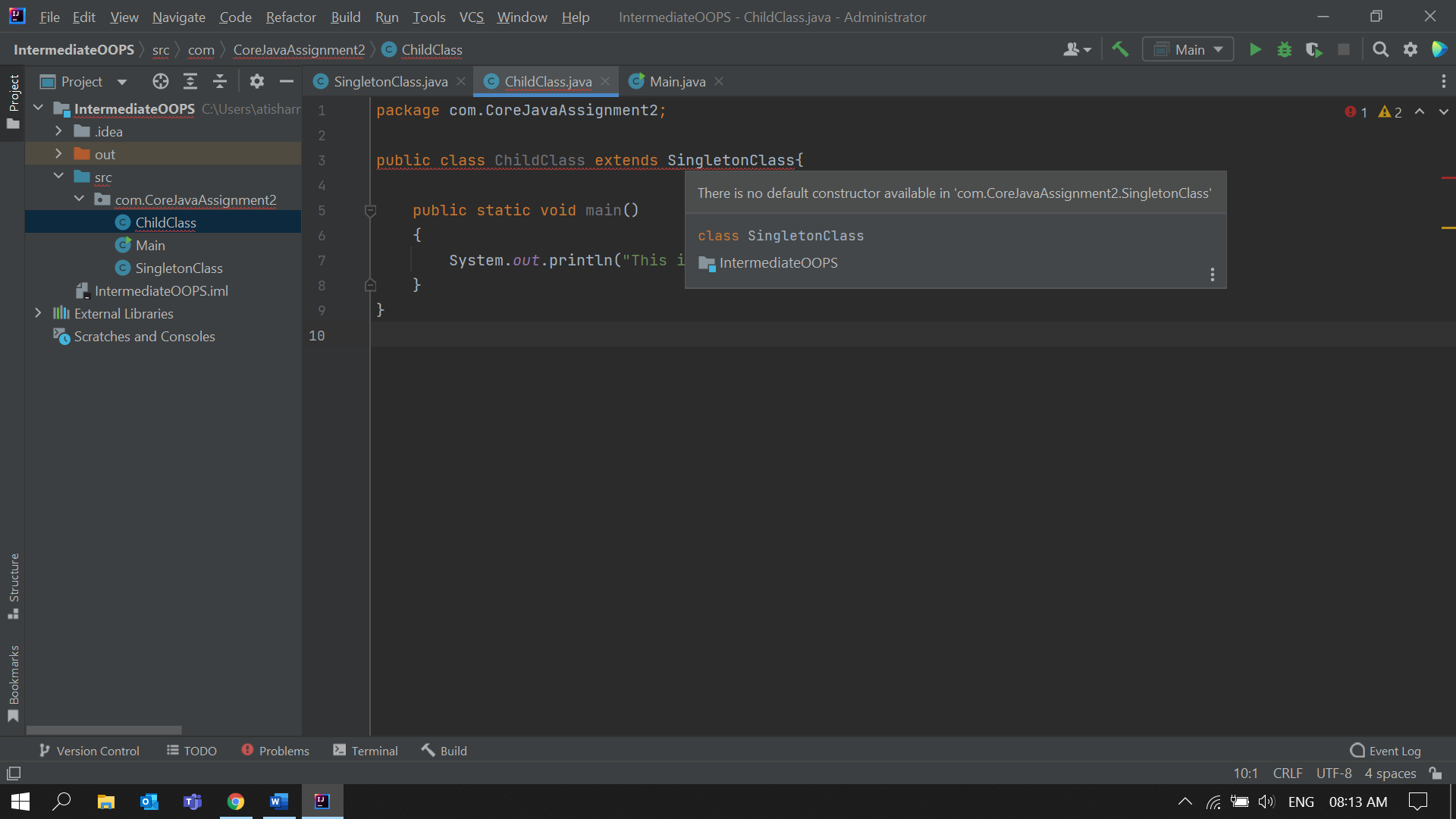
**CORE JAVA ASSIGNMENT 2 – Intermediate OOPS**

1. Write a singleton class. Confirm that singleton class cannot be inherited

**Solution:**

package com.CoreJavaAssignment2;  
  
class SingletonClass {  
 private static SingletonClass *instance*;  
  
 private SingletonClass()  
 {  
 System.*out*.println("This is the Constructor from Singleton Class");  
 }  
  
 public static SingletonClass getInstance()  
 {  
 if(*instance* == null)  
 {  
 *instance* = new SingletonClass();  
 }  
 return *instance*;  
 }  
}  
  
**In ChildClass.java**



2. 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.

**Solution:**Employee.java

package com.OrganizationHierarchy;  
  
public class Employee {  
  
 String name;  
 int age;  
 double salary;  
 public Employee(String name , int age , double salary )  
 {  
 this.name = name;  
 this.age = age;  
 this.salary = salary;  
 }  
 void getTotalSalary()  
 {  
 System.*out*.println("Total Salary of Employee is : " + salary );  
 }  
}  
  
Manager.java

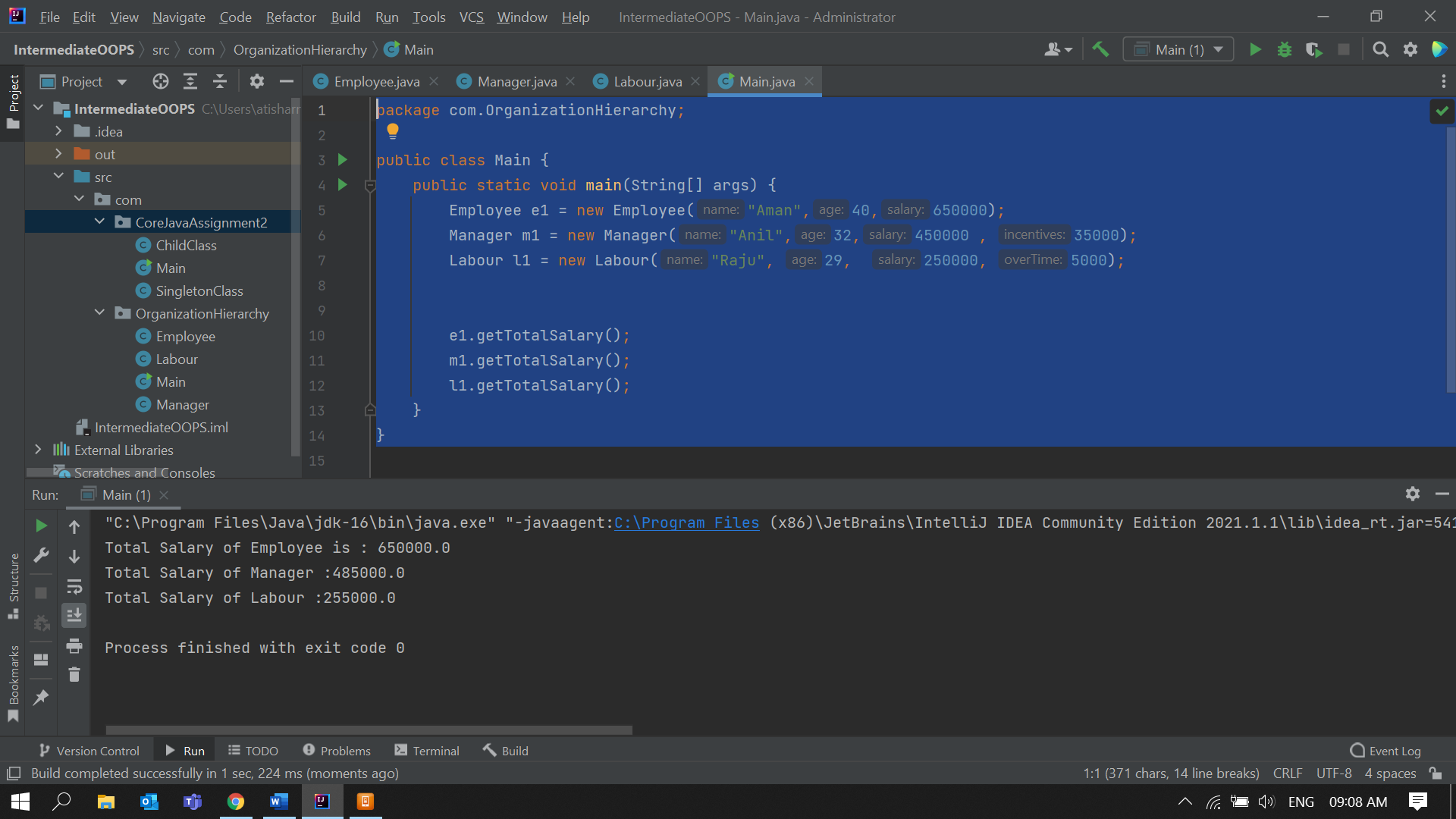
package com.OrganizationHierarchy;  
  
public class Manager extends Employee  
{  
 double incentives;  
 public Manager(String name , int age , double salary , double incentives)  
 {  
 super(name , age , salary);  
 this.incentives = incentives;  
 }  
 void getTotalSalary()  
 {  
 this.salary += incentives;  
 System.*out*.println("Total Salary of Manager :" + salary);  
 }  
  
}

Labour.java

package com.OrganizationHierarchy;  
  
public class Labour extends Employee{  
  
 double overTime;  
  
 public Labour(String name , int age , double salary , double overTime)  
 {  
 super(name , age , salary);  
 this.overTime = overTime;  
 }  
  
 void getTotalSalary()  
 {  
 this.salary += overTime;  
 System.*out*.println("Total Salary of Labour :" + salary);  
 }  
}

Main.java  
  
package com.OrganizationHierarchy;  
  
public class Main {  
 public static void main(String[] args) {  
 Employee e1 = new Employee("Aman",40,650000);  
 Manager m1 = new Manager("Anil",32,450000 , 35000);  
 Labour l1 = new Labour("Raju", 29, 250000, 5000);  
  
  
 e1.getTotalSalary();  
 m1.getTotalSalary();  
 l1.getTotalSalary();  
 }  
}

Output:

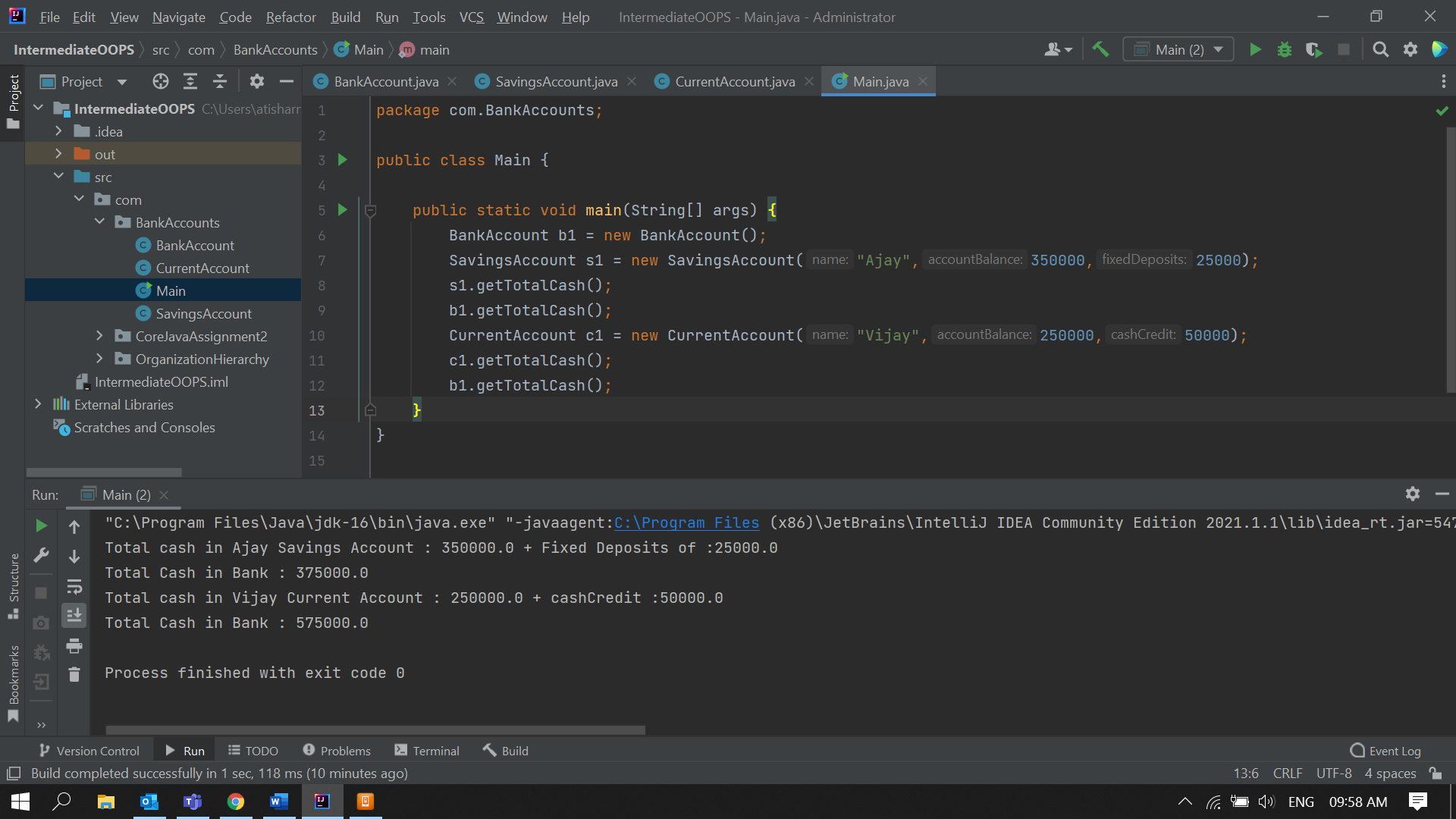
  
  
  
3. Write a program to consider saving & current account in the bank. Saving account holder has

'Fixed Deposits' whereas Current account holder has cash credit. Apply polymorphism to find

out total cash in the bank.  
  
**Solution:**BankAccount.java  
  
package com.BankAccounts;  
  
public class BankAccount {  
 String name;  
 double accountBalance;  
 static double *totalCashInBank*;  
 public void getTotalCash()  
 {  
 System.*out*.println("Total Cash in Bank : " + *totalCashInBank*);  
 }  
}  
  
SavingsAccount.java  
  
package com.BankAccounts;  
public class SavingsAccount extends BankAccount{  
 double fixedDeposits;  
 public SavingsAccount(String name, double accountBalance , double fixedDeposits )  
 {  
 this.name = name;  
 this.accountBalance = accountBalance;  
 this.fixedDeposits = fixedDeposits;  
 BankAccount.*totalCashInBank* += this.fixedDeposits + this.accountBalance;  
 }  
 public void getTotalCash()  
 {  
 System.*out*.println("Total cash in " + this.name + " Savings Account : " + accountBalance + " + FixedDeposits of :" + fixedDeposits);  
 }  
}

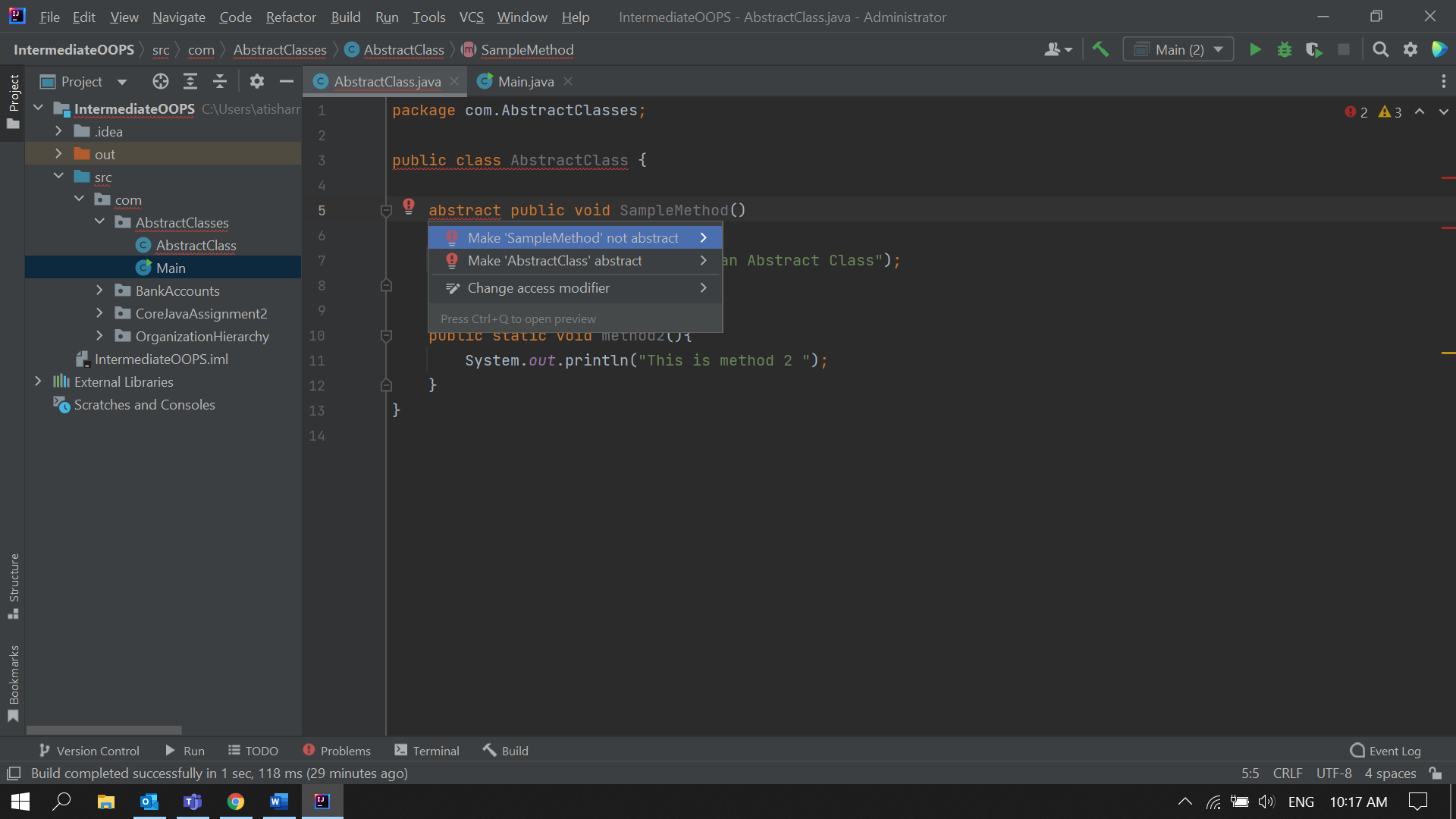
CurrentAccount.java

package com.BankAccounts;  
public class CurrentAccount extends BankAccount{  
 double cashCredit;  
 public CurrentAccount(String name, double accountBalance , double cashCredit )  
 {  
 this.name = name;  
 this.accountBalance = accountBalance;  
 BankAccount.*totalCashInBank* += this.accountBalance ;  
 this.cashCredit = cashCredit;  
 BankAccount.*totalCashInBank* -= this.cashCredit ;  
 }  
 public void getTotalCash()  
 {  
 System.*out*.println("Total cash in " + this.name + " Current Account : " + accountBalance + " + cashCredit :" +cashCredit);  
 }  
}  
  
Main.java  
  
package com.BankAccounts;  
public class Main {  
 public static void main(String[] args) {  
 BankAccount b1 = new BankAccount();  
 SavingsAccount s1 = new SavingsAccount("Ajay",350000,25000);  
 s1.getTotalCash();  
 b1.getTotalCash();  
 CurrentAccount c1 = new CurrentAccount("Vijay",250000,50000);  
 c1.getTotalCash();  
 b1.getTotalCash();  
 }  
}  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Output:**



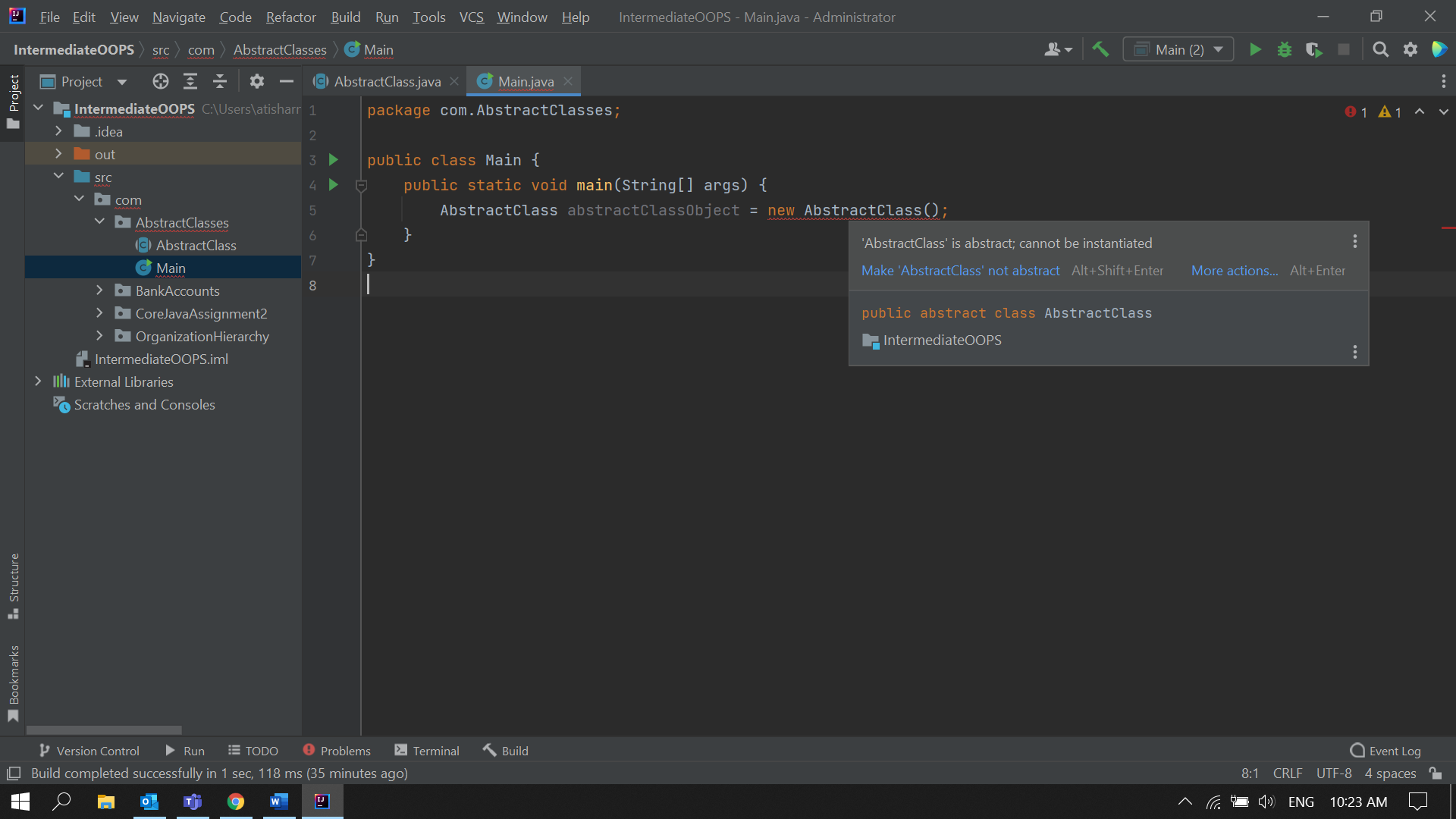
4. Test the following principles of an abstract class:   
  
• If any class has any of its method abstract then you must declare entire class abstract.

**Solution:**AbstractClass.javapackage com.AbstractClasses;  
  
public class AbstractClass {  
  
 abstract public void SampleMethod()  
 {  
 System.*out*.println("This is an Abstract Class");  
 }  
   
 public static void method2(){  
 System.*out*.println("This is method 2 ");  
 }  
}  
  
Error:



• Abstract class cannot be instantiated.   
  
**Solution:**  
  
AbstractClass.java  
  
package com.AbstractClasses;  
  
abstract public class AbstractClass {  
  
 public void SampleMethod()  
 {  
 System.*out*.println("This is an Abstract Class");  
 }  
  
 public static void method2(){  
 System.*out*.println("This is method 2 ");  
 }  
}

Main.java  
  
package com.AbstractClasses;  
  
public class Main {  
 public static void main(String[] args) {  
 AbstractClass abstractClassObject = new AbstractClass();  
 }  
}  
  
**Error:**



• When we extend an abstract class, we must either override all the abstract methods in sub class or declare subclass as abstract.   
  
**Solution:**  
**Case 1: Overrided all the abstract methods of AbstractClass in ChildClass.**  
   
AbstractClass.java

package com.AbstractClasses;  
  
public abstract class AbstractClass {

AbstractClass(){  
 System.*out*.println("AbstractClass object is created");  
 }

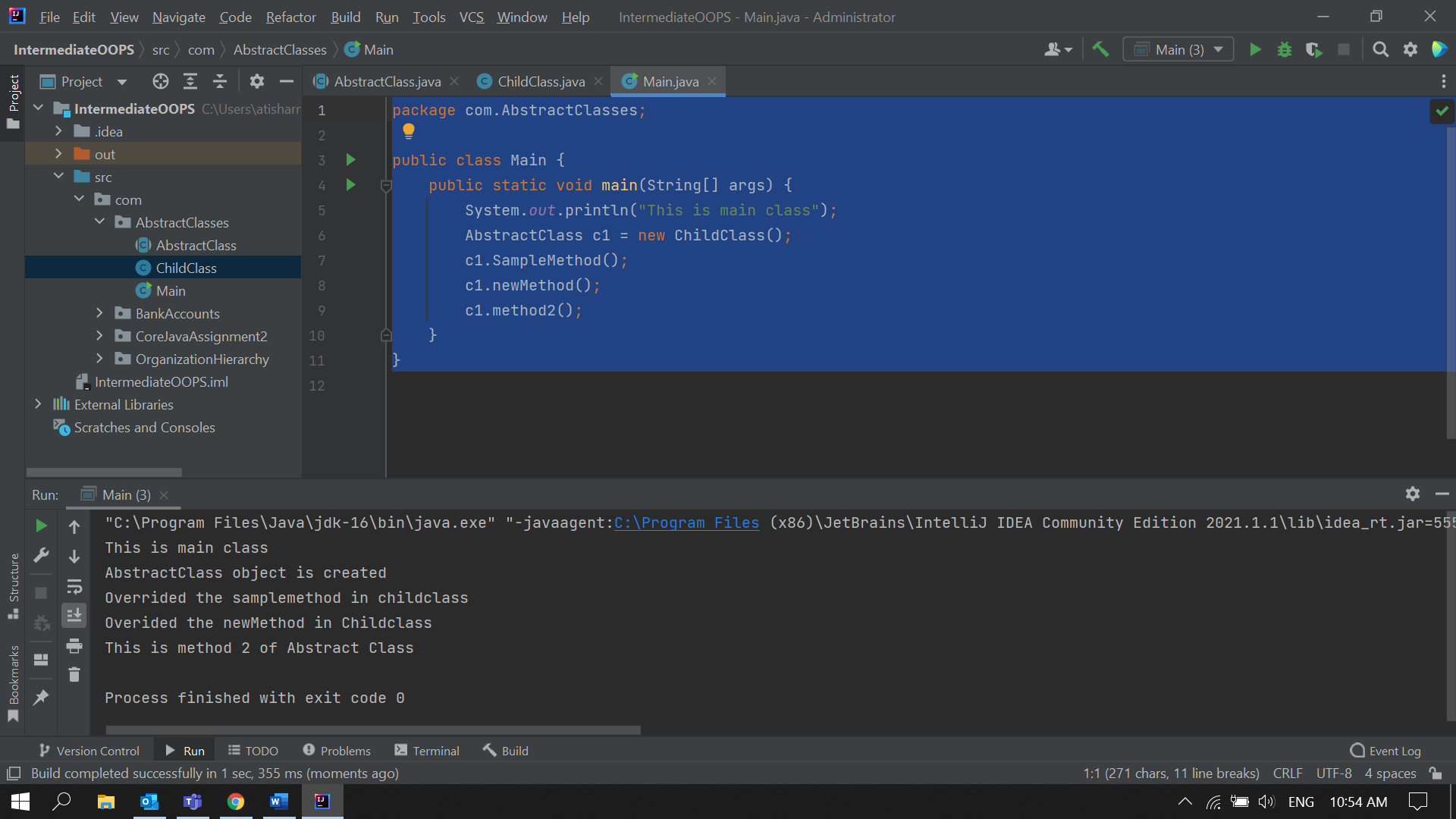
abstract public void SampleMethod();

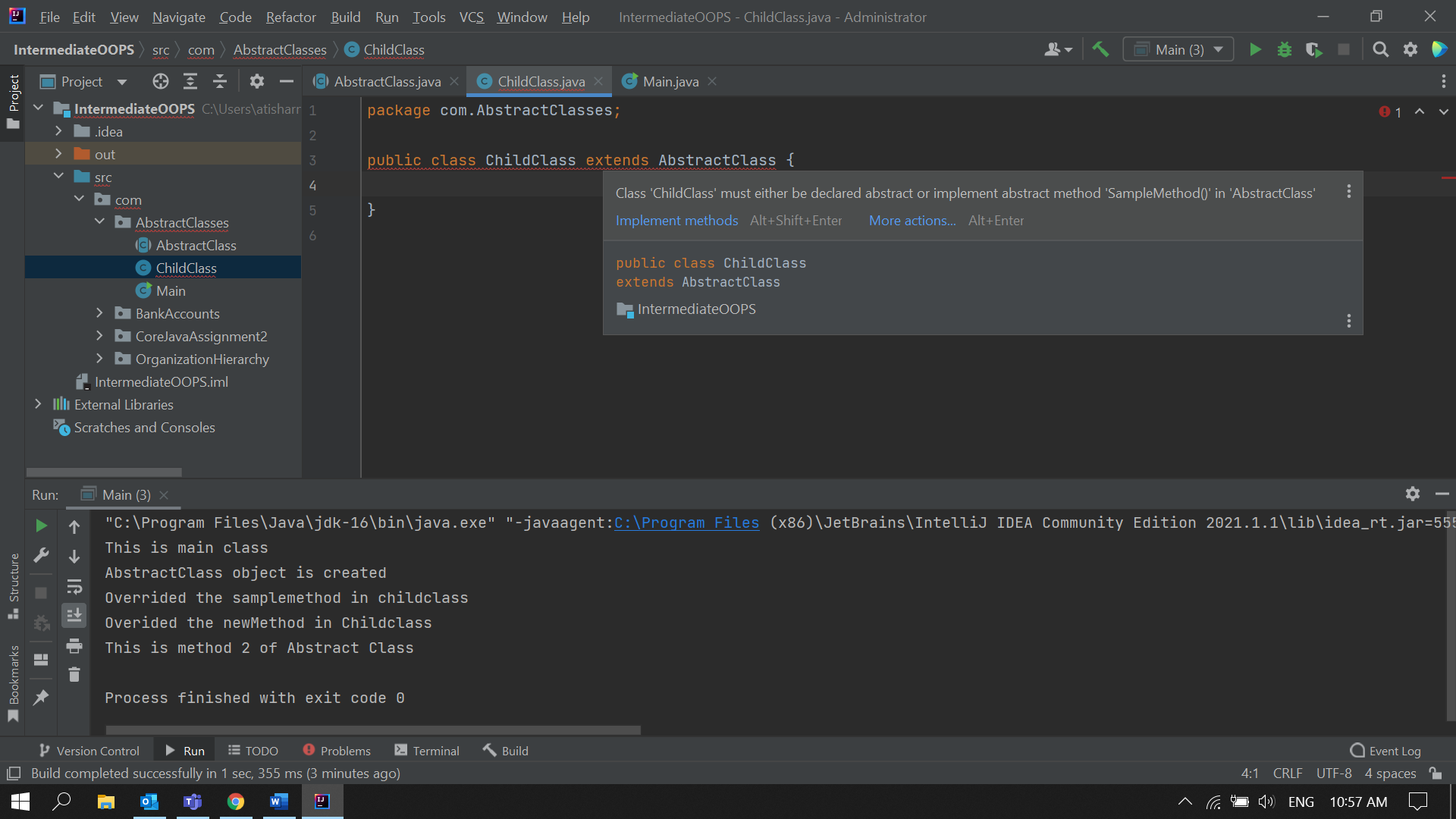
abstract public void newMethod();

public void method2(){  
 System.*out*.println("This is method 2 of Abstract Class");  
 }  
}

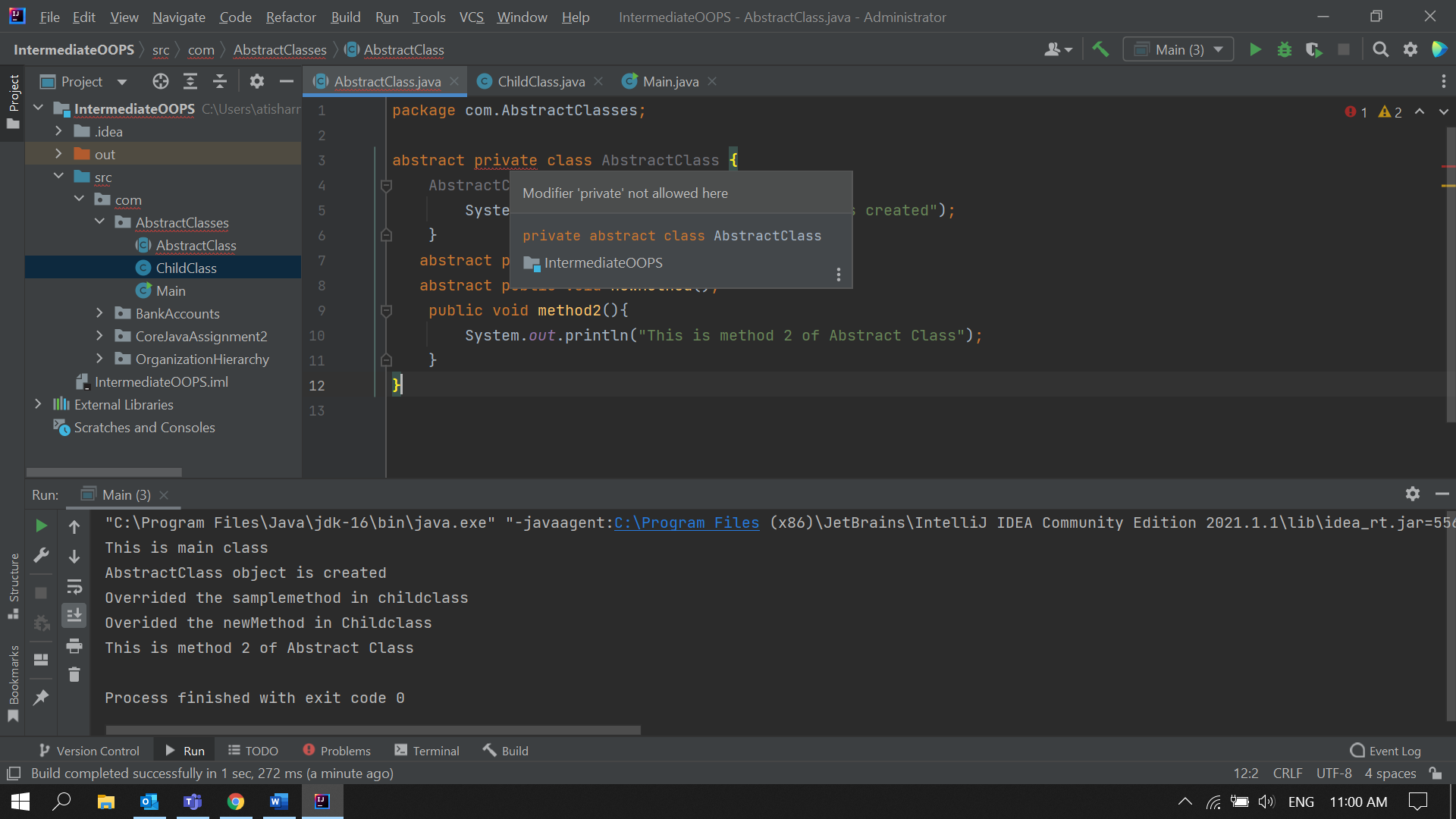
ChildClass.java  
  
package com.AbstractClasses;  
  
public class ChildClass extends AbstractClass {  
  
 @Override  
 public void SampleMethod() {  
 System.*out*.println("Overrided the samplemethod in childclass");  
 }  
 @Override  
 public void newMethod()  
 {  
 System.*out*.println("Overided the newMethod in Childclass");  
 }  
  
}  
  
  
Main.java  
  
package com.AbstractClasses;  
  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("This is main class");  
 AbstractClass c1 = new ChildClass();  
 c1.SampleMethod();  
 c1.newMethod();  
 c1.method2();  
 }  
}

**Output:**

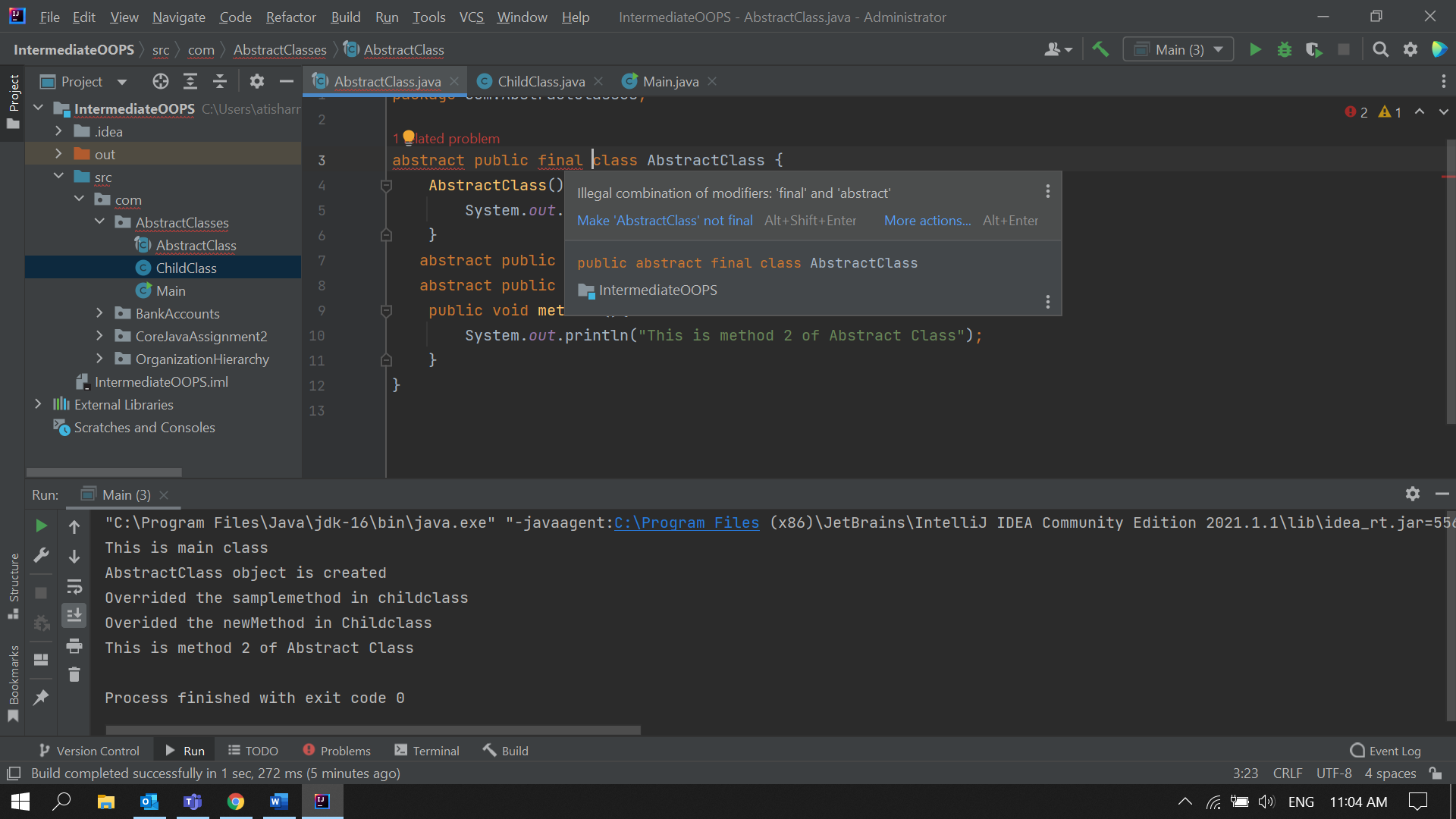
  
  
  
**Case 2: Not Overrided any abstract methods of AbstractClass in ChildClass.**  
  
**Output:**



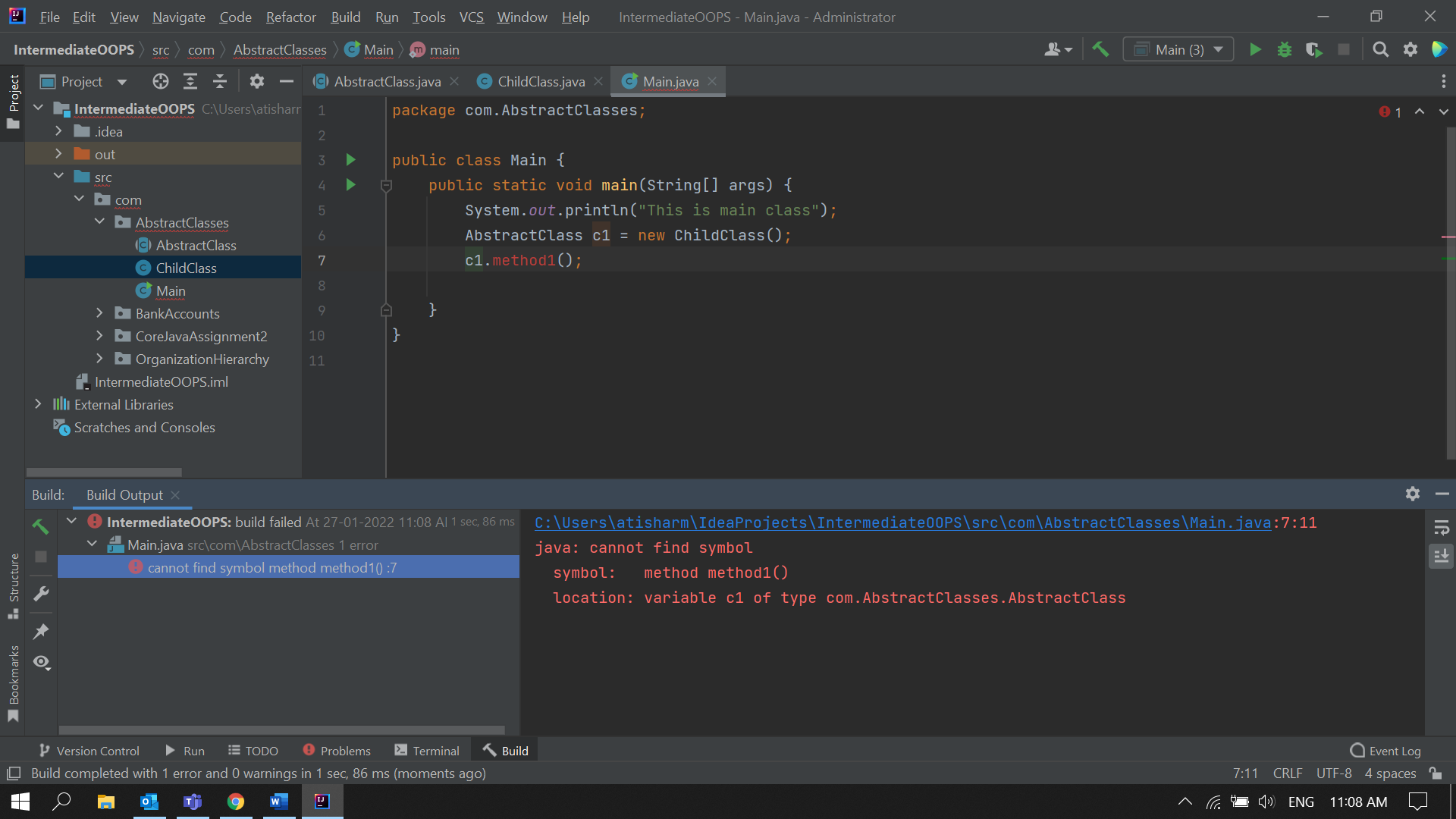
• Abstract class cannot be private.



• Abstract class cannot be final.

  
  
• You can declare a class abstract without having any abstract method.  
  
**Solution:**  
Abstract.java  
  
package com.AbstractClasses;  
  
abstract public class AbstractClass {  
  
}

ChildClass.java  
  
package com.AbstractClasses;  
  
public class ChildClass extends AbstractClass {  
  
 public static void method1()  
 {  
 System.*out*.println("Method of ChildClass");  
 }  
  
}  
  
  
Main.java  
  
package com.AbstractClasses;  
  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("This is main class");  
 AbstractClass c1 = new ChildClass();  
 c1.method1();  
  
 }  
}

**Output:**

5. 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.  
  
**Solution:**Shape.java

package com.DrawShapes;  
abstract public class Shape {  
 abstract public void draw();  
}  
  
Line.java

package com.DrawShapes;  
public class Line extends Shape{  
 public void draw() {  
 System.*out*.println("Method to draw a Line");  
 }  
}

Rectangle.java

package com.DrawShapes;  
public class Rectangle extends Shape{  
 public void draw()  
 {  
 System.*out*.println("Method to Draw a Rectangle");  
 }  
}

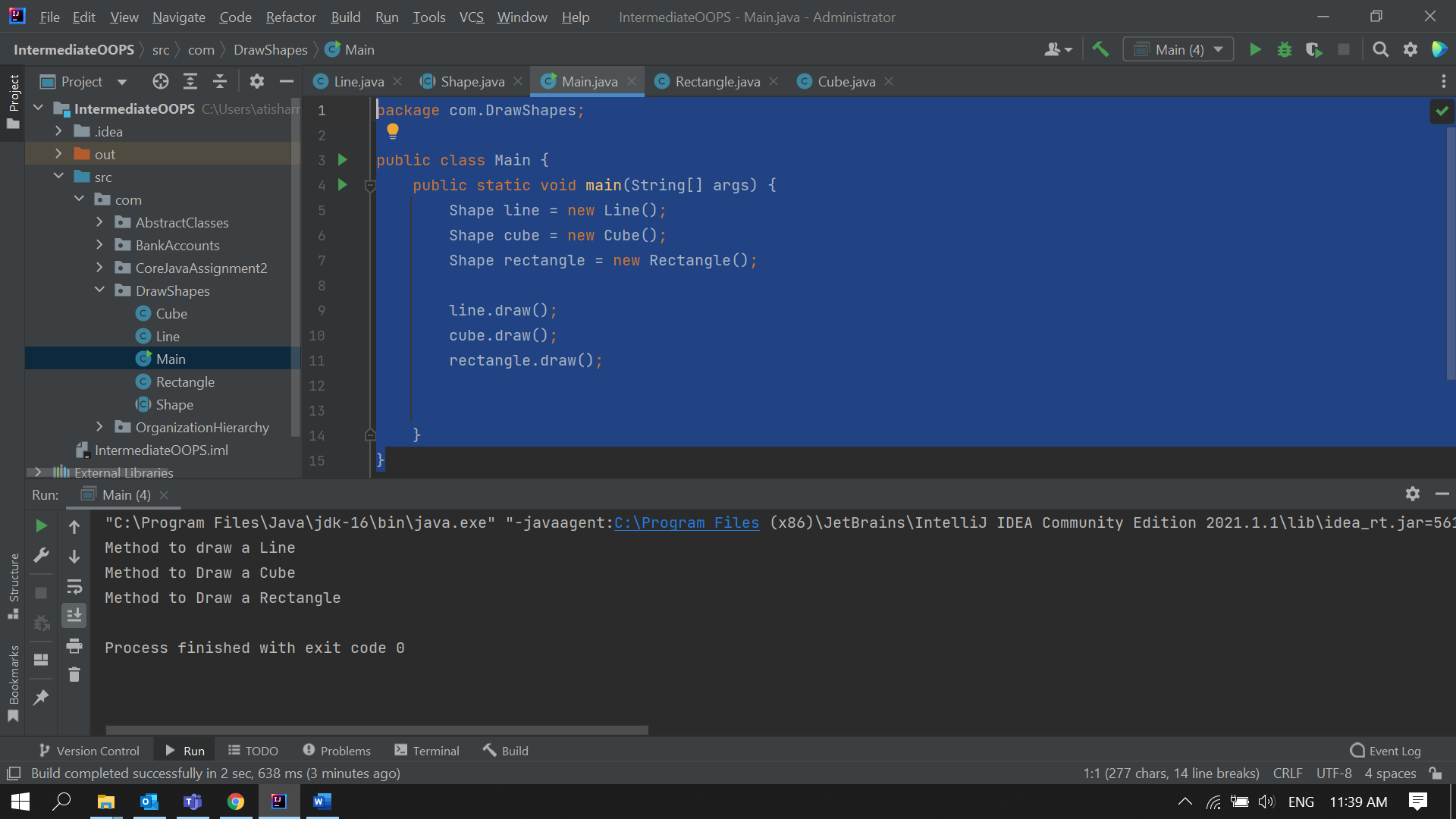
Cube.java

package com.DrawShapes;  
public class Rectangle extends Shape{  
 public void draw()  
 {  
 System.*out*.println("Method to Draw a Rectangle");  
 }  
}

Main.java

package com.DrawShapes;  
  
public class Main {  
 public static void main(String[] args) {  
 Shape line = new Line();  
 Shape cube = new Cube();  
 Shape rectangle = new Rectangle();  
  
 line.draw();  
 cube.draw();  
 rectangle.draw();  
  
  
 }  
}

**Output:**



6. 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.  
  
**Solution:**

Persistance.java

package com.Persistance;  
  
abstract public class Persistance {  
  
 abstract void persist(int input);  
  
}

FilePersistance.java

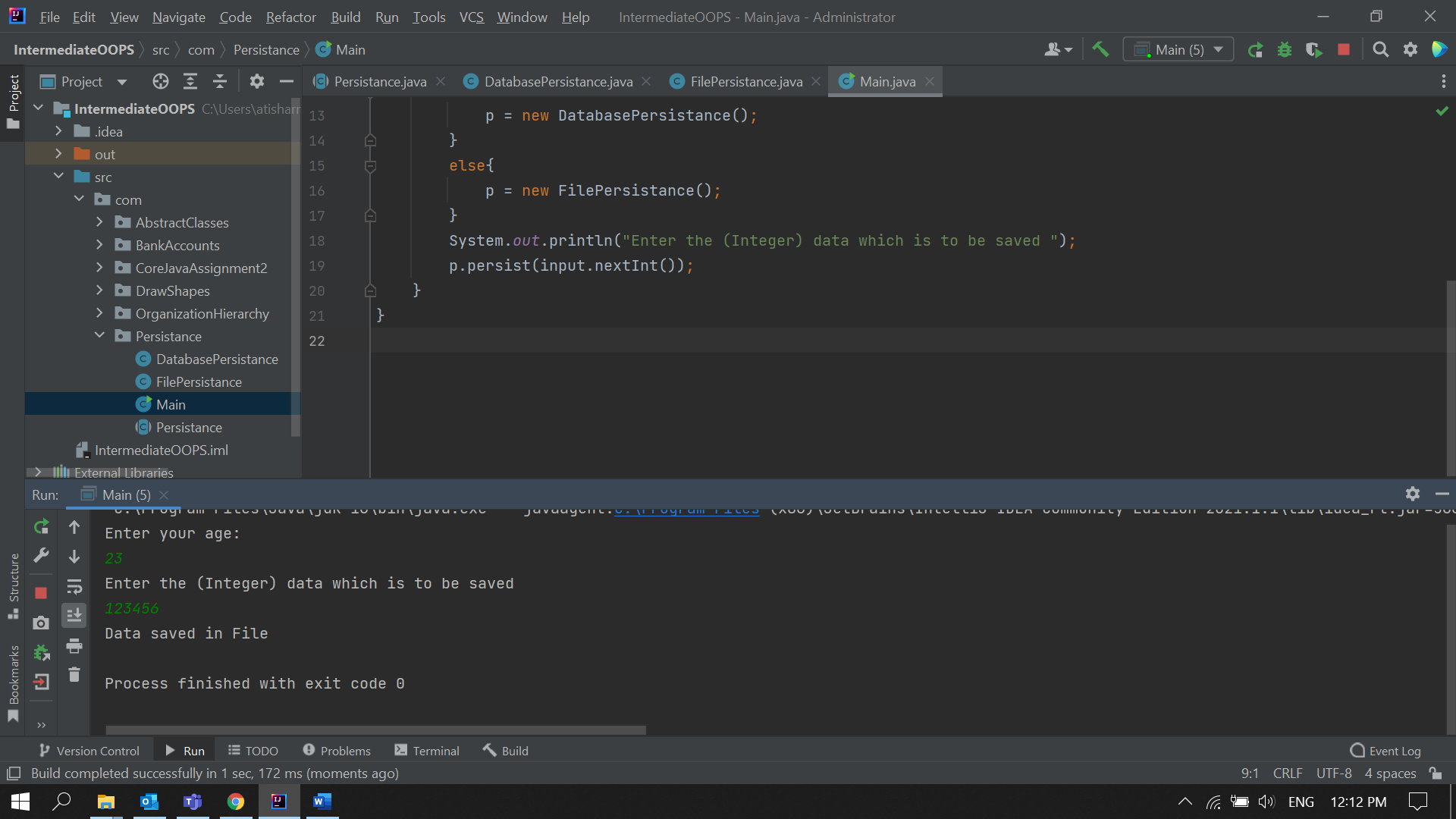
package com.Persistance;  
  
public class FilePersistance extends Persistance{  
  
 int input;  
 void persist(int input)  
 {  
 this.input = input;  
 System.*out*.println("Data saved in File");  
 }  
}

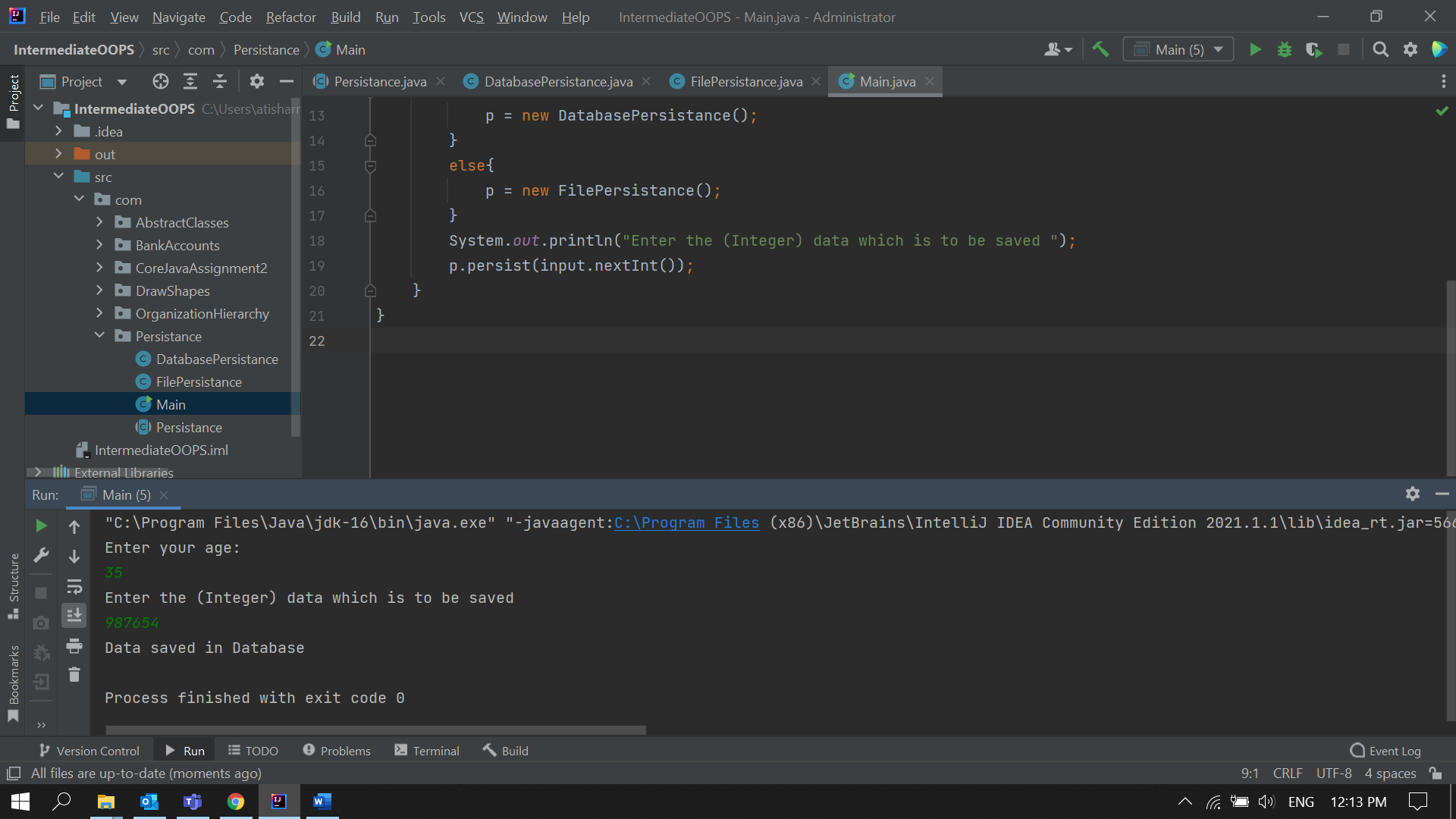
DatabasePersistance.java

package com.Persistance;  
public class DatabasePersistance extends Persistance{  
 int input;  
 void persist(int input)  
 {  
 this.input = input;  
 System.*out*.println("Data saved in Database");} }

Main.java

package com.Persistance;  
  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner input = new Scanner(System.*in*);  
 System.*out*.println("Enter your age: ");  
 int Age = input.nextInt();  
 Persistance p;  
 if(Age > 25)  
 {  
 p = new DatabasePersistance();  
 }  
 else{  
 p = new FilePersistance();  
 }  
 System.*out*.println("Enter the (Integer) data which is to be saved ");  
 p.persist(input.nextInt());  
 }  
}  
  
**Output:**





7. 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 & IceCream 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.  
  
  
**Solution:**

**Main.java**

package com.DessertShop;  
  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner input = new Scanner(System.*in*);  
 int choice;  
 do{  
 *DisplayMenu*();  
 choice = input.nextInt();  
 switch (choice) {  
 case 1:  
 Owner.*AddItems*();  
 break;  
 case 2:  
 Customer.*orderItems*();  
 break;  
 case 3:  
 System.*out*.println("Bye");  
 break;  
 default:  
 System.*out*.println("Invalid Input ! Choose a correct Role. Try Again");  
 break;  
 }  
 }while (choice!=3);  
 }  
  
 public static void DisplayMenu()  
 {  
 System.*out*.println();  
 System.*out*.println("Welcome to DessertShop!");  
 System.*out*.println("Choose your Role: ");  
 System.*out*.println("1. Owner ");  
 System.*out*.println("2. Customer ");  
 System.*out*.println("3. Exit ");  
 System.*out*.print("Enter your Choice: ");  
 }  
}

**Owner.java**  
  
package com.DessertShop;  
  
import java.util.Scanner;  
  
public class Owner {  
 public static void AddItems()  
 {  
 Scanner input = new Scanner(System.*in*);  
 System.*out*.println();  
 System.*out*.println("Choose the Item to Add: ");  
 System.*out*.println("1. Candy ");  
 System.*out*.println("2. Cookie ");  
 System.*out*.println("3. Ice Cream ");  
 System.*out*.print("Enter your Choice: ");  
 int choice = input.nextInt();  
 switch (choice) {  
 case 1:  
 *AddTheItem*("Candy");break;  
 case 2:  
 *AddTheItem*("Cookie");break;  
 case 3:  
 *AddTheItem*("IceCream");break;  
 }  
 }  
  
 public static void AddTheItem(String ItemName)  
 {  
 Scanner item = new Scanner(System.*in*);  
 if(ItemName.equals("Candy"))  
 {  
 *AddItemDetails*("Candy");  
 DessertItem Candy = new Candy(item.next(), item.nextDouble());  
 }  
 else if(ItemName.equals("Cookie"))  
 {  
 *AddItemDetails*("Cookie");  
 DessertItem Candy = new Cookie(item.next(), item.nextDouble());  
 }  
 else if(ItemName.equals("IceCream"))  
 {  
 *AddItemDetails*("IceCream");  
 DessertItem Candy = new IceCream(item.next(), item.nextDouble());  
 }  
 }  
  
 private static void AddItemDetails(String ItemName)  
 {  
 System.*out*.println();  
 System.*out*.print("Enter the Name of " + ItemName + " and Cost(in Rupees) : ");  
 }  
  
}

**Customer.java**

package com.DessertShop;  
import java.util.Scanner;  
public class Customer {  
 public static void orderItems()  
 {  
 Scanner input = new Scanner(System.*in*);  
 System.*out*.println();  
 System.*out*.println("Available Dessert Choices : ");  
 System.*out*.println("1. Candy ");  
 System.*out*.println("2. Cookie ");  
 System.*out*.println("3. Ice Cream ");  
 System.*out*.println("4. Enter 4 to exit ");  
 System.*out*.print("Enter your Choice: ");  
 int choice = input.nextInt();  
 switch (choice) {  
 case 1:  
 *itemSelected*("Candy");break;  
 case 2:  
 *itemSelected*("Cookie");break;  
 case 3:  
 *itemSelected*("IceCream");break;  
 }  
 }  
  
 public static void itemSelected(String ItemName)  
 {  
 if(ItemName.equals("Candy"))  
 Order.*orderItem*("Candy");  
 else if(ItemName.equals("Cookie"))  
 Order.*orderItem*("Cookie");  
 else if(ItemName.equals("IceCream"))  
 Order.*orderItem*("IceCream");  
 }  
}

**DessertItem.java**

package com.DessertShop;  
  
abstract public class DessertItem {  
 String itemName;  
 abstract public double getCost();  
}

**IceCream.java**

package com.DessertShop;  
  
public class IceCream extends DessertItem{  
 static double *cost*;  
  
 IceCream(){}  
 IceCream(String Name, double cost)  
 {  
 this.itemName = Name;  
 IceCream.*cost* = cost;  
 System.*out*.println(this.itemName + " has been added to Storage");  
 }  
  
 @Override  
 public double getCost() {  
 if(IceCream.*cost*==0)  
 {  
 System.*out*.println("NO stocks for IceCream Sorry!");  
 return 0;  
 }  
 return *cost*;  
 }  
}

**Candy.java**

package com.DessertShop;  
  
public class Candy extends DessertItem{  
 static double *cost*;  
  
 Candy(){}  
 Candy(String Name, double cost)  
 {  
 this.itemName = Name;  
 Candy.*cost* = (cost/60);  
 System.*out*.println(this.itemName + " has been added to Storage");  
 }  
  
 @Override  
 public double getCost() {  
 if(Candy.*cost*==0)  
 {  
 System.*out*.println("NO stocks for Candy Sorry!");  
 return 0;  
 }  
 return *cost*;  
 }  
}

**Cookie.java**

package com.DessertShop;  
  
public class Cookie extends DessertItem{  
 static double *cost*;  
  
 Cookie(){}  
 Cookie(String Name, double cost)  
 {  
 this.itemName = Name;  
 Cookie.*cost* = (cost/70);  
 System.*out*.println(this.itemName + " has been added to Storage");  
 }  
  
 @Override  
 public double getCost() {  
 if(Cookie.*cost*==0)  
 {  
 System.*out*.println("NO stocks for Cookie Sorry!");  
 return 0;  
 }  
 return *cost*;  
 }  
}

**Order.java**  
public class Order{  
 public static void orderItem(String ItemName)  
 {  
 DessertItem itemToOrder = null;  
 String currency = null;  
  
 if(ItemName.equals("Candy")) {  
 itemToOrder = new Candy();  
 currency = "Dollar";  
 }  
 else if(ItemName.equals("Cookie"))  
 {  
 itemToOrder = new Cookie();  
 currency = "Euros";  
 }  
 else if(ItemName.equals("IceCream")) {  
 itemToOrder = new IceCream();  
 currency = "Rupees";  
 }  
  
 if(itemToOrder.getCost() != 0)  
 {  
 System.*out*.println("Price: " + itemToOrder.getCost() + " " + currency );  
 System.*out*.println("Order has been placed for " + ItemName);  
 }  
 }  
}

**Output:**

