Object-Oriented

Software Engineering

Instructor : Huang, Chuen-Min

**Teamwork2 ver.1**

Group 3

|  |  |
| --- | --- |
| ID | Name |
| A10723029 | Natalia |
| A10723035 | Sunny |
| A10723011 | Bryan |
| A10723026 | Elian |
| A10723044 | Ricky |
| B10523039 | Jess |
| B10523019 | Jason |
| B10523030 | Jerry |
|  |  |
|  |  |
| Date 2019/12/14 | |

**Content**

[**Content** I](#_Toc27322907)

[Pattern 1](#_Toc27322908)

[ **Iterator** combines with **Factory Method** 1](#_Toc27322909)

[ **State** 5](#_Toc27322910)

[ **Template** 10](#_Toc27322911)

[ **Singleton** 13](#_Toc27322912)

[ **Chain of Responsibility** 14](#_Toc27322913)

[ **Memento** 20](#_Toc27322914)

[**Big class diagram** 23](#_Toc27322915)

[**Highlight/mark the new applied patterns in the class diagram.** 24](#_Toc27322916)

[1. Display some snapshots of the new result in the report. 28](#_Toc27322917)

[2. You need to evaluate the design quality of the new design by using object-oriented 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. 34](#_Toc27322918)

[WMC(Weighted Methods per Class) 34](#_Toc27322919)

[DIT (Depth of Inheritance) 35](#_Toc27322920)

[NOC (Number of Children) 36](#_Toc27322921)

[CBO (Coupling between Object Classes) 36](#_Toc27322922)

[RFC (Response for a Class) 37](#_Toc27322923)

[LCOM (Lack of Cohesion Methods) 38](#_Toc27322924)

[3. Create Junit test cases and Junit test suite to test one new selected class. 39](#_Toc27322925)

[**Junit test** 39](#_Toc27322926)

[**Junit test suite** 40](#_Toc27322927)

[4. Conduct a new part of the software testing including white box and black box. 42](#_Toc27322928)

[**Black box** 42](#_Toc27322929)

[Equivalence Partitioning 42](#_Toc27322930)

[Boundary Value Analysis 42](#_Toc27322931)

[**White Box** 42](#_Toc27322932)

[Basis Path Testing 42](#_Toc27322933)

[Cyclomatic Complexity 44](#_Toc27322934)

[5. Please analyze the invocation chains of the new design and compare the result with the first teamwork. 45](#_Toc27322935)

[6. Please clearly indicate the number of classes, inheritance, aggregation, association relationships, and functions of your design for the two teamwork, respectively. 51](#_Toc27322936)

[The number of classes 51](#_Toc27322937)

[The number of inheritances 51](#_Toc27322938)

[The number of aggregations 51](#_Toc27322939)

[The number of associations 52](#_Toc27322940)

[Functions of your design for two teamwork 52](#_Toc27322941)

[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. 52](#_Toc27322942)

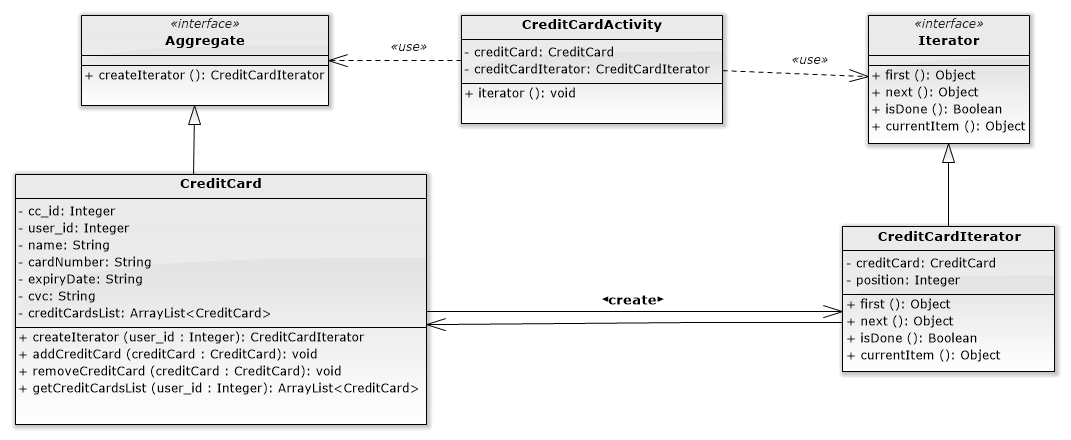
[**Participation** 53](#_Toc27322943)

Pattern

* **Iterator** combines with **Factory Method**

Intent:

The user can have more than one credit card in ElePay. So the Iterator Pattern can iterate every credit card that user has, then we can know how many credit card user has.



▲Iterator structure in Elepay

**Interface: Aggregate**

|  |
| --- |
| **package com.example.elepay.Iterator;  public interface Aggregate {  public CreditCardIterator createIterator(int user\_id); }** |

**Class: CreditCard**

|  |
| --- |
| **package com.example.elepay.Iterator;  import com.example.elepay.Singleton.DB\_Connect;  import java.util.ArrayList;  public class CreditCard implements Aggregate{  int cc\_id;  int user\_id;  String name;  String cardNumber;  String expiryDate;  String cvc;   private ArrayList<CreditCard> creditCardsList = new ArrayList<CreditCard>();   public CreditCard(){   }   public CreditCard(int cc\_id,int user\_id, String name, String cardNumber, String expiryDate, String cvc) {  this.cc\_id = cc\_id;  this.user\_id = user\_id;  this.name = name;  this.cardNumber = cardNumber;  this.expiryDate = expiryDate;  this.cvc = cvc;  }  public void setUser\_id(int user\_id){  this.user\_id = user\_id;  }   public int getUser\_id(){  return this.user\_id;  }   public void setCc\_id(int cc\_id) {  this.cc\_id = cc\_id;  }   public int getCc\_id() {  return cc\_id;  }   public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }   public String getCardNumber() {  return cardNumber;  }   public void setCardNumber(String cardNumber) {  this.cardNumber = cardNumber;  }   public String getExpiryDate() {  return expiryDate;  }   public void setExpiryDate(String expiryDate) {  this.expiryDate = expiryDate;  }   public String getCvc() {  return cvc;  }   public void setCvc(String cvc) {  this.cvc = cvc;  }   @Override  public CreditCardIterator createIterator(int user\_id) {  this.user\_id = user\_id;  return new CreditCardIterator(this);  }   public void addCreditCard(CreditCard creditCard){  creditCardsList.add(creditCard);  }   public void removeCreditCard(CreditCard creditCard){  creditCardsList.remove(creditCard);  }   public ArrayList<CreditCard> getCreditCardsList(int user\_id){  DB\_Connect db = new DB\_Connect();  return db.getCreditCardByUserId(user\_id);  } }** |

**interface: Iterator**

|  |
| --- |
| **package com.example.elepay.Iterator;  public interface Iterator {  public Object first();  public Object next();  public boolean isDone();  public Object currentItem(); }** |

**Class: CreditCardIterator**

|  |
| --- |
| **package com.example.elepay.Iterator;  public class CreditCardIterator implements Iterator {  private CreditCard creditCard;  private int position = 0;   public CreditCardIterator(CreditCard cc){  this.creditCard = cc;  }   @Override  public Object first() {  return creditCard.getCreditCardsList(creditCard.user\_id).get(position);  }   @Override  public Object next() {  Object next = null;  position++;  if(position < creditCard.getCreditCardsList(creditCard.user\_id).size()){  next = creditCard.getCreditCardsList(creditCard.user\_id).get(position);  }  return next;  }   @Override  public boolean isDone() {  //Integer count = creditCard.getCreditCardsList(creditCard.user\_id).size();  return position >= creditCard.getCreditCardsList(creditCard.user\_id).size() ? true : false;  }   @Override  public Object currentItem() {  return creditCard.getCreditCardsList(creditCard.user\_id).get(position);  }   public int getPosition() {  return position;  }   public void setPosition(int position) {  this.position = position;  } }** |

**Client in CreditCardActivity**

|  |
| --- |
| **public void iterator(){  while(!creditCardIterator.isDone()){  Log.*e*("iterator", ((CreditCard)creditCardIterator.currentItem()).getName() + " "  + ((CreditCard)creditCardIterator.currentItem()).getCardNumber() + " "  + ((CreditCard)creditCardIterator.currentItem()).getExpiryDate() + " "  + ((CreditCard)creditCardIterator.currentItem()).getCvc() );  sumCredit++;  creditCardIterator.next();  }  Toast.*makeText*(this, "總共有" + sumCredit + "張卡片", Toast.*LENGTH\_LONG*).show(); }** |

* **State**

Intent:

There’ll be several states changes when the users are making a transfer. We want to let the users know which step they’re currently at, so we use the State Pattern to present the status change to user.

We have five state in Transaction: 1.No Transfer Yet, 2.No Transfer Information Filled, 3. Setting Up Transfer Information, 4.Checking Transfer, 5.Confirm Successful Transfer. And there’re four events: 1.Press Confirm Button, 2.Press Cancel Buton, 3.Enter Transfer Info, 4.Press Transfer Button.

When the user wanted to make a transfer, the user will first be in “No Transfer Yet” state, after pressing the Transfer Button, the “Press Transfer Button” event activated, then the state will change to abstract state “Transfer Procedure”, and its initial state is “No Transfer Information Filled” means no information has been filled in.

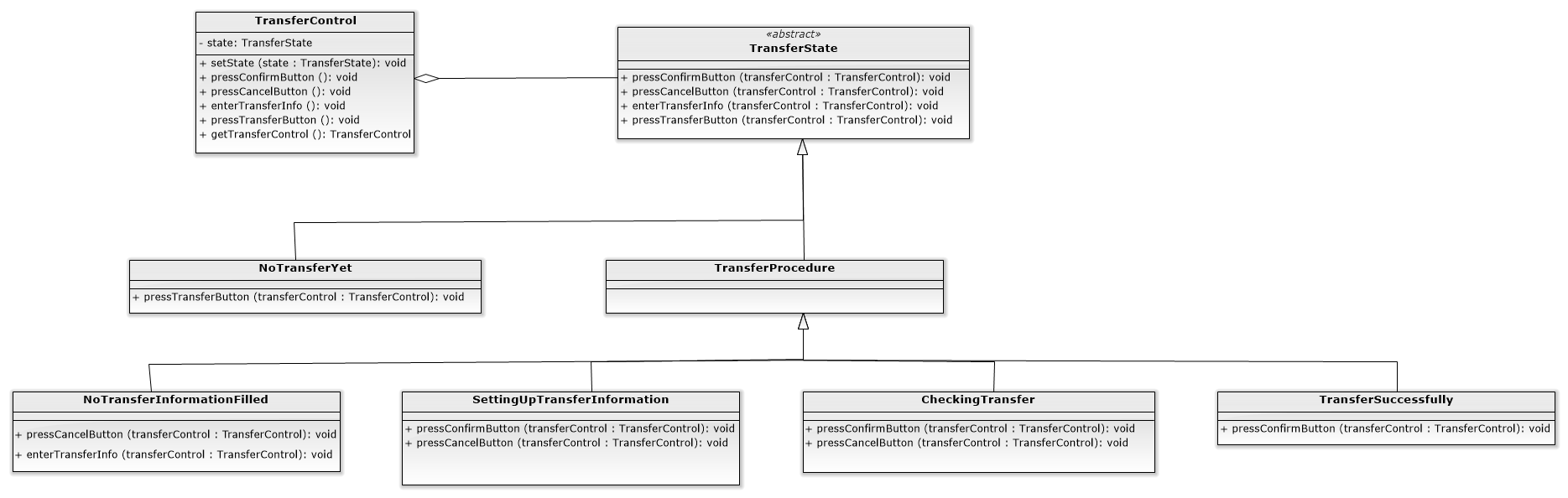
There’re four sub-states in “Transfer Procedure”: 1.No Transfer Information Filled”, 2. Setting Up Transfer Information, 3.Checking Transfer, 4.Confirm Successful Transfer. If the user P

When the “Enter Transfer Info” event happened, the state will change from initial state “No Transfer Information Filled” to “Setting Up Transfer Information”. There’ll be two condition in “Setting Up Transfer Information”: 1.Confirmation Button Pressed, 2.Cancellation Button Pressed. If the user press Confirm, then the state will turn into “Checking transfer”. On the other hand, the Cancellation Button been pressed, the state goes back to the initial state “No transfer Information Filled”.

Assuming we have done setting the transfer information, we’re now at “Checking Transfer” state, in this state, we have two different options: 1.Press Confirm Button to proceed the transaction or 2.Press cancel Button to cancel the transfer. If the user pressed the confirmation button, then the state will be in “Transfer Successfully”. On the contrary, the transaction check being cancel, the state will back to “Setting Up Transfer Information”.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Event  State Substate | | Press Confirm Button | Press Cancel Button | Enter Transfer Info. | Press Transfer Button |
| No Transfer Yet (Init) | | NA | NA | NA | Transfer Procedure |
| Transfer Procedure | No Transfer Information Filled (Init) | NA | No Transfer Yet | Setting up transfer information | NA |
| Setting Up Transfer Information | Checking transfer | No Transfer Information Filled | NA |
| Checking Transfer | Transfer Successfully / send email | Setting Up Transfer Information | NA |
| Transfer Successfully | No Transfer Yet | NA | NA |

**Structure:**



Class: TransferControl

|  |
| --- |
| **package com.example.elepay.State;  public class TransferControl {  private TransferState state;  private static TransferControl *transferControl* = new TransferControl();   public void setState(TransferState state){  this.state = state;  }   public void pressConfirmButton(){  state.pressConfirmButton(this);  }   public void pressCancelButton(){  state.pressCancelButton(this);  }   public void enterTransferInfo(){  state.enterTransferInfo(this);  }   public void pressTransferButton(){  state.pressTransferButton(this);  }   public static TransferControl getTransferControl(){  return *transferControl*;  } }** |

Abstract Class: TransferState

|  |
| --- |
| **package com.example.elepay.State;  abstract class TransferState {   public void pressConfirmButton(TransferControl transferControl){}   public void pressCancelButton(TransferControl transferControl){}   public void enterTransferInfo(TransferControl transferControl){}   public void pressTransferButton(TransferControl transferControl){} }** |

Class: NoTransferYet

|  |
| --- |
| **package com.example.elepay.State;  import android.util.Log;  public class NoTransferYet extends TransferState {   public NoTransferYet(){  Log.*e*("test", "NoTransferYet");  }   @Override  public void pressTransferButton(TransferControl transferControl){  transferControl.setState(new NoTransferInformationFilled());  } }** |

Abstract Class: TransferProcedure

|  |
| --- |
| **package com.example.elepay.State;  abstract class TransferProcedure extends TransferState{   @Override  public void pressTransferButton(TransferControl transferControl){  transferControl.setState(new NoTransferYet());  } }** |

Class: NoTransferInformationFilled

|  |
| --- |
| **package com.example.elepay.State;  public class NoTransferInformationFilled extends TransferProcedure {   @Override  public void pressCancelButton(TransferControl transferControl){  transferControl.setState(new NoTransferYet());  }   @Override  public void enterTransferInfo(TransferControl transferControl){  transferControl.setState(new SettingUpTransferInformation());  } }** |

Class: SettingUpTransferInformation

|  |
| --- |
| **package com.example.elepay.State;  public class SettingUpTransferInformation extends TransferProcedure {   @Override  public void pressConfirmButton(TransferControl transferControl){  transferControl.setState(new CheckingTransfer());  }   public void pressCancelButton(TransferControl transferControl){  transferControl.setState(new NoTransferInformationFilled());  } }** |

Class: CheckingTransfer

|  |
| --- |
| **package com.example.elepay.State;  public class CheckingTransfer extends TransferProcedure {   @Override  public void pressConfirmButton(TransferControl transferControl){  transferControl.setState(new TransferSuccessfully());  //send mail: AOTP  }   public void pressCancelButton(TransferControl transferControl){  transferControl.setState(new SettingUpTransferInformation());  } }** |

Class: TransferSuccessfully

|  |
| --- |
| **package com.example.elepay.State;  public class TransferSuccessfully extends TransferProcedure {   @Override  public void pressConfirmButton(TransferControl transferControl){  transferControl.setState(new NoTransferYet());  } }** |

* **Template**

Intent:

We have two transaction method: Credit in ElePayAccount or Credit Card, and they have the same verification process: First, the “isEmpty()” method will check whether the value is empty. Second, the “isValid()” will check the length is valid or not. Third, the “isDoHook()” method determine if the hook method “useCoupon()” will be executed. Last step is to verify the account by “isVerificate()” method.

By default, the hook method “useCoupon()” is set to empty, when the transaction is paid by ElePayAccount, the “isDoHook()” method is inherit from abstract class “Verificate”, no override action. However, if the transaction is paid by Credit Card, then the “isDoHook()” will be override, this result in executing the “hookMethod(useCoupon())”.

Abstract Class: Verificate

|  |
| --- |
| **package com.example.elepay.Template;  import android.util.Log;  public abstract class Verificate {  private String str;   public final Boolean CheckInfo(){ //Template Method  Log.*e*("CheckInfo",isEmpty(str).toString()+"|"+isValid(str).toString()+"|"+isVerificate(str).toString());  isEmpty(str); //是否為空值?  isValid(str); // 是否為有效長度?  if (isDoHook()){ //判斷是否執行Hook Method(使用優惠券)  useCoupon(); // 使用優惠券  }  isVerificate(str); //是否為有效帳戶   if (isEmpty(str) && isValid(str) && isVerificate(str)){  return true; //交易完成  }else{  return false; //交易未達成  }  }   public void setStr(String str){  this.str = str;  }   abstract Boolean isEmpty(String str);  abstract Boolean isValid(String str);  abstract Boolean isVerificate(String str);  public void useCoupon(){   }  public Boolean isDoHook(){  return false;  } }** |

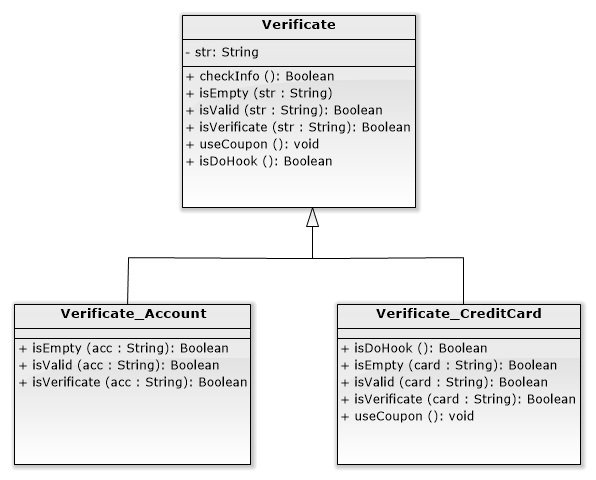
Class: Verificate\_CreditCard

|  |
| --- |
| **package com.example.elepay.Template;  import android.util.Log;  public class Verificate\_CreditCard extends Verificate{   public Verificate\_CreditCard(String str){  String temp = str.replace(" ","");  super.setStr(temp);  }   //CreditCard使用coupon  @Override  public Boolean isDoHook(){  return true;  }   public Boolean isEmpty(String card){ //是否為空值  if (!card.equals("")){  return true;  }else{  return false;  }  }  public Boolean isValid(String card){ //是否為有效長度  if (card.length() == 16){  return true;  }else{  return false;  }  }  public Boolean isVerificate(String card){ //是否為有效卡  String[] temp1 = new String[17];  int[] temp2 = new int[17];  int sum = 0;  if (isValid(card)){  for(int i = 1 ; i<17;i++){  temp1[i] = card.substring(i-1,i);  }  for(int i = 1 ; i<16;i++){  if (i%2 == 0){  temp2[i] = (Integer.*valueOf*(temp1[i]) \* 1);  }else{  temp2[i] = (Integer.*valueOf*(temp1[i]) \* 2);  if (temp2[i] >9){  temp2[i] = temp2[i]/10 + temp2[i]%10;  }  }  }  for (int i = 1 ; i<16;i++){  sum = sum + temp2[i];  }  sum = sum % 10;  if (Integer.*valueOf*(temp1[16]).equals(10-sum)){  return true;  }else{  return false;  }  }  return false;  }   @Override  public void useCoupon(){  Log.*e*("e", "useCoupon");  } }** |

Class: Verificate\_Account

|  |
| --- |
| **package com.example.elepay.Template;  public class Verificate\_Account extends Verificate{  public Verificate\_Account(String str){  super.setStr(str);  }  public Boolean isEmpty(String acc){  if (!acc.equals("")){  return true;  }else{  return false;  }  }  public Boolean isValid(String acc){  if (acc.length() == 16){  return true;  }else{  return true;  }  }  public Boolean isVerificate(String acc){  if (30000>10000){  return true;  }else{  return false;  }  } }** |

**Structure：**



* **Singleton**

Intent:

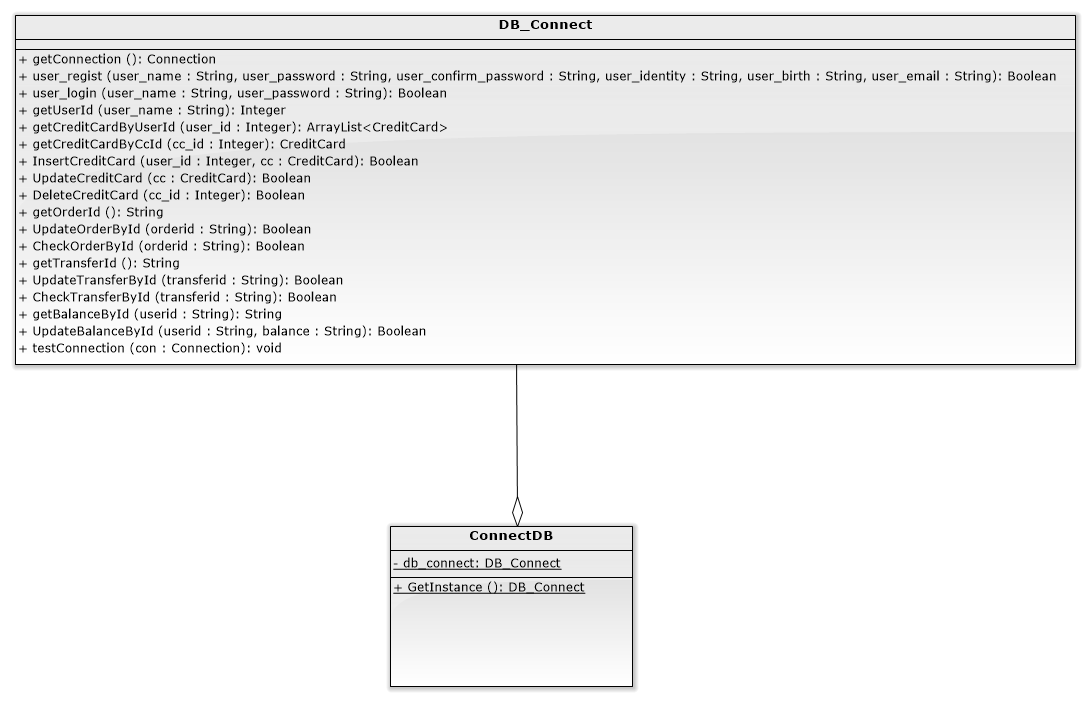
The DB connection need only one instance, Singleton Pattern can make the DB object as a unique access point when connecting to database, this can reduce other unnecessary objects also improve the stability of the connection to DB.

“DB\_Connect” is a bundle of method that actually connect to SQL. The “connectDB” class is the present of the Singleton, it limit the “DB\_Connect”, so the DB\_Connect will have only one instance.

Class: ConnectDB

|  |
| --- |
| **package com.example.elepay.Singleton;  public class ConnectDB {  private static DB\_Connect *db\_connect*;  private ConnectDB(){   }   public static DB\_Connect GetInstance(){  if (*db\_connect* == null){  synchronized(ConnectDB.class){  if (*db\_connect* == null){  *db\_connect* = new DB\_Connect();  }  }  }  return *db\_connect*;  } }** |

**Structure：**



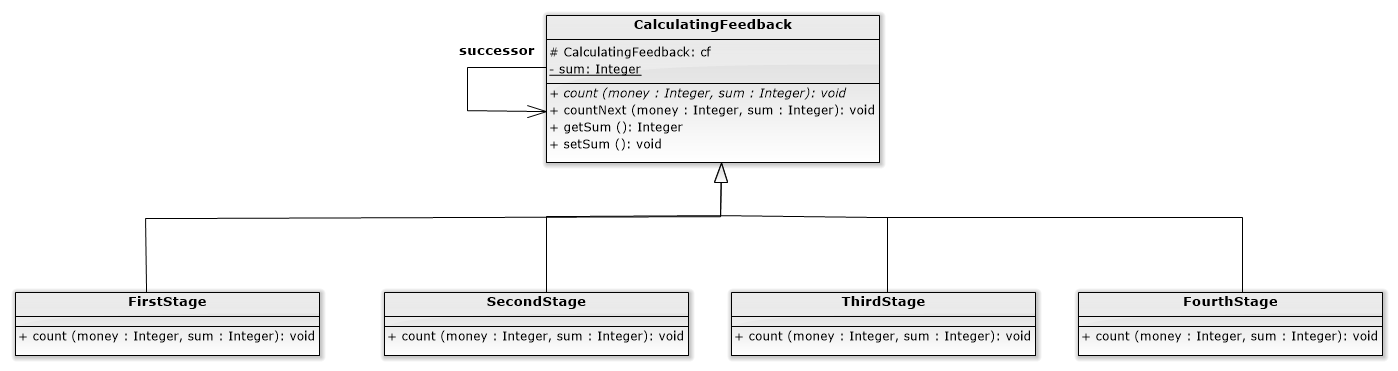
* **Chain of Responsibility**

Intent:

The Chain of Responsibility can help give the amount of credits to customer according to their bill. The credit is calculated in the table below.

|  |  |
| --- | --- |
| Consumption | Credits |
| $0 ~ $7,500 | 5% |
| $7,501 ~ $15,000 | 10% |
| $15,001 ~ $22,500 | 15% |
| $22,501 ~ $30,000 | 20% |

**Structure:**



Abstract Class: CalculatingFeedback

|  |
| --- |
| **package com.example.elepay.ChainOfResponsibility;  import android.util.Log;  public abstract class CalculatingFeedback {  protected CalculatingFeedback cf;  static int *sum* ;   public CalculatingFeedback(CalculatingFeedback cf){  this.cf = cf;  }   public abstract void count(int money, int sum);   public void countNext(int money, int sum){  cf.count(money, sum);  }   public int getSum() {  return this.*sum*;  }   public void setSum(int sum) {  this.*sum* = sum;  Log.*e*("SetSum", String.*valueOf*(sum));  } }** |

Class: FirstStage

|  |
| --- |
| **package com.example.elepay.ChainOfResponsibility;  import android.util.Log;  public class FirstStage extends CalculatingFeedback{   public FirstStage(CalculatingFeedback cf){  super(cf);  }   @Override  public void count(int money, int sum) {   int num = 0;  num = (int) (money \* 0.05);  sum += money \* 0.05; //回饋5%  super.setSum(sum);  Log.*e*("sum\_one", String.*valueOf*(sum));   } }** |

Class: SecondStage

|  |
| --- |
| **package com.example.elepay.ChainOfResponsibility;  import android.util.Log;  public class SecondStage extends CalculatingFeedback{   public SecondStage(CalculatingFeedback cf){  super(cf);  }   @Override  public void count(int money, int sum) {  int num = 0;  if(money > 7500){  num = money - 7500;  sum += num \* 0.1; //回饋10%  countNext(7500, sum);  }else{  countNext(money, sum);  }  Log.*e*("sum\_second", String.*valueOf*(sum));  } }** |

Class: ThirdStage

|  |
| --- |
| **package com.example.elepay.ChainOfResponsibility;  import android.util.Log;  public class ThirdStage extends CalculatingFeedback{   public ThirdStage(CalculatingFeedback cf){  super(cf);  }   @Override  public void count(int money, int sum) {   int num = 0;  if(money > 15000){  num = money - 15000;  sum += num \* 0.15; //回饋15%  countNext(15000, sum);  }else{  countNext(money, sum);  }  Log.*e*("sum\_three", String.*valueOf*(sum));  } }** |

Class: FourthStage

|  |
| --- |
| **package com.example.elepay.ChainOfResponsibility;  import android.util.Log;  public class** **FourthStage extends CalculatingFeedback{  public FourthStage(CalculatingFeedback cf){  super(cf);  }   @Override  public void count(int money, int sum) {   int num = 0;  if(money > 22500){  num = money - 22500;  sum += num \* 0.2; //回饋20%  countNext(22500, sum);  }else{  countNext(money, sum);  }  Log.*e*("sum\_four", String.*valueOf*(sum));  } }** |

Class: Client

|  |
| --- |
| **public void aotp(){  try{  //取得訂單編號以及驗證碼  String[] result = db.getTransferId().split(",");  String random = result[1];  glo\_transferid = result[0];  final TextView tvCheckInfo = choosePayCheckView.findViewById(R.id.*tvCheckInfo*);  tvCheckInfo.setText("Certificate:"+random+"\nEmail:wuyou0815@gmail.com");  final TextView tvTime = choosePayCheckView.findViewById(R.id.*tvTime*);  countDownTimer = new CountDownTimer(40000,1000){   @Override  public void onFinish() {  tvTime.setText("Timeout!");  Toast.*makeText*(ConfirmTransInfoActivity.this, "Time out!", Toast.*LENGTH\_LONG*).show();  dialog2.dismiss();  finish();  }  @Override  public void onTick(long millisUntilFinished) {  tvTime.setText(""+millisUntilFinished/1000);  if(checkCancel){  countDownTimer.cancel();  }  }   }.start();   AlertDialog.Builder builder1 = new AlertDialog.Builder(ConfirmTransInfoActivity.this)  .setCancelable(false)  .setIcon(R.drawable.*confirm*)  .setTitle("Check Payment")  .setView(choosePayCheckView)  .setPositiveButton("Check", new DialogInterface.OnClickListener(){  @Override  public void onClick(DialogInterface dialog, int which) {  checkCancel = true; //確認是否關閉   DB\_Connect db = new DB\_Connect();  //取得驗證狀態  if(db.CheckTransferById(glo\_transferid)){  Log.*e*("Show","Success"); // Toast.makeText(ConfirmTransInfoActivity.this, "Pay success", Toast.LENGTH\_LONG).show();   //state Pattern  transferControl.pressConfirmButton();  Toast.*makeText*(ConfirmTransInfoActivity.this, "State: TransferSuccessfully", Toast.*LENGTH\_LONG*).show(); // TransferActivity ta = new TransferActivity(); // ta.finishTransfer();  Log.*e*("music", tvMoney.getText().toString());  CalculatingFeedback cf = new FourthStage(new ThirdStage(new SecondStage(new FirstStage(null))));  String temp[] = tvMoney.getText().toString().split("：");  Integer money = Integer.*parseInt*(temp[1]);  cf.count(money, 0); //回饋金額是2750元  Integer lastBalance = Integer.*valueOf*(db.getBalanceById("2"));  db.UpdateBalanceById("2",String.*valueOf*(lastBalance - money + cf.getSum()));  Toast.*makeText*(ConfirmTransInfoActivity.this, "這次一共獲得" + cf.getSum() + "元的回饋", Toast.*LENGTH\_SHORT*).show();  Log.*e*("feedback", String.*valueOf*(cf.getSum()));   display(); //跳轉display transferInfo的資訊  finish();  }else{  Log.*e*("Show","Failed");  Toast.*makeText*(ConfirmTransInfoActivity.this, "pay failure", Toast.*LENGTH\_LONG*).show();  finish();  }   }  });  dialog2 = builder1.show();  dialog1.dismiss();  }catch (Exception e){   }  }** |

* **Memento**

Intent:

Memeto let the user Undo or Redo when they’re adding, modifying or delete the Credit Card.

Class: CareTaker

|  |
| --- |
| **package com.example.elepay.Memento;  import java.util.ArrayList;  public class CareTaker {  private static final int *MAX* = 3;  public ArrayList<CreditCardMemento> memList = new ArrayList<>();   public CreditCardMemento getMemento(int index){  if (index > memList.size() -1 ){  index = memList.size() -1 ;  }  return memList.get(index);  }  public void addMemento(CreditCardMemento m){  if (memList.size() >= *MAX*){  memList.remove(0);  }  memList.add(m);  }  public int getCount(){  return memList.size();  }   }** |

Class: CreditCardMemento

|  |
| --- |
| **package com.example.elepay.Memento;  import com.example.elepay.Iterator.CreditCard;  import java.util.ArrayList;  public class CreditCardMemento {  private ArrayList<CreditCard> cclist;  public CreditCardMemento(ArrayList<CreditCard> cclist){  this.cclist = cclist;  }   public ArrayList<CreditCard> getCclist() {  return cclist;  }   public void setCclist(ArrayList<CreditCard> cclist) {  this.cclist = cclist;  } }** |

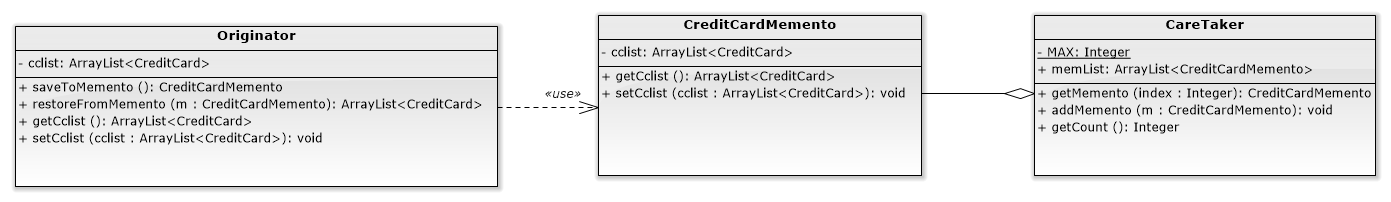
Class: Originator

|  |
| --- |
| **package com.example.elepay.Memento;  import com.example.elepay.Iterator.CreditCard;  import java.util.ArrayList;  public class Originator {  private ArrayList<CreditCard> cclist;   public CreditCardMemento saveToMemento(){  return new CreditCardMemento(cclist);  }   public ArrayList<CreditCard> restoreFromMemento(CreditCardMemento m){  this.cclist = m.getCclist();  return cclist;  }   public ArrayList<CreditCard> getCclist() {  return cclist;  }   public void setCclist(ArrayList<CreditCard> cclist) {  this.cclist = cclist;  } }** |

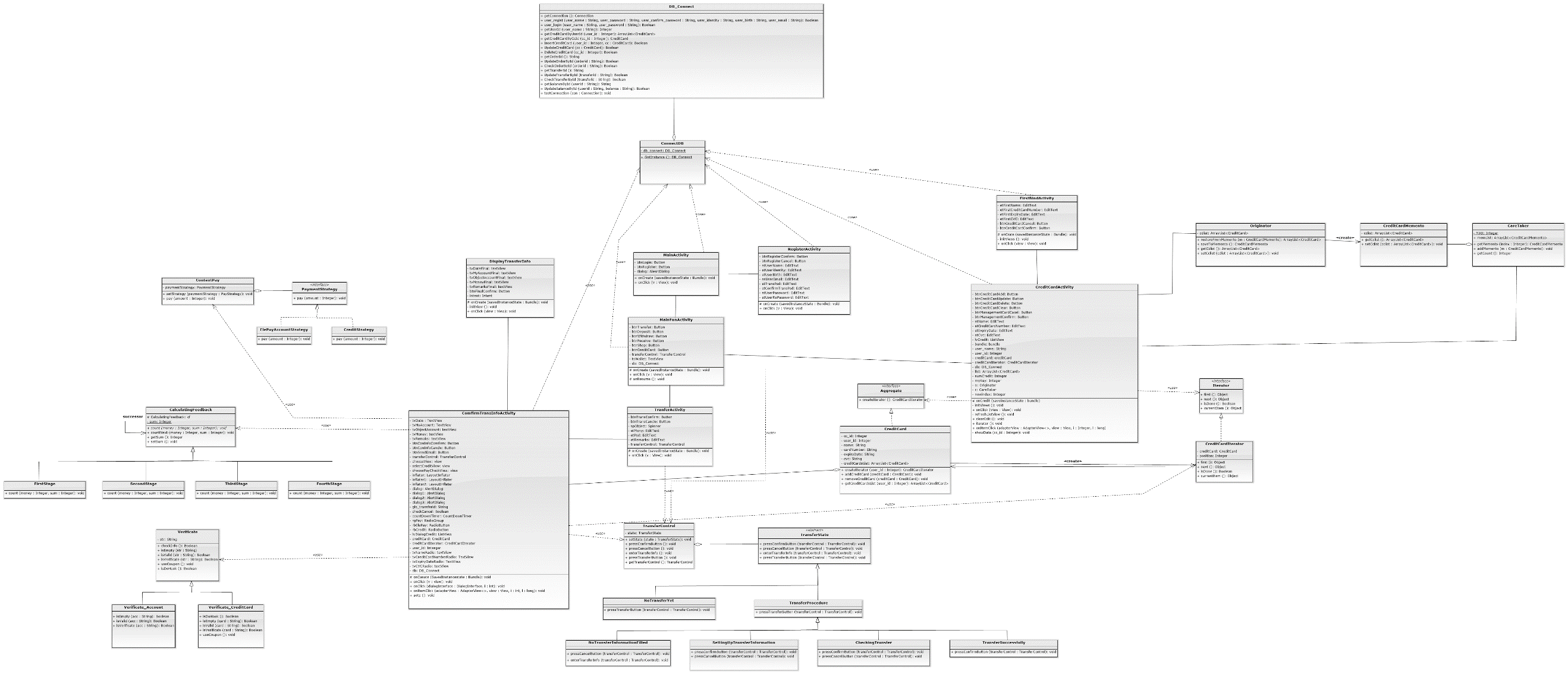
Client

|  |
| --- |
| **case R.id.*btnCreditCardAdd*:  CreditCard creditCard1 = new CreditCard(0,user\_id, etName.getText().toString(), etCreditCardNumber.getText().toString(), etExpiryDate.getText().toString(), etCVC.getText().toString());  creditCard.addCreditCard(creditCard1);  nowindex = -1;  //連資料庫  //insert(creditCard)  try{  o.setCclist(creditCard.getCreditCardsList(user\_id));  c.addMemento(o.saveToMemento());  db.InsertCreditCard(user\_id,creditCard1);  }catch (Exception e){  Log.*e*("Error",Log.*getStackTraceString*(e));  }  refreshListView();  break;** |

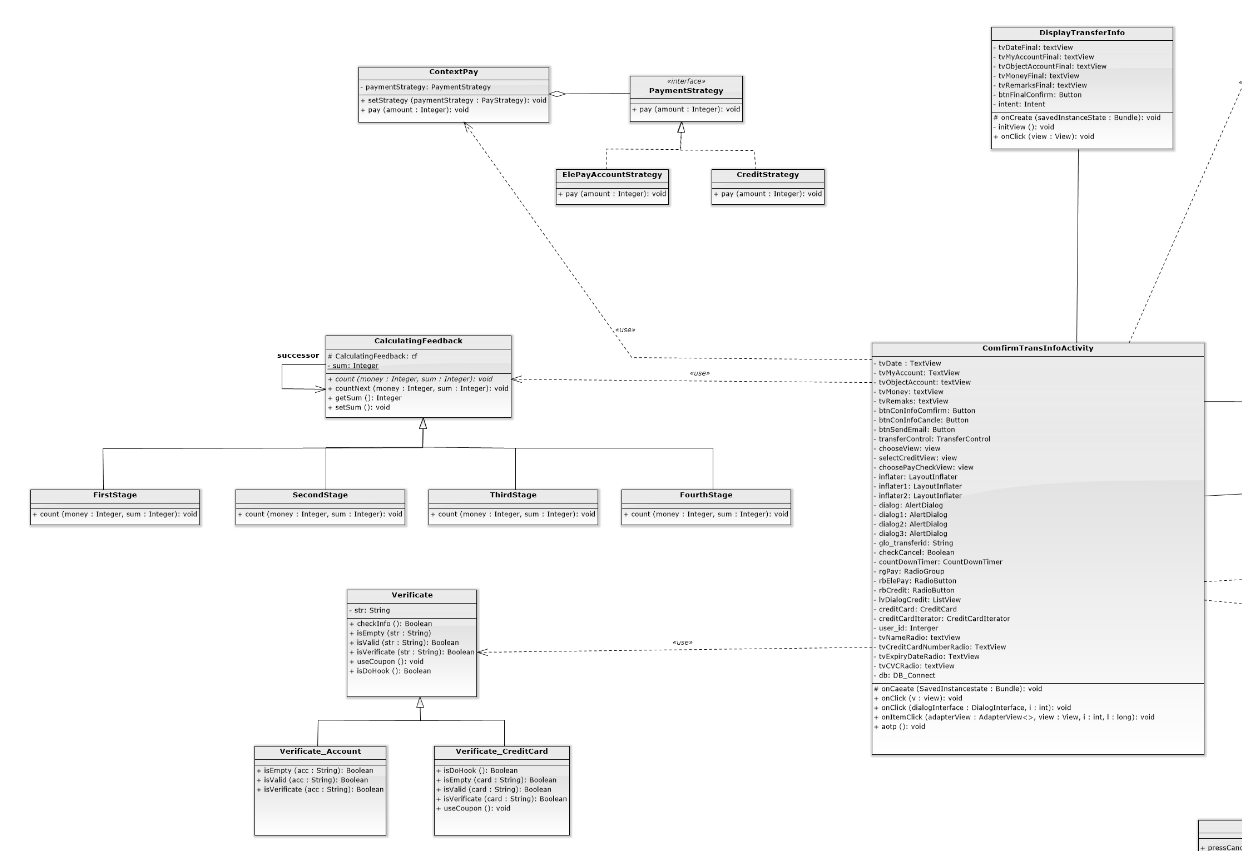
**Structure：**



**Big class diagram**

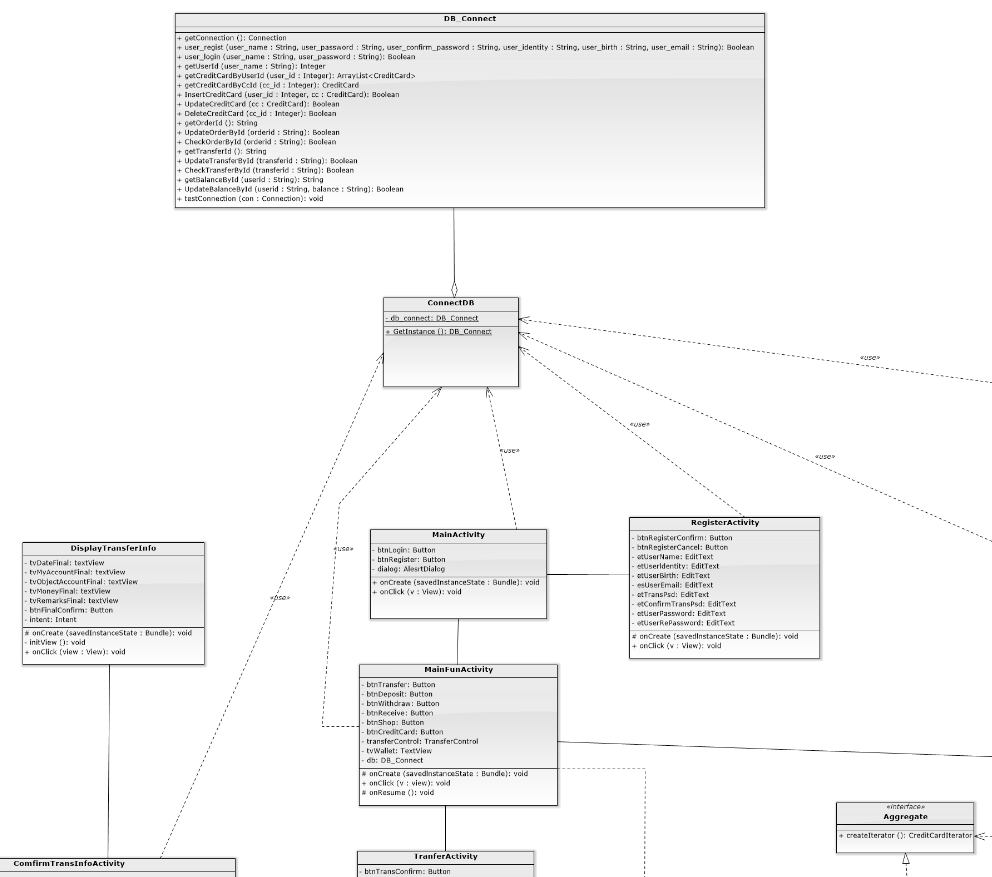


**Highlight/mark the new applied patterns in the class diagram.**

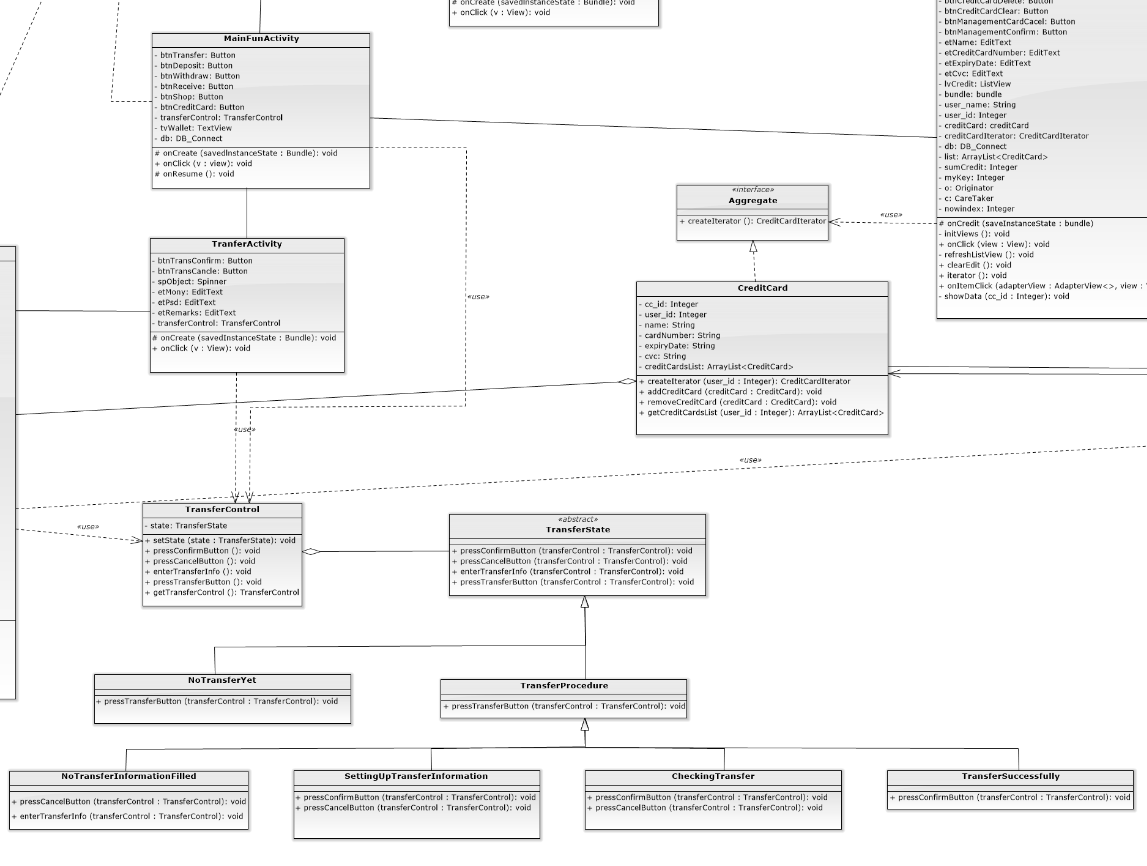


**Template Pattern**

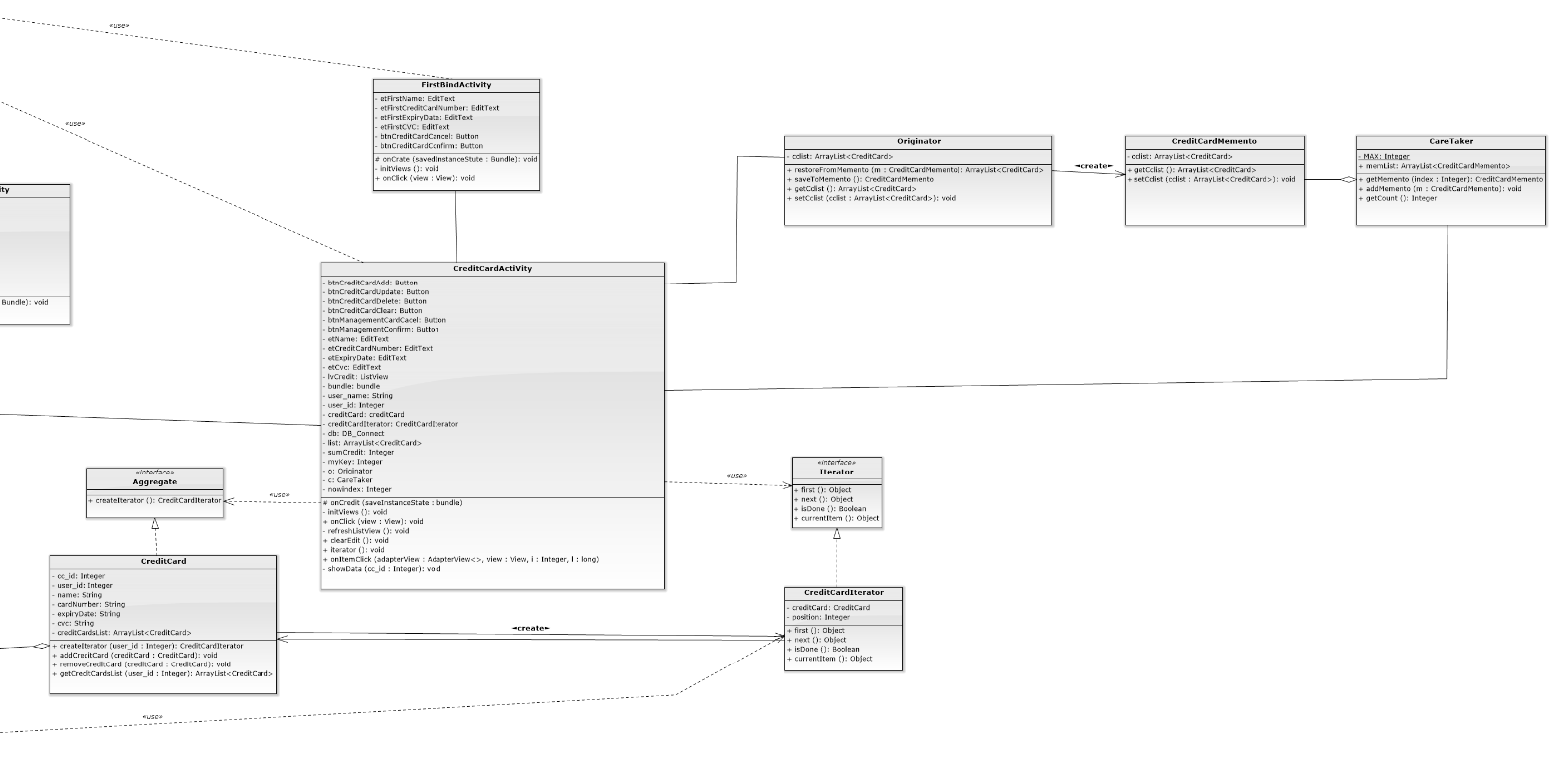
**Chain Of Responsibility**



**Singleton Pattern**



**State Pattern**



**Iterator Pattern**

**Factory Method**

**Memento Pattern**

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

|  |  |
| --- | --- |
|  |  |
| Figure 1. state: NoTransferYet | Figure2.state:NoTransferInformationEntered |
|  |  |
| Figure 3. enter transfer info. | Figure 4. state: SettingUpTransferInfo. |

|  |  |
| --- | --- |
|  |  |
| Figure 5. state: CheckingTransfer | Figure 6. choose ElePayAccount |
|  |  |
| Figure 7. choose deposit or not | Figure 8. AOTP verification |

|  |  |
| --- | --- |
|  |  |
| Figure 9. send email | Figure 10. state: TransferSuccessfully |
|  |  |
| Figure 11. display feedback | Figure 12. display transfer detail |

|  |  |
| --- | --- |
|  |  |
| Figure 13. choose CreditCard | Figure 14. choose CreditCard |
|  |  |
| Figure 15. CreditCard | Figure 16. select item |

|  |  |
| --- | --- |
|  |  |
| Figure 17. modify CreditCard | Figure 18. modify success |
|  |  |
| Figure 19. modify cvc: 727 | Figure 20. modify cvc: 730 |

|  |  |
| --- | --- |
|  |  |
| Figure 21. memento undo | Figure 22. memento undo |
|  |  |
| Figure 23. memento redo | Figure 24. memento redo |

|  |  |
| --- | --- |
|  |  |
| Figure 25. iterator display number | Figure 26. receive email (AOTP) |

1. You need to evaluate the design quality of the new design by using object-oriented 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(Weighted Methods per Class)

Formula for WMC:

WMC(C) = Cm1 + Cm2 + ··· + Cmn (Cmi=complexity metrics of methods of C)

Significance: Time and effort required to understand, test, and maintain class C increases exponentially with WMC.

WMC measures the complexity of a class. Complexity of a class can for example be calculated by the cyclomatic complexities of its methods. High value of WMC indicates the class is more complex than that of low values.

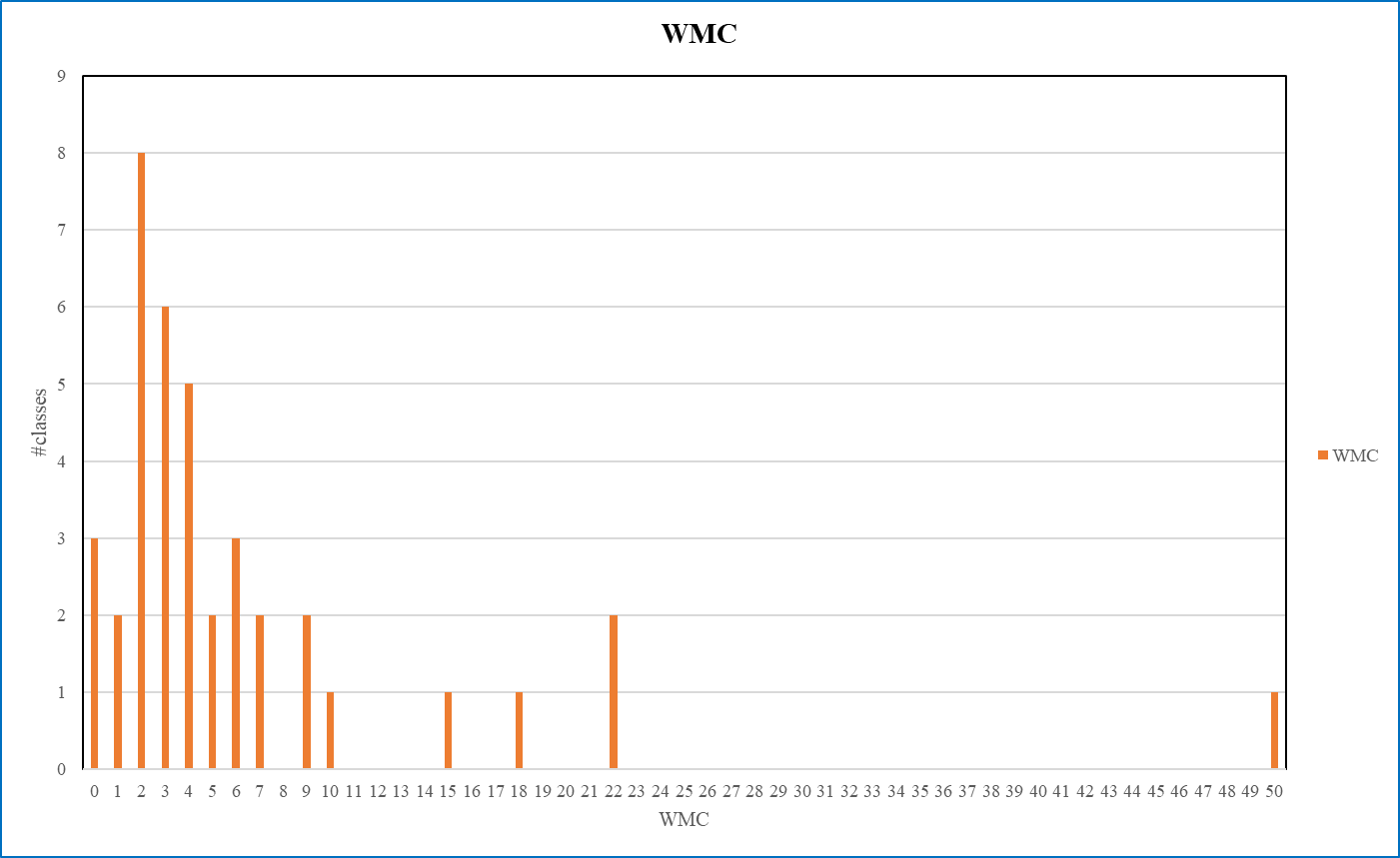


Figure27. Weighted Methods per Class

The 80% of classes in our system with WMC lower than 10, it represents the complexity of our system is quite low. The highest WMC in our system is 50, it’s a great representative for a well-function system.

On the WMC perspective, the development of our system can be controlled easily. Last but not least, we want to lower the relationship in the class with WMC over 10 in the future development.

* DIT (Depth of Inheritance)

DIT: Distance from a derived class to the root class in the inheritance hierarchy.

Figure28. Depth of Inheritance

We can see that most of the DIT is not so high, so the behavior in each class is less affected, because the parent class of a class might redefine some methods used to implement this class. If the DIT is high, that might indicates the class inherits more and use these methods, it will result in difficulty of class behavior prediction

As the chart above, although most of the DIT are 0 here. Therefore, our system performs well on DIT.

* NOC (Number of Children)

Formula for NOC:

NOC(C) = | { C’ : C’ is an immediate child of C }|

The dependencies of child classes on class C increases proportionally with NOC.

Increase in dependencies increases the change impact, and behavior impact of C on its child classes.

These make the program more difficult to understand, test, and maintain.

Figure29. Number of Children

As the chart above, the values of NOC in our system most are 0, that means the subclasses are affected less by super classes in our system. Therefore, our system performs well on NOC.

* CBO (Coupling between Object Classes)

Formula for CBO:

CBO(C) = |{ C’ : C depends on C’ }|

The higher the CBO for class C the more difficult to understand, test, maintain, and reuse class C.

Figure30. Coupling between Object Classes

We can see the most of the CBO is not high here, so it is easier to understand, test, maintain and reuse class. However, the larger the CBO, the harder it is to test.

As the chart above, the CBO are mostly 1, and there are only a few values higher than 3. Therefore, our system performs not bad on CBO.

* RFC (Response for a Class)

Formula for RFC:

RFC (C) = |{ m : m is a method of C, or m is called by a method of C }|

The higher the RFC, the more difficult to understand, test, maintain, and reuse the class due to higher dependencies of the class on other classes.

RFC: the number of methods of a class plus the number of methods called by a method of the class.

Figure31. Response for a Class

Most of our class with the RFC lower than 17, because they have less complex relationship. If the RFC is high, then it will take more time to understand and setup the test cases. As a result, our class not only easy to understand, test and maintain but the class reusing is suitable.

On the RFC perspective, our program is a good design.

* LCOM (Lack of Cohesion Methods)

Formula for LCOM:

LCOM (C) = n \* (n-1) / 2 - 2 \* |{ (mi, mj) : mi and mj share an attribute of C }|

LCOM measures the number of pairs of methods of C that do not share a common attribute.

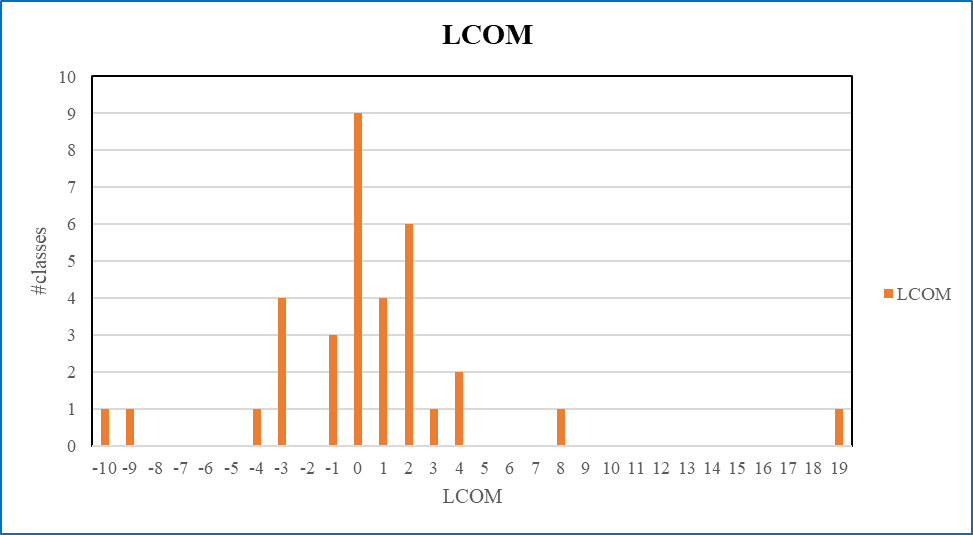


Figure32. Lack of Cohesion Methods

LCOM is the lower the better. As the chart above, LCOM most are between -3 to 2, that means it is high cohesion. Therefore, this is a good design. High LCOM means the cohesion of the class is low, we want the cohesion of the class be higher.

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

**Junit test**

|  |
| --- |
| **package com.example.elepay;  import com.example.elepay.Template.Verificate\_Account; import com.example.elepay.Template.Verificate\_CreditCard;  import org.junit.Assert; import org.junit.Test;  */\*\*  \* Example local unit test, which will execute on the development machine (host).  \*  \* @see <a href="http://d.android.com/tools/testing">Testing documentation</a>  \*/* public class JunitTest {  @Test  public void isEmptyAccount(){  Verificate\_Account va = new Verificate\_Account("natalia");  Assert.*assertEquals*(true,va.isEmpty("natalia"));  }  @Test  public void isEmptyCreditCard(){  Verificate\_CreditCard vc = new Verificate\_CreditCard("5195219206071342");  Assert.*assertEquals*(true,vc.isEmpty("5195219206071342"));  }  @Test  public void isValidAccount(){  Verificate\_Account va = new Verificate\_Account("natalia");  Assert.*assertEquals*(true,va.isValid("natalia"));  }  @Test  public void isValidCreditCard(){  Verificate\_CreditCard vc = new Verificate\_CreditCard("5195219206071342");  Assert.*assertEquals*(true,vc.isValid("5195219206071342"));  }  @Test  public void isVerificateAccount(){  Verificate\_Account va = new Verificate\_Account("natalia");  Assert.*assertEquals*(true,va.isVerificate("natalia"));  }  @Test  public void isVerificateCreditCard(){  Verificate\_CreditCard vc = new Verificate\_CreditCard("5195219206071342");  Assert.*assertEquals*(true,vc.isVerificate("5195219206071342"));  }  @Test  public void VerificateAccount(){  Verificate\_Account va = new Verificate\_Account("natalia");  Assert.*assertEquals*(true,va.checkInfo());  }  @Test  public void VerificateCreditCard(){  Verificate\_CreditCard vc = new Verificate\_CreditCard("5195219206071342");  Assert.*assertEquals*(true,vc.checkInfo());  }  }** |

|  |
| --- |
|  |
| Figure33. Junit test result |

**Junit test suite**

|  |
| --- |
| **package com.example.elepay;  import com.example.elepay.Template.Verificate\_Account; import com.example.elepay.Template.Verificate\_CreditCard;  import junit.framework.TestCase; import junit.framework.TestSuite;  import org.junit.Assert; import org.junit.Test;  public class JunitTestSuite extends TestCase {   public JunitTestSuite(){};   public JunitTestSuite(String name){  super(name);  }   @Test  public void isEmptyAccount(){  Verificate\_Account va = new Verificate\_Account("natalia");  Assert.*assertEquals*(true,va.isEmpty("natalia"));  }  @Test  public void isEmptyCreditCard(){  Verificate\_CreditCard vc = new Verificate\_CreditCard("5195219206071342");  Assert.*assertEquals*(true,vc.isEmpty("5195219206071342"));  }  @Test  public void isValidAccount(){  Verificate\_Account va = new Verificate\_Account("natalia");  Assert.*assertEquals*(true,va.isValid("natalia"));  }  @Test  public void isValidCreditCard(){  Verificate\_CreditCard vc = new Verificate\_CreditCard("5195219206071342");  Assert.*assertEquals*(true,vc.isValid("5195219206071342"));  }  @Test  public void isVerificateAccount(){  Verificate\_Account va = new Verificate\_Account("natalia");  Assert.*assertEquals*(true,va.isVerificate("natalia"));  }  @Test  public void isVerificateCreditCard(){  Verificate\_CreditCard vc = new Verificate\_CreditCard("5195219206071342");  Assert.*assertEquals*(true,vc.isVerificate("5195219206071342"));  }  @Test  public void VerificateAccount(){  Verificate\_Account va = new Verificate\_Account("natalia");  Assert.*assertEquals*(true,va.checkInfo());  }  @Test  public void VerificateCreditCard(){  Verificate\_CreditCard vc = new Verificate\_CreditCard("5195219206071342");  Assert.*assertEquals*(true,vc.checkInfo());  }   public static TestSuite suite(){  TestSuite suite = new TestSuite();  suite.addTest(new JunitTestSuite("isEmptyAccount"));  suite.addTest(new JunitTestSuite("isEmptyCreditCard"));  suite.addTest(new JunitTestSuite("isValidAccount"));  suite.addTest(new JunitTestSuite("isValidCreditCard"));  suite.addTest(new JunitTestSuite("isVerificateAccount"));  suite.addTest(new JunitTestSuite("isVerificateCreditCard"));  suite.addTest(new JunitTestSuite("VerificateAccount"));  suite.addTest(new JunitTestSuite("VerificateCreditCard"));  return suite;  } }** |

|  |
| --- |
|  |
| Figure34. Junit test suite result |

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

**Black box**

* Equivalence Partitioning

Equivalence partitioning divides the input and output domains into a number of disjoint subsets, and selects one test case from each of these disjoint subsets.

* In ElePay, users have an account, when they transfer money, the transfer amount cannot over 30,000. According to our transfer amount regulation, we set the Equivalence Partitioning Testing as below:

1. Partition 1(the transfer amount equal and less than 30,000)

First subset consists of the transfer amount equal and less than 30,000. The test cases in this partition will be accepted.

1. Partition 2(the transfer amount greater than 30,000)

Second subset consists of the transfer amount greater than 30,000. The test cases in this partition will be rejected.

* Boundary Value Analysis

The boundary value analysis selects test cases at near the boundaries of the equivalence classes.

Suppose that “transfer amount” is “TM”.

* For Partition1(the transfer amount equal and less than 30,000), the test cases are

{TM = 29999, TM= 30000, TM = 30001}

**White Box**

* Basis Path Testing

Basis path testing generates test cases to exercise to exercise the independent control flow paths, called basis paths, of the CUT.

A basis path is a path from the B node to the E node and exercises a directed cycle at most once.

|  |
| --- |
| **@Override public void onClick(View view) { *1* switch (view.getId()){  case R.id.*btnShopPayment*: *2* String strDC = ""; *3* String strGC = ""; *4* if(cbDC.isChecked()){ *5* ComputerBuilder computerBuilder = new DocumentComputerBuilder(); *6* ComputerDirector director = new ComputerDirector(computerBuilder); *7* director.build(); *8* Computer DC = computerBuilder.getComputer(); *9* strDC = DC.display(); *10* } *11* else(cbGC.isChecked()){ *12* ComputerBuilder computerBuilder = new GamingComputerBuilder(); *13* ComputerDirector director = new ComputerDirector(computerBuilder);  *14* director.build(); *15* Computer GC = computerBuilder.getComputer(); *16* strGC = GC.display(); *17* } *18* Intent intent = new Intent(this, PayActivity.class); *19* intent.putExtra("total", sum);  *20* intent.putExtra("strDC", strDC); *21* intent.putExtra("strGC", strGC); *22* startActivity(intent); *23* break;  case R.id.*btnShopCancel*: *24* finish(); *25* break;  default: *26* Toast.*makeText*(this, "default", Toast.*LENGTH\_SHORT*).show(); *27* }** |

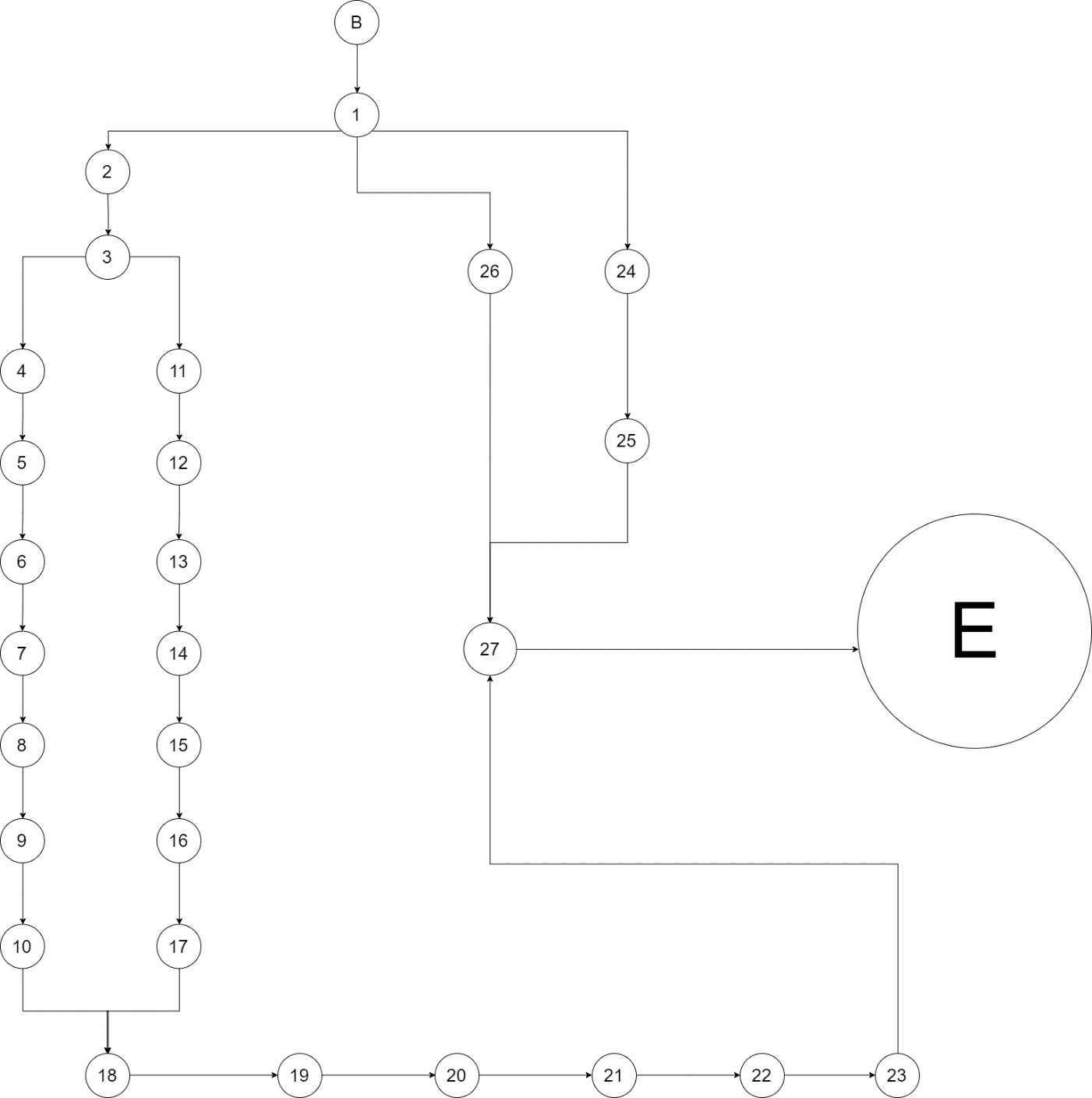


Figure35. Flow graph of the method onClick()

This definition produces four basis paths for the onClick() method using its flow graph:

1. B→1→2→3→4→5→6→7→8→9→10→18→19→20→21→22→23→27→E
2. B→1→2→3→11→12→13→14→15→16→17→18→19→20→21→22→23→27→E
3. B→1→2→24→25→27→E
4. B→1→2→26→27→E

* Cyclomatic Complexity

The number of basis paths of the CUT is defined as the cyclomatic complexity of the CUT.

It is determined in three equivalent way.

1. Number of closed regions plus one.

This approach obtains the cyclomatic complexity by adding one to the number of closed regions in the flow graph.

According to the flow graph above have three closed regions, so the cyclomatic complexity is 3 + 1 = 4.

1. Number of nodes and edges.

In this approach, the cyclomatic complexity is the number of edges minus the number of nodes plus 2.

According to the flow graph have eighteen edges and sixteen nodes, so the cyclomatic complexity is 31 – 29 + 2 = 4.

1. Number of atomic binary conditions plus one.

The cyclomatic complexity is the number of atomic binary conditions plus 1.

According to the flow graph have three atomic binary conditions(1 and 3 and 12), so the cyclomatic complexity is 3 + 1 = 4.

1. Please analyze the invocation chains of the new design and compare the result with the first teamwork.

**ChainOfResponsibility.FirstStage**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| count()→CaculatingFeedback.setSum() | 1 |

**ChainOfResponsibility.SecondStage**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| count()→CaculatingFeedback.countNext() | 1 |

**ChainOfResponsibility.ThirdStage**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| count()→CaculatingFeedback.countNext() | 1 |

**ChainOfResponsibility.FourthStage**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| count()→CaculatingFeedback.countNext() | 1 |

**Memento.Originator**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| restoreFromMemento(CreditCardMemento m) → CreditCardMemento.getCclist() | 1 |

**State.TransferControl**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| pressConfirmButton()→TransferState.pressConfirmButton() | 1 |
| pressCancelButton()→TransferState.pressCancelButton() | 1 |
| enterTransferInfo()→TransferState.enterTransferInfo() | 1 |
| pressTransferButton()→TransferState.pressTransferButton() | 1 |

**State.TransferProcedure**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| pressTransferButton(TransferControl transferControl) → TransferControl.setState() | 1 |

**State.NoTransferYet**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| pressTransferButton(TransferControl transferControl) → TransferControl.setState() | 1 |

**State.CheckingTransfer**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| pressConfirmButton(TransferControl transferControl) → TransferControl.setState() | 1 |
| pressCancelButton(TransferControl transferControl) → TransferControl.setState() | 1 |

**State.SettingUpTransferInformation**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| pressConfirmButton(TransferControl transferControl) → TransferControl.setState() | 1 |
| pressCancelButton(TransferControl transferControl) → TransferControl.setState() | 1 |

**State.NoTransferInformationFilled**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| enterTransferInfoButton(TransferControl transferControl) → TransferControl.setState() | 1 |
| pressCancelButton(TransferControl transferControl) → TransferControl.setState() | 1 |

**State.TransferSuccessfully**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| pressConfirmButton(TransferControl transferControl) → TransferControl.setState() | 1 |

**Iterator.CreditCardIterator**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| first() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId() → CreditCard.setCvc() | 3 |
| first() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId() → CreditCard.setExpiryDate() | 3 |
| first() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId() → CreditCard.setCardNumber() | 3 |
| first() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId() → CreditCard.setName() | 3 |
| first() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId() → CreditCard.setCc\_id() | 3 |
| next() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→ CreditCard.setCvc() | 3 |
| next() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→ CreditCard.setExpiryDate() | 3 |
| next() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→ CreditCard.setCardNumber() | 3 |
| next() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→ CreditCard.setName() | 3 |
| next() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→ CreditCard.setCc\_id() | 3 |
| isDone() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→CreditCard.setCc\_id() | 3 |
| isDone() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→CreditCard.setName() | 3 |
| isDone() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→CreditCard.setCardNumber() | 3 |
| isDone() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→CreditCard.setExpiryDate() | 3 |
| isDone() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→CreditCard.setCvc() | 3 |
| currentItem() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→CreditCard.setCvc() | 3 |
| currentItem() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→CreditCard.setExpiryDate() | 3 |
| currentItem() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→CreditCard.setCardNumber() | 3 |
| currentItem() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→CreditCard.setName() | 3 |
| currentItem() → CreditCard.getCreditCardsList() → DB\_Connect.getCreditCardByUserId()→CreditCard.setCc\_id() | 3 |

**Iterator.CreditCard**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| getCreditCardsList()→DB\_Connect.getCreditCardByUserId()→CreditCard.setCc\_id() | 2 |
| getCreditCardsList()→DB\_Connect.getCreditCardByUserId()→CreditCard.setName() | 2 |
| getCreditCardsList()→DB\_Connect.getCreditCardByUserId()→CreditCard.setCardNumber() | 2 |
| getCreditCardsList()→DB\_Connect.getCreditCardByUserId()→CreditCard.setExpiryDate() | 2 |
| getCreditCardsList()→DB\_Connect.getCreditCardByUserId()→CreditCard.setCvc() | 2 |

**Singleton.DB\_Connect**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| getCreditCardByUserId()→CreditCard.setCvc() | 1 |
| getCreditCardByUserId()→CreditCard.setExpiryDate() | 1 |
| getCreditCardByUserId()→CreditCard.setCardNumber() | 1 |
| getCreditCardByUserId()→CreditCard.setName() | 1 |
| getCreditCardByUserId()→CreditCard.setCc\_id() | 1 |

**TransferActivity**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| onClick() → TransferControl.enterTransferInfo() → TransferState.enterTransferInfo() | 2 |
| onClick() → TransferControl.pressConfrimButton() → TransferState.pressConfrimButton() | 2 |
| onClick() → TransferControl.pressCancelButton() → TransferState.pressCancelButton() | 2 |

**RegisterActivity**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| onClick() → DB\_Connect.user\_regist() | 1 |

**FirstBindActivity**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| onClick() → DB\_Connect.getUserId() | 1 |
| onClick() → DB\_Connect.InsertCreditCard() | 1 |

**MainActivity**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| onClick() → Verification.CheckCreditCard() | 1 |
| onClick() → DB\_Connect.user\_login() | 1 |

**CreditCardActivity**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| onCreate()→DB\_Connect.getUserId() | 1 |
| onCreate()→CreditCard.createIterator() | 1 |
| onClick()→CreditCard.addCreditCard() | 1 |
| onClick()→Originator.setCclist() | 1 |
| onClick()→CareTaker.addMemento() | 1 |
| onClick()→DB\_Connect.InsertCreditCard() | 1 |
| onClick()→CareTaker.getCount() | 1 |

**MainFuncActivity**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| onCreate()→TransferControl.setState() | 1 |
| onClick()→TransferControl.pressTransferButton()→TransferState.pressTransferButton() | 2 |

**ConfirmTransInfoActivity**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| onCreate()→CreditCard.createIterator() | 1 |
| onClick()→DB\_Connect.UpdateTransferById() | 1 |
| onClick()→TransferControl.pressCancelButton()→TransferState.pressCancelButton() | 2 |
| onItemClick()→DB\_Connect.getCreditCardByUserId()→CreditCard.setCc\_id() | 2 |
| onItemClick()→DB\_Connect.getCreditCardByUserId()→CreditCard.setName() | 2 |
| onItemClick()→DB\_Connect.getCreditCardByUserId()→CreditCard.setCardNumber | 2 |
| onItemClick()→DB\_Connect.getCreditCardByUserId()→CreditCard.setExpiryDate | 2 |
| onItemClick()→DB\_Connect.getCreditCardByUserId()→CreditCard.setCvc | 2 |
| aotp()→DB\_Connect.getTransferId() | 1 |

**Strategy.ContextPay**

|  |  |
| --- | --- |
| Invocation chains | Invocation chains length |
| pay() → paymentStrategy.pay() | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| Invocation Chain Length | 1 | 2 | 3 |
| Number of chains | 41 | 15 | 20 |

Table1. invocation chains in the ElePay (teamwork2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Invocation Chain Length | 1 | 2 | 3 | 4 |
| Number of chains | 32 | 11 | 5 | 3 |

Table2. invocation chains in the ElePay (teamwork1)

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

* The number of classes:

We have 37 classes in our ElePay system.

* The number of inheritances:

There are 12 inheritance in our ElePay system, that are Verificate\_Account、Verificate\_CreditCard、FirstStage、SecondStage、ThirdStage、FourthStage、NoTransferInformationFilled、SettingUpTransferInformation、CheckingTransfer、TransferSuccessfully、NoTransferYet、TransferProcedure.

* The number of aggregations:

There are 3 aggregations in our ElePay system, that are ConnectDB、TransferControl、CareTaker.

* The number of associations:

There are 28 associations in our ElePay System, that are Verificate and ComfirmTransInfoActivity、CalculatingFeedback and ComfirmTransInfoActivity、ContextPay and ComfirmTransInfoActivity、DisplayTransferInfo and ComfirmTransInfoActivity、ComfirmTransInfoActivity and ConnectDB、MainFunActivity and ConnectDB、MainActivity and ConnectDB、RegisterActivity and ConnectDB、CreditCardActiVity and ConnectDB、FirstBindActivity and ConnectDB、MainFunActivity and MainActivity、MainActivity and RegisterActivity、MainFunActivity and TranferActivity、ComfirmTransInfoActivity and TranferActivity、TranferActivity and TransferControl、MainFunActivity and TransferControl、ComfirmTransInfoActivity and TransferControl、ComfirmTransInfoActivity and CreditCardIterator、MainFunActivity and CreditCardActiVity、MainActivity and RegisterActivity、FirstBindActivity and CreditCardActiVity、CreditCardActiVity and Aggregate、CreditCardActiVity and Iterator、CreditCard and CreditCardIterator、CreditCardIterator and CreditCard、CreditCardActiVity and CareTaker、CreditCardActiVity and Originator、Originator and CreditCardMemento.

* Functions of your design for two teamwork

Because our teamwork1 directions is wrong so we modify our system functions, we add additional 7 patterns in our teamwork2.

In teamwork2 we add additional functions like transfer can choose which way you want to pay and if you choose creditCard , you can choose which card you want to pay、you can management your creditCard, add, remove, update…and if you did wrong, you can undo and redo the steps、transfer add an AOTP verification process、and when you in transfer process, the system will display the transfer state let you know, at the end, the system will display transfer detail, and it can count how many credit card you have.

1. 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.
2. The first pieces:

TranferControl depends on TransferState, so if TransferState change, TranferControl will change.

For example, if we add a longClick() method in TransferState, then TransferControl need to add a longClick() method too.

1. The second pieces:

SettingUpTransferInformation depends on TransferProcedure, and TransferProcedure depends on TransferState, so if TransferState change, TransferProcedure and SettingUpTransferInformation will changes.

For example, if we add a pressCancel() method in TransferProcedure, then SettingUpTransferInformation need to add a pressCancel() method.

1. The third pieces:

CreditCardIterator depends on Iterator, so if Iterator changes, CreditCardIterator will change.

For example, if we add or remove a method, then CreditCardIterator need to add or remove a method too.

**Participation**

|  |  |  |
| --- | --- | --- |
| StudentID / Name | Work content | participation |
| A10723029  Natalia (Leader) | * Thinking and design the system * Write the pattern code and suite test draw diagram * Draw the class diagram * Do the black / white box testing * Indicates the relationships * Indicate the affect classes * Integration | H |
| A10723035  Sunny | * Thinking and design the system * Count and analyze the invocation chain * Draw and analyze the metrics * Do the black / white box testing * Draw the class diagram | H |
| A10723011  Bryan | * Thinking and design the system * Indicates the relationships * Do the black / white box testing * Draw the class diagram * Indicate the affect classes | H |
| A10723026  Elian | * Thinking and design the system * Count and analyze the invocation chain * Draw and analyze the metrics * Do the black / white box testing * Draw the class diagram | H |
| A10723044  Ricky | * Thinking and design the system * Count the invocation chain * Draw the class diagram | H |
| B10523039  Jess | * Thinking and design the system * Modify, write and Translation the document | H |
| B10523019  Jason | * Thinking and design the system * Write the pattern, AOTP, Juint code and database | H |
| B10523030  Jerry | * Thinking and design the system * Draw the class diagram * Do the ppt | H |