Object-Oriented Software Engineering

Instructor : Huang, Chuen-Min

Teamwork2 ver.1

Group 4

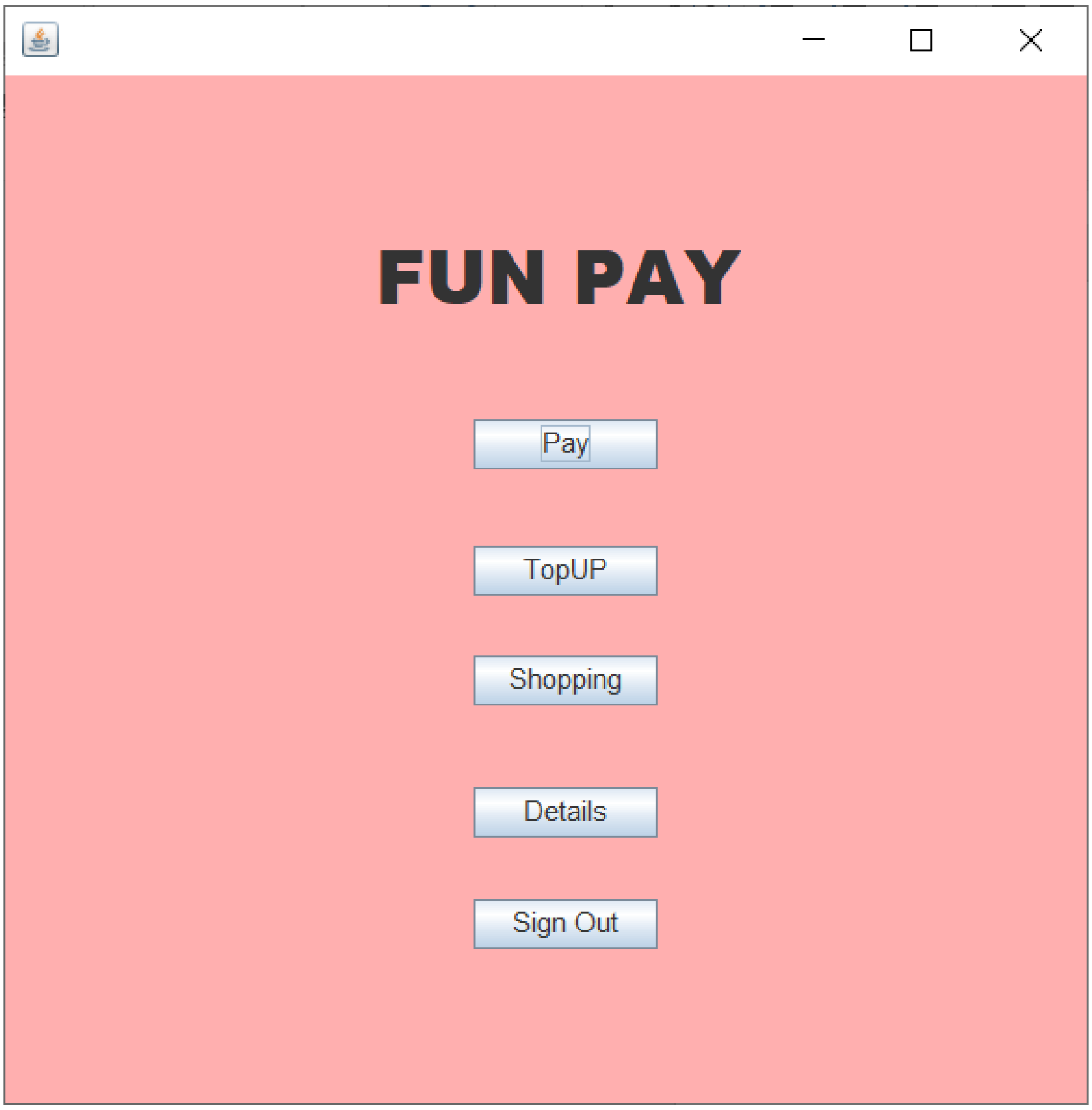
ID	Name
B10523038	Edward
B10523024	Steven
B10523032	Xavier
B10523033	Wing
B10523021	Johnny
B10523037	Yee
B10523006	Peggy
B10523007	Bess
B10523005	Aliss
B10523056	Sandy

Date 2019/ 12 / 14

Context

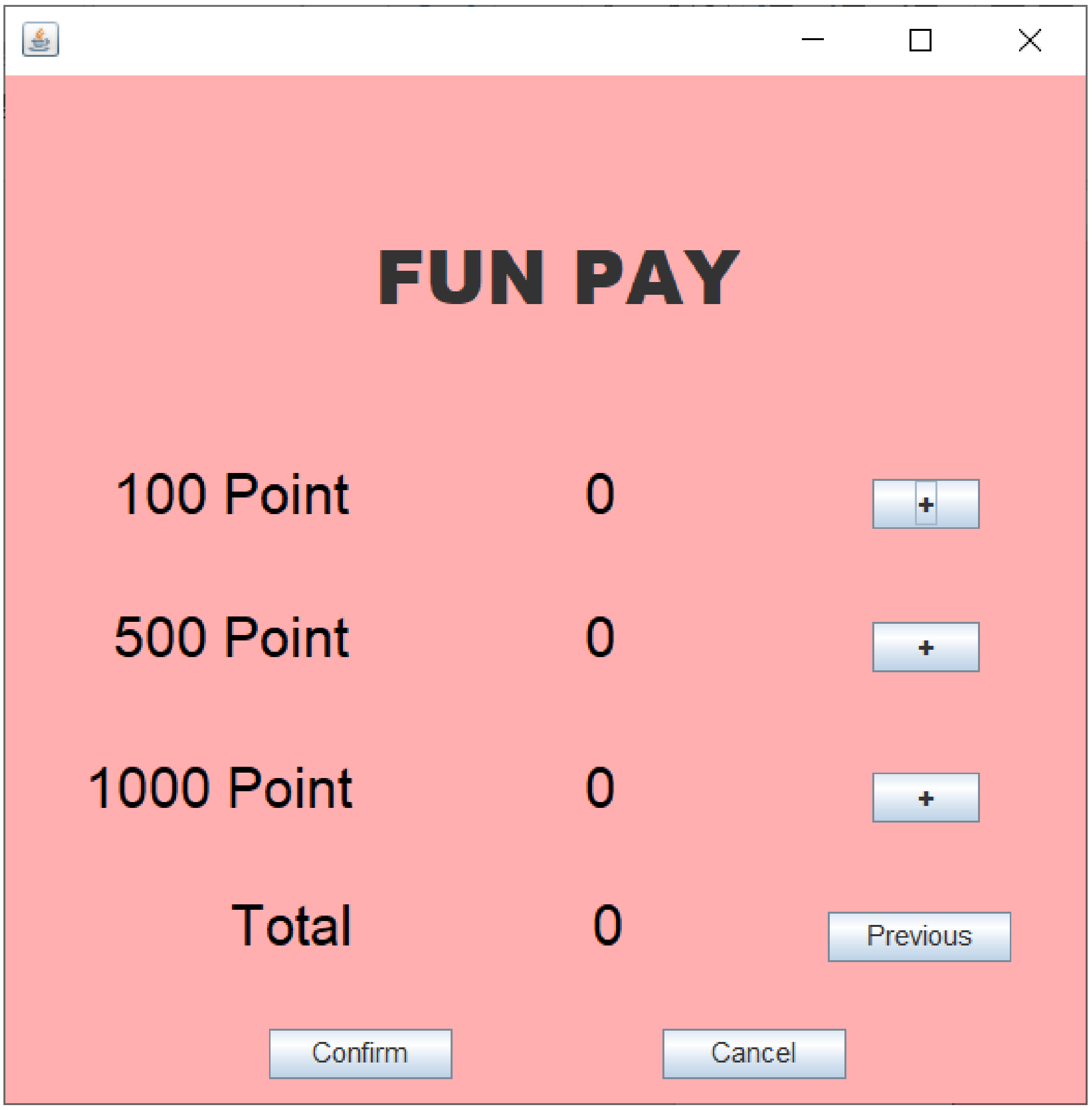
1) Display some snapshots of the new result in the report.

Main GUI



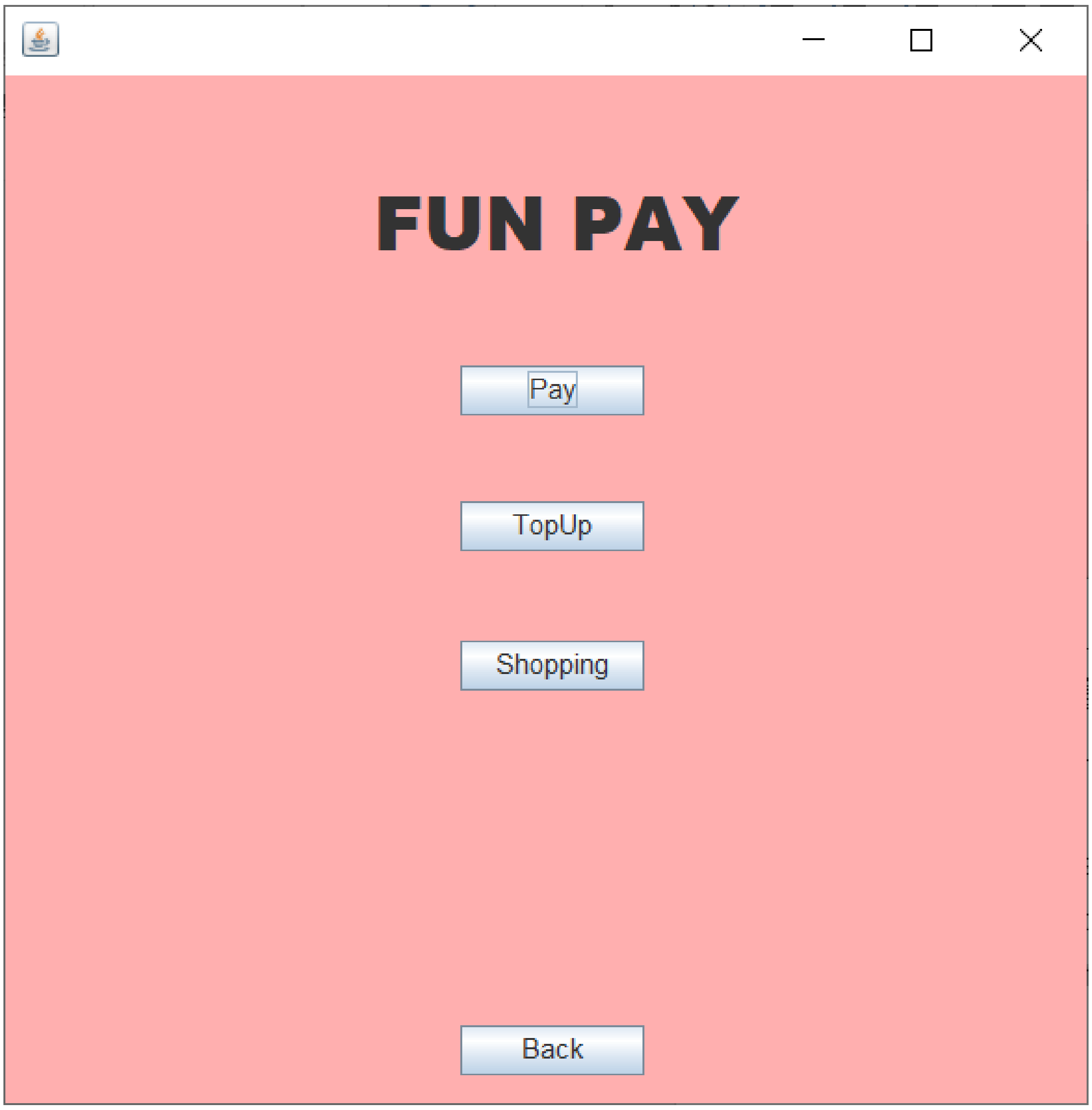
We have two new functions, including Shopping and Details. Shopping is used to purchase points, and Details is used to display various details.

Shopping GUI



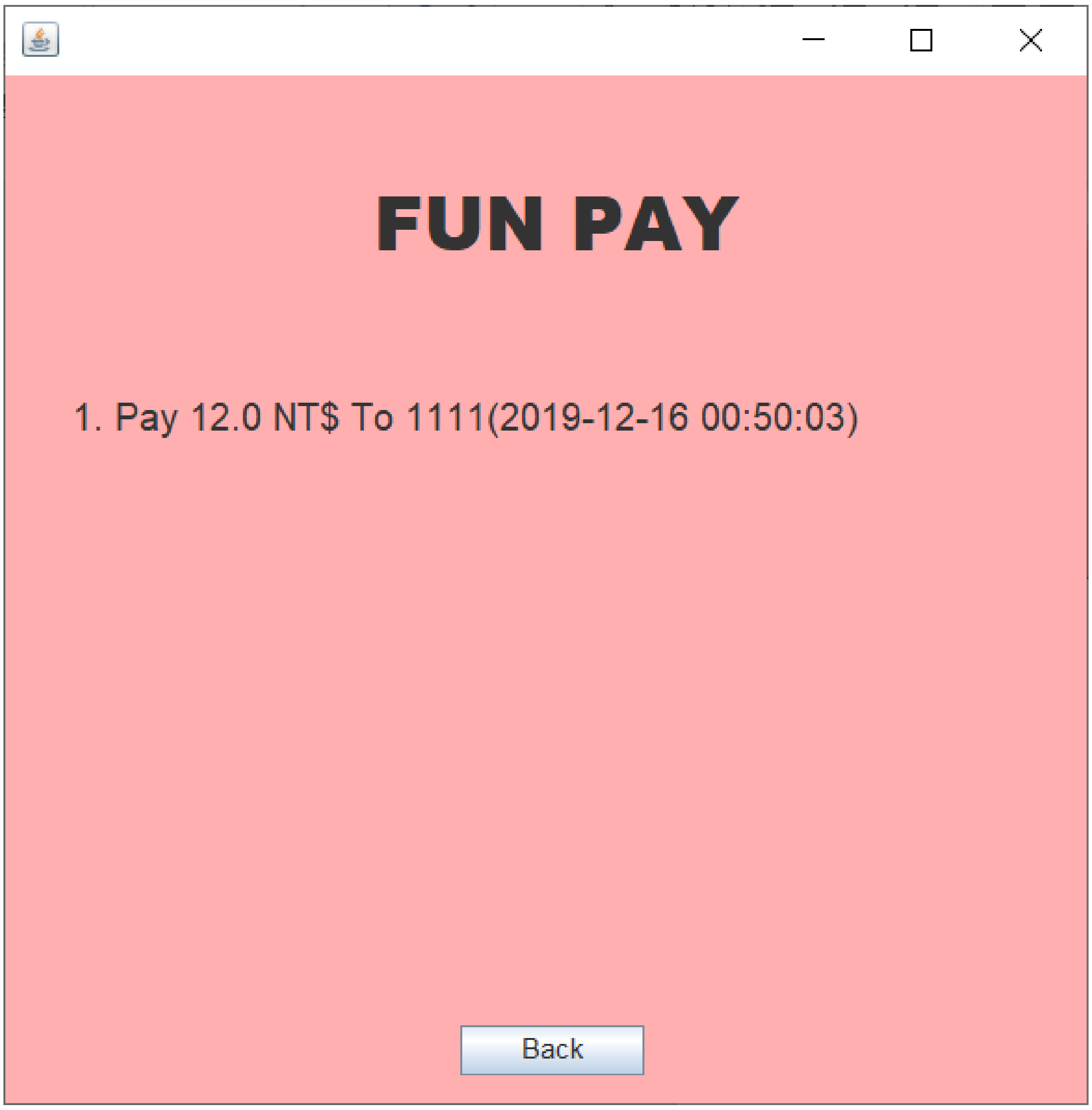
Here you can choose the number of denominations you want. If you add wrong, you can press the Previous button to return. After confirming, you can click Confirm button to complete the transaction. This part of the payment is made by credit card, so it is different from Pay function

Select Details GUI



Here you can select the Details you want, and after pressing the button, you will jump to the corresponding page.

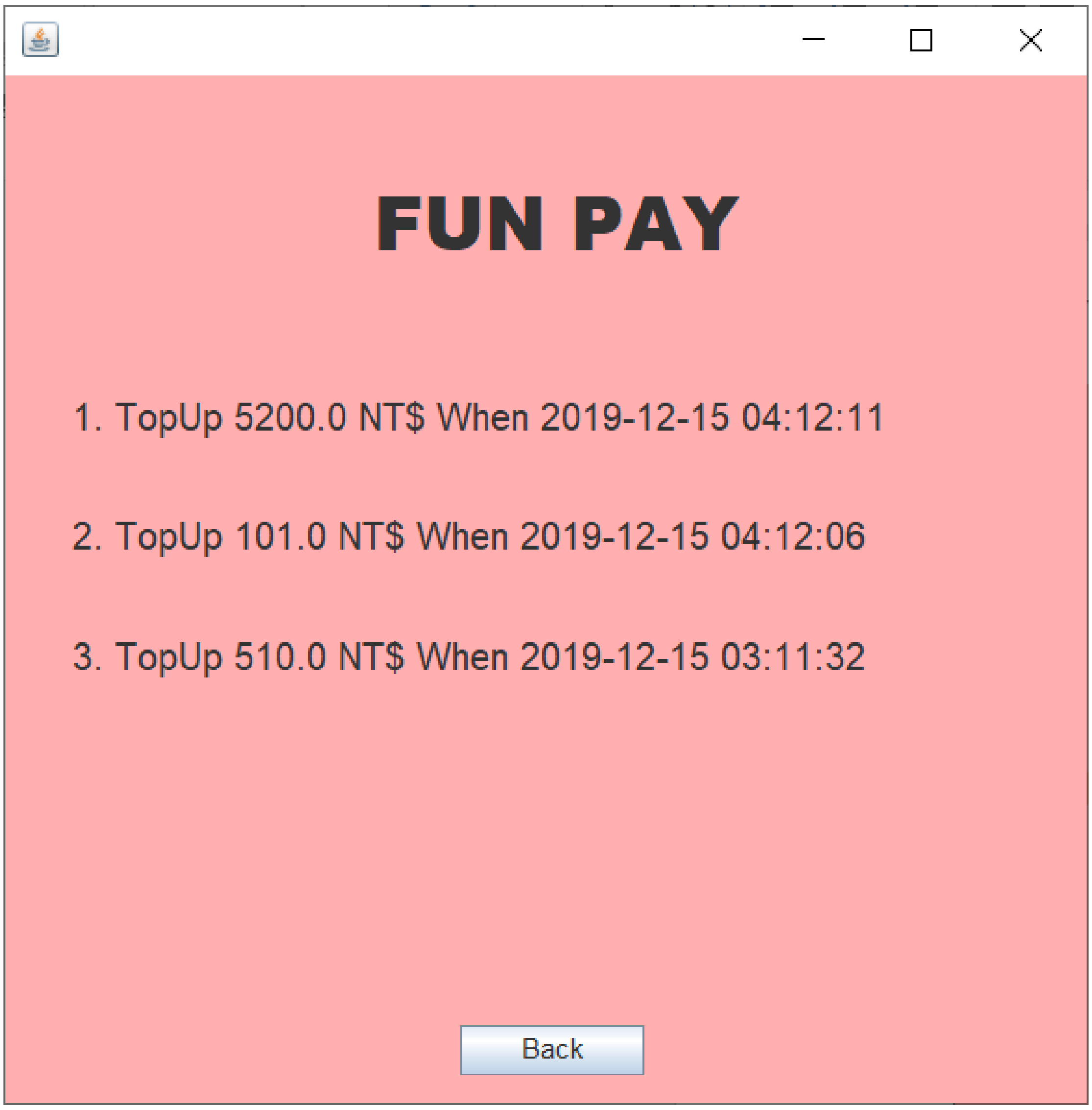
Pay Details GUI



1. Pay 12.0 NT$ To 1111(2019-12-16 00:50:03)

Display Pay's Details.

TopUp Details GUI



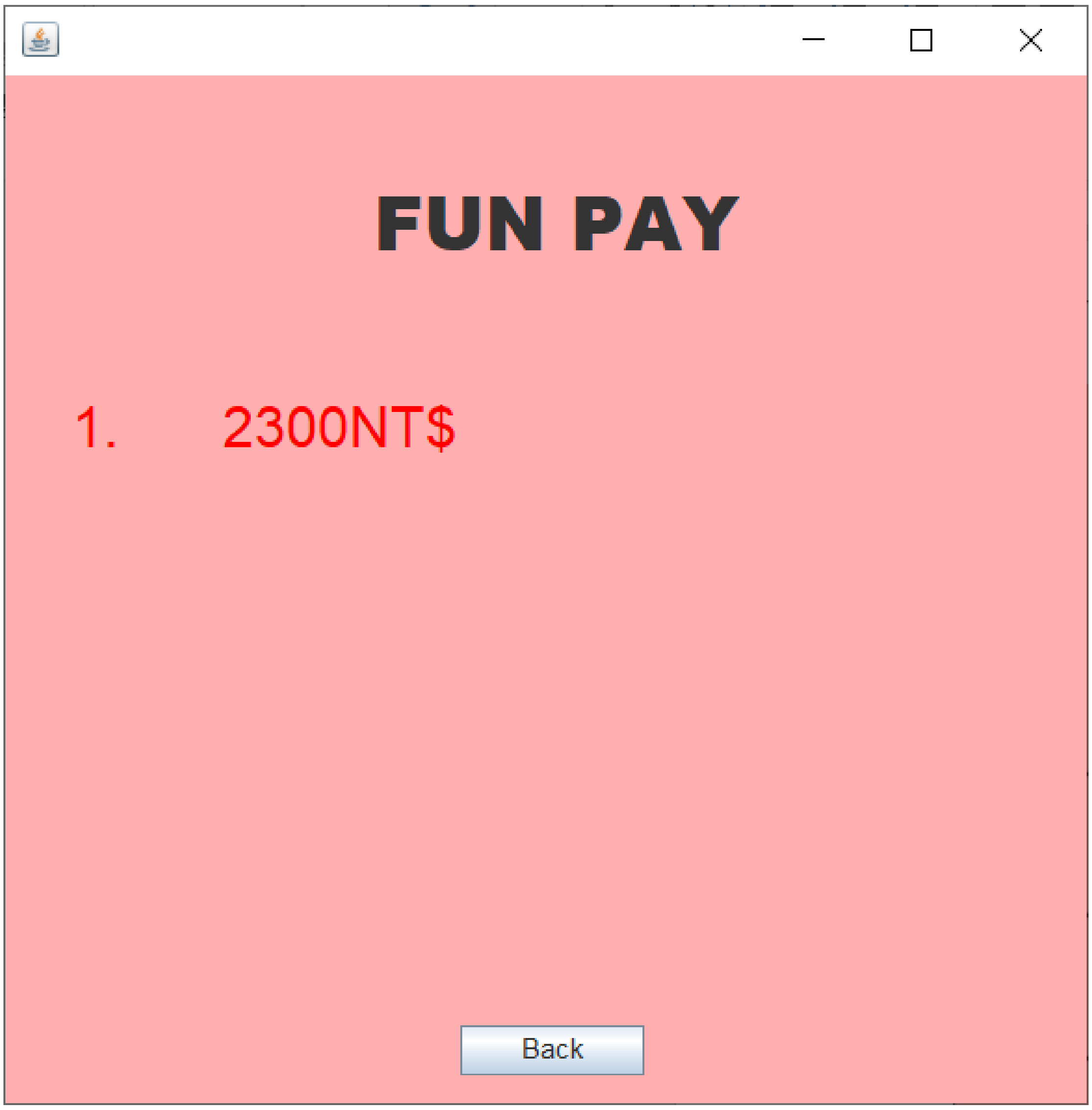
1. TopUp 5200.0 NT$ When 2019-12-15 04:12:11

2. TopUp 101.0 NT$ When 2019-12-15 04:12:06

3. TopUp 510.0 NT$ When 2019-12-15 03:11:32

Display TopUp's Details.

Shopping Details GUI

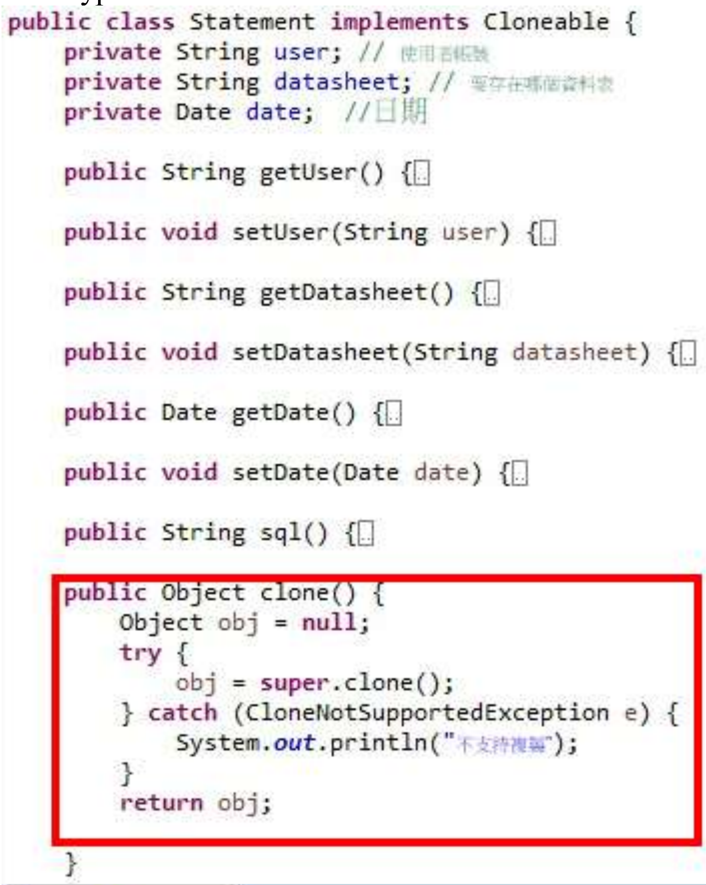


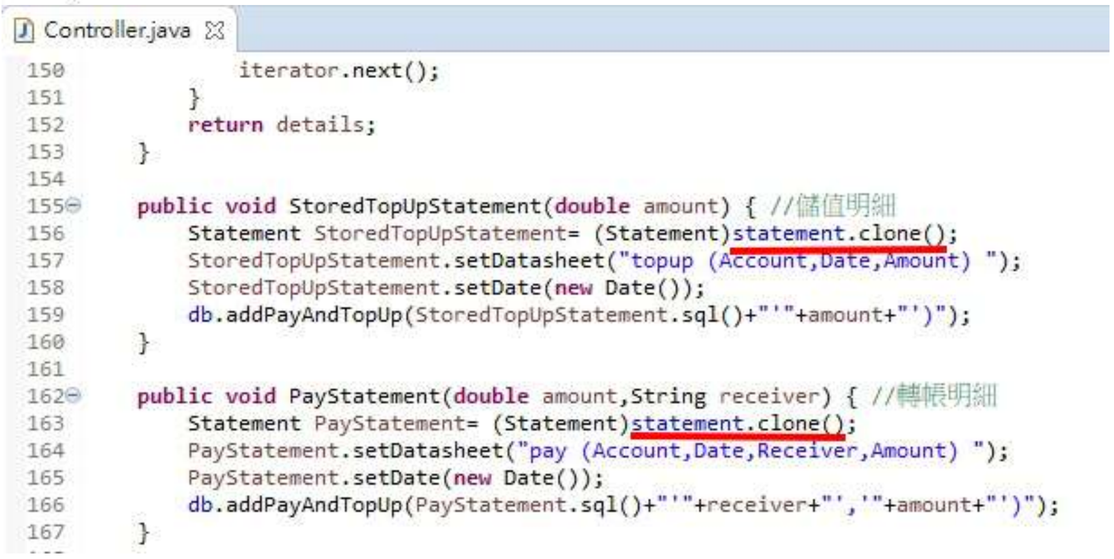
1.   2300NT$

Display Shopping's Details.

We have added Prototype, Decorator & Memento, Iterator, Chain of Responsibility, Singleton and State. Here is our code.

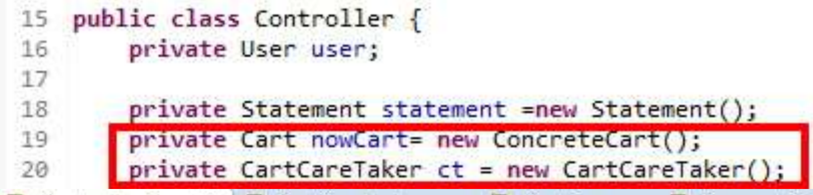
Prototype


```java
public class Statement implements Cloneable {
    private String user; // 使用者帳號
    private String datasheet; // 要存在哪個資料表
    private Date date;  //日期

    public String getUser() {[]

    public void setUser(String user) {[]

    public String getDatasheet() {[]

    public void setDatasheet(String datasheet) {[]

    public Date getDate() {[]

    public void setDate(Date date) {[]

    public String sql() {[]

    public Object clone() {
        Object obj = null;
        try {
            obj = super.clone();
        } catch (CloneNotSupportedException e) {
            System.out.println("不支持複製");
        }
        return obj;
    }
```

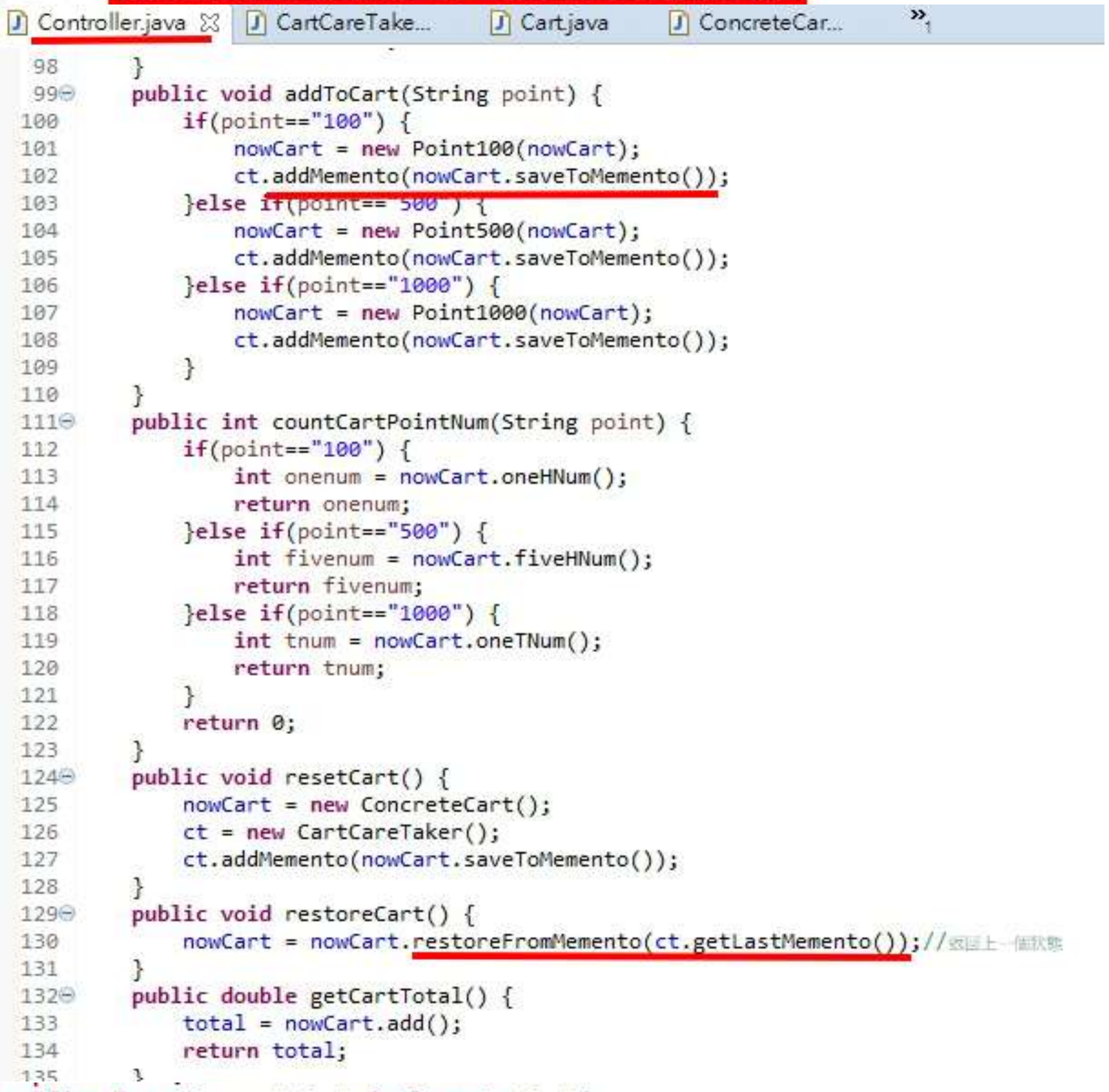


```java
                iterator.next();
            }
            return details;
        }

        public void StoredTopUpStatement(double amount) { //儲值明細
            Statement StoredTopUpStatement= (Statement)statement.clone();
            StoredTopUpStatement.setDatasheet("topup (Account,Date,Amount) ");
            StoredTopUpStatement.setDate(new Date());
            db.addPayAndTopUp(StoredTopUpStatement.sql()+"'"+amount+"')");
        }

        public void PayStatement(double amount,String receiver) { //轉帳明細
            Statement PayStatement= (Statement)statement.clone();
            PayStatement.setDatasheet("pay (Account,Date,Receiver,Amount) ");
            PayStatement.setDate(new Date());
            db.addPayAndTopUp(PayStatement.sql()+"'"+receiver+"','"+amount+"')");
        }
```

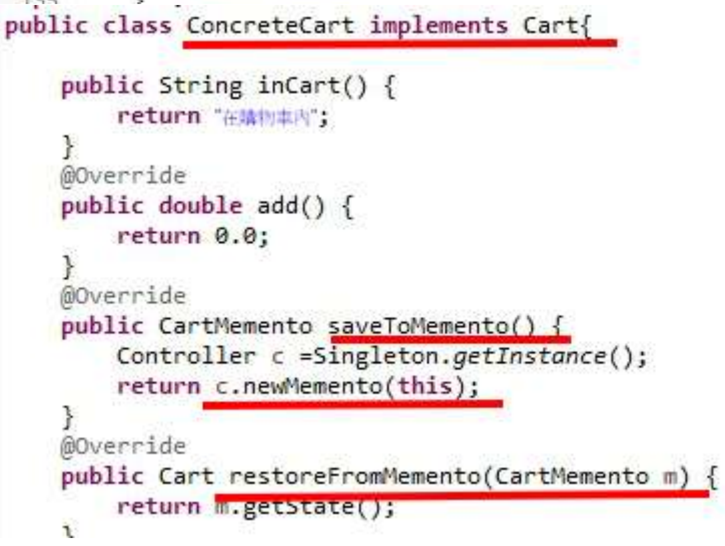
Decorator&Memento



```java
public class Controller {
    private User user;

    private Statement statement =new Statement();
    private Cart nowCart= new ConcreteCart();
    private CartCareTaker ct = new CartCareTaker();
```



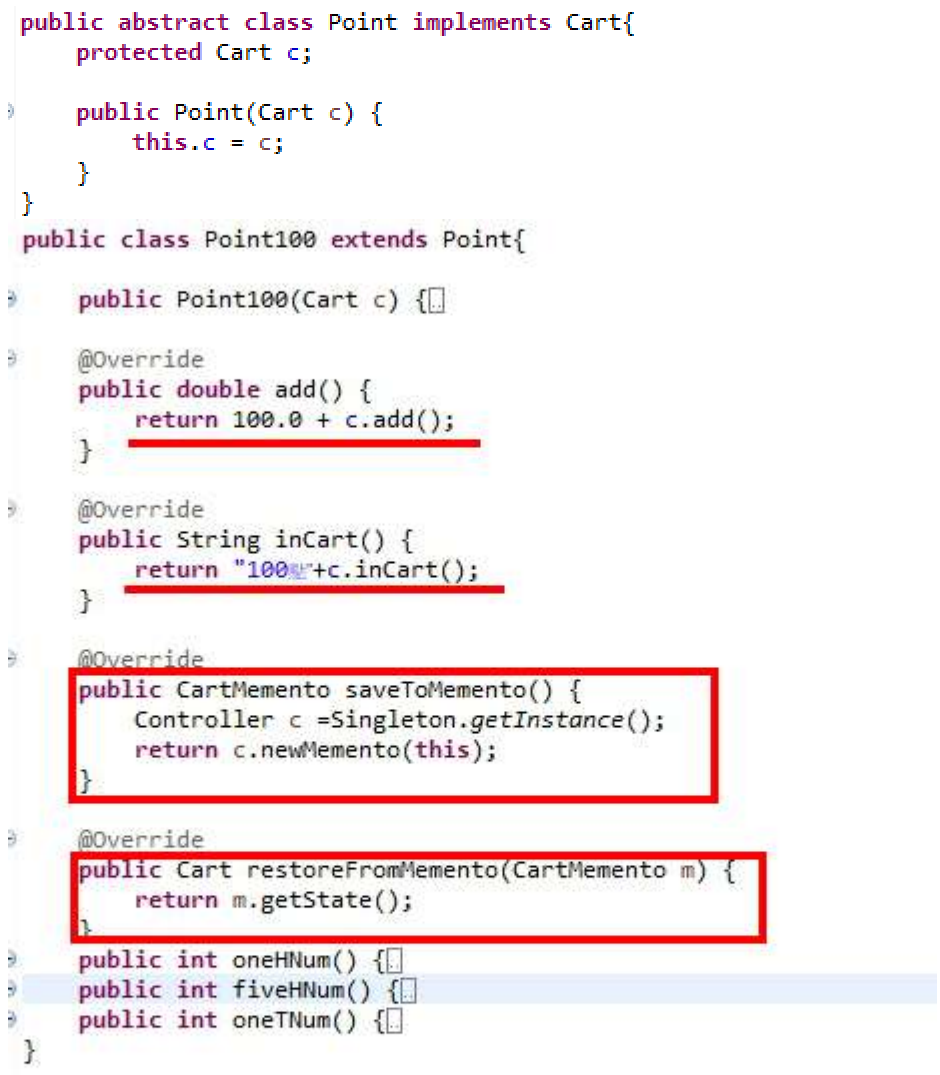
```java
    }
    public void addToCart(String point) {
        if(point=="100") {
            nowCart = new Point100(nowCart);
            ct.addMemento(nowCart.saveToMemento());
        }else if(point=="500") {
            nowCart = new Point500(nowCart);
            ct.addMemento(nowCart.saveToMemento());
        }else if(point=="1000") {
            nowCart = new Point1000(nowCart);
            ct.addMemento(nowCart.saveToMemento());
        }
    }
    public int countCartPointNum(String point) {
        if(point=="100") {
            int onenum = nowCart.oneHNum();
            return onenum;
        }else if(point=="500") {
            int fivenum = nowCart.fiveHNum();
            return fivenum;
        }else if(point=="1000") {
            int tnum = nowCart.oneTNum();
            return tnum;
        }
        return 0;
    }
    public void resetCart() {
        nowCart = new ConcreteCart();
        ct = new CartCareTaker();
        ct.addMemento(nowCart.saveToMemento());
    }
    public void restoreCart() {
        nowCart = nowCart.restoreFromMemento(ct.getLastMemento());//返回上一個狀態
    }
    public double getCartTotal() {
        total = nowCart.add();
        return total;
    }
```



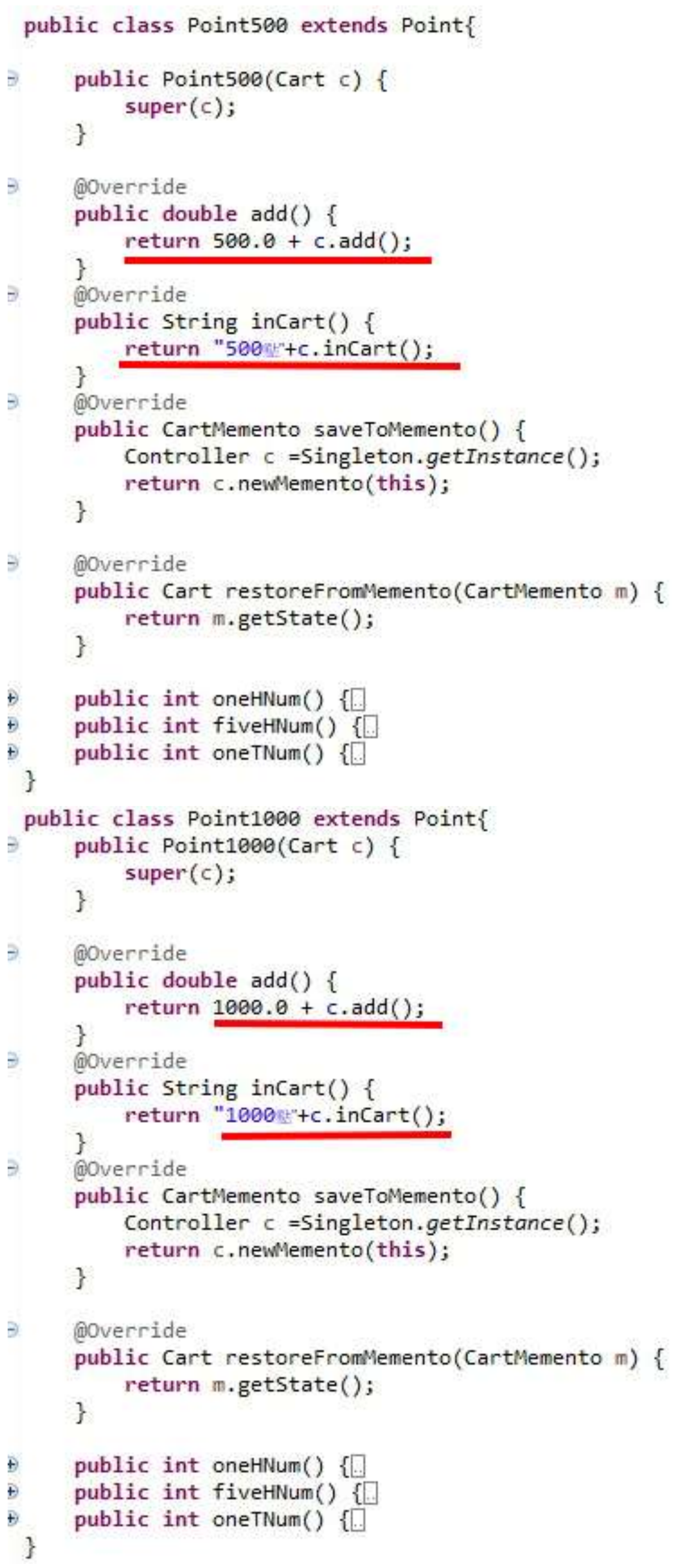
```java
public class ConcreteCart implements Cart{

    public String inCart() {
        return "在購物車內";
    }
    @Override
    public double add() {
        return 0.0;
    }
    @Override
    public CartMemento saveToMemento() {
        Controller c =Singleton.getInstance();
        return c.newMemento(this);
    }
    @Override
    public Cart restoreFromMemento(CartMemento m) {
        return m.getState();
    }
```

```java
public class CartCareTaker {
    ArrayList<CartMemento> memList = new ArrayList<CartMemento>();
    int index = memList.size();

    public void addMemento(CartMemento m) {
        memList.add(m);
    }
    public CartMemento getLastMemento() {
        memList.remove(memList.size()-1);
        return memList.get(memList.size()-1);
    }
}
public class CartMemento {
private Cart state;

    public CartMemento(Cart state) {
        this.state = state;
    }

    public Cart getState() {
        return this.state;
    }
}
```

```java
public interface Cart {
    public String inCart();
    public double add();
    public int oneHNum();
    public int fiveHNum();
    public int oneTNum();
    public CartMemento saveToMemento();
    public Cart restoreFromMemento(CartMemento m);
}
public class ConcreteCart implements Cart{

    public String inCart() {
        return "在購物車內";
    }
    @Override
    public double add() {
        return 0.0;
    }
    @Override
    public CartMemento saveToMemento() {
        Controller c =Singleton.getInstance();
        return c.newMemento(this);
    }
    @Override
    public Cart restoreFromMemento(CartMemento m) {
        return m.getState();
    }
    public int oneHNum() {
    public int fiveHNum() {
    public int oneTNum() {
}
```

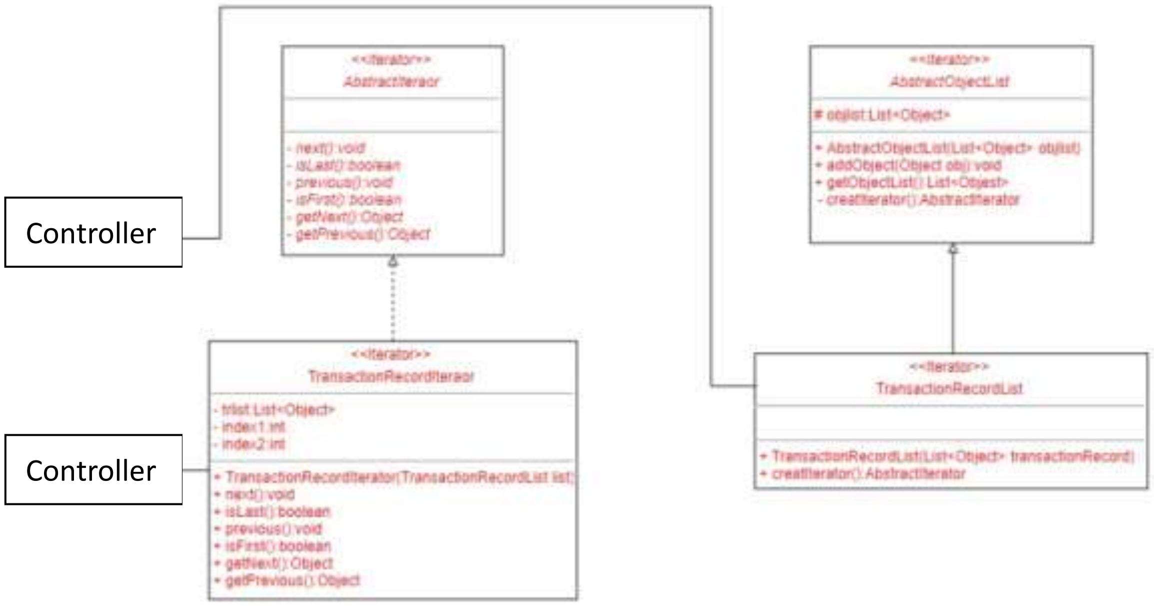


```java
public abstract class Point implements Cart{
    protected Cart c;

    public Point(Cart c) {
        this.c = c;
    }
}
public class Point100 extends Point{

    public Point100(Cart c) {

    @Override
    public double add() {
        return 100.0 + c.add();
    }

    @Override
    public String inCart() {
        return "100點"+c.inCart();
    }

    @Override
    public CartMemento saveToMemento() {
        Controller c =Singleton.getInstance();
        return c.newMemento(this);
    }

    @Override
    public Cart restoreFromMemento(CartMemento m) {
        return m.getState();
    }
    public int oneHNum() {
    public int fiveHNum() {
    public int oneTNum() {
}
```

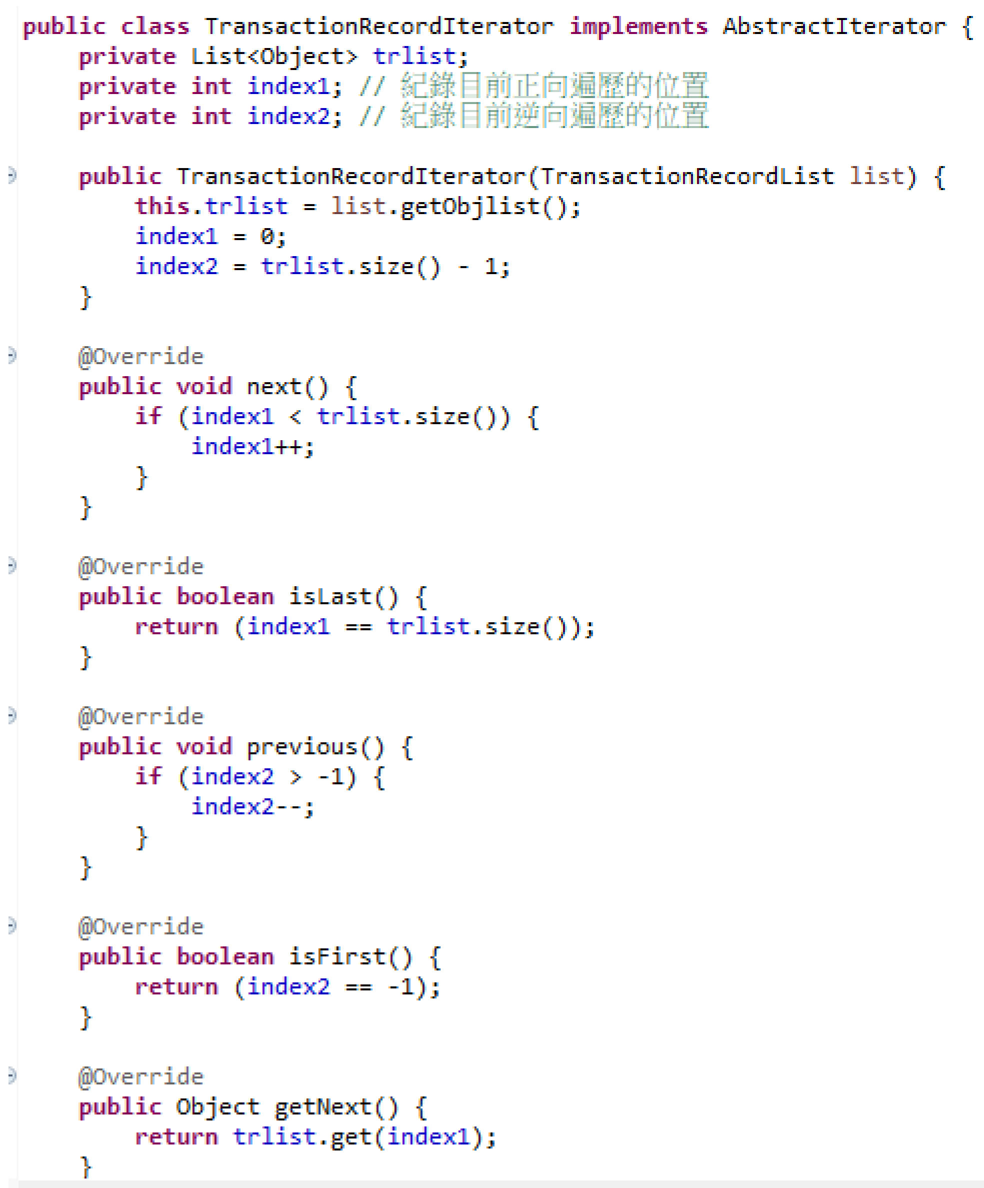


```java
public class Point500 extends Point{

    public Point500(Cart c) {
        super(c);
    }

    @Override
    public double add() {
        return 500.0 + c.add();
    }
    @Override
    public String inCart() {
        return "500點"+c.inCart();
    }
    @Override
    public CartMemento saveToMemento() {
        Controller c =Singleton.getInstance();
        return c.newMemento(this);
    }

    @Override
    public Cart restoreFromMemento(CartMemento m) {
        return m.getState();
    }

    public int oneHNum() {
    public int fiveHNum() {
    public int oneTNum() {
}
public class Point1000 extends Point{
    public Point1000(Cart c) {
        super(c);
    }

    @Override
    public double add() {
        return 1000.0 + c.add();
    }
    @Override
    public String inCart() {
        return "1000點"+c.inCart();
    }
    @Override
    public CartMemento saveToMemento() {
        Controller c =Singleton.getInstance();
        return c.newMemento(this);
    }

    @Override
    public Cart restoreFromMemento(CartMemento m) {
        return m.getState();
    }

    public int oneHNum() {
    public int fiveHNum() {
    public int oneTNum() {
}
```

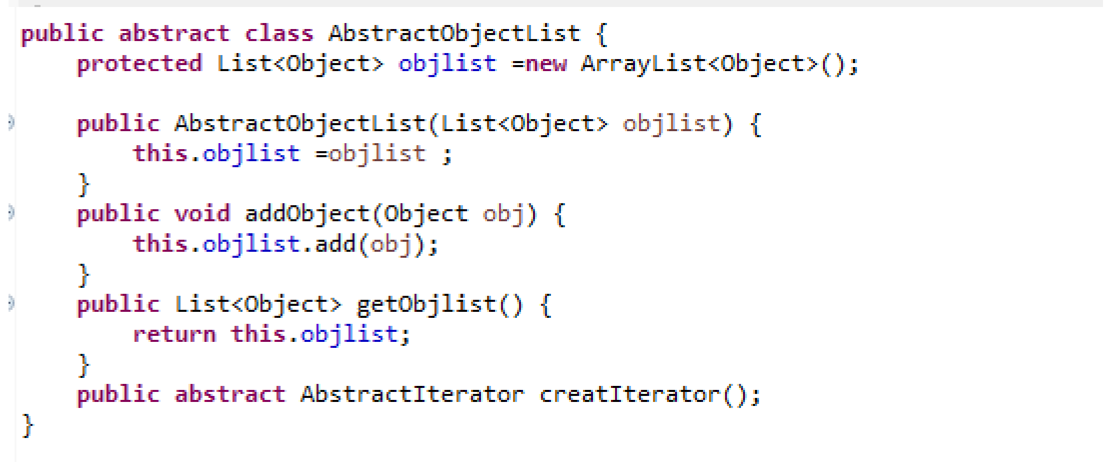
Iterator
Because we have Mediator, TransactionRecordIterator and TransactionObjectList will coordinate through the Controller. So the extra two are connected to the controller



```
public interface AbstractIterator {
    public void next(); // 到下一個元素

    public boolean isLast(); // 判斷是否為最後一個元素

    public void previous(); // 到上一個元素

    public boolean isFirst(); // 判斷是否為第一個元素

    public Object getNext(); // 擷取下一個元素

    public Object getPrevious(); // 擷取上一個元素
}
```



```
public class TransactionRecordIterator implements AbstractIterator {
    private List<Object> trlist;
    private int index1; // 紀錄目前正向遍歷的位置
    private int index2; // 紀錄目前逆向遍歷的位置

    public TransactionRecordIterator(TransactionRecordList list) {
        this.trlist = list.getObjlist();
        index1 = 0;
        index2 = trlist.size() - 1;
    }

    @Override
    public void next() {
        if (index1 < trlist.size()) {
            index1++;
        }
    }

    @Override
    public boolean isLast() {
        return (index1 == trlist.size());
    }

    @Override
    public void previous() {
        if (index2 > -1) {
            index2--;
        }
    }

    @Override
    public boolean isFirst() {
        return (index2 == -1);
    }

    @Override
    public Object getNext() {
        return trlist.get(index1);
    }
```



```
public abstract class AbstractObjectList {
    protected List<Object> objlist =new ArrayList<Object>();

    public AbstractObjectList(List<Object> objlist) {
        this.objlist =objlist ;
    }
    public void addObject(Object obj) {
        this.objlist.add(obj);
    }
    public List<Object> getObjlist() {
        return this.objlist;
    }
    public abstract AbstractIterator creatIterator();
}
```

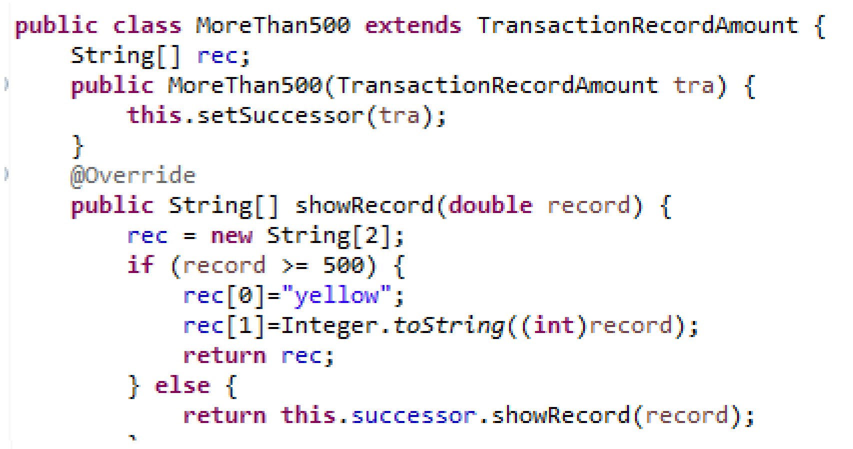
```java
public class TransactionRecordList extends AbstractObjectList {

    public TransactionRecordList(List<Object> transactionRecord) {
        super(transactionRecord);
    }

    @Override
    public AbstractIterator creatIterator() {
        Controller c= Singleton.getInstance();
        return c.newIterator(this);
    }

}
```

Chain of Responsibility

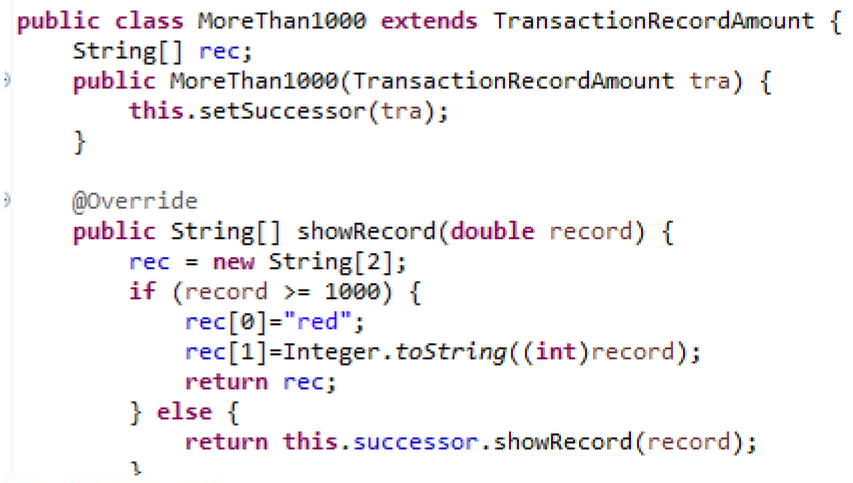
```java
public abstract class TransactionRecordAmount {   //helper
    protected TransactionRecordAmount successor; //定義誰是後繼對象

    public void setSuccessor(TransactionRecordAmount successor) {
        this.successor = successor;
    }

    public abstract String[] showRecord(double record); //顯示紀錄

}
```

```java
public class LessThan100 extends TransactionRecordAmount {
    String[] rec;
    @Override
    public String[] showRecord(double record) {
        rec = new String[2];
        rec[0]="black";
        rec[1]=Integer.toString((int)record);
        return rec;
    }
}
```

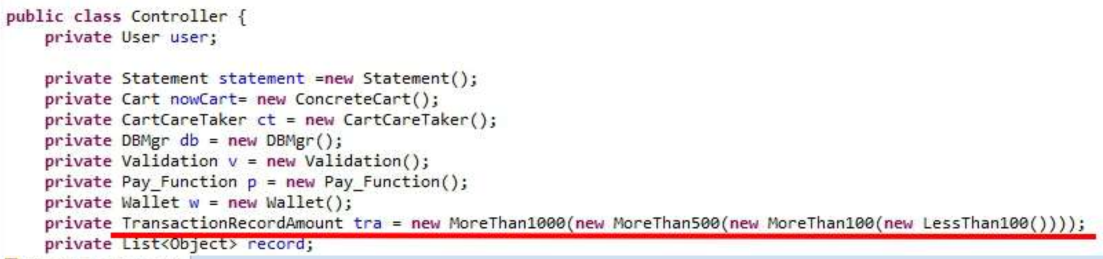
```java
public class MoreThan100 extends TransactionRecordAmount{
    String[] rec;
    public MoreThan100(TransactionRecordAmount tra) {
        this.setSuccessor(tra);
    }
    @Override
    public String[] showRecord(double record) {
        rec = new String[2];
        if (record >= 100) {
            rec[0]="green";
            rec[1]=Integer.toString((int)record);
            return rec;
        }else {
            return this.successor.showRecord(record);
        }
    }

}
```



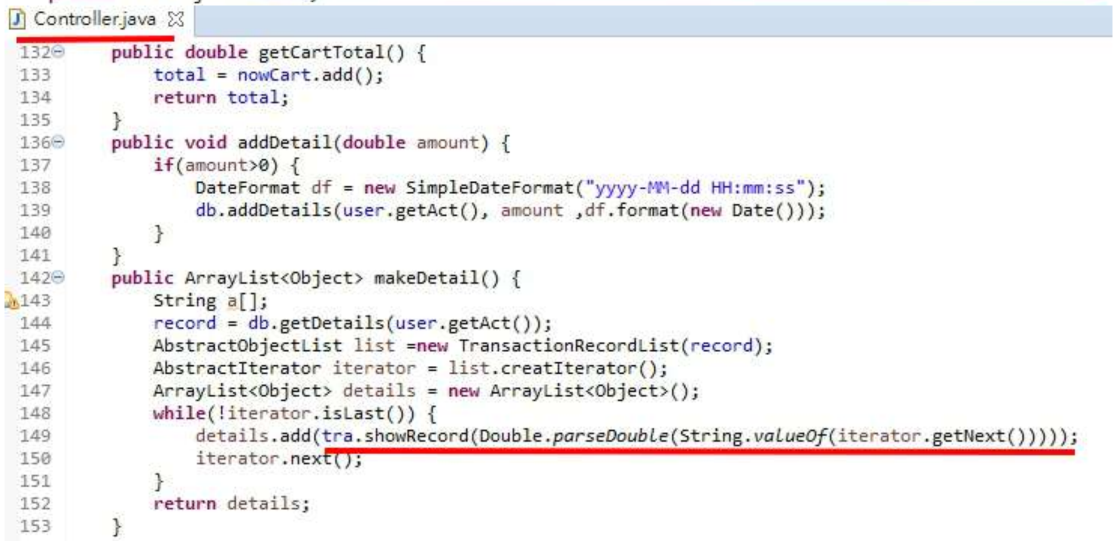
```java
public class MoreThan500 extends TransactionRecordAmount {
    String[] rec;
    public MoreThan500(TransactionRecordAmount tra) {
        this.setSuccessor(tra);
    }
    @Override
    public String[] showRecord(double record) {
        rec = new String[2];
        if (record >= 500) {
            rec[0]="yellow";
            rec[1]=Integer.toString((int)record);
            return rec;
        } else {
            return this.successor.showRecord(record);
```



```java
public class MoreThan1000 extends TransactionRecordAmount {
    String[] rec;
    public MoreThan1000(TransactionRecordAmount tra) {
        this.setSuccessor(tra);
    }

    @Override
    public String[] showRecord(double record) {
        rec = new String[2];
        if (record >= 1000) {
            rec[0]="red";
            rec[1]=Integer.toString((int)record);
            return rec;
        } else {
            return this.successor.showRecord(record);
```

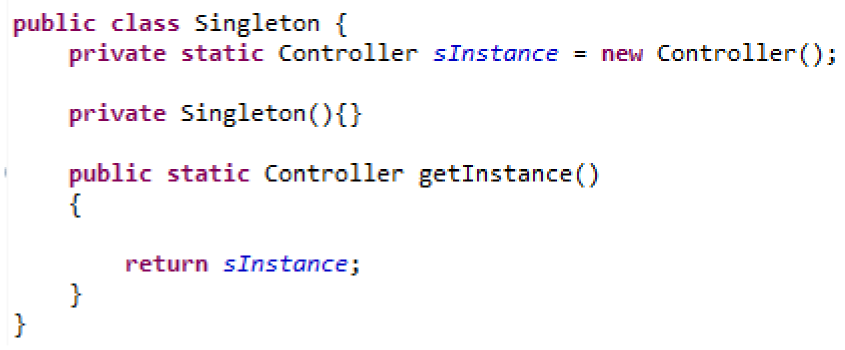


```java
public class Controller {
    private User user;

    private Statement statement =new Statement();
    private Cart nowCart= new ConcreteCart();
    private CartCareTaker ct = new CartCareTaker();
    private DBMgr db = new DBMgr();
    private Validation v = new Validation();
    private Pay_Function p = new Pay_Function();
    private Wallet w = new Wallet();
    private TransactionRecordAmount tra = new MoreThan1000(new MoreThan500(new MoreThan100(new LessThan100())));
    private List<Object> record;
```



```java
    public double getCartTotal() {
        total = nowCart.add();
        return total;
    }
    public void addDetail(double amount) {
        if(amount>0) {
            DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            db.addDetails(user.getAct(), amount ,df.format(new Date()));
        }
    }
    public ArrayList<Object> makeDetail() {
        String a[];
        record = db.getDetails(user.getAct());
        AbstractObjectList list =new TransactionRecordList(record);
        AbstractIterator iterator = list.creatIterator();
        ArrayList<Object> details = new ArrayList<Object>();
        while(!iterator.isLast()) {
            details.add(tra.showRecord(Double.parseDouble(String.valueOf(iterator.getNext()))));
            iterator.next();
        }
        return details;
    }
```

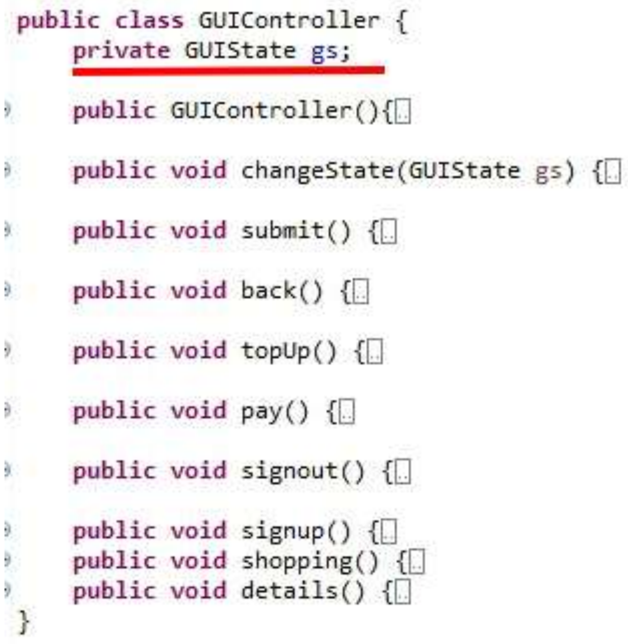
Singleton



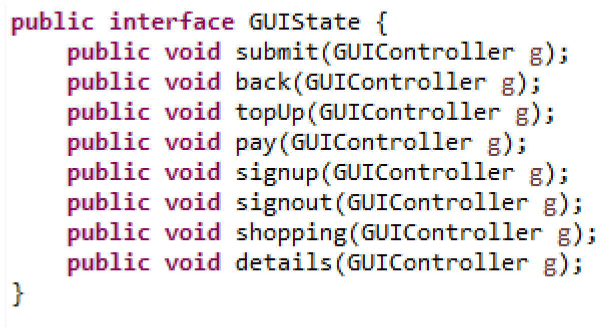
```
public class Singleton {
    private static Controller sInstance = new Controller();

    private Singleton(){}

    public static Controller getInstance()
    {

        return sInstance;
    }
}
```

State

Our state pattern have 13 states. They are Login, Main, Pay, PayDetails, PayOTPCheck, Receipt, SelectDetails, Shopping, SignUp, SignUpOTPCheck, TopUp, TopUpDetails, TopUpReceipt. This is used to transition the state of the GUI. We take Login and Main as an example.



```
public class GUIController {
    private GUIState gs;

    public GUIController(){□

    public void changeState(GUIState gs) {□

    public void submit() {□

    public void back() {□

    public void topUp() {□

    public void pay() {□

    public void signout() {□

    public void signup() {□
    public void shopping() {□
    public void details() {□
}
```



```
public interface GUIState {
    public void submit(GUIController g);
    public void back(GUIController g);
    public void topUp(GUIController g);
    public void pay(GUIController g);
    public void signup(GUIController g);
    public void signout(GUIController g);
    public void shopping(GUIController g);
    public void details(GUIController g);
}
```

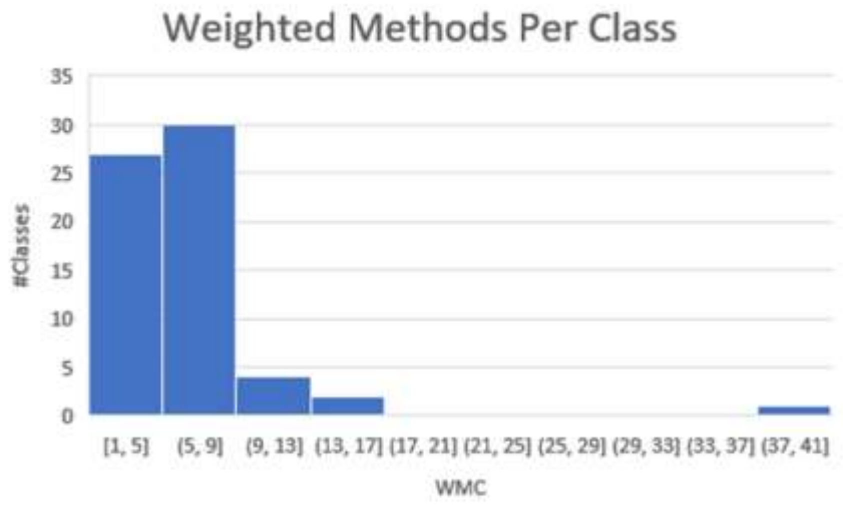
```java
public class Main implements GUIState{
    public void submit(GUIController g) {
        System.out.println("Doesn't support this function.");
    }
    public void back(GUIController g) {
        System.out.println("Doesn't support this function.");
    }
    public void topUp(GUIController g) {
        System.out.println("Go to TopUpGUI.");
        g.changeState(new TopUp());
    }
    public void pay(GUIController g) {
        System.out.println("Go to PayGUI.");
        g.changeState(new Pay());
    }
    public void signup(GUIController g) {
        System.out.println("Doesn't support this function.");
    }
    public void signout(GUIController g) {
        System.out.println("Back to Login.");
        g.changeState(new Login());
    }
    public void shopping(GUIController g) {
        System.out.println("Go to Shopping");
        g.changeState(new Shopping());
    }
    public void details(GUIController g) {
        System.out.println("Go to Details");
        g.changeState(new SelectDetails());
    }
}
```

```java
public class Login implements GUIState{
    public void submit(GUIController g) {
        System.out.println("Login success.");
        g.changeState(new Main());
    }
    public void back(GUIController g) {
        System.out.println("Doesn't support this function.");
    }
    public void topUp(GUIController g) {
        System.out.println("Doesn't support this function.");
    }
    public void pay(GUIController g) {
        System.out.println("Doesn't support this function.");
    }
    public void signup(GUIController g) {
        System.out.println("Start to SignUp.");
        g.changeState(new SignUp());
    }
    public void signout(GUIController g) {
        System.out.println("Doesn't support this function.");
    }
    public void shopping(GUIController g) {
        System.out.println("Doesn't support this function.");
    }
    public void details(GUIController g) {
        System.out.println("Doesn't support this function.");
    }
}
```

2) **You need to evaluate the design quality of the new design by using objectoriented quality metrics (WMC, DIT, NOC, CBO, RFC, LCOM). The figure shall be drawn like the previous provided references. You shall explain each metric by giving examples of your design.**

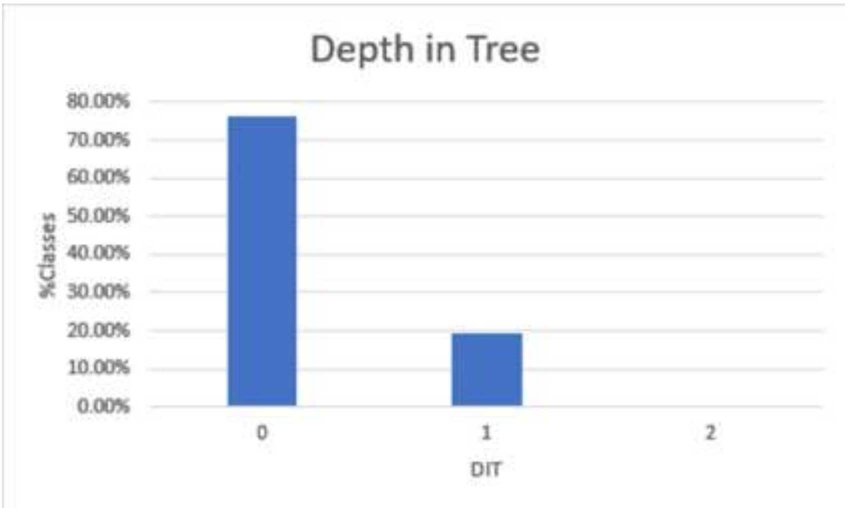
WMC

WMC relates directly to Bunge's definition of complexity of a thing, since methods are properties of object classes and complexity is determined by the cardinality of its set of properties.
In this metrics, our code is possible to reuse.



DIT

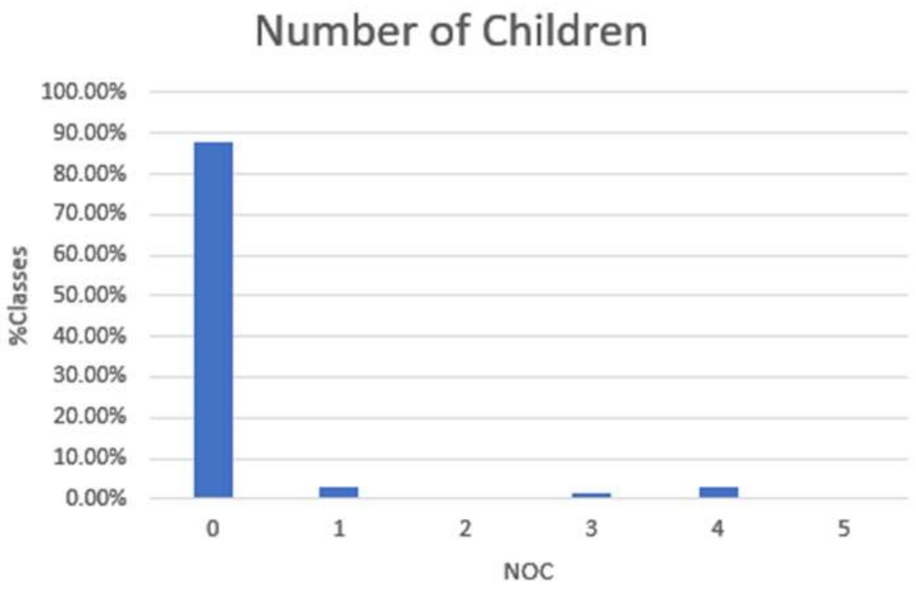
The greater the DIT the more likely the class will inherit and use such method. The behavior of the class is affected and could be more difficult to predict.
Our DIT almost are 1 and 2, it means our code aren't difficult to predict.



NOC

The greater the NOC the greater likelihood to reusing features.

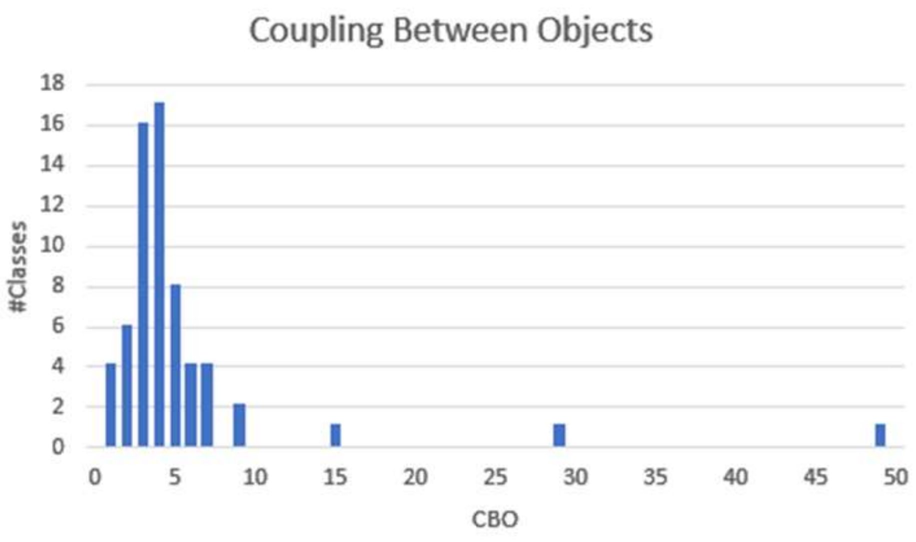
Our NOC within the range of 0 to 1 have 61 , it means our code reusability is not high.



CBO

According to Godin & Miceli article, CBO above 14 is hard to maintain.

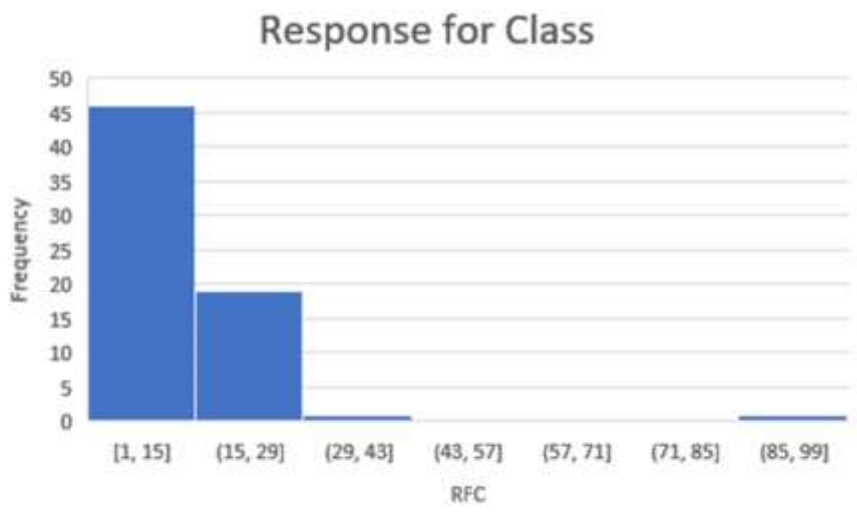
Our highest CBO is 49 and then is 29, most are between 1 to 7, so our project doesn't hard to test and reuse.



RFC

The larger the RFC the more difficult to test and debug due to more complex interaction relationships and more effort required to understand the methods and prepare test cases and test stubs.

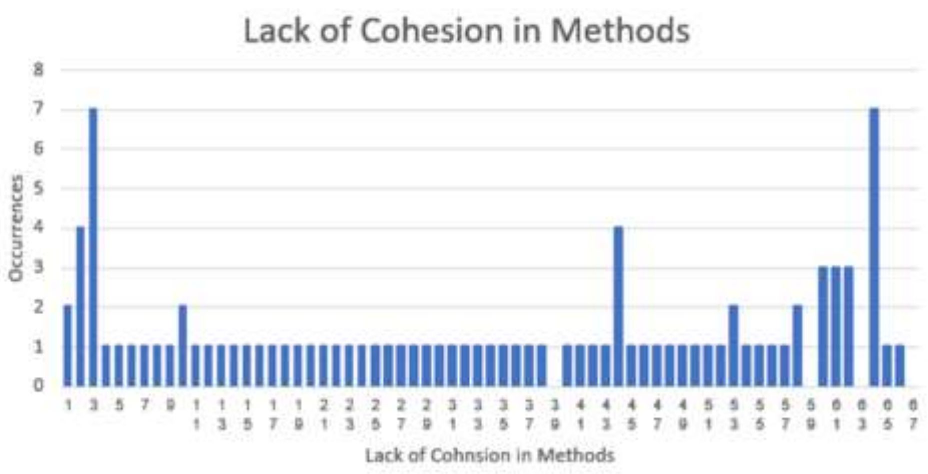
Our RFC highest is Controller, it is 96. It means Controller is hard to test and debug.



LCOM

This metric measures the correlation between the methods and the local instance variables of a class. High cohesion indicates good class subdivision. Lack of cohesion or low cohesion increases complexity.

Our LCOM most values are 1, it means that cohesion is normal.



3) Create Junit test cases and Junit test suite to test one new selected class.

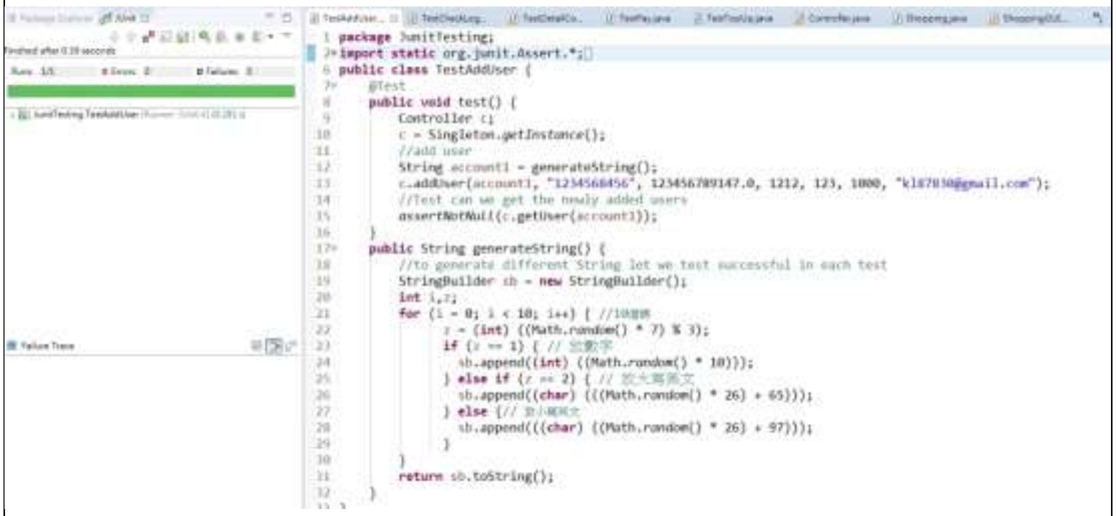
Junit test

We choose controller() to conduct Junit and suite test.

1 TestAddUser：

Result of Controller calls addUser() by using correct data is correct because we Succeed to get the user that we add just now.

*Using gerenateString() to generate different string let test easier.

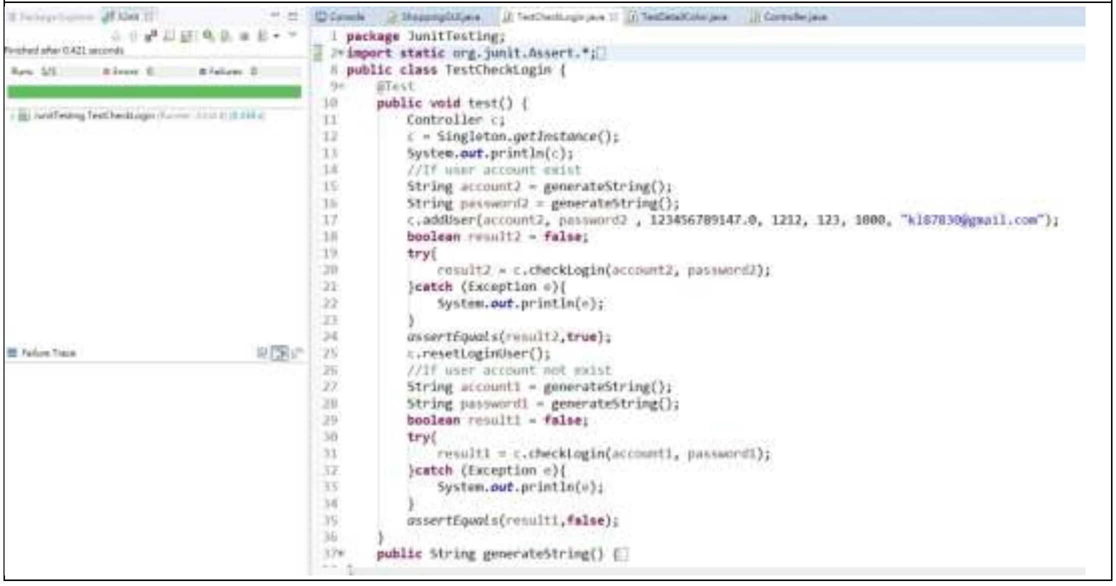


```java
package JunitTesting;
import static org.junit.Assert.*;
public class TestAddUser {
    @Test
    public void test() {
        Controller c;
        c = Singleton.getInstance();
        //add user
        String account1 = generateString();
        c.addUser(account1, "123456B456", 123456789147.0, 1212, 123, 1000, "k187830@gmail.com");
        //Test can we get the newly added users
        assertNotNull(c.getUser(account1));
    }
    public String generateString() {
        //to generate different String let we test successful in each test
        StringBuilder sb = new StringBuilder();
        int i,r;
        for (i = 0; i < 10; i++) { //10個碼
                r = (int) ((Math.random() * 7) % 3);
                if (r == 1) { // 出數字
                  sb.append((int) ((Math.random() * 10)));
                } else if (r == 2) { // 出大寫英文
                  sb.append((char) (((Math.random() * 26) + 65)));
                } else {// 出小寫英文
                  sb.append(((char) ((Math.random() * 26) + 97)));
                }
        }
        return sb.toString();
    }
}
```

2 TestCheckLogin：

Controller calls checkLogin() by using existing user account and correct password then result is correct because the return value of checkLogin() is true. If controller calls checkLogin() by not existed user account then result is correct because the return value of checkLogin become false too.

*This generateString() is exactly the same as TestaddUser()'s generateString()

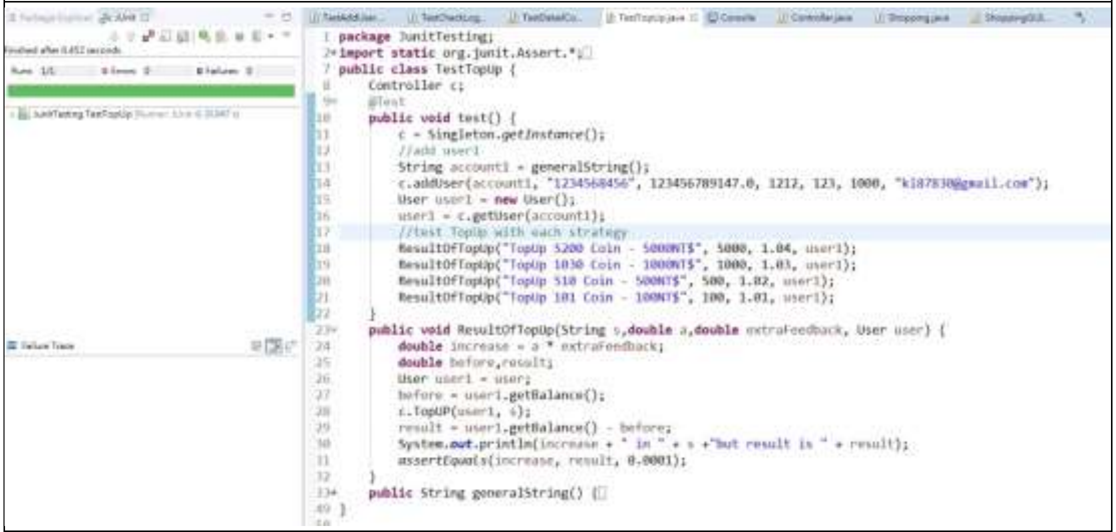


```java
package JunitTesting;
import static org.junit.Assert.*;
public class TestCheckLogin {
    @Test
    public void test() {
        Controller c;
        c = Singleton.getInstance();
        System.out.println(c);
        //If user account exist
        String account2 = generateString();
        String password2 = generateString();
        c.addUser(account2, password2 , 123456789147.0, 1212, 123, 1000, "k187830@gmail.com");
        boolean result2 = false;
        try{
            result2 = c.checkLogin(account2, password2);
        }catch (Exception e){
            System.out.println(e);
        }
        assertEquals(result2,true);
        c.resetLoginUser();
        //If user account not exist
        String account1 = generateString();
        String password1 = generateString();
        boolean result1 = false;
        try{
            result1 = c.checkLogin(account1, password1);
        }catch (Exception e){
            System.out.println(e);
        }
        assertEquals(result1,false);
    }
    public String generateString() 
```

3 TestTopUp：

Results of controller calls TopUp() by using different strategy is correct because balance increased correctly.

*This generateString() is exactly the same as TestaddUser()'s generateString()



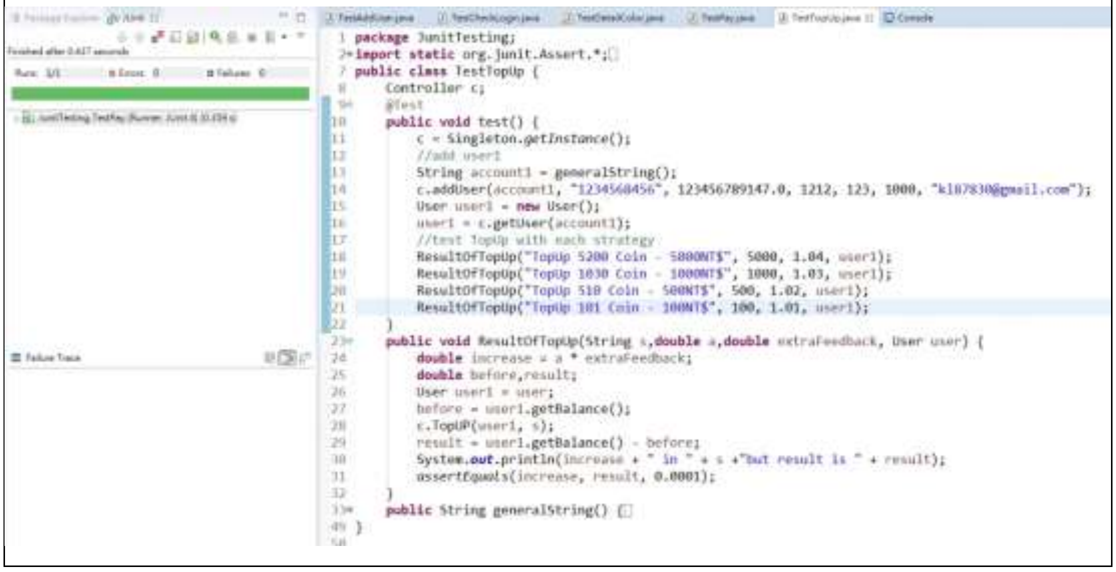
```
package JunitTesting;
import static org.junit.Assert.*;
public class TestTopUp {
    Controller c;
    @Test
    public void test() {
        c = Singleton.getInstance();
        //add user1
        String account1 = generalString();
        c.addUser(account1, "1234568456", 123456789147.0, 1212, 123, 1000, "k187830@gmail.com");
        User user1 = new User();
        user1 = c.getUser(account1);
        //test TopUp with each strategy
        ResultOfTopUp("TopUp 5200 Coin - 5000NT$", 5000, 1.04, user1);
        ResultOfTopUp("TopUp 1030 Coin - 1000NT$", 1000, 1.03, user1);
        ResultOfTopUp("TopUp 510 Coin - 500NT$", 500, 1.02, user1);
        ResultOfTopUp("TopUp 101 Coin - 100NT$", 100, 1.01, user1);
    }
    public void ResultOfTopUp(String s,double a,double extraFeedback, User user) {
        double increase = a * extraFeedback;
        double before,result;
        User user1 = user;
        before = user1.getBalance();
        c.TopUP(user1, s);
        result = user1.getBalance() - before;
        System.out.println(increase + " in " + s +"but result is " + result);
        assertEquals(increase, result, 0.0001);
    }
    public String generalString() {
}
```

4 TestPay：

PayC() is the function that let user can pay money to others.

Results of controller calls PayC() by using correct data is correct because balance decreased correctly.

*This generateString() is exactly the same as TestaddUser()'s generateString()



```
package JunitTesting;
import static org.junit.Assert.*;
public class TestTopUp {
    Controller c;
    @Test
    public void test() {
        c = Singleton.getInstance();
        //add user1
        String account1 = generalString();
        c.addUser(account1, "1234568456", 123456789147.0, 1212, 123, 1000, "k187830@gmail.com");
        User user1 = new User();
        user1 = c.getUser(account1);
        //test TopUp with each strategy
        ResultOfTopUp("TopUp 5200 Coin - 5000NT$", 5000, 1.04, user1);
        ResultOfTopUp("TopUp 1030 Coin - 1000NT$", 1000, 1.03, user1);
        ResultOfTopUp("TopUp 510 Coin - 500NT$", 500, 1.02, user1);
        ResultOfTopUp("TopUp 101 Coin - 100NT$", 100, 1.01, user1);
    }
    public void ResultOfTopUp(String s,double a,double extraFeedback, User user) {
        double increase = a * extraFeedback;
        double before,result;
        User user1 = user;
        before = user1.getBalance();
        c.TopUP(user1, s);
        result = user1.getBalance() - before;
        System.out.println(increase + " in " + s +"but result is " + result);
        assertEquals(increase, result, 0.0001);
    }
    public String generalString() {
}
```

5 TestDetailColor：

The result that run makeDetail() is correct because each color is equals we set.

*Each color condition we set as:

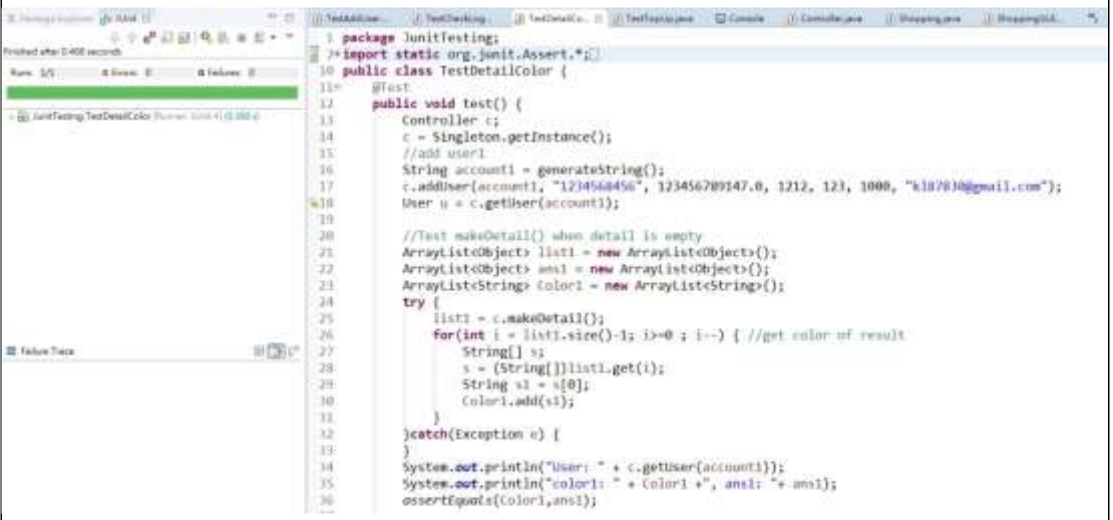
If (amount <100) then color is black.

If (amount >= 100 and amount < 500) then color is green.

If (amount >= 500 and amount < 1000) then color is yellow.

If (amount >= 1000) then color is red.

*This generateString() is exactly the same as TestaddUser()'s generateString()

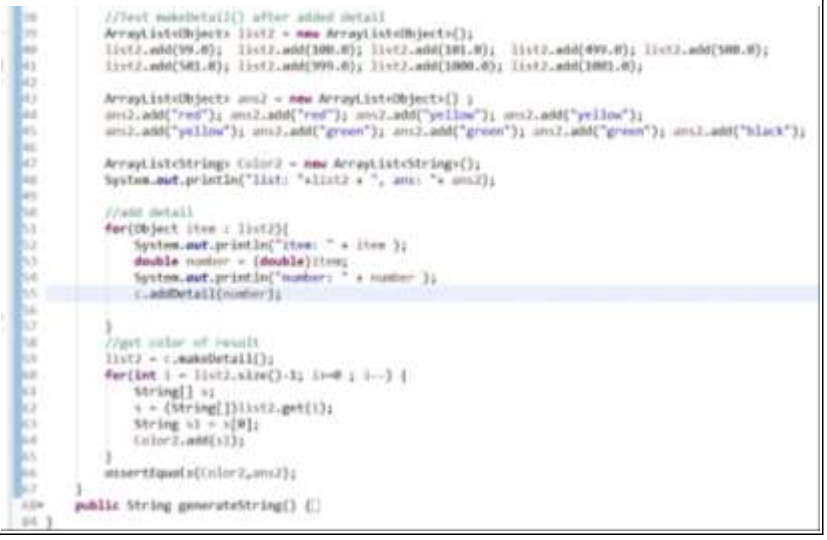


Finished after 0.408 seconds

Runs: 1/1   Errors: 0   Failures: 0

JunitTesting.TestDetailColor

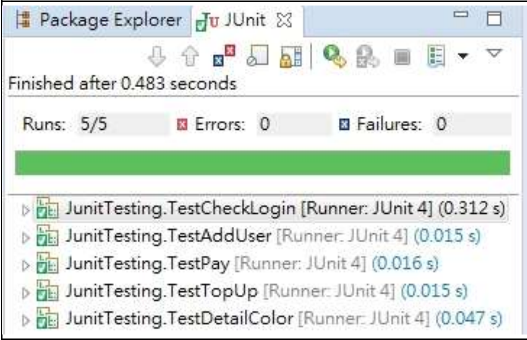
```java
package JunitTesting;
import static org.junit.Assert.*;
public class TestDetailColor {
    @Test
    public void test() {
        Controller c;
        c = Singleton.getInstance();
        //add user1
        String account1 = generateString();
        c.addUser(account1, "1234560456", 123456789147.0, 1212, 123, 1000, "k187030@gmail.com");
        User u = c.getUser(account1);

        //Test makeDetail() when detail is empty
        ArrayList<Object> list1 = new ArrayList<Object>();
        ArrayList<Object> ans1 = new ArrayList<Object>();
        ArrayList<String> Color1 = new ArrayList<String>();
        try {
            list1 = c.makeDetail();
            for(int i = list1.size()-1; i>=0 ; i--) { //get color of result
                String[] s;
                s = (String[])list1.get(i);
                String s1 = s[0];
                Color1.add(s1);
            }
        }catch(Exception e) {
        }
        System.out.println("User: " + c.getUser(account1));
        System.out.println("color1: " + Color1 +", ans1: "+ ans1);
        assertEquals(Color1,ans1);
```



```java
        //Test makeDetail() after added detail
        ArrayList<Object> list2 = new ArrayList<Object>();
        list2.add(99.0);  list2.add(100.0); list2.add(101.0);  list2.add(499.0); list2.add(500.0);
        list2.add(501.0); list2.add(999.0); list2.add(1000.0); list2.add(1001.0);

        ArrayList<Object> ans2 = new ArrayList<Object>() ;
        ans2.add("red"); ans2.add("red"); ans2.add("yellow"); ans2.add("yellow");
        ans2.add("yellow"); ans2.add("green"); ans2.add("green"); ans2.add("green"); ans2.add("black");

        ArrayList<String> Color2 = new ArrayList<String>();
        System.out.println("list: "+list2 + ", ans: "+ ans2);

        //add detail
        for(Object item : list2){
            System.out.println("item: " + item );
            double number = (double)item;
            System.out.println("number: " + number );
            c.addDetail(number);

        }
        //get color of result
        list2 = c.makeDetail();
        for(int i = list2.size()-1; i>=0 ; i--) {
            String[] s;
            s = (String[])list2.get(i);
            String s1 = s[0];
            Color2.add(s1);
        }
        assertEquals(Color2,ans2);
    }
    public String generateString() {
}
```

Junit test suite

Result after Integrated all Junit test together to test is correct too.



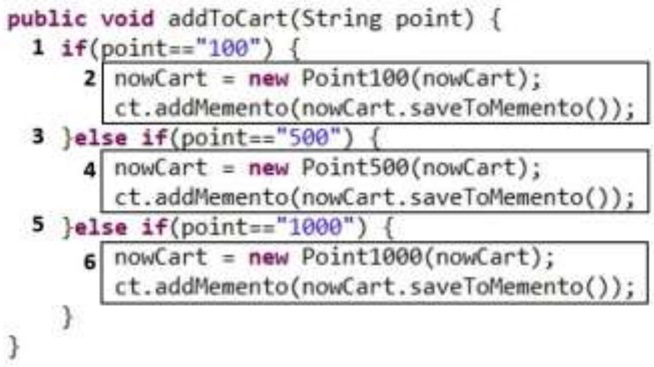
```
Finished after 0.483 seconds

Runs: 5/5    Errors: 0    Failures: 0

JunitTesting.TestCheckLogin [Runner: JUnit 4] (0.312 s)
JunitTesting.TestAddUser [Runner: JUnit 4] (0.015 s)
JunitTesting.TestPay [Runner: JUnit 4] (0.016 s)
JunitTesting.TestTopUp [Runner: JUnit 4] (0.015 s)
JunitTesting.TestDetailColor [Runner: JUnit 4] (0.047 s)
```

4) Conduct a new part of the software testing including white box and black box.

White box testing

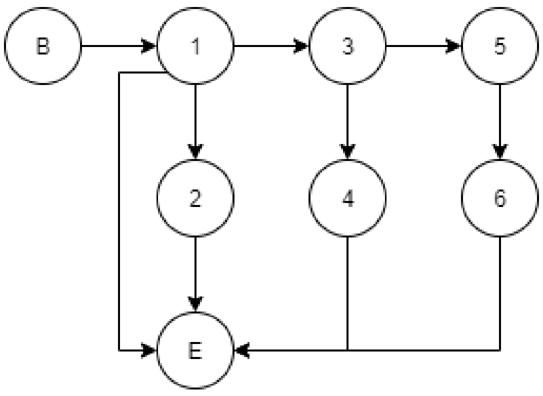
Basis Path Testing

Basis path testing is a technique for designing test cases based on paths. It is based on the program control flow graph. By analyzing the Cyclomatic Complexity of the control structure, the basic executable path set is derived. The test case is designed to ensure that at least each executable statement of the program is executed at least once.



```
public void addToCart(String point) {
  1 if(point=="100") {
      2 nowCart = new Point100(nowCart);
        ct.addMemento(nowCart.saveToMemento());
  3 }else if(point=="500") {
      4 nowCart = new Point500(nowCart);
        ct.addMemento(nowCart.saveToMemento());
  5 }else if(point=="1000") {
      6 nowCart = new Point1000(nowCart);
        ct.addMemento(nowCart.saveToMemento());
    }
}
```

Derive Basis Path:



Path1: B→1→E

Path2: B→1→2→E

Path3: B→1→3→4→E

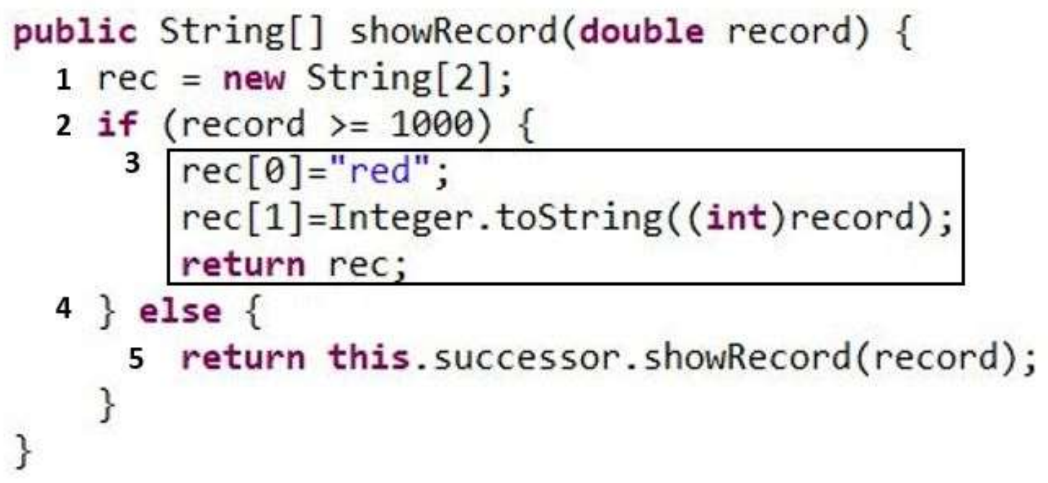
Path4: B→1→3→5→6→E

The Cyclomatic Complexity = 4

Number of edges – Number of nodes = 10 – 8 + 2 = 4

Number of closed region = 3 + 1 = 4

Number of atomic binary predicate = 3 + 1 = 4



```
public String[] showRecord(double record) {
  1 rec = new String[2];
  2 if (record >= 1000) {
     3 rec[0]="red";
       rec[1]=Integer.toString((int)record);
       return rec;
  4 } else {
     5 return this.successor.showRecord(record);
    }
}
```

Derive Basis Path:

$$\begin{tikzcd}
B \arrow[r] & 1 \arrow[r] & 2 \arrow[r] \arrow[d] & 4 \arrow[d] & \\
 & & 3 \arrow[rr, bend right] & 5 \arrow[r] & E
\end{tikzcd}$$

Path1: B→1→2→3→E

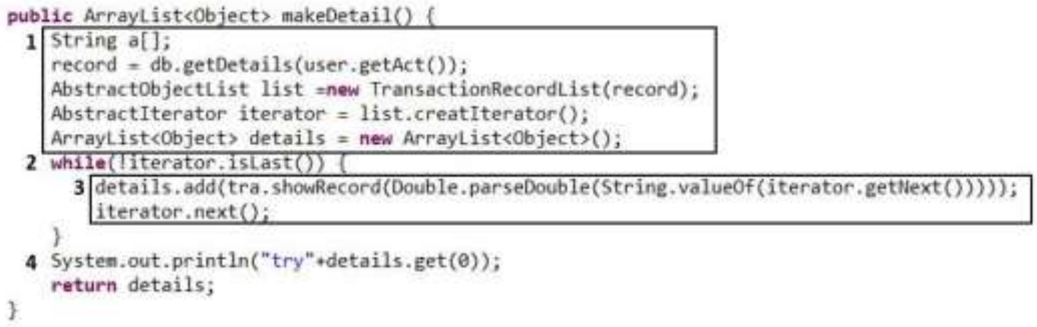
Path2: B→1→2→4→5→E

The Cyclomatic Complexity = 2

Number of edges – Number of nodes = 7 - 7 + 2= 2

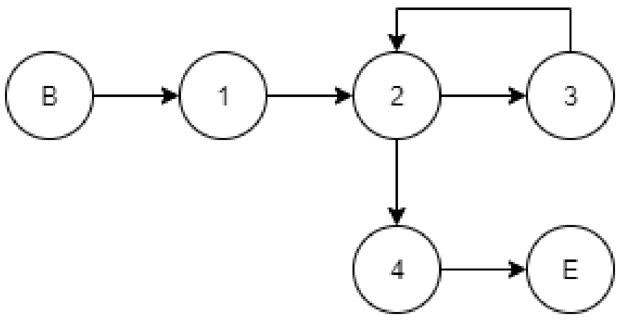
Number of closed region = 1 + 1 = 2

Number of atomic binary predicate = 1 + 1 = 2



```
public ArrayList<Object> makeDetail() {
  1 String a[];
    record = db.getDetails(user.getAct());
    AbstractObjectList list =new TransactionRecordList(record);
    AbstractIterator iterator = list.creatIterator();
    ArrayList<Object> details = new ArrayList<Object>();
  2 while(!iterator.isLast()) {
      3 details.add(tra.showRecord(Double.parseDouble(String.valueOf(iterator.getNext()))));
        iterator.next();
    }
  4 System.out.println("try"+details.get(0));
    return details;
}
```

Derive Basis Path:



Path1: B→1→2→4→E

Path2: B→1→2→3→2→4→E

The Cyclomatic Complexity = 2

Number of edges – Number of nodes = 6 – 6 + 2 = 2

Number of closed region = 1 + 1 = 2

Number of atomic binary predicate = 1 + 1 = 2

Black box testing

We choose the color of detail record show on GUI to do our software testing. GUI will show the latest 5 detail record that query from database when client get into DetailGUI. Condition dependents on response chain of TransactionRecordAmount() that defined in controller.

Equivalence Partitioning:

1. Partition 1: (amount <100)
2. Partition 2: (amount >= 100 and amount < 500)
3. Partition 3: (amount >= 500 and amount < 1000)
4. Partition 4: (amount >= 1000)

Boundary Value Analysis:

*Analysis by using Equivalence Partitioning above.

1. Partition 1: (number <100)
   { amount == 99, amount == 100, amount == 101}
2. Partition 2: (amount >= 100 and amount < 500)
   { amount == 99, amount == 100, amount == 101, amount ==499, amount ==500, amount ==501}
3. Partition 3: (amount >= 500 and amount < 1000)
   { amount ==499, amount ==500, amount ==501, amount == 999, amount ==1000, amount ==1001}
4. Partition 4: (amount >= 1000)
   { amount == 999, amount == 1000, amount == 1001}

Testing Table:

*Produced from the test case above.

*EO = Expected Output, AO = Actual Output

*This test table result is dependent on Junit test – CheckDetailColor().

#	Description	Input	EO	AO	Passing Criteria	Test Result
1	Test for n1-1	99	black	black	EO=AO	correct
2	Test for n1	100	green	green	EO=AO	correct
3	Test for n1+1	101	green	green	EO=AO	correct
4	Test for n2-1	499	green	green	EO=AO	correct
5	Test for n2	500	yellow	yellow	EO=AO	correct
6	Test for n2+1	501	yellow	yellow	EO=AO	correct
7	Test for n3-1	999	yellow	yellow	EO=AO	correct
8	Test for n3	1000	red	red	EO=AO	correct
9	Test for n3+1	1001	red	red	EO=AO	correct

5) **Please analyze the invocation chains of the new design and compare the result with the first teamwork.**

Comparing this teamwork with the first one, we found that the algorithm of the first teamwork was wrong. We added the irrelevant methods and the repeated parts together. The result of the second teamwork after correction.

We analyzed whole project's concrete class to compute invocation chains by tracing through their method. We got 66 chains that length is 1, 16 chains that length is 2, 1 chains that length is 3.

addUser(…)-> addUser(…)-> addUser(…)	2
getUserAccount()-> getAct()	1
CheckLogin(…)-> setUser(…)	1
CheckLogin(…)-> getUser(…)	1
CheckLogin(…)-> validateUser(…)-> getAct()	2
CheckLogin(…)-> validateUser(…)-> equals(…)	2
CheckLogin(…)-> validateUser(…)-> getPsd()	2
CheckLogin(…)-> getUser(…)	1
getBalance()-> getBalance()	1
PayC(…)-> getUser(…)->getUser(…)	2
PayC(…)-> setTransfer(…)	1
PayC(…)-> getBalance(…)	1
PayC(…)-> saveUser(…)-> saveUser(…)	2
PayC(…)-> getAct()	1
PayC(…)-> getPsd()	1
PayC(…)-> getCardNum()	1
PayC(…)-> getGoodThru()	1
PayC(…)-> getCVV	1

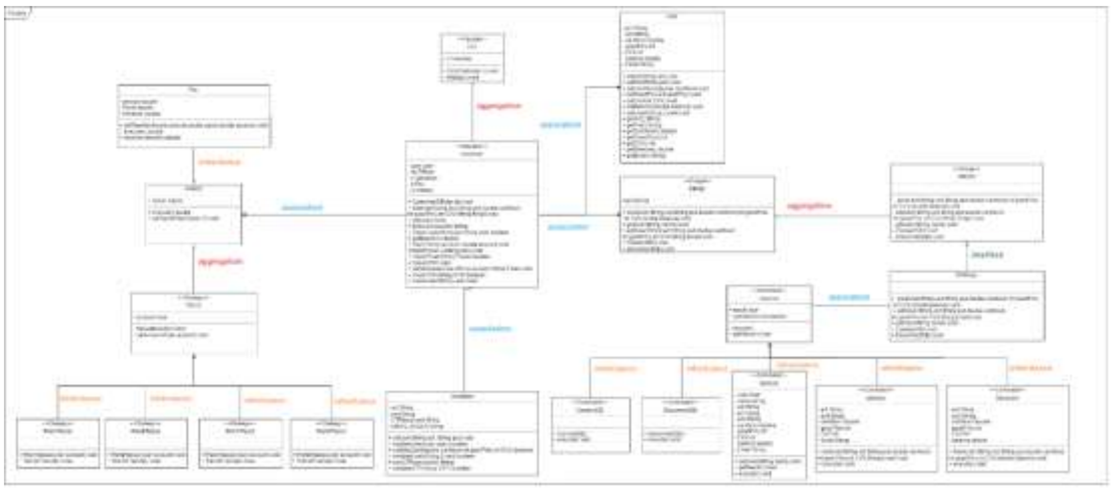
PayC(…)-> execute()	1
PayC(…)-> receiverAmount()	1
TopUp(…)-> getBalance(…)	1
TopUp(…)-> saveUser(…)-> saveUser(…)	1
TopUp(…)-> getAct()	1
TopUp(…)-> getCardNum()	1
TopUp(…)-> getGoodThru()	1
TopUp(…)-> getCVV	1
TopUp(…)-> setTopUpPlan()	1
TopUp(…)-> PlanATopUp(…)	1
TopUp(…)-> PlanBTopUp(…)	1
TopUp(…)-> PlanCTopUp(…)	1
TopUp(…)-> PlanDTopUp(…)	1
TopUp(…)-> TopUpExecute()	1
checkEmail(…)-> vaildateEmail(…)	1
makeOTP()->getAct()	1
makeOTP()-> newOTPpassword()	1
setTemporaryUser(…)-> setAct(…)	1
setTemporaryUser(…)->setEmail(…)	1
checkOTP(…)-> validateOTP(…)	1
checkUser(…)-> getUser(….)-> getUser(…)-> setBalance(…)	3
addToCart(…)-> Point100(…)-> Point(…)	2
addToCart(…)-> addMemento(…)	1
addToCart(…)-> SaveToMemento()	1
addToCart(…)-> Point500(…)-> Point(…)	2
addToCart(…)-> Point1000(…)-> Point(…)	2

iniMemento()→addMemento(…)	1
iniMemento()→saveToMemento()	1
newMemento(…)→ CartMemento(…)	1
newIterator(…) →TransactionRecordIterator(…)→ getObjlist()	2
getPayOrTopupDetails(…)→ getPayOrTopupDetails(…)	1
getPayOrTopupDetails(…)→ getAct()	1
addPayAndTopUp(…)→ addPayAndTopUp(…)→ addPayAndTopUp(…)	2
PayStatement(…)→ addPayAndTopUp(…)→ addPayAndTopUp(…)	2
PayStatement(…)→ clone()	1
PayStatement(…)→ setDatasheet(…)	1
PayStatement(…)→ setDate(…)	1
PayStatement(…)→ Date()	1
PayStatement(…)→ sql()	1
StoredTopUpStatement(double amount) → addPayAndTopUp(…)→ addPayAndTopUp(…)	2
StoredTopUpStatement(double amount) → clone()	1
StoredTopUpStatement(double amount) → setDatasheet(…)	1
StoredTopUpStatement(double amount) →setDate(…)	1
StoredTopUpStatement(double amount) → Date()	1
StoredTopUpStatement(double amount) → sql()	1
makeDetail()→getAct()	1
makeDetail()→getDetails(…) →getDetails(…)	2
makeDetail()→TransactionRecordList(…) →AbstractObjectList(…)	2
makeDetail()→creatIterator()	1
makeDetail()→isLast()	1
makeDetail()→showRecord(…)	1
makeDetail()→getNext()	1
makeDetail()→next()	1
addDetail(…) →getAct()	1
addDetail(…) →Date()	1
addDetail(…) → addDetails(…)→ addDetails(…)	2
getCartTotal()→add()	1
restoreCart()→restoreFromMemento(…)	1
restoreCart()→getLastMemento()	1
resetCart()→saveToMemento()	1
resetCart()→ConcreteCart()	1

resetCart()→CartCareTaker()	1
countCartPointNum(…) → oneHNum()	1
countCartPointNum(…) → fiveHNum()	1
countCartPointNum(…) → oneTNum()	1

Invocation Chain Length	1	2	3
Number of Chains	66	16	1

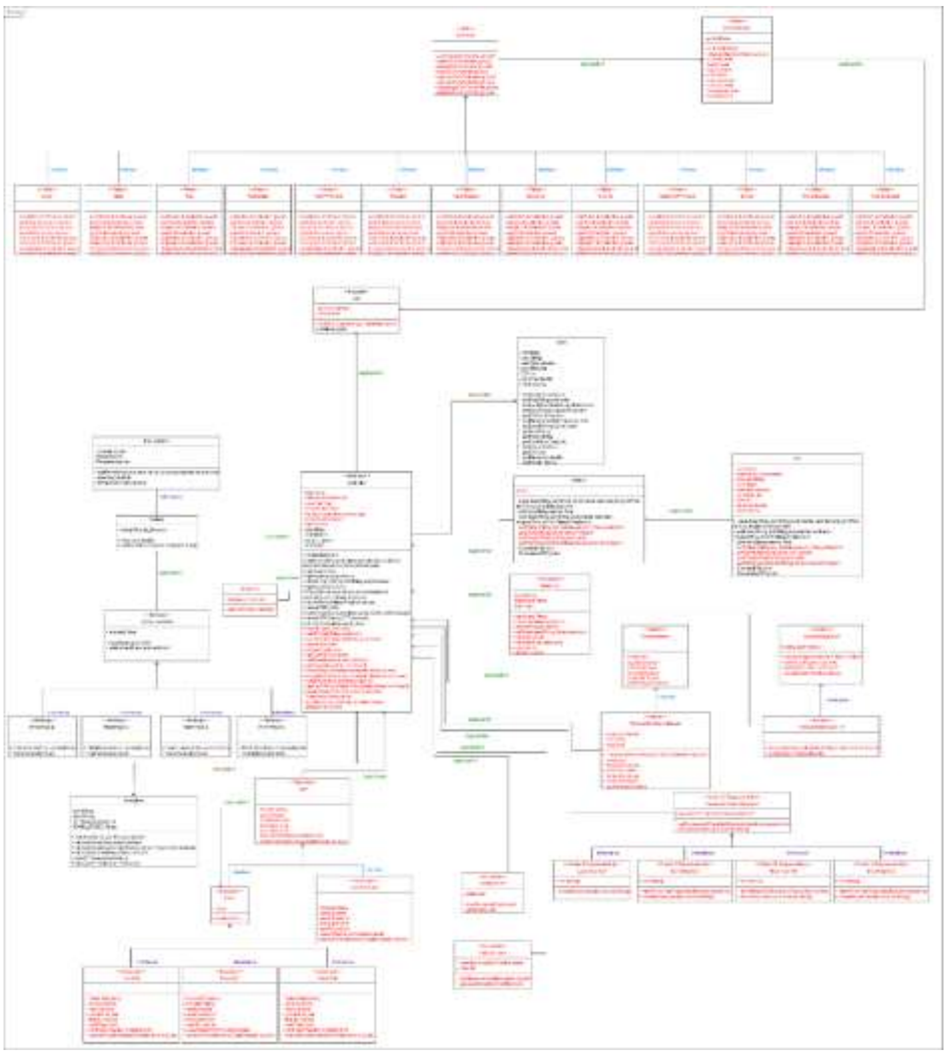
6) **Please clearly indicate the number of classes, inheritance, aggregation, association relationships, and functions of your design for the two teamwork, respectively.**



In teamwork1, we have many GUI for different features, but they do the same thing is show out state to user, so our class diagram combines all GUIs in GUI class.

According to above, there are

- 20 classes
- 10 inheritance
- 1 interface
- 3 aggregation
- 5 association
- 78 functions



In teamwork2, we also merge all GUIs in GUI class.
After modification and redesign, our class diagram has

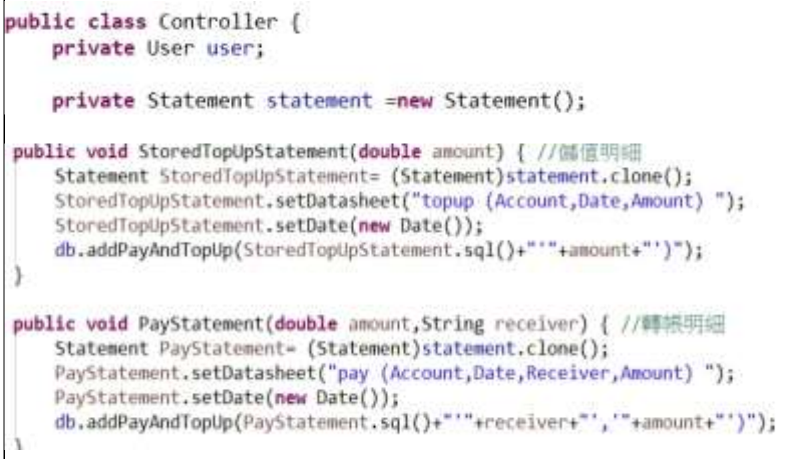
- 47 classes
- 13 inheritance
- 16 interface
- 15 aggregation
- 2 association
- 281 functions

7) Please describe three pieces of the needed changes based on the change events with examples and code. Such as class A depends on class B, then changes to class A affect class B, etc.

Part 1 Show receipt by using prototype.

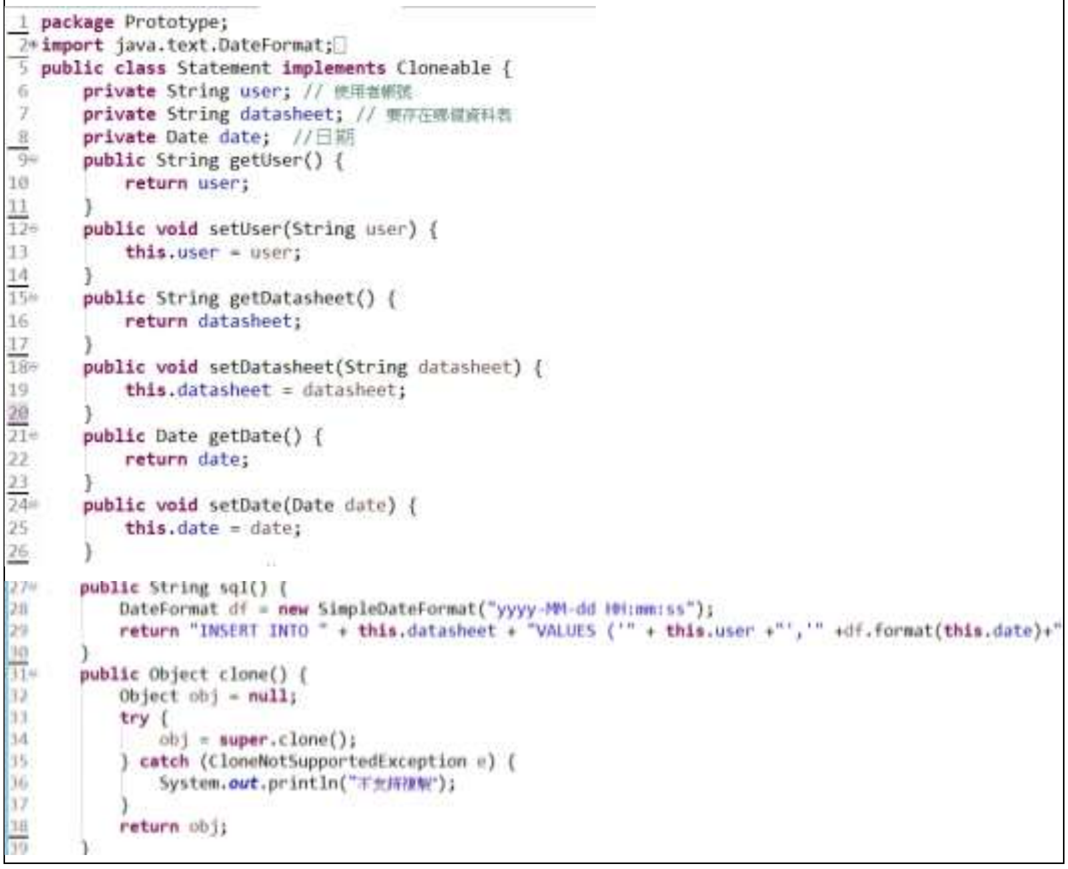
In our teamwork1 we don’t have receipt function. Now we add this function. After user TopUp or Pay money, our system will show the result. We can clone statement and modify to perform different SQL and show different result.

Controller:



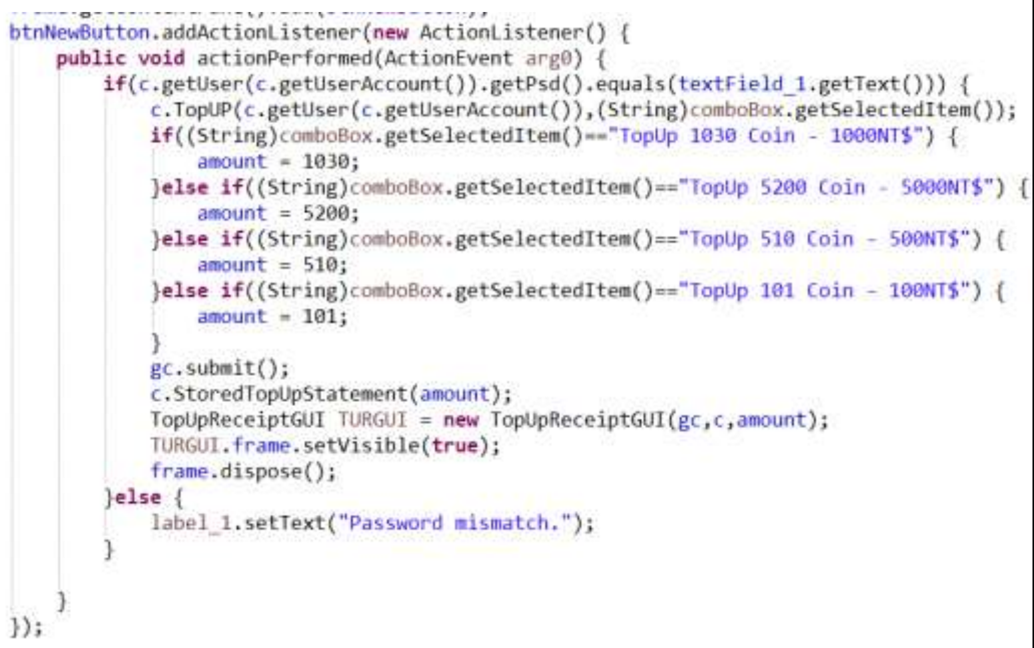
```
public class Controller {
    private User user;

    private Statement statement =new Statement();

 public void StoredTopUpStatement(double amount) { //儲值明細
     Statement StoredTopUpStatement= (Statement)statement.clone();
     StoredTopUpStatement.setDatasheet("topup (Account,Date,Amount) ");
     StoredTopUpStatement.setDate(new Date());
     db.addPayAndTopUp(StoredTopUpStatement.sql()+"'"+amount+"')");
 }

 public void PayStatement(double amount,String receiver) { //轉帳明細
     Statement PayStatement= (Statement)statement.clone();
     PayStatement.setDatasheet("pay (Account,Date,Receiver,Amount) ");
     PayStatement.setDate(new Date());
     db.addPayAndTopUp(PayStatement.sql()+"'"+receiver+"','"+amount+"')");
 }
```

Statement:



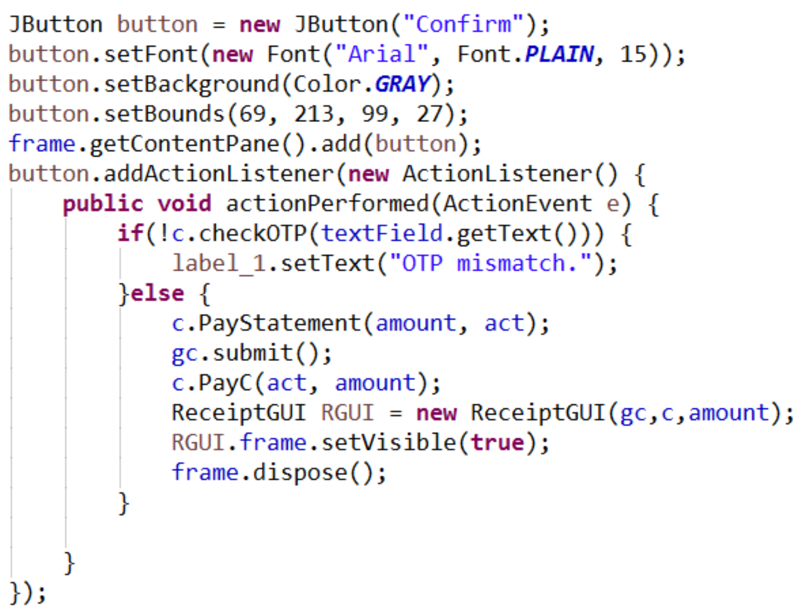
```
package Prototype;
import java.text.DateFormat;□
public class Statement implements Cloneable {
    private String user; // 使用者帳號
    private String datasheet; // 要存在哪個資料表
    private Date date;  //日期
    public String getUser() {
        return user;
    }
    public void setUser(String user) {
        this.user = user;
    }
    public String getDatasheet() {
        return datasheet;
    }
    public void setDatasheet(String datasheet) {
        this.datasheet = datasheet;
    }
    public Date getDate() {
        return date;
    }
    public void setDate(Date date) {
        this.date = date;
    }

    public String sql() {
        DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        return "INSERT INTO " + this.datasheet + "VALUES ('" + this.user +"','" +df.format(this.date)+"
    }
    public Object clone() {
        Object obj = null;
        try {
            obj = super.clone();
        } catch (CloneNotSupportedException e) {
          System.out.println("不支持複製");
        }
        return obj;
    }
```

TopUpGUI:
Using c.StoredTopUpStatement() to clone statement to show..



```
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        if(c.getUser(c.getUserAccount()).getPsd().equals(textField_1.getText())) {
            c.TopUP(c.getUser(c.getUserAccount()),(String)comboBox.getSelectedItem());
            if((String)comboBox.getSelectedItem()=="TopUp 1030 Coin - 1000NT$") {
                amount = 1030;
            }else if((String)comboBox.getSelectedItem()=="TopUp 5200 Coin - 5000NT$") {
                amount = 5200;
            }else if((String)comboBox.getSelectedItem()=="TopUp 510 Coin - 500NT$") {
                amount = 510;
            }else if((String)comboBox.getSelectedItem()=="TopUp 101 Coin - 100NT$") {
                amount = 101;
            }
            gc.submit();
            c.StoredTopUpStatement(amount);
            TopUpReceiptGUI TURGUI = new TopUpReceiptGUI(gc,c,amount);
            TURGUI.frame.setVisible(true);
            frame.dispose();
        }else {
            label_1.setText("Password mismatch.");
        }

    }
});
```

PayOTPCheckGUI:
Using c.PayStatement() to clone statement to show.

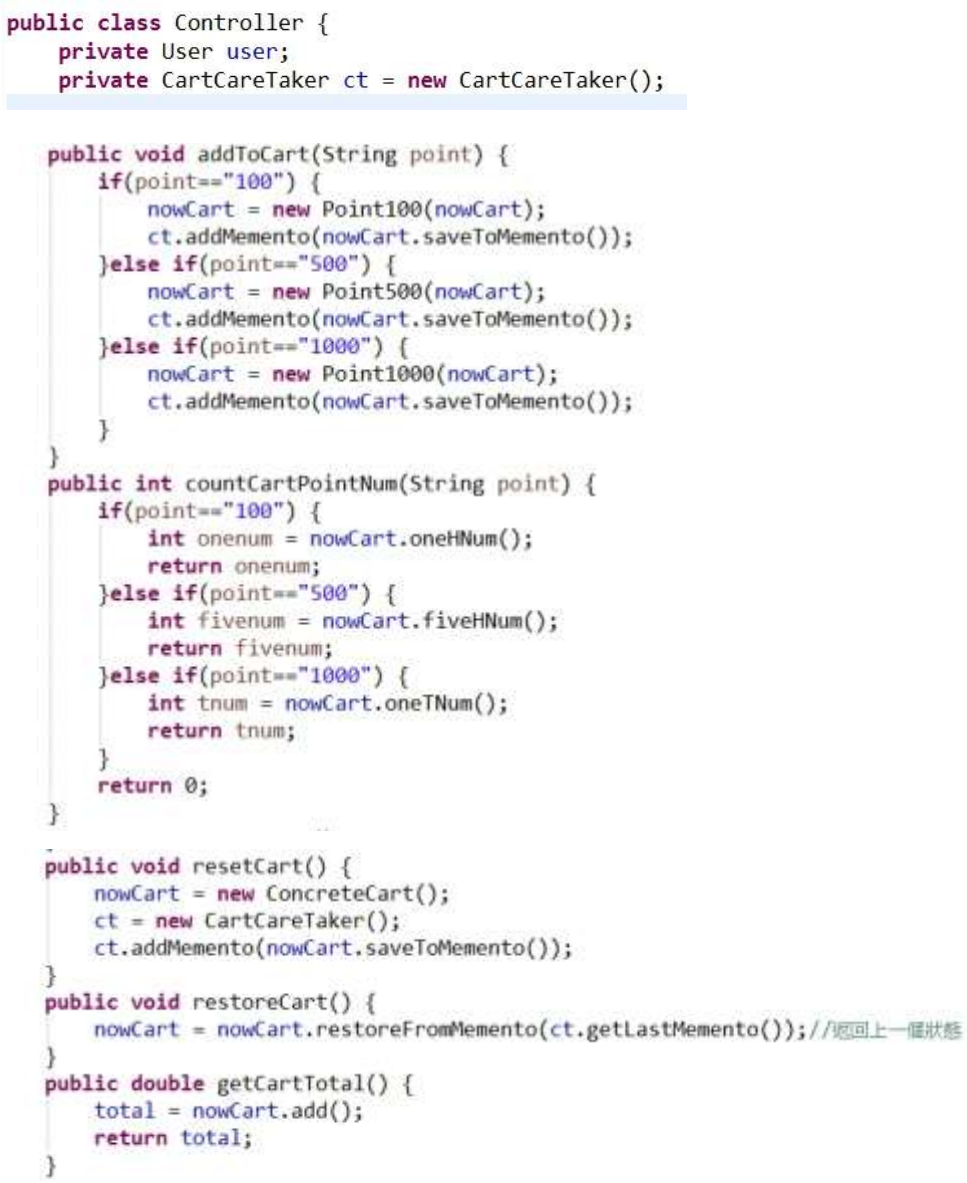


```
JButton button = new JButton("Confirm");
button.setFont(new Font("Arial", Font.PLAIN, 15));
button.setBackground(Color.GRAY);
button.setBounds(69, 213, 99, 27);
frame.getContentPane().add(button);
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(!c.checkOTP(textField.getText())) {
            label_1.setText("OTP mismatch.");
        }else {
            c.PayStatement(amount, act);
            gc.submit();
            c.PayC(act, amount);
            ReceiptGUI RGUI = new ReceiptGUI(gc,c,amount);
            RGUI.frame.setVisible(true);
            frame.dispose();
        }

    }
});
```

Part 2 Shopping function by using memento and decorator.

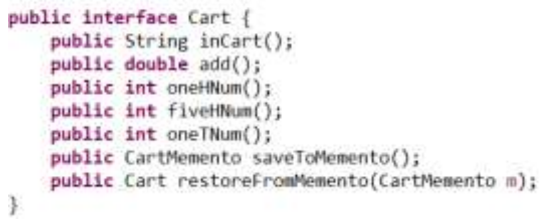
In our teamwork1 we don't have this function. Now our system provide point let user can buy in shopping function. When user click "+" button will trigger c.addToCart() then decorate this point on cart and store this cart state. When user click "Previous" will trigger c.restoreCart() then undo operation. When user click "Confirm" will trigger c. addDetail(). The point that user buy will update to DB.

Controller:



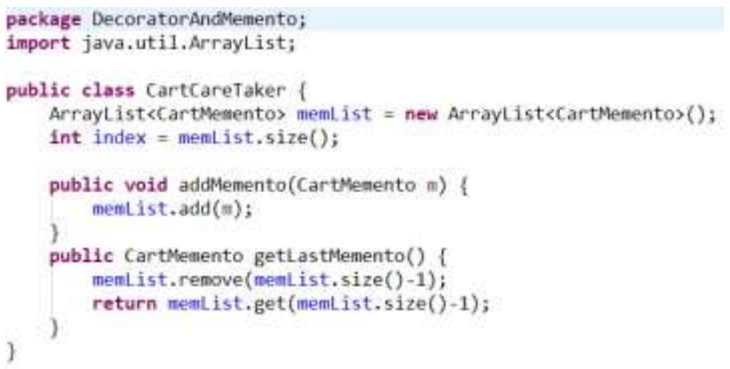
```java
public class Controller {
    private User user;
    private CartCareTaker ct = new CartCareTaker();

    public void addToCart(String point) {
        if(point=="100") {
            nowCart = new Point100(nowCart);
            ct.addMemento(nowCart.saveToMemento());
        }else if(point=="500") {
            nowCart = new Point500(nowCart);
            ct.addMemento(nowCart.saveToMemento());
        }else if(point=="1000") {
            nowCart = new Point1000(nowCart);
            ct.addMemento(nowCart.saveToMemento());
        }
    }
    public int countCartPointNum(String point) {
        if(point=="100") {
            int onenum = nowCart.oneHNum();
            return onenum;
        }else if(point=="500") {
            int fivenum = nowCart.fiveHNum();
            return fivenum;
        }else if(point=="1000") {
            int tnum = nowCart.oneTNum();
            return tnum;
        }
        return 0;
    }

    public void resetCart() {
        nowCart = new ConcreteCart();
        ct = new CartCareTaker();
        ct.addMemento(nowCart.saveToMemento());
    }
    public void restoreCart() {
        nowCart = nowCart.restoreFromMemento(ct.getLastMemento());//返回上一個狀態
    }
    public double getCartTotal() {
        total = nowCart.add();
        return total;
    }
```

Cart:



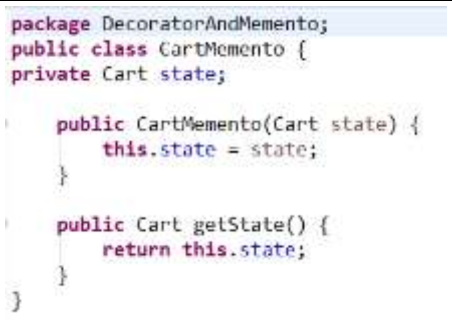
```java
public interface Cart {
    public String inCart();
    public double add();
    public int oneHNum();
    public int fiveHNum();
    public int oneTNum();
    public CartMemento saveToMemento();
    public Cart restoreFromMemento(CartMemento m);
}
```

CareTaker:



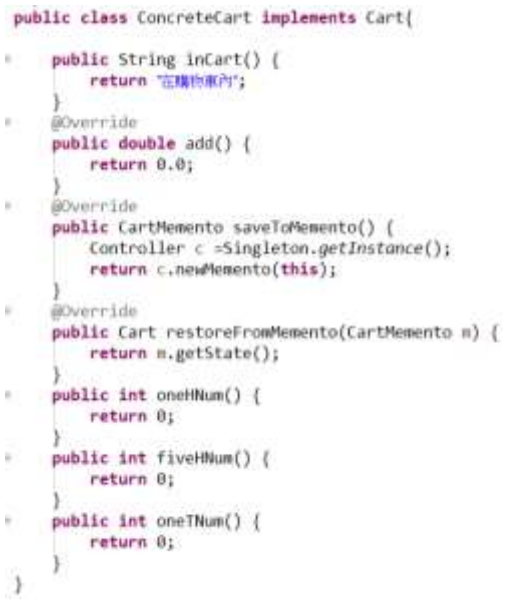
```java
package DecoratorAndMemento;
import java.util.ArrayList;

public class CartCareTaker {
    ArrayList<CartMemento> memList = new ArrayList<CartMemento>();
    int index = memList.size();

    public void addMemento(CartMemento m) {
        memList.add(m);
    }
    public CartMemento getLastMemento() {
        memList.remove(memList.size()-1);
        return memList.get(memList.size()-1);
    }
}
```

CartMenemto:



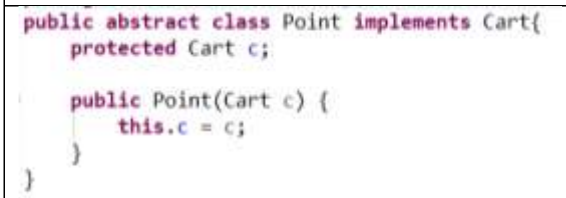
```java
package DecoratorAndMemento;
public class CartMemento {
private Cart state;

    public CartMemento(Cart state) {
        this.state = state;
    }

    public Cart getState() {
        return this.state;
    }
}
```

ConcreteCart:



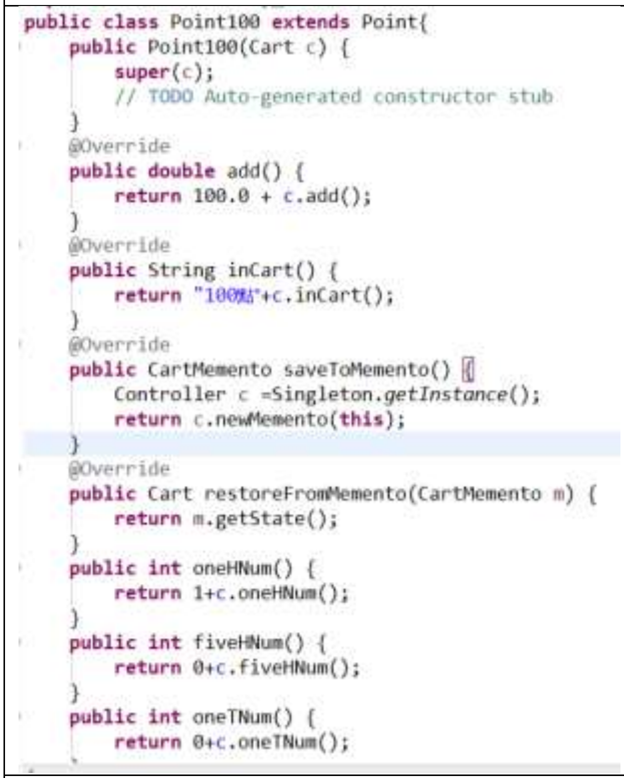
```java
public class ConcreteCart implements Cart{

    public String inCart() {
        return "在購物車內";
    }
    @Override
    public double add() {
        return 0.0;
    }
    @Override
    public CartMemento saveToMemento() {
        Controller c =Singleton.getInstance();
        return c.newMemento(this);
    }
    @Override
    public Cart restoreFromMemento(CartMemento m) {
        return m.getState();
    }
    public int oneHNum() {
        return 0;
    }
    public int fiveHNum() {
        return 0;
    }
    public int oneTNum() {
        return 0;
    }
}
```

Point:



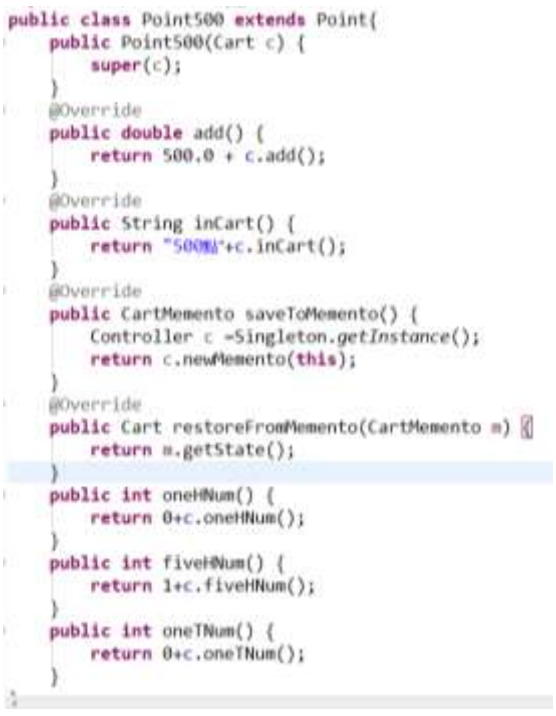
```java
public abstract class Point implements Cart{
    protected Cart c;

    public Point(Cart c) {
        this.c = c;
    }
}
```

Point100:



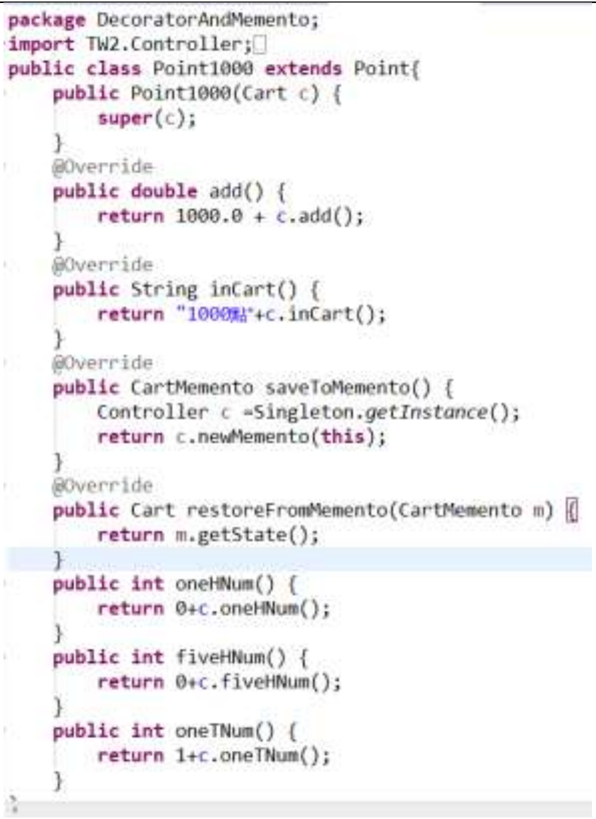
```java
public class Point100 extends Point{
    public Point100(Cart c) {
        super(c);
        // TODO Auto-generated constructor stub
    }
    @Override
    public double add() {
        return 100.0 + c.add();
    }
    @Override
    public String inCart() {
        return "100點"+c.inCart();
    }
    @Override
    public CartMemento saveToMemento() {
        Controller c =Singleton.getInstance();
        return c.newMemento(this);
    }
    @Override
    public Cart restoreFromMemento(CartMemento m) {
        return m.getState();
    }
    public int oneHNum() {
        return 1+c.oneHNum();
    }
    public int fiveHNum() {
        return 0+c.fiveHNum();
    }
    public int oneTNum() {
        return 0+c.oneTNum();
```

Point500:



```java
public class Point500 extends Point{
    public Point500(Cart c) {
        super(c);
    }
    @Override
    public double add() {
        return 500.0 + c.add();
    }
    @Override
    public String inCart() {
        return "500點"+c.inCart();
    }
    @Override
    public CartMemento saveToMemento() {
        Controller c =Singleton.getInstance();
        return c.newMemento(this);
    }
    @Override
    public Cart restoreFromMemento(CartMemento m) {
        return m.getState();
    }
    public int oneHNum() {
        return 0+c.oneHNum();
    }
    public int fiveHNum() {
        return 1+c.fiveHNum();
    }
    public int oneTNum() {
        return 0+c.oneTNum();
    }
```

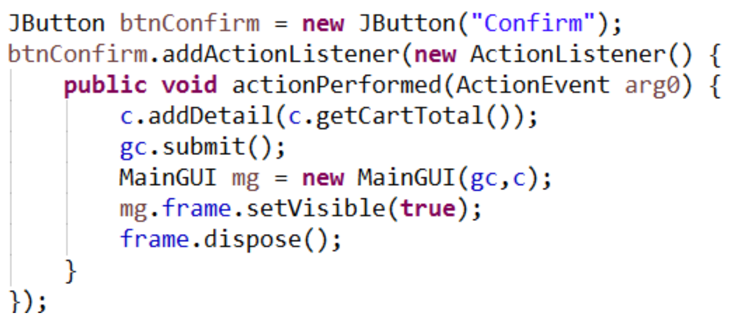
Point1000:



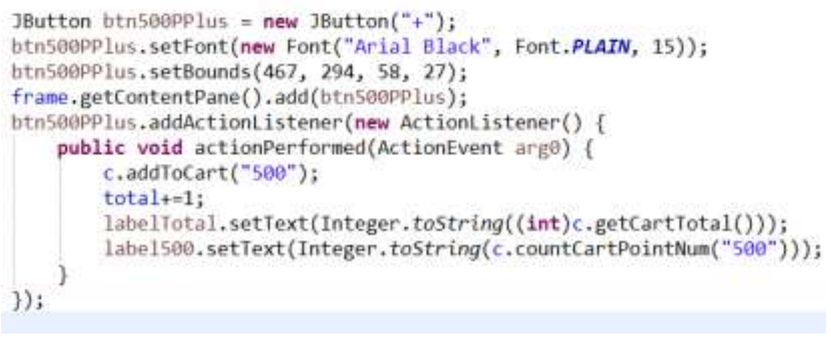
```
package DecoratorAndMemento;
import TW2.Controller;
public class Point1000 extends Point{
    public Point1000(Cart c) {
        super(c);
    }
    @Override
    public double add() {
        return 1000.0 + c.add();
    }
    @Override
    public String inCart() {
        return "1000點"+c.inCart();
    }
    @Override
    public CartMemento saveToMemento() {
        Controller c =Singleton.getInstance();
        return c.newMemento(this);
    }
    @Override
    public Cart restoreFromMemento(CartMemento m) {
        return m.getState();
    }
    public int oneHNum() {
        return 0+c.oneHNum();
    }
    public int fiveHNum() {
        return 0+c.fiveHNum();
    }
    public int oneTNum() {
        return 1+c.oneTNum();
    }
}
```

ShoppingGUI:

Using c.addDetail() to add this record to DB. Using c.addToCart() to decorate point to cart.



```
JButton btnConfirm = new JButton("Confirm");
btnConfirm.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        c.addDetail(c.getCartTotal());
        gc.submit();
        MainGUI mg = new MainGUI(gc,c);
        mg.frame.setVisible(true);
        frame.dispose();
    }
});
```

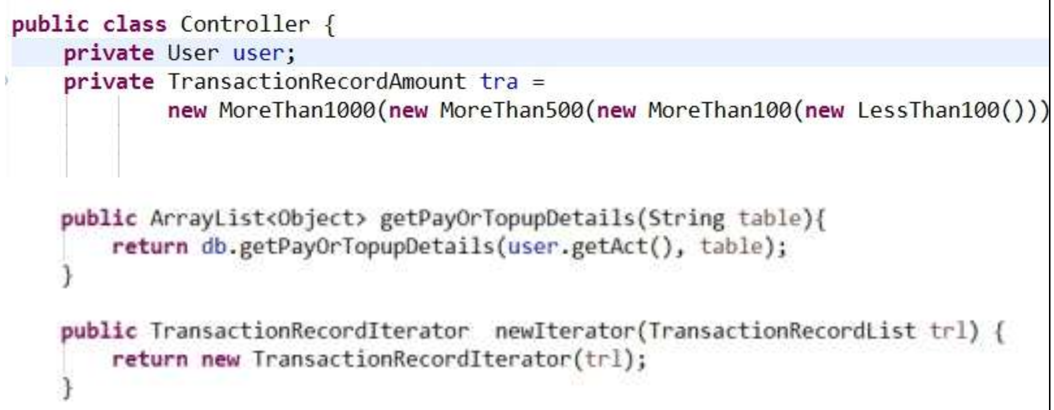


```
JButton btn500PPlus = new JButton("+");
btn500PPlus.setFont(new Font("Arial Black", Font.PLAIN, 15));
btn500PPlus.setBounds(467, 294, 58, 27);
frame.getContentPane().add(btn500PPlus);
btn500PPlus.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        c.addToCart("500");
        total+=1;
        labelTotal.setText(Integer.toString((int)c.getCartTotal()));
        label500.setText(Integer.toString(c.countCartPointNum("500")));
    }
});
```

Part 3   Detail show with different color.

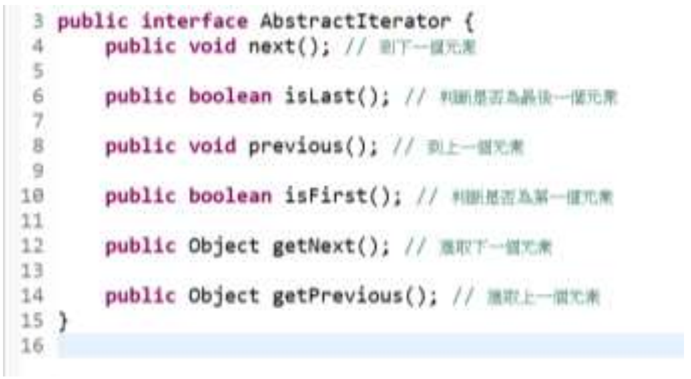
In our teamwork1 we don't have this function. Now our system have account detail including pay detail, TopUp detail, shopping detail. In shopping detail is show the latest 5 record that user buy how many point. Each record will show different color after Iterator processed. Iterator is process data with chain of responsibility.

Controller:



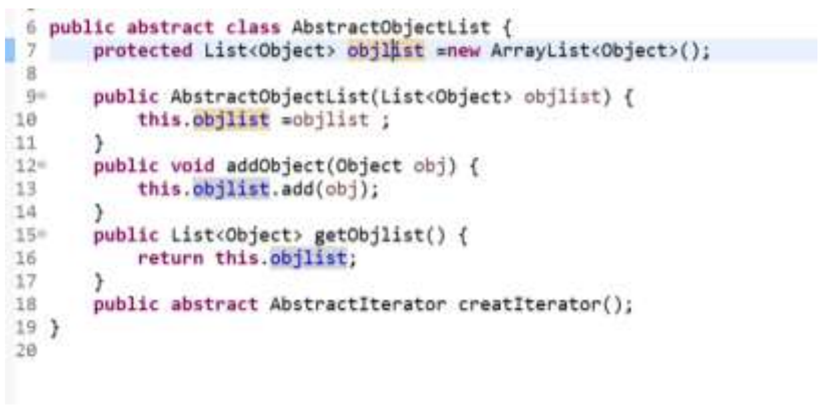
```java
public class Controller {
    private User user;
    private TransactionRecordAmount tra =
            new MoreThan1000(new MoreThan500(new MoreThan100(new LessThan100()))

    public ArrayList<Object> getPayOrTopupDetails(String table){
        return db.getPayOrTopupDetails(user.getAct(), table);
    }

    public TransactionRecordIterator  newIterator(TransactionRecordList trl) {
        return new TransactionRecordIterator(trl);
    }
```

AbstractIterator:



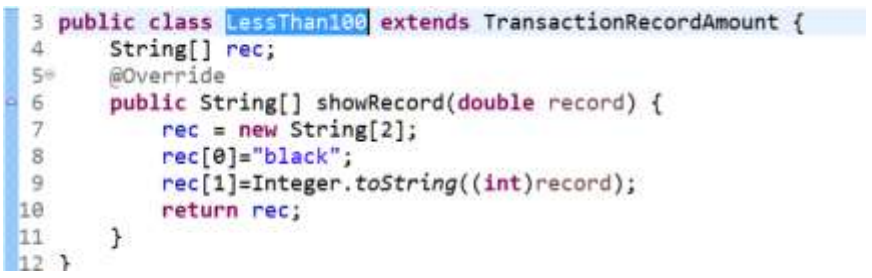
```java
public interface AbstractIterator {
    public void next(); // 到下一個元素

    public boolean isLast(); // 判斷是否為最後一個元素

    public void previous(); // 到上一個元素

    public boolean isFirst(); // 判斷是否為第一個元素

    public Object getNext(); // 讀取下一個元素

    public Object getPrevious(); // 讀取上一個元素
}
```

AbstractObjectList:



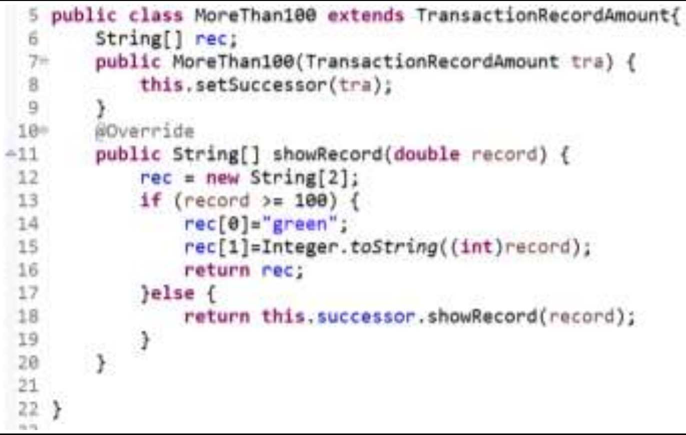
```java
public abstract class AbstractObjectList {
    protected List<Object> objlist =new ArrayList<Object>();

    public AbstractObjectList(List<Object> objlist) {
        this.objlist =objlist ;
    }
    public void addObject(Object obj) {
        this.objlist.add(obj);
    }
    public List<Object> getObjlist() {
        return this.objlist;
    }
    public abstract AbstractIterator creatIterator();
}
```

LessThan100:



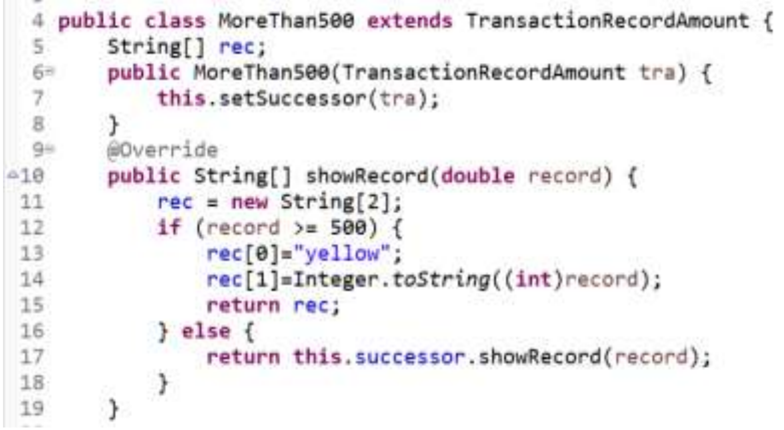
```java
public class LessThan100 extends TransactionRecordAmount {
    String[] rec;
    @Override
    public String[] showRecord(double record) {
        rec = new String[2];
        rec[0]="black";
        rec[1]=Integer.toString((int)record);
        return rec;
    }
}
```

MoreThan100:



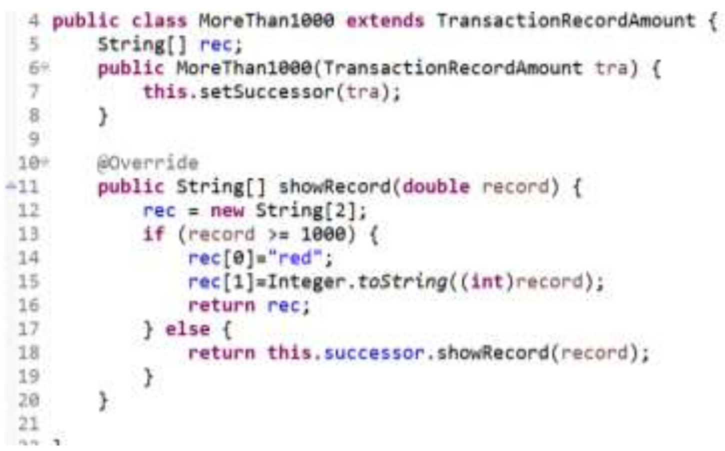
```java
public class MoreThan100 extends TransactionRecordAmount{
    String[] rec;
    public MoreThan100(TransactionRecordAmount tra) {
        this.setSuccessor(tra);
    }
    @Override
    public String[] showRecord(double record) {
        rec = new String[2];
        if (record >= 100) {
            rec[0]="green";
            rec[1]=Integer.toString((int)record);
            return rec;
        }else {
            return this.successor.showRecord(record);
        }
    }

}
```

MoreThan500:



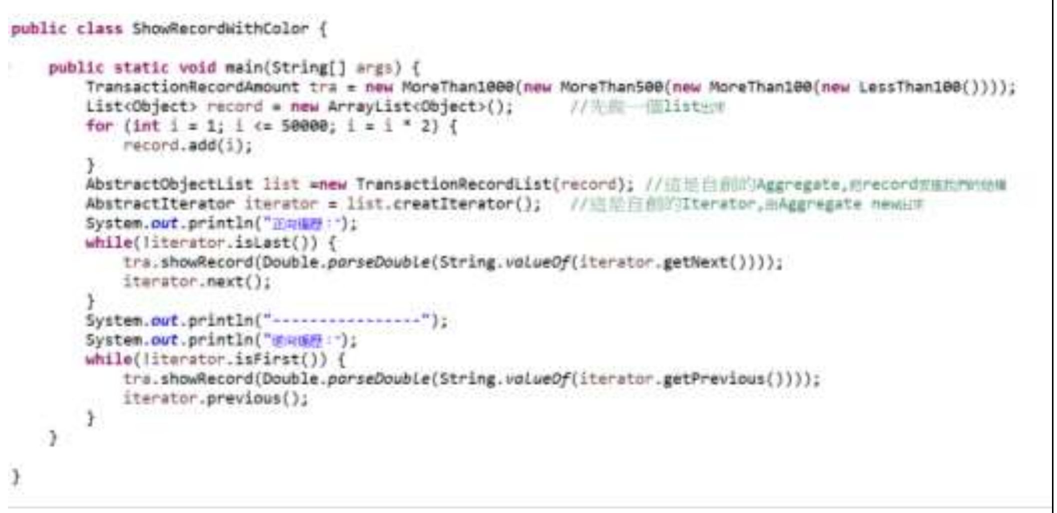
```java
public class MoreThan500 extends TransactionRecordAmount {
    String[] rec;
    public MoreThan500(TransactionRecordAmount tra) {
        this.setSuccessor(tra);
    }
    @Override
    public String[] showRecord(double record) {
        rec = new String[2];
        if (record >= 500) {
            rec[0]="yellow";
            rec[1]=Integer.toString((int)record);
            return rec;
        } else {
            return this.successor.showRecord(record);
        }
    }
```

MoreThan1000:



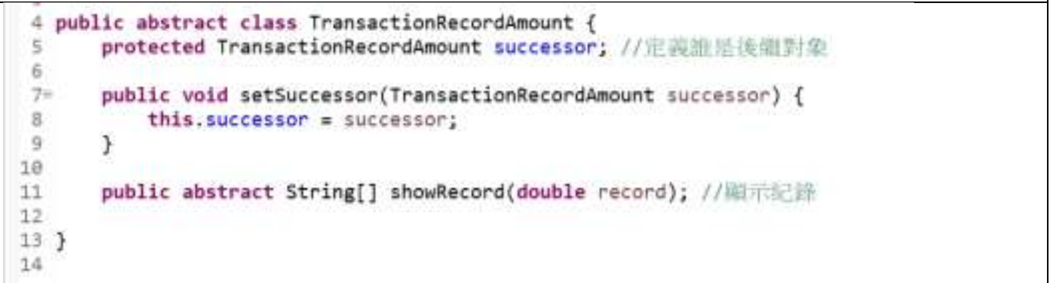
```java
public class MoreThan1000 extends TransactionRecordAmount {
    String[] rec;
    public MoreThan1000(TransactionRecordAmount tra) {
        this.setSuccessor(tra);
    }

    @Override
    public String[] showRecord(double record) {
        rec = new String[2];
        if (record >= 1000) {
            rec[0]="red";
            rec[1]=Integer.toString((int)record);
            return rec;
        } else {
            return this.successor.showRecord(record);
        }
    }

```

ShowRecordWithColor:



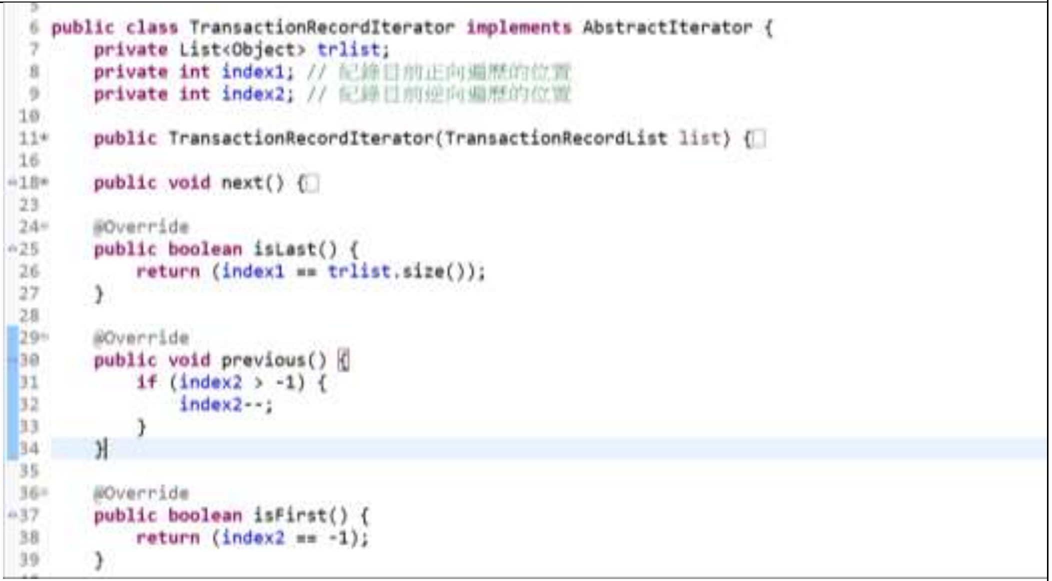
```java
public class ShowRecordWithColor {

    public static void main(String[] args) {
        TransactionRecordAmount tra = new MoreThan1000(new MoreThan500(new MoreThan100(new LessThan100())));
        List<Object> record = new ArrayList<Object>();      //先做一個list出來
        for (int i = 1; i <= 50000; i = i * 2) {
            record.add(i);
        }
        AbstractObjectList list =new TransactionRecordList(record); //這是自創的Aggregate,將record[illegible]
        AbstractIterator iterator = list.creatIterator();   //這是自創的Iterator,由Aggregate new出來
        System.out.println("正向遍歷：");
        while(!iterator.isLast()) {
            tra.showRecord(Double.parseDouble(String.valueOf(iterator.getNext())));
            iterator.next();
        }
        System.out.println("---------------");
        System.out.println("逆向遍歷：");
        while(!iterator.isFirst()) {
            tra.showRecord(Double.parseDouble(String.valueOf(iterator.getPrevious())));
            iterator.previous();
        }
    }

}
```

TransactionRecordAmount:

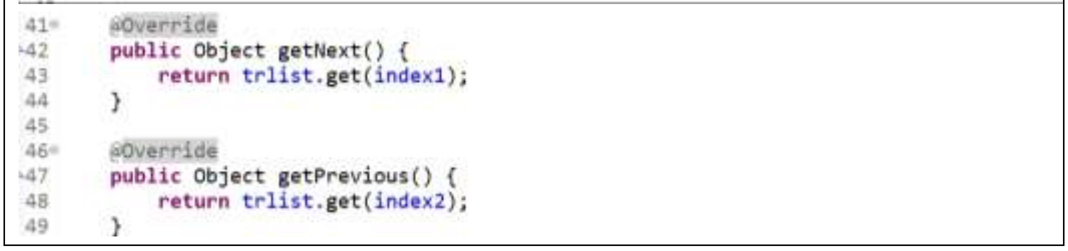


```java
public abstract class TransactionRecordAmount {
    protected TransactionRecordAmount successor; //定義誰是後繼對象

    public void setSuccessor(TransactionRecordAmount successor) {
        this.successor = successor;
    }

    public abstract String[] showRecord(double record); //顯示紀錄

}
```

TransactionRecordIterator:

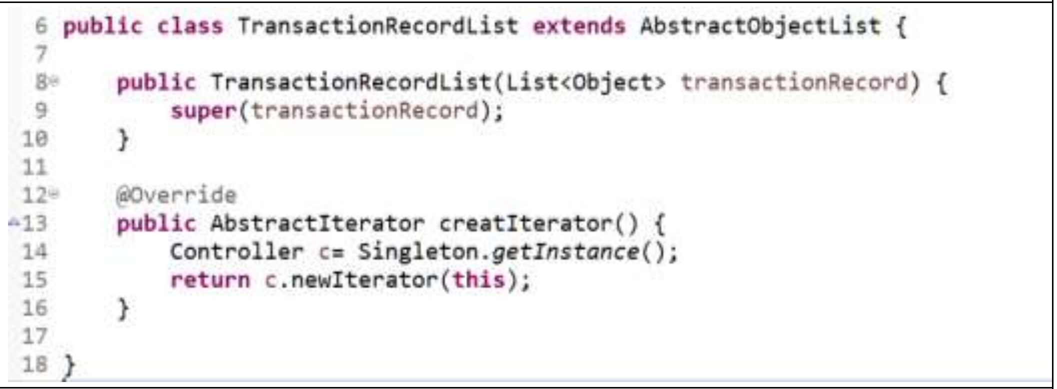


```java
public class TransactionRecordIterator implements AbstractIterator {
    private List<Object> trlist;
    private int index1; // 紀錄目前正向遍歷的位置
    private int index2; // 紀錄目前逆向遍歷的位置

    public TransactionRecordIterator(TransactionRecordList list) {

    public void next() {

    @Override
    public boolean isLast() {
        return (index1 == trlist.size());
    }

    @Override
    public void previous() {
        if (index2 > -1) {
            index2--;
        }
    }

    @Override
    public boolean isFirst() {
        return (index2 == -1);
    }
```



```java
    @Override
    public Object getNext() {
        return trlist.get(index1);
    }

    @Override
    public Object getPrevious() {
        return trlist.get(index2);
    }
```

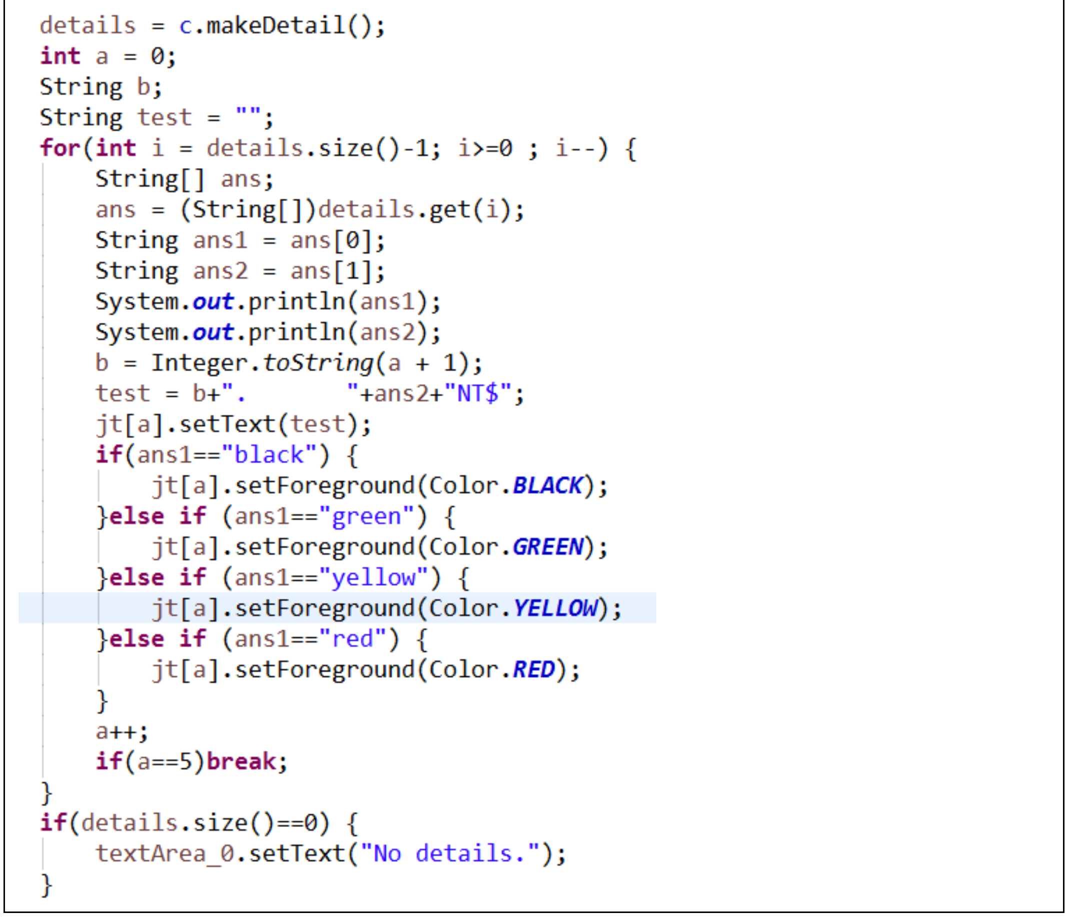
TransactionRecordList:



```java
public class TransactionRecordList extends AbstractObjectList {

    public TransactionRecordList(List<Object> transactionRecord) {
        super(transactionRecord);
    }

    @Override
    public AbstractIterator creatIterator() {
        Controller c= Singleton.getInstance();
        return c.newIterator(this);
    }

}
```

DetailGUI:

Set different foreground color according to the result of c.makeDetail().



```java
details = c.makeDetail();
int a = 0;
String b;
String test = "";
for(int i = details.size()-1; i>=0 ; i--) {
    String[] ans;
    ans = (String[])details.get(i);
    String ans1 = ans[0];
    String ans2 = ans[1];
    System.out.println(ans1);
    System.out.println(ans2);
    b = Integer.toString(a + 1);
    test = b+".        "+ans2+"NT$";
    jt[a].setText(test);
    if(ans1=="black") {
        jt[a].setForeground(Color.BLACK);
    }else if (ans1=="green") {
        jt[a].setForeground(Color.GREEN);
    }else if (ans1=="yellow") {
        jt[a].setForeground(Color.YELLOW);
    }else if (ans1=="red") {
        jt[a].setForeground(Color.RED);
    }
    a++;
    if(a==5)break;
}
if(details.size()==0) {
    textArea_0.setText("No details.");
}
```

Participation

Number	Name	Grade
B10523038	Edward	H
B10523024	Steven	H
B10523032	Xavier	H
B10523033	Wing	H
B10523021	Johnny	H
B10523037	Yee	H
B10523006	Peggy	H
B10523007	Bess	H
B10523005	Aliss	H
B10523056	Sandy	H

Appendix

Highlight/mark the new applied patterns in the class diagram.