Object-Oriented

Software Engineering

Instructor : Huang, Chuen-Min

**Teamwork1 ver.1**

Group 3

|  |  |
| --- | --- |
| ID | Name |
| A10723029 | Natalia |
| A107230 | Sunny |
| A107230 | Bryan |
| A107230 | Elian |
| A107230 | Ricky |
| B105230 | Jess |
| B105230 | Jason |
| B105230 | Jerry |
|  |  |
|  |  |
| Date 2019/11/7 | |

1. To have a user-friendly GUI, please create a Window containing a set of pane objects. It may include TextPane, ListBox, Button, and so on. Display apply Façade, Mediator to design the GUI. In addition to this, please also apply another 3 patterns to your project.

* **Façade** combines with **Mediator**

Façade intent:

Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.

Mediator Intent:

Define an object that encapsulates how a set of object interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

We use Façade simplifies the interface for the Client.

Our LoginGUI consist of the component build by Façade, LoginGUI includes: JLabel: “Account”, JTextField: “Account”, JLabel: “Password”, JTextField: “Password”, JButton: “Login”. On the backend, the Mediator will call “AccountChecker” and “PasswordChecker” to check whether the login is success or not. The Mediator will coordinate three component: “AccountChecker”, “PasswordChecker” and “BalanceChecker”(both of them are M-to-M interaction). Finally, display the result to user.

After user login successfully, user can make a transfer. Façade will display the “TransferGUI”, including: “Transfer Account”, JTextField: “Transfer Account”, JLabel: “Transfer Amount”、JTextField: “Transfer Amount”, JButton: “Transfer”.

On the other hand, if the login was failed, the system will display LOGIN FAILURE.

In the transaction page, user needs to provide the payee and the amount of transfer. The Mediator will call “BalanceChecker” to help us confirm whether the payee is existed and the amount is enough or not. When the confirmation is done, the transaction will be executed, the balance will be deducted and the GUI will shows the successful message and balance. Otherwise, GUI will tell user “Transfer was Failed”.

**Client.java**

|  |
| --- |
| **package** Facade;  **public** **class** Client{    **public** **static** **void** main(String[] args) {  Facade facade = **new** Facade(); //use Façade pattern  }  } |

**Façade.java**

|  |
| --- |
| **package** Facade;  **import** java.awt.GridLayout;  **import** java.awt.event.ActionEvent;  **import** java.awt.event.ActionListener;  **import** javax.swing.JButton;  **import** javax.swing.JFrame;  **import** javax.swing.JLabel;  **import** javax.swing.JTextField;  **import** javax.swing.SwingConstants;  **public** **class** Facade **extends** JFrame **implements** ActionListener{    JFrame myframe = **new** JFrame("ElePay - Login"); //建立JFrame容器物件    JLabel jlAccount = **new** JLabel("account", SwingConstants.***CENTER***); //文字方塊  JTextField tfAccount = **new** JTextField(); //輸入文字方塊  JLabel jlPsd = **new** JLabel("password", SwingConstants.***CENTER***);  JTextField tfPsd = **new** JTextField();  JLabel jlResult = **new** JLabel(" ");  JButton jbLogin = **new** JButton("login"); //建立按鈕物件    **public** Facade() {  myframe.getContentPane().setLayout(**new** GridLayout(3, 2));  myframe.add(jlAccount);  myframe.add(tfAccount);  myframe.add(jlPsd);  myframe.add(tfPsd);  myframe.add(jbLogin);  myframe.add(jlResult);    jbLogin.addActionListener(**this**); //通知按鈕物件：本物件要當傾聽者  //myframe.getContentPane().add(btn); //取得ContentPane並加入按鈕    myframe.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  myframe.setSize(320, 240); //設定視窗的寬與高  myframe.setVisible(**true**); //將視窗設為要顯示  }    **public** **void** displayTransfer() {  TransferGUI gui = **new** TransferGUI();  }    // public void displayGUI() {  // GUI test = new GUI();  // }  @Override  **public** **void** actionPerformed(ActionEvent e) {  // **TODO** Auto-generated method stub  **if**(e.getSource() == jbLogin) {;  Mediator mediator = **new** Mediator();  **if**(mediator.loginCheck(tfAccount.getText().toString(), tfPsd.getText().toString(), jlResult)) {  displayTransfer();  }  }  }  } |

**Mediator.java**

|  |
| --- |
| **package** Facade;  **import** javax.swing.JLabel;  **public** **class** Mediator {    AccountChecker acctChecker; //Login  PasswordChecker psdChecker; //Login  BalanceChecker balChecker; //Transfer  String account[] = {"000001", "000002", "000003", "000004", "000005", "000006", "000007",};    **public** Mediator() {  acctChecker = **new** AccountChecker();  psdChecker = **new** PasswordChecker();  balChecker = **new** BalanceChecker();  }    **public** Boolean loginCheck(String acount, String psd, JLabel jlResult){  **if**(acctChecker.accountCheck(acount) && psdChecker.psdCheck(psd)) {  System.***out***.println("Login Success\n");  jlResult.setText("Login Success");  **return** **true**;  } **else** {  System.***out***.println("Login Failure\n");  jlResult.setText("Login Failure");  **return** **false**;  }  }    **public** **void** transferCheck(String transAccount, String transAmount, JLabel jlResult) {  jlResult.setText("transfer failure");  **for**(**int** i = 0; i < account.length; i++) {  **if**(transAccount.equals(account[i]) && balChecker.checkBalance(Integer.*parseInt*(transAmount))) {  jlResult.setText("transfer successful" + "\n" + "Your account balance is " + balChecker.getBalance());  **break**;  }  }  }  } |

**PasswordChecker.java**

|  |
| --- |
| **package** Facade;  **public** **class** PasswordChecker {  **private** String psd = "happy123";    **public** String getPsd() {  **return** psd;  }  **public** **boolean** psdCheck(String psd) {  **if**(psd.equals(getPsd())) {  **return** **true**;  } **else** {  **return** **false**;  }  }  } |

**AccountChecker.java**

|  |
| --- |
| **package** Facade;  **public** **class** AccountChecker {  **private** String account = "123456";    **public** String getAccountNumber() {  **return** account;  }    **public** **boolean** accountCheck(String account){  **if**(account.equals(getAccountNumber())) {  **return** **true**;  } **else** {  **return** **false**;  }  }  } |

**TransferGUI.java**

|  |
| --- |
| **package** Facade;  **import** java.awt.GridLayout;  **import** java.awt.event.ActionEvent;  **import** java.awt.event.ActionListener;  **import** javax.swing.JButton;  **import** javax.swing.JFrame;  **import** javax.swing.JLabel;  **import** javax.swing.JTextField;  **import** javax.swing.SwingConstants;  **public** **class** TransferGUI **implements** ActionListener{  JFrame myframe = **new** JFrame("ElePay - Transfer"); //建立JFrame容器物件    JLabel jlTransAccount = **new** JLabel("Transfer Account", SwingConstants.***CENTER***); //文字方塊  JTextField tfTransAccount = **new** JTextField(); //輸入文字方塊  JLabel jlTransAmount = **new** JLabel("Transfer Amount", SwingConstants.***CENTER***);  JTextField tfTransAmount = **new** JTextField();  JLabel jlResult = **new** JLabel(" ");  JButton jbTransfer = **new** JButton("Transfer"); //建立按鈕物件    **public** TransferGUI() {  myframe.getContentPane().setLayout(**new** GridLayout(3, 2));  myframe.add(jlTransAccount);  myframe.add(tfTransAccount);  myframe.add(jlTransAmount);  myframe.add(tfTransAmount);  myframe.add(jbTransfer);  myframe.add(jlResult);    jbTransfer.addActionListener(**this**); //通知按鈕物件：本物件要當傾聽者  //myframe.getContentPane().add(btn); //取得ContentPane並加入按鈕    myframe.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  myframe.setSize(320, 240); //設定視窗的寬與高  myframe.setVisible(**true**); //將視窗設為要顯示  }  @Override  **public** **void** actionPerformed(ActionEvent e) {  // **TODO** Auto-generated method stub  **if**(e.getSource() == jbTransfer) {  Mediator mediator = **new** Mediator();  mediator.transferCheck(tfTransAccount.getText().toString(), tfTransAmount.getText().toString(), jlResult);  }  }  } |

**BalanceChecker.java**

|  |
| --- |
| **package** Facade;  **public** **class** BalanceChecker {    **int** balance = 3064;    **public** Boolean checkBalance(**int** transAmount) {  **if**(transAmount <= balance) {  balance -= transAmount;  System.***out***.println(balance + " ");  **return** **true**;  }**else** {  **return** **false**;  }  }    **public** **int** getBalance() {  **return** balance;  }  } |

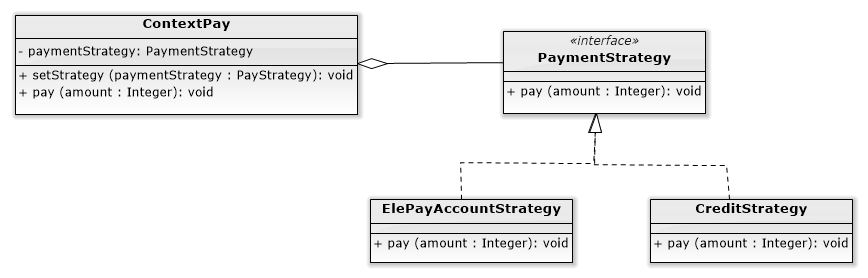
* **Strategy**

Strategy Intent:

Define a family of algorithm, encapsulate each one, and make them interchangeable.

Strategy lets the algorithm vary independently that use it.

**Structure：**



We have two Strategy to handle different payment method, the first one is ElePayAccountStrategy, and the other one is CreditStrategy.

When the user decides how they wanna pay, then the method “setStrategy()” will depend on “PaymentStrategy” to create the payment handle object “ElePayAccountStrategy” or “CreditStrategy”, as a result user can pay in different way.

**PaymentStrategy.java**

|  |
| --- |
| **package com.example.elepay.Strategy;  public interface PaymentStrategy {  public void pay(int amount); }** |

**ElePayAccountStrategy.java**

|  |
| --- |
| **package com.example.elepay.Strategy;  import android.util.Log;  public class ElePayAccountStrategy implements PaymentStrategy{  private String id; //帳戶id  private String name; //姓名  int balance; //餘額   public ElePayAccountStrategy(String id, String name, int balance){  this.id = id;  this.name = name;  this.balance = balance;  }   public void pay(int amount){  Log.*e*("ElePay", "You use ElePay to pay $" + amount);  } }** |

**CreditStrategy.java**

|  |
| --- |
| **package com.example.elepay.Strategy;  import android.util.Log;  public class CreditStrategy implements PaymentStrategy{  private String name; //持卡人姓名  private String creditNo; //信用卡號碼  private String cvc; //信用卡驗證碼  private String expiryDate; //信用卡到期日   public CreditStrategy(String name, String creditNo, String cvc, String expiryDate){  this.name = name;  this.creditNo = creditNo;  this.cvc = cvc;  this.expiryDate = expiryDate;  }   public void pay(int amount){  Log.*e*("credit", "You use credit to pay $" + amount);  } }** |

**ContextPay.java**

|  |
| --- |
| **package com.example.elepay.Strategy;  public class ContextPay {  PaymentStrategy paymentStrategy = null;   public void setStrategy(PaymentStrategy paymentStrategy){  this.paymentStrategy = paymentStrategy;  }   public void pay(int amount){  paymentStrategy.pay(amount);  } }** |

Client in **PayActivity.java**

When we choose want to pay for ElePayAccount, then setStrategy to ElePayAccountStrategy, and invoke pay() method, so it can pay for that way.

When we choose want to pay for CreditCard, then setStrategy to CreditStrategy, and invoke pay() method, so it can pay for that way.

|  |
| --- |
| **@Override**  **public void onClick(DialogInterface dialogInterface, int i) {**  **if(i == DialogInterface.*BUTTON\_POSITIVE*){ //按下Confirm的時候**  **//Strategy Pattern：在付款的時候選擇不同的pay方式**  **ContextPay c = new ContextPay();**  **switch (rgPay.getCheckedRadioButtonId()){**  **case R.id.*rbElePay*:**  **c.setStrategy(new ElePayAccountStrategy("123456", "Natalia", 1500));**  **c.pay(550);**  **break;**  **case R.id.*rbCredit*:**  **c.setStrategy(new CreditStrategy("Natalia", "1234432156788765", "123", "08/25"));**  **c.pay(1000);**  **break;**  **}**  **}else if(i == DialogInterface.*BUTTON\_NEGATIVE*){ //按下Cancel的時候**  **dialog.dismiss();**  **}**  **}** |

* **Factory Method** combine **Abstract Factory** combine **Builder**

Factory Method Intent:

Define an interface for creating an object, but let subclasses decide which class to instantiate.

Factory Method lets a class defer instantiation to subclasses.

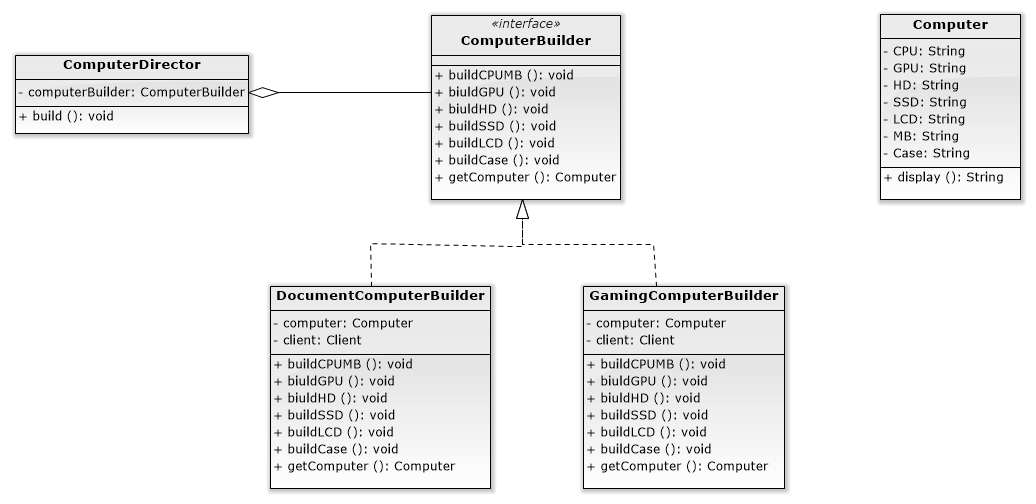
Abstract Factory Intent:

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Builder Intent:

Separate the construction of a complex object from its representation so that the same construction process can create different representations.

* **Builder Structure:**



We have a shopping app sailing the computer. But the computer consists of many different parts.

We separate the construction of a complex object(Computer) from its representation so that the same construction process (the method of ComputerDirector : build()) can create different representations like DocumentComputer or GamingComputer.

Our **Builder** combined with **Abstract Factory**:

**Builder** can use **Abstract Factory** to vary the parts used by the build steps. The method “buildCPUMB()” in class “ComputerBuilder” use **Abstract Factory** to vary the CPU and MB, “DocumentComputerBuilder” uses “AMDFactory” to create “AMDCPU” and “AMDMB”, “GamingComputerBuilder” uses “IntelFactory” to create “IntelCPU” and “IntelMB”.

**Builder code:**

**Computer.java**

|  |
| --- |
| **package com.example.elepay.Builder;  public class Computer {  String CPU;  String GPU;  String HD;  String SSD;  String LCD;  String MB; //com.example.elepay.AbstractFactory.MB MB; //Mother Board主機板  String Case; //Computer Case  String type;   public Computer(String type){  this.type = type;  }   public String getCPU() {  return CPU;  }   public void setCPU(String CPU) {  this.CPU = CPU;  }   public String getGPU() {  return GPU;  }   public void setGPU(String GPU) {  this.GPU = GPU;  }   public String getHD() {  return HD;  }   public void setHD(String HD) {  this.HD = HD;  }   public String getSSD() {  return SSD;  }   public void setSSD(String SSD) {  this.SSD = SSD;  }   public String getLCD() {  return LCD;  }   public void setLCD(String LCD) {  this.LCD = LCD;  }   public String getMB() {  return MB;  }   public void setMB(String MB) {  this.MB = MB;  }   public String getCase() {  return Case;  }   public void setCase(String aCase) {  Case = aCase;  }   public String display(){  StringBuilder sb = new StringBuilder();  sb.append(type + "：");  sb.append("\n CPU:");  sb.append(CPU);  sb.append("\n GPU:");  sb.append(GPU);  sb.append("\n HD:");  sb.append(HD);  sb.append("\n SSD:");  sb.append(SSD);  sb.append("\n LCD:");  sb.append(LCD);  sb.append("\n MB:");  sb.append(MB);  sb.append("\n Case:");  sb.append(Case);  sb.append("\n");   return sb.toString();  } }** |

**ComputerBuilder.java**

|  |
| --- |
| **package com.example.elepay.Builder;  public interface ComputerBuilder {  void buildCPUMB();  void buildGPU();  void buildHD();  void buildSSD();  void buildLCD();  void buildCase();  Computer getComputer(); }** |

**DocumentComputerBuilder.java**

|  |
| --- |
| **package com.example.elepay.Builder;  import com.example.elepay.AbstractFactory.AMDFactory; import com.example.elepay.AbstractFactory.Client;  public class DocumentComputerBuilder implements ComputerBuilder{  private final Computer computer = new Computer("Document Computer");   Client client = new Client(); //AbstractFactory的Client   @Override  public void buildCPUMB() {  client.setFactory(new AMDFactory()); //使用AMDFactory這個ConcreteFactory  client.use(); //建立一系列產品  computer.setCPU(client.getCPU().getType()); //取得ConcreteFactory幫我們建立的CPU，讓Builder去建立Computer的其中一個part(CPU)  computer.setMB(client.getMB().getType()); //取得ConcreteFactory幫我們建立的MB，讓Builder去建立Computer的其中一個part(MB)  }   @Override  public void buildGPU() {  computer.setGPU("GTX 650");  }   @Override  public void buildHD() {  computer.setHD("240G SATA3");  }   @Override  public void buildSSD() {  computer.setSSD("128GB SATA3 M.2");  }   @Override  public void buildLCD() {  computer.setLCD("16 inch");  }   @Override  public void buildCase() {  computer.setCase("405x195x410mm");  }   @Override  public Computer getComputer() {  return computer;  } }** |

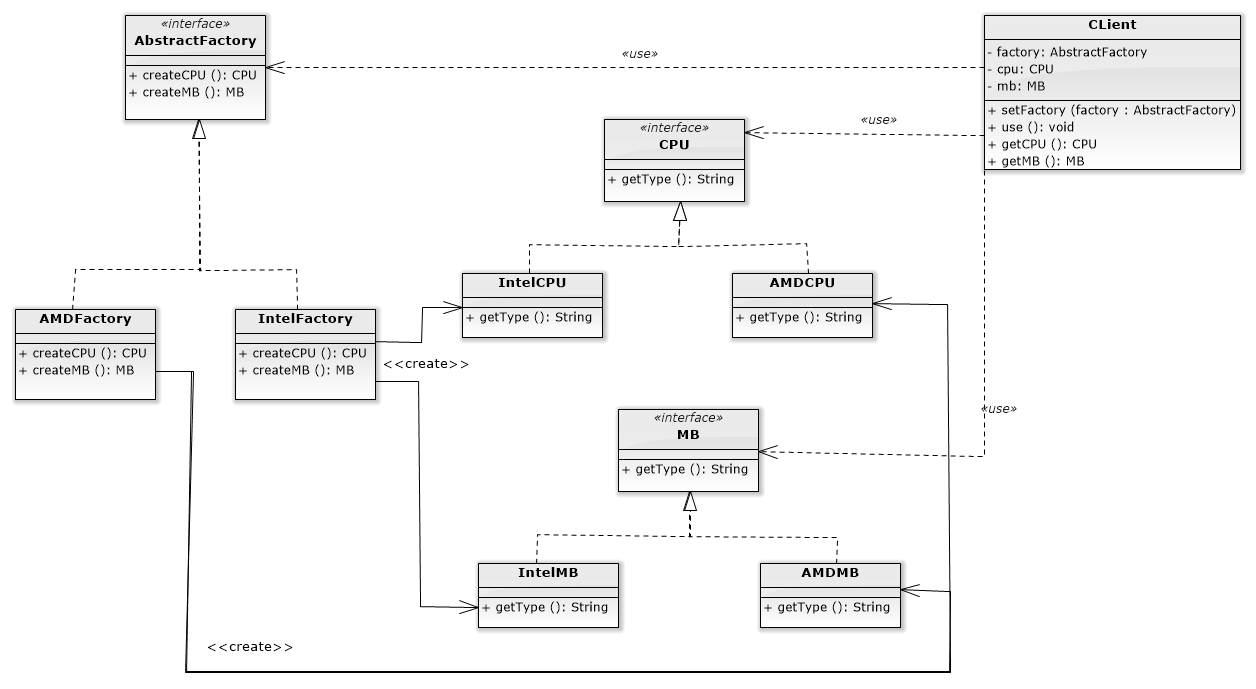
**GamingComputerBuilder.java**

|  |
| --- |
| **package com.example.elepay.Builder;  import com.example.elepay.AbstractFactory.Client; import com.example.elepay.AbstractFactory.IntelFactory;  public class GamingComputerBuilder implements ComputerBuilder{  private final Computer computer = new Computer("Gaming Computer");   Client client = new Client(); //AbstractFactory的Client   @Override  public void buildCPUMB() {  client.setFactory(new IntelFactory()); //使用IntelFactory這個ConcreteFactory  client.use(); //建立一系列產品  computer.setCPU(client.getCPU().getType()); //取得ConcreteFactory幫我們建立的CPU，讓Builder去建立Computer的其中一個part(CPU)  computer.setMB(client.getMB().getType()); //取得ConcreteFactory幫我們建立的MB，讓Builder去建立Computer的其中一個part(MB)  }   @Override  public void buildGPU() {  computer.setGPU("GeForce RTX 2080Ti");  }   @Override  public void buildHD() {  computer.setHD("4TB WD4005FZBX");  }   @Override  public void buildSSD() {  computer.setSSD("250G MX500");  }   @Override  public void buildLCD() {  computer.setLCD("30 inch");  }   @Override  public void buildCase() {  computer.setCase("468.8x200x454.5mm");  }   @Override  public Computer getComputer() {  return computer;  } }** |

**ComputerDirector.java**

|  |
| --- |
| **package com.example.elepay.Builder;  public class ComputerDirector {  private ComputerBuilder computerBuilder;   public ComputerDirector(ComputerBuilder computerBuilder){ //指定要建立哪一種電腦  this.computerBuilder = computerBuilder;  }   public void build(){  computerBuilder.buildCPUMB();  computerBuilder.buildGPU();  computerBuilder.buildHD();  computerBuilder.buildSSD();  computerBuilder.buildLCD();  computerBuilder.buildCase();  }  }** |

* **Abstract Factory Structure：**



Abstract Factory is concerned with the creation of similar products of different brands.

We provide two parts of the computer, CPU and MotherBoard(MB). Each product have two series, intel and AMD.

Our Abstract Factory includes Factory Method:

There’re two methods in class “AbstractFactory” “createCPU()” and “CreateMB()” is the representation of **Factory Method**.

Subclass “AMDFactory” and “IntelFactory” will decide which class to instantiate. “AMDFactory” will create “AMDCPU” and “AMDMB”, and “IntelFactory” create “IntelCPU” and “IntelMB”.

**Abstract Factory Code:**

**AbstractFactory.java**

|  |
| --- |
| **package com.example.elepay.AbstractFactory;  public interface AbstractFactory {  CPU createCPU(); //Factory Method  MB createMB(); //Factory Method }** |

**IntelFactory.java**

|  |
| --- |
| **package com.example.elepay.AbstractFactory;  public class IntelFactory implements AbstractFactory {  @Override  public CPU createCPU() {  return new IntelCPU("Intel Core i9-9900K");  }   @Override  public MB createMB() {  return new IntelMB("Intel Z390 LGA 1151 ATX");  } }** |

**AMDFactory.java**

|  |
| --- |
| **package com.example.elepay.AbstractFactory;  public class AMDFactory implements AbstractFactory {  @Override  public CPU createCPU() {  return new AMDCPU("AMD Ryzen™ Threadripper");  }   @Override  public MB createMB() {  return new AMDMB("AMD AM4 ATX");  } }** |

**CPU.java**

|  |
| --- |
| **package com.example.elepay.AbstractFactory;  //Abstract Product public interface CPU {  String getType(); }** |

**IntelCPU.java**

|  |
| --- |
| **package com.example.elepay.AbstractFactory;  public class IntelCPU implements CPU {  String type;   public IntelCPU(String type) {  this.type = type;  }   public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  } }** |

**AMDCPU.java**

|  |
| --- |
| **package com.example.elepay.AbstractFactory;  public class AMDCPU implements CPU {  String type;   public AMDCPU(String type) {  this.type = type;  }   public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  } }** |

**MB.java**

|  |
| --- |
| **package com.example.elepay.AbstractFactory;  //Abstract Product public interface MB {  String getType(); }** |

**IntelMB.java**

|  |
| --- |
| **package com.example.elepay.AbstractFactory;  public class IntelMB implements MB {  String type;   public IntelMB(String type) {  this.type = type;  }   public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  } }** |

**AMDMB.java**

|  |
| --- |
| **package com.example.elepay.AbstractFactory;  public class AMDMB implements MB {  String type;   public AMDMB(String type) {  this.type = type;  }   public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  } }** |

**Client.java**

|  |
| --- |
| **package com.example.elepay.AbstractFactory;  public class Client {  AbstractFactory factory;  CPU cpu;  MB mb;   public void setFactory(AbstractFactory factory){  this.factory = factory;  }   public void use(){  cpu = factory.createCPU();  mb = factory.createMB();  }   public CPU getCPU(){  return cpu;  }   public MB getMB(){  return mb;  } }** |

We use these combine patterns in **ShoppingActivity.java**

|  |
| --- |
| **@Override**  **public void onClick(View view) {**  **switch (view.getId()){**  **case R.id.*btnShopConfirm*:**  **String strDC = "";**  **String strGC = "";**  **if(cbDC.isChecked()){**  **//Builder Pattern**  **ComputerBuilder computerBuilder = new DocumentComputerBuilder(); //選擇Builder**  **ComputerDirector director = new ComputerDirector(computerBuilder); // 使用文書筆電這個builder來建立產品**  **director.build(); //建立產品**  **Computer DC = computerBuilder.getComputer(); //取得產品**  **strDC = DC.display(); //取得產品資訊**  **sum += 25000 \* Integer.*parseInt*((String) spDCQuan.getSelectedItem());**  **}**  **if(cbGC.isChecked()){**  **ComputerBuilder computerBuilder = new GamingComputerBuilder(); //選擇Builder**  **ComputerDirector director = new ComputerDirector(computerBuilder); // 使用文書筆電這個builder來建立產品**  **director.build(); //建立產品**  **Computer GC = computerBuilder.getComputer(); //取得產品**  **strGC = GC.display(); //取得產品資訊**  **sum += 40000 \* Integer.*parseInt*((String) spGCQuan.getSelectedItem());**  **}**  **Intent intent = new Intent(this, PayActivity.class);**  **intent.putExtra("total", sum);**  **intent.putExtra("strDC", strDC);**  **intent.putExtra("strGC", strGC);**  **startActivity(intent);**  **break;**  **case R.id.*btnShopCancel*:**  **finish();**  **break;**  **}**  **}** |

1. You need to evaluate the pieces of design quality by using object-oriented quality metrics (WMC, DIT, NOC, CBO, RFC, LCOM). The figure shall be drawn like the provided references below. 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.

根據我們所開發出來的軟體，可以發現我們近90%的WMC都低於9，而且大多數的WMC都是在0和2，代表整個軟體複雜度都是偏低的，而目前最高的WMC也只有17左右，對於大型軟體而言這是個漂亮的數字，代表目前整個系統的開發狀況都還是可控制的，若以未來開發來講，會希望把WMC高於10的WMC

降低至10以內，漸漸降低其關聯度以降低整體的潛在風險

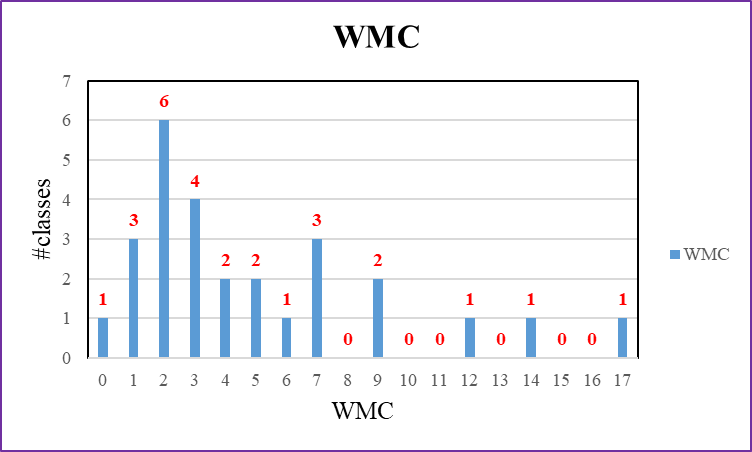


Figure1. Weighted Methods per Class

* DIT (Depth of Inheritance)

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

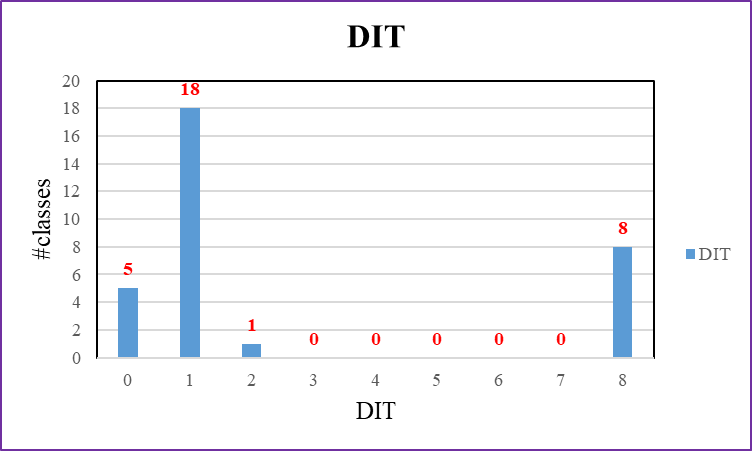


Figure2. Depth of Inheritance

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 1 here, we still have 8 classes with DIT of 8. Therefore, our design can still be improved.

* 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.

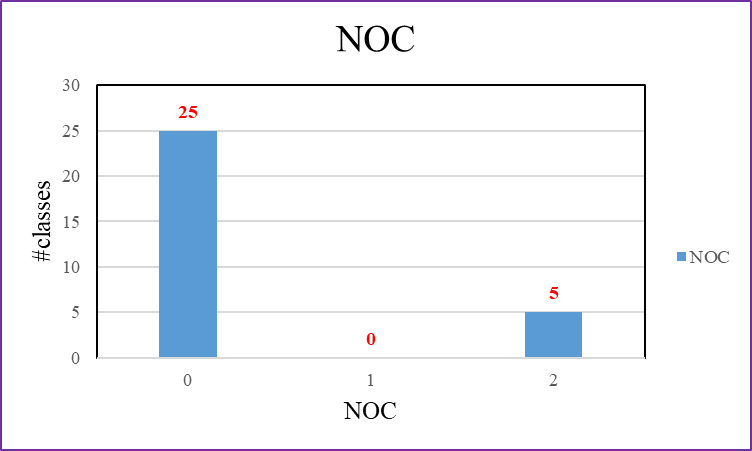


Figure3. Number of Children

* 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.

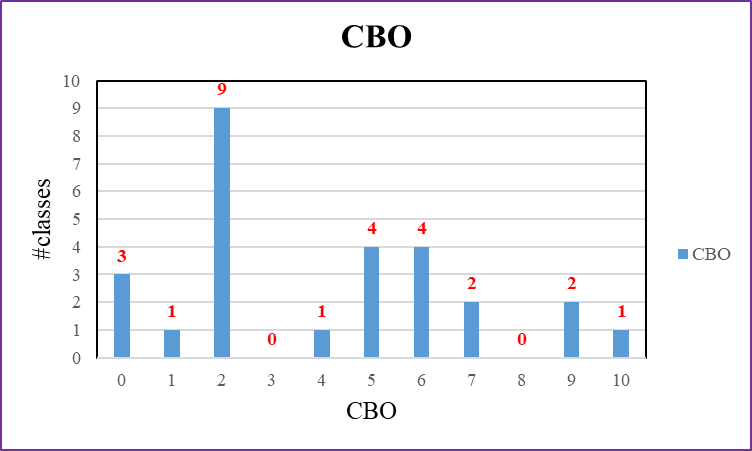


Figure4. Coupling between Object Classes

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 2, but some values are still higher than 5, so we think this is a so-so design.

* 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.

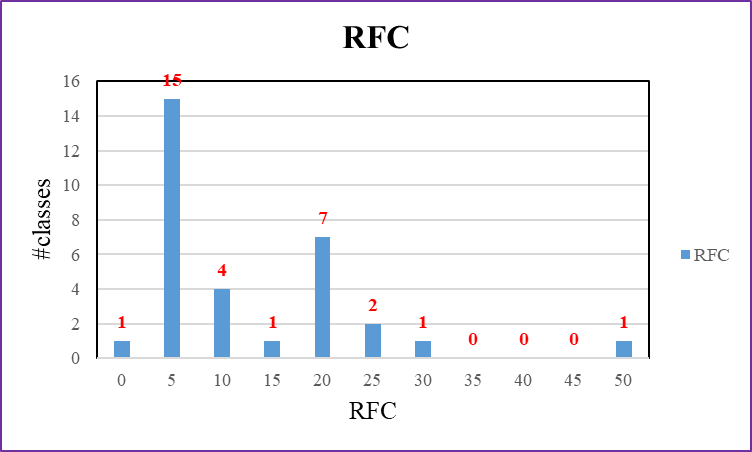


Figure5. Response for a Class

Most of our class with the RFC lower than 25, because they have less complex relationship. If the RFC is high, then it will takes 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.

我們可以看到在RFC這裡，普遍還是低於25的，所以在理解、測試、維護和重用該類別會顯得比較容易一些，這是因為他沒有複雜的交互關係，需要花更多的精力去理解方法並準備測試用例，因此RFC如果越大，就越難去測試和調試這個類別。

在RFC這裡因為大部分為1~5和16~20的數值，這些數值普遍不高，所以我們認為這是一個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 attribte.

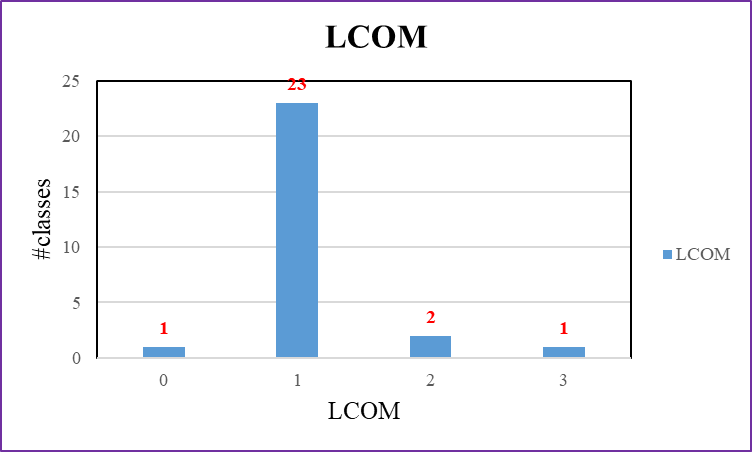


Figure6. Lack of Cohesion Methods

LCOM is the lower the better. As the chart above, LCOM most are one, that means it is high cohesion, two classes LCOM are two, and one class LCOM is three. 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 selected class.

Junit test Code:

|  |
| --- |
| package com.example.elepay;  import com.example.elepay.Other.Verification;  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 ExampleUnitTest {  @Test  public void isValid\_Login(){  Verification va = new Verification();  Assert.*assertEquals*(true,va.CheckLogin("123456","123456"));  }  @Test  public void isValid\_Password(){  Verification va = new Verification();  Assert.*assertEquals*(true,va.CheckPassword("123456","123456"));  }  @Test  public void isValid\_CreditCard(){  Verification va = new Verification();  Assert.*assertEquals*(true,va.CheckCreditCard("5412345678901232"));  }  } |

Junit test suite to test one selected class

|  |
| --- |
|  |

1. Conduct 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, and they have their own password, according to our password regulation, the password must between 6-12 digits. So we set the Equivalence Partitioning Testing as below:

1. Partition 1(length of password < 6)

First subset consists of the length of password less than 6 digits. The test cases in this partition will be rejected.

1. Partition 2(length of password >= 6 && length of password <= 12)

Second subset consists of the length of password between 6 to 12 digits. The test cases in this partition will be accepted.

1. Partition 3(length of password > 12)

Third subset consists of the length of password more than 12 digits. 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 “length of password” is “psdL”.

* For Partition2(length of password >= 6 and length of password <= 12), the test cases are

{psdL = 5, psdL = 6, psdL = 7, psdL = 11, psdL = 12, psdL = 13}

If we set the psdL = 5, the test case is belongs to Partition1(length of password < 6), if we set the psdL = 13, the test case is belongs to Partition3(length of password > 12), the test cases at the boundary of 6 and 12 will cover all the situation.

* Cause Effect Analysis

We choose the event “**login**”to do cause-effect analysis, assume the Account and Password Type is Nonnumeric, and Domain is a single String, so our decision table will be:

Two input condition, Account and Password.

The Account input has four conditions: Astring, null Astring, empty Astring and very long Astring. (Astring means the string name of Account)

Input of Password has four conditions: Pstring, null Pstring, empty Pstring and very long Pstring. (Pstring means the string name of Password)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Account | Astring | Astring | Astring | Astring | null Astring | null Astring | null Astring | null Astring |
| Password | Pstring | null Pstring | empty Pstring | very long Pstring | Pstring | null Pstring | empty Pstring | very long Pstring |
| Rule count | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| result | login success | login failure | login failure | login failure | login failure | login failure | login failure | login failure |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| empty Astring | empty Astring | empty Astring | empty Astring | very long Astring | very long Astring | very long Astring | very long Astring |
| Pstring | null Pstring | empty Pstring | very long Pstring | Pstring | null Pstring | empty Pstring | very long Pstring |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| login failure | login failure | login failure | login failure | login failure | login failure | login failure | login failure |

Table 1. Decision Table

If the first condition is invalid, the result will be login failure. The second condition can be ignored, and it is indifference. Therefor we can merge them into the new decision table as below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Acount | Astring | Astring | Astring | Astring | null Astring | empty Astring | very long Astring |
| Password | Pstring | null Pstring | empty Pstring | very long Pstring | - | - | - |
| Rule count | 1 | 1 | 1 | 1 | 4 | 4 | 4 |
| result | login success | login failure | login failure | login failure | login failure | login failure | login failure |

Table 2. after merge Decision Table

We can see that rule 5,6 and 7 in the new decision table are merged.

The rule count row is useful for completeness checking. That is, the sum of the integers on the row should equal to the total of all possible combinations of the values of the input variable.

Follow the description above, sum of the rules count is 1 + 1 + 1 + 1 + 4 + 4 + 4 = **16**, and the possible combination is 4 \* 4 = **16**, they are the same, so the Decision Table is completeness.

**White Box**

* Basis Path Testing

Basis path testing generates test cases 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.

This code is in **PayActivity.java** onClick() method

|  |
| --- |
| @Override public void onClick(DialogInterface dialogInterface, int i) { ***1*** if(i == DialogInterface.*BUTTON\_POSITIVE*){ //When the confirm pressed   //Strategy Pattern：Choosing the different Payment while checking out  ***2*** ContextPay c = new ContextPay(); ***3*** switch (rgPay.getCheckedRadioButtonId()){  case R.id.*rbElePay*: ***4*** c.setStrategy(new ElePayAccountStrategy("123456", "Natalia", 200000)); ***5*** c.pay(550); ***6*** break;  case R.id.*rbCredit*: ***7*** c.setStrategy(new CreditStrategy("Natalia", "1234432156788765", "123", "08/25")); ***8*** c.pay(1000); ***9*** break; ***10*** } ***11*** }  ***12*** else { //When the cancel pressed  ***13*** dialog.dismiss(); ***14*** } } |

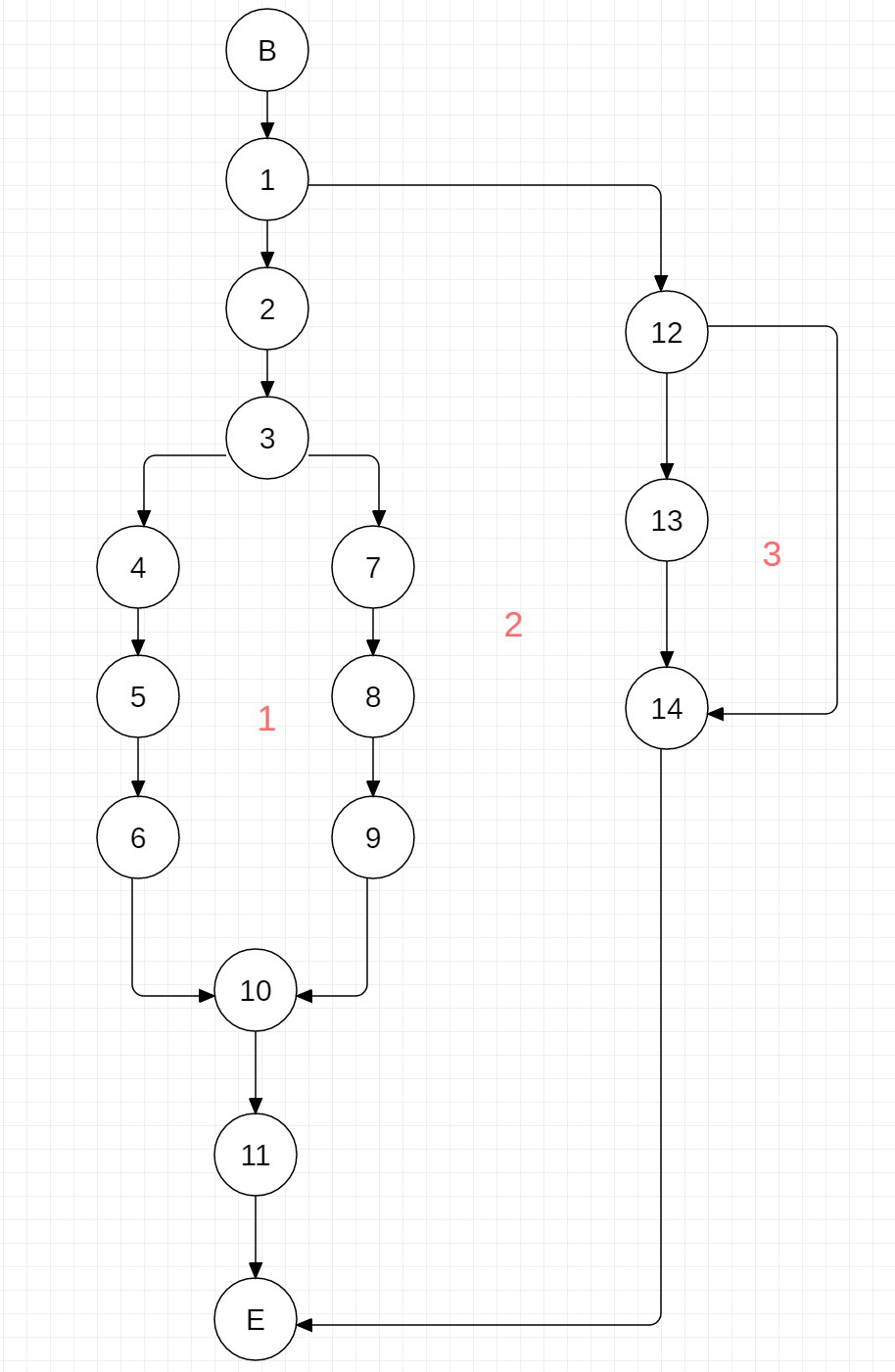


Figure7. Flow graph of the method onClick()

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

1. B→1→2→3→4→5→6→10→11→E
2. B→1→2→3→7→8→9→10→11→E
3. B→1→12→13→14→E
4. B→1→12→14→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 18 – 16 + 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.

* Data Flow Testing
* Loop Testing

1. Please analyze the invocation chains of your design.

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

table1. invocation chains in the ElePay

The classes call each other, result in the so-called invocation chains. That is function f1 calls function f2, which in turn calls function f3, and so on. Table 1 shows the lengths of the invocation chains in the ElePay.

If the number of long Invocation Chain is high, means the coupling is high. As the table above, our system doesn’t have too much long invocation chains, and most of the invocation chains calls one to two functions. Therefore, Elepay is a “Low coupling, high cohesion design”.

1. Based on the metrics result, please identify places that need improvement, Selecting an improvement strategy,

**Participation**

|  |  |  |
| --- | --- | --- |
| StudentID / Name | Work content | participation |
| A10723029  Natalia (Leader) |  |  |
| A10723035  Sunny |  |  |
| A10723011  Bryan |  |  |
| A10723026  Elian |  |  |
| A10723044  Ricky |  |  |
| B10523039  Jess |  |  |
| B10523019  Jason |  |  |
| B10523030  Jerry |  |  |