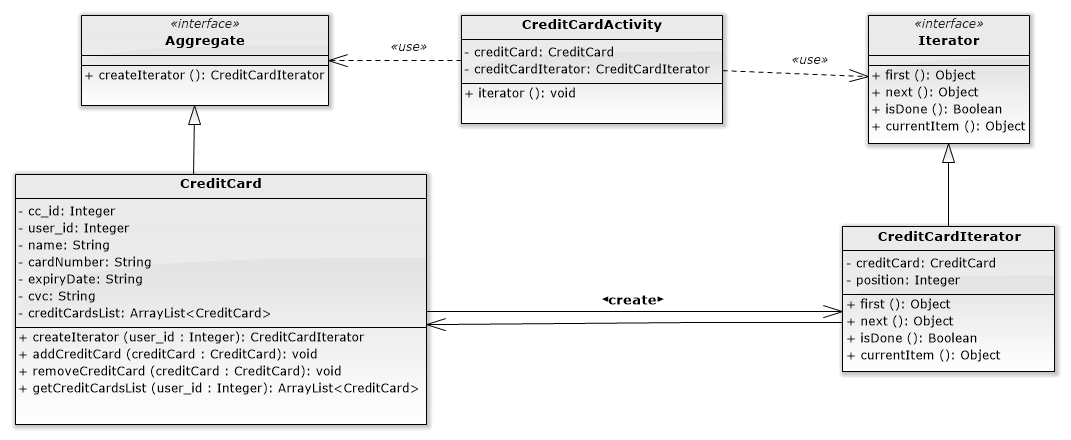
Pattern

* **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(); }** |

* **Adapter**

Intent:

The Adapter Pattern can help us display the listView in the way we want.

We have “CreditCardAdapter” to make the listView display the credit card that user has.

Class: ArrayAdapter

|  |
| --- |
| **public class ArrayAdapter<T> extends BaseAdapter implements Filterable, ThemedSpinnerAdapter {**  **public View getView(int position, @RecentlyNullable View convertView, @RecentlyNonNull ViewGroup parent) {  throw new RuntimeException("Stub!");  }**  **}** |

Class: CreditCardActivity

|  |
| --- |
| **public class CreditCardAdapter extends ArrayAdapter<CreditCard> {   private static final int *mResourceId*=R.layout.*card\_item*;  private LayoutInflater mLayoutInflater;   public CreditCardAdapter(@NonNull Context context, @LayoutRes int resource, @NonNull List<CreditCard> objects) {  super(context, resource, objects);  mLayoutInflater= (LayoutInflater) context.getSystemService(*LAYOUT\_INFLATER\_SERVICE*);  //callIterator();  }   @NonNull  @Override  public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {  View view;   final CreditCard item=getItem(position);    if(convertView==null){  view=mLayoutInflater.inflate(*mResourceId*,parent,false);  }else{  view=convertView;  }  TextView tvName = view.findViewById(R.id.*tvName*);  TextView tvCreditCardNumber = view.findViewById(R.id.*tvCreditCardNumber*);  TextView tvExpiryDate = view.findViewById(R.id.*tvExpiryDate*);  TextView tvCVC = view.findViewById(R.id.*tvCVC*);   tvName.setText(item.getName());  tvCreditCardNumber.setText(item.getCardNumber());  tvExpiryDate.setText(item.getExpiryDate());  tvCVC.setText(item.getCvc());   return view;  }  }**  **}** |

Class: ListView

|  |
| --- |
| **public class ListView extends AbsListView {**  **public void setAdapter(ListAdapter adapter) {  throw new RuntimeException("Stub!");  }**  **}** |

* **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.

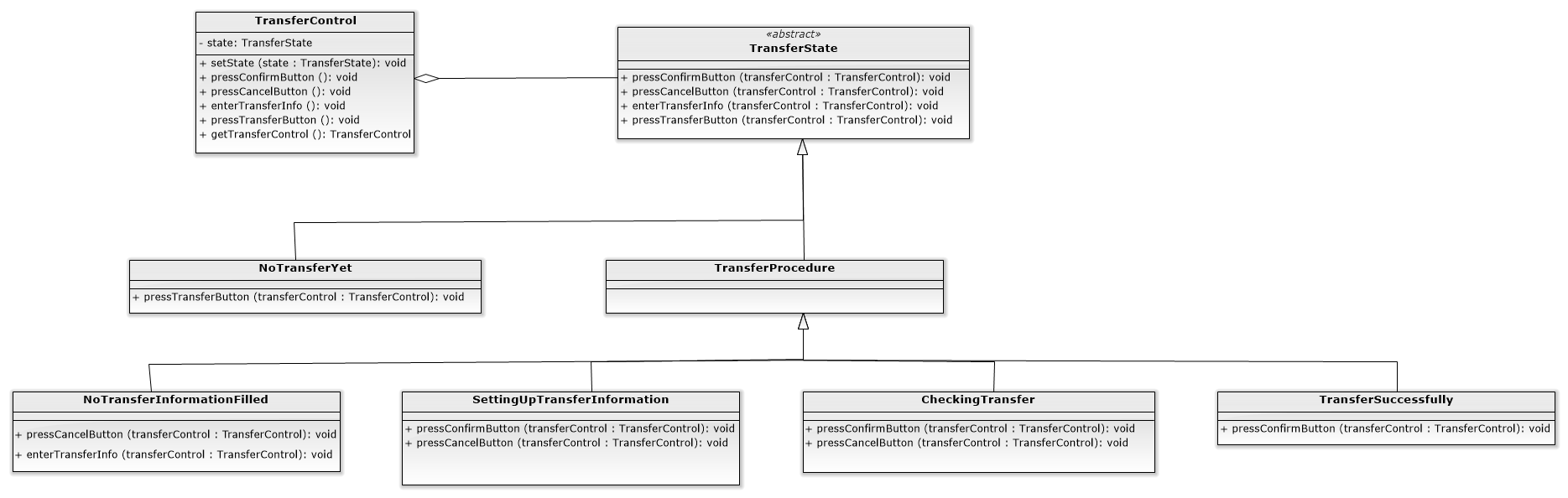
There’re four sub-states in “Transfer Procedure”: 1.No Transfer Information Filled”, 2. Setting Up Transfer Information, 3.Checking Transfer, 4.Confirm Successful Transfer. If the user P

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**

在使用ElePayAccount或是CreditCard進行交易的話，會進行一個SOP的驗證流程，第一步為isEmpty()：是否為空值？、第二步為isValid()：是否為有效長度？、第三步isDoHook()用來判斷是否執行hookMethod：useCoupon()，最後一個步驟為：isVerificate()驗證是否為有效帳戶？我們讓useCoupon()這個hookMethod預設為空，在使用ElePayAccount進行交易的時候，不去override isDoHook()的方法，直接繼承Verifacate的isDoHook()的return false，但如果使用CreditCard進行交易的話，就去override isDoHook()的方法，將它return true，去執行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;  }  } }** |

* **Singleton**

Intent:

We can have only one DB object, the Singleton Pattern can make the DB object as a unique access point when connecting to database.

系統上的DB連線僅需要建立一個實例即可，有效降低不必要的物件建立，並且確立了DB連線的穩定，DB\_Connect是實際對SQL進行連線的一系列方法，而connectDB則為Singleton限制了DB\_Connect實際存在數，因而達到DB\_Connect只會有唯一實例。

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*;  } }** |

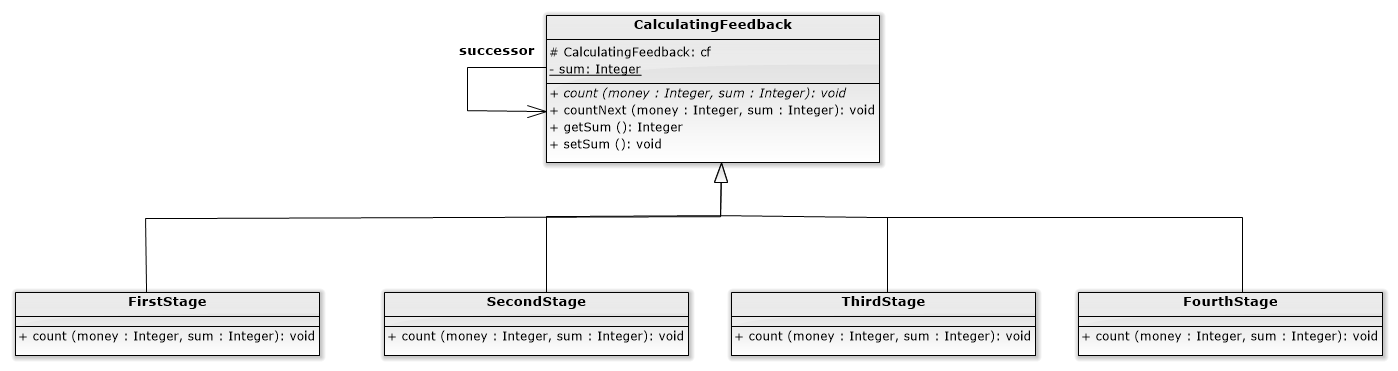
* **Chain of Responsibility**

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

|  |
| --- |
|  |

1. Display some snapshots of the new result in the report.
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.
3. Create Junit test cases and Junit test suite to test one new selected class.
4. 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* }** |

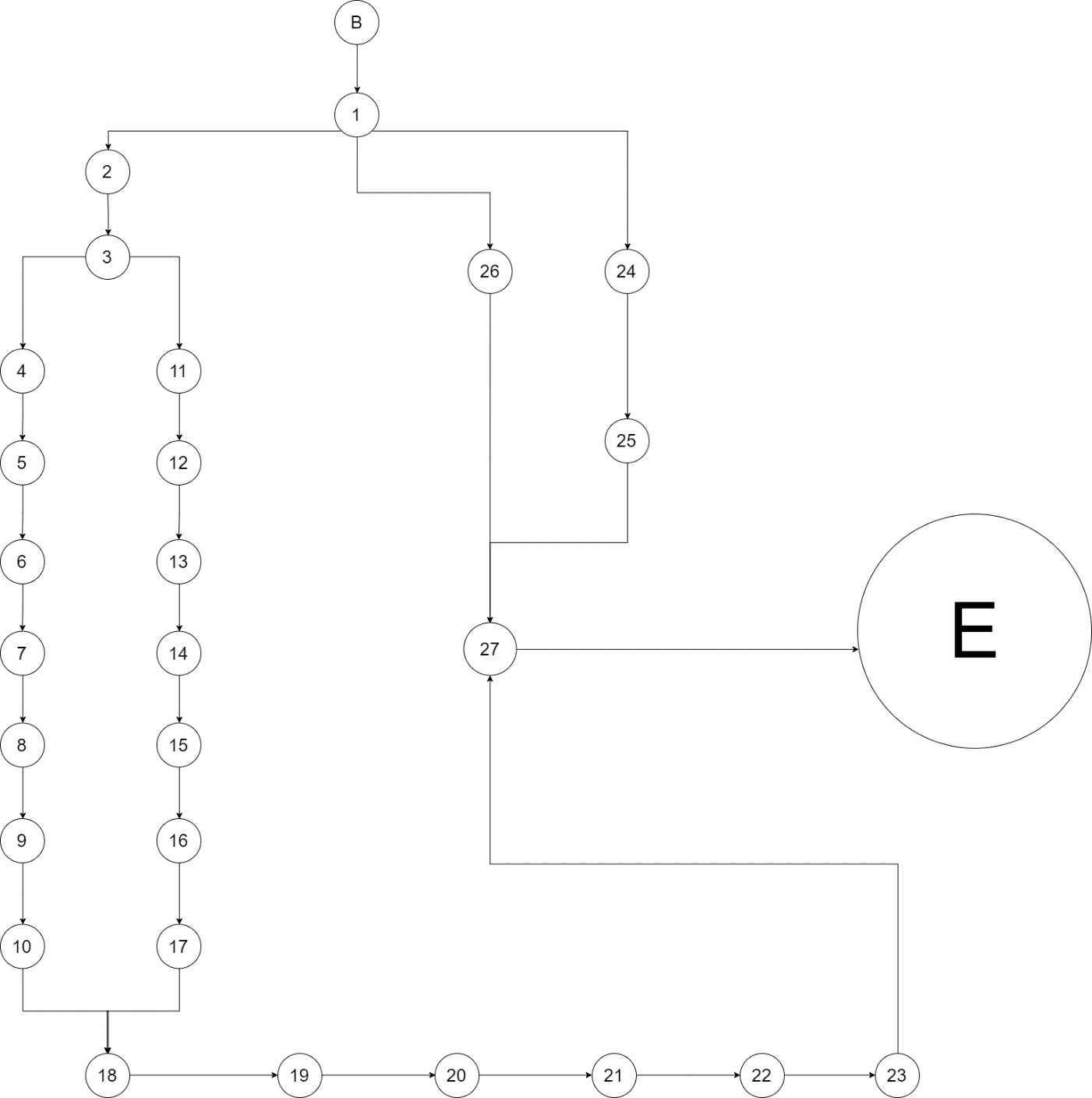


Figure7. Flow graph of the method onClick()

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.
2. Please clearly indicate the number of classes, inheritance, aggregation, association relationships, and functions of your design for the two teamwork, respectively.
3. 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.