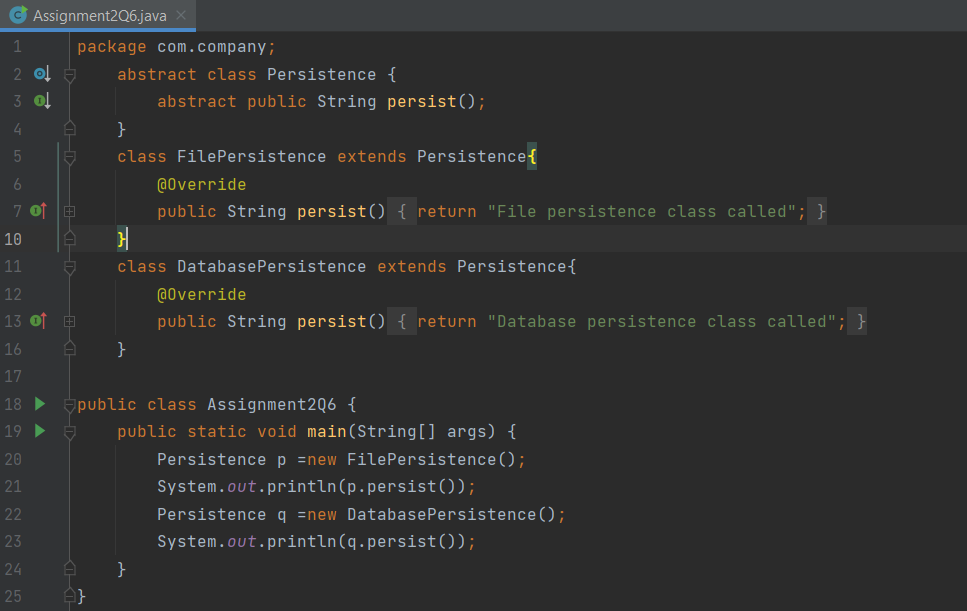
**Description:-**

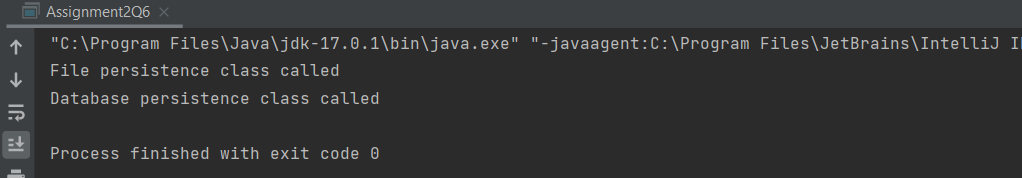
Write a program having an abstract class "Persistence" which have two child classes "FilePersistence","DatabasePersistence".The base class will have a persist() method and it is overridden by sub classes.wite a seperate class of client which will get persistence object and invoke persist method on it.(Polymorphism)

**Specifications:**

abstract class Persistence {  
    abstract public String persist();  
}  
class FilePersistence extends Persistence{  
    @Override  
    public String persist() {}  
}  
class DatabasePersistence extends Persistence{  
    @Override  
    public String persist() {}  
}  
  
public class Assignment2Q6 {  
    public static void main(String[] args) {}  
}

**Code:**





**Q7. Develop an application for Dessert shop. The application should allow owner to add items like Candy, Cookie or Ice Cream in the shop storage. Also, customers should be able to place an order.**

**DessertItem is an abstract class having an abstract method getCost(). Every dessert item has tax associated. Candy item is sold in dollar currency, Cookie in Euro currency & Ice Cream in Rupees currency. The sub classes are supposed to override these methods. When we run the application, it should ask us our role i.e. owner or customer. If role is owner, we should be able to add dessert items in our storage. If role is customer, then we should be able to place an order. The currency conversion rates are:**

**1 dollar = 60 rupees.**

**1 euro = 70 rupees.**

**Specifications:-**

abstract class DesertItem {

    abstract public int getCost();  
}  
class Candy extends DesertItem {  
    public int addCandies(int candies){}  
}  
class Cookie extends DesertItem {  
    public int addCookies(int candies){}  
}  
class IceCream extends DesertItem {  
    public int addIceCreams(int candies){}  
}  
public class Assignment2Q7 {  
    public static void main(String[] args) {}  
    private void selectRoles(){}  
    private void roles(String role){}  
    private void addItems() {}  
    private void addItemsOperation(int choice) {}  
    private void placeOrder() {}  
    private void placeOrderOperation(int choice) {}  
}

**Code:**

