Object-Oriented

Software Engineering

Instructor : Huang, Chuen-Min

**Teamwork ver.1**

Group3

|  |  |
| --- | --- |
| ID | Name |
| B10523001 | Carol |
| B10523002 | Ivan |
| B10523003 | Hank |
| B10523004 | Rick |
| B10523005 | Aliss |
| B10523006 | Peggy |
| B10523007 | Bess |
| B10423019 | Ivan |
| B10423051  B10423052  B10423054 | Ginobili  Linda  John |
| Date 2018/ 11 / 11 | |

Contents

[1. Before run our interface 2](#_Toc529966519)

[2. Some snapshots of our web 4](#_Toc529966520)

[3. Class Diagram 10](#_Toc529966521)

[4. Patterns and Code 11](#_Toc529966522)

[ Singleton 11](#_Toc529966523)

[ Memento 13](#_Toc529966524)

[ State 14](#_Toc529966525)

[ Observer 24](#_Toc529966526)

[5. Evaluate the design by using object-oriented quality metrics 26](#_Toc529966527)

[6. Junit test cases and Junit test suite 29](#_Toc529966528)

[7. Software testing including black box and white box 30](#_Toc529966529)

[ Black Box Test 30](#_Toc529966530)

[ White Box Test 33](#_Toc529966531)

[7. Invocation chain of our design 35](#_Toc529966532)

[8. Participate In Assignments 36](#_Toc529966533)

# Before run our interface

Because we use external Database-phpMyAdmin. First, you need to install this Database’s software.

|  |
| --- |
| 1. Go to its download page : https://www.phpmyadmin.net/ |
|  |
| 1. Install it, then open it. Create a database named onesize(don’t have space), and import our one\_size.sql. Our account and password use default value, so you don’t need to change it.   Account : root  Password : “” |

Second, set Build Path. Decompress mysql-connector-java-8.0.13.zip, then set it.

|  |
| --- |
| 1. Right click project name→Build Path→Configure Build Path… |
|  |
| 1. Follow the step, then you can run our program. |
| 5.  4.  3.  2.  1. |

# Some snapshots of our web

* These are our old interface, but our interface can’t work very successful. So we just put same snapshots in document, and we make a new interface behind those.

|  |
| --- |
| 1. Index |
|  |
| 1. Login |
|  |
| 1. Register |
|  |
| 1. Product Page |
|  |
| 1. Cart Page |
|  |

* These are our remake interface.

|  |
| --- |
| 1. Index |
|  |
| 1. Login |
|  |
| 1. Register |
|  |
| 1. Search |
|  |
| 1. Shopping Cart |
|  |
| 1. Checkout |
|  |

# Class Diagram

# Patterns and Code

|  |
| --- |
| DB-DBMgr |
| DB |

## Singleton

|  |
| --- |
| Singleton-Controller |
| controller1controller2controller3controller4 |

## Memento

|  |
| --- |
| Memento-CareTaker |
| memento1 |
| Memento-Memento |
| C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\careTaker1.jpg |
| Memento- Originator |
| C:\Users\user\Desktop\Test info\tw文件圖片\oringinator1.jpgC:\Users\user\Desktop\Test info\tw文件圖片\oringinator2.jpg |

## State

|  |
| --- |
| State\_Pattern- Account\_State |
| C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\AccountState1.jpg  C:\Users\user\Desktop\Test info\tw文件圖片\AccountState2.jpg  C:\Users\user\Desktop\Test info\tw文件圖片\AccountState3.jpg |
| State\_Pattern- Sign\_in\_State |
| SignInState1SignInState2SignInState3SignInState4SignInState5SignInState6SignInState7SignInState8SignInState9SignInState10 |
| State\_Pattern- Sign\_out\_State |
| SignOutState1SignOutState2SignOutState3SignOutState4 |

## Observer

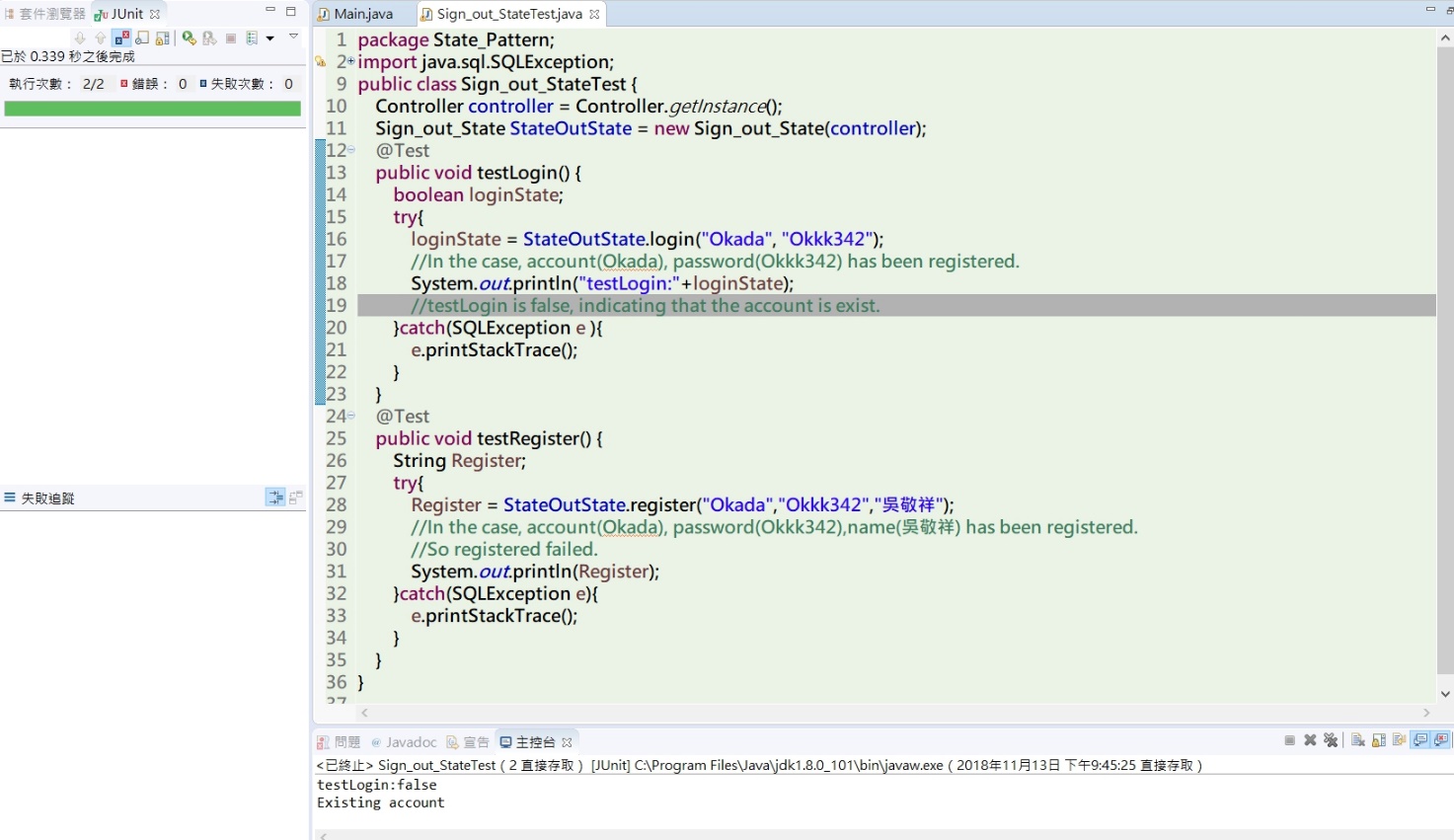
|  |
| --- |
| Observer- Observer |
| observer1 |
| Observer-Product |
| product1product2 |
| Observer- Subject |
| subject1 |
| Observer- User |
| C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\user1.jpg |

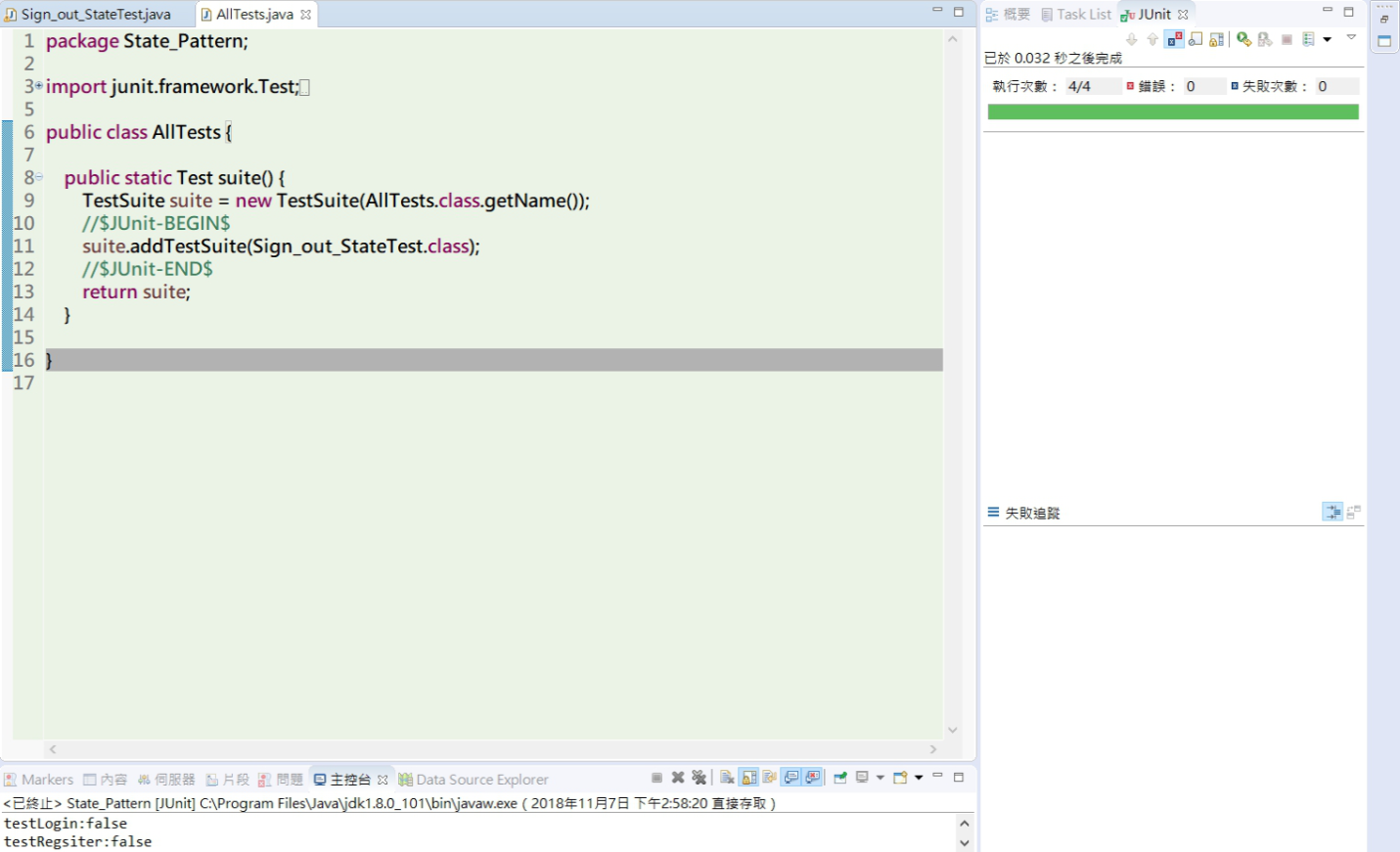
# Evaluate the design by using object-oriented quality metrics

|  |
| --- |
| 1. WMC |
| In our graph, you can see class Sign\_in\_State’s WMC is highest, it will hard to maintain, and reusable. So we have to decrease its complexity. |
| 1. DIT |
| In our graph, we don’t have too high DIT, it means we can easy to predict subclass’s action, and our program’s complexity is very low, it’s a good design. |
| 1. NOC |
| In our graph, there are just three classes-Account\_State, Subject, Observer have subclass, especially Account\_State’s has more reusable. |
| 1. CBO |
| In our graph, all classes coupling are low, although class Sign\_in\_State has the highest coupling, but it is in acceptable range. |
| 1. RFC |
| In our graph, you can see class Sign\_in\_State has too high RFC, it means this class will more difficult to understand, test, maintain, and reuse the class due to higher dependencies of the class on other class. |
| 1. LCOM |
| In our graph, all classes LCOM are very low, it means cohesion of these classes is high, so it’s a good design. |

# **Junit test cases and Junit test suite**

* Junit test case

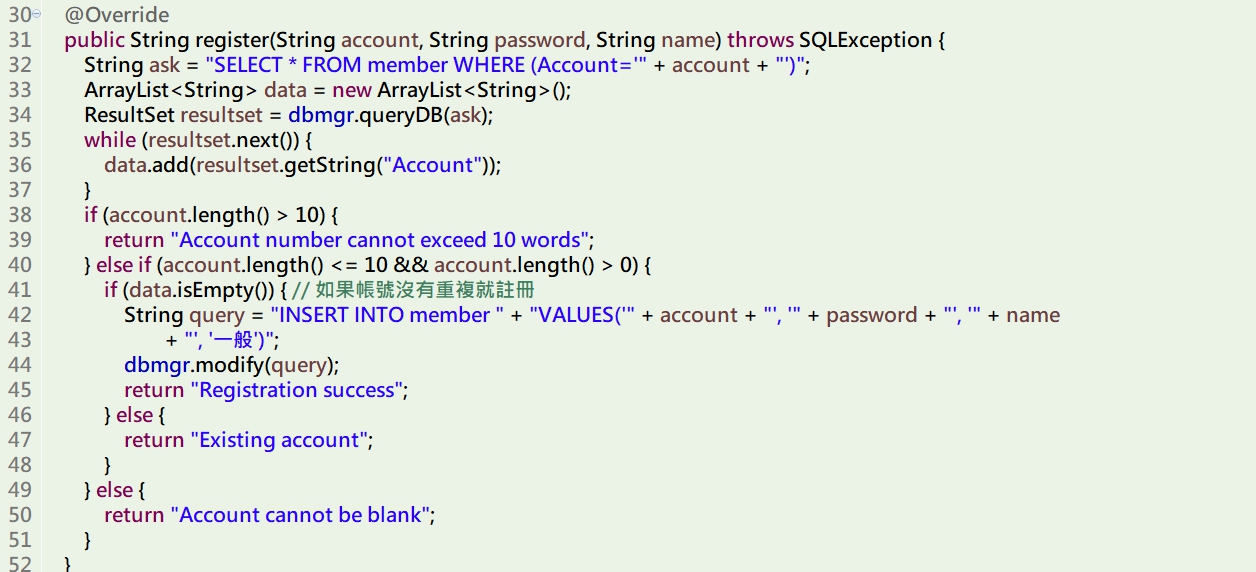
We use class Sign\_out\_State as the test object, the test results are successful, it means it doesn’t have errors.

* Junit test suite

1. Software testing including black box and white box

## Black Box Test

1. Boundary Value Analysis



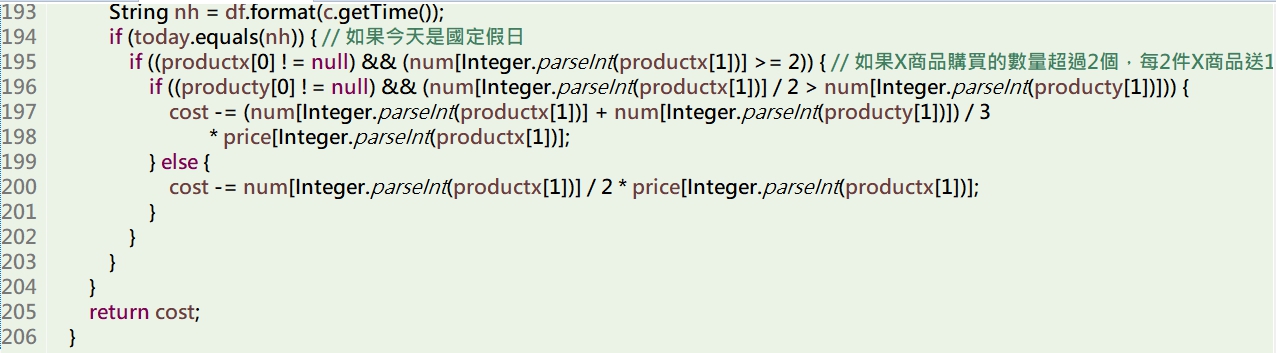
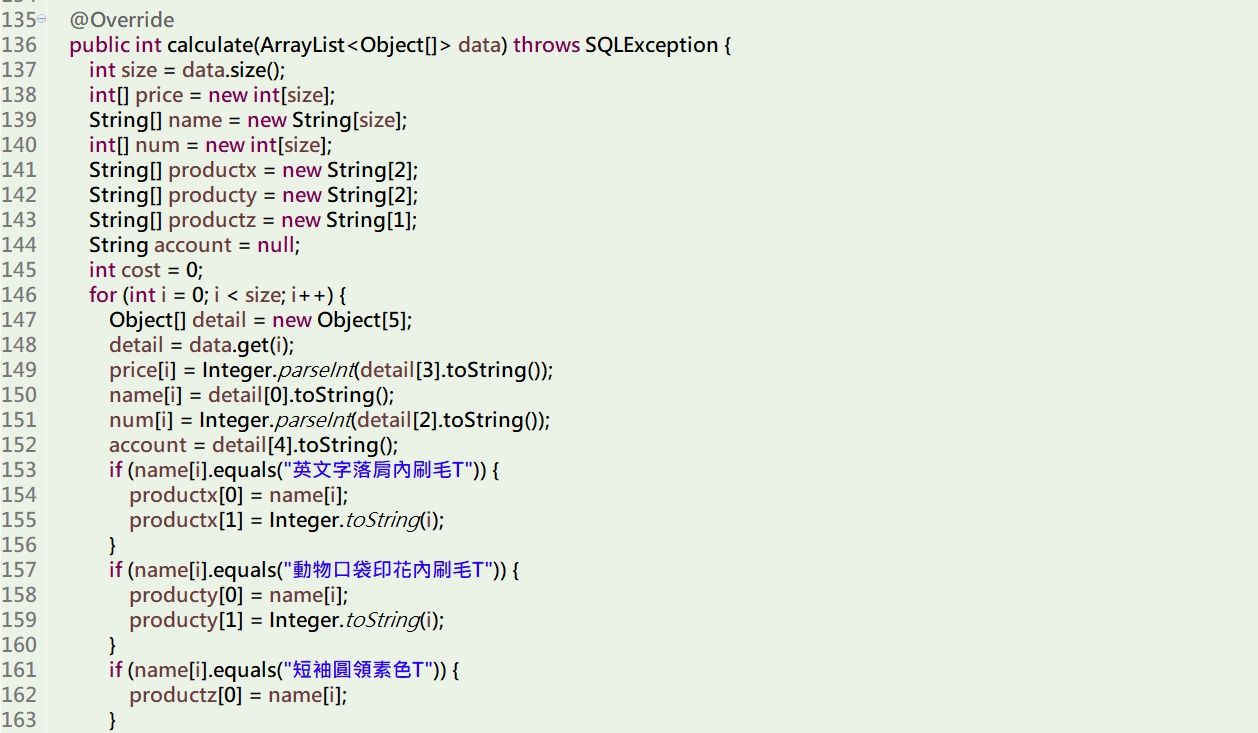
|  |  |  |
| --- | --- | --- |
| No. | Input | Output |
| 1 | “Abcd123” | Registration success |
| 2 | “Abcd123456789” | Account number cannot exceed 10 words |
| 3 | “” | Account cannot be null |
| 4 | “Ban123”(already have this account) | Existing account |

We use method register() as the analysis object, the limit of the account must be:

* User can't enter more than ten characters,
* The content can't be null.
* User can't enter duplicate account with others.

According to above limit, if it enters more than ten characters, it will display "Account number cannot exceed 10 words"; if the content is null, it will display "Account cannot be null"; if it is a duplicate account, it will display "Existing account".

1. Cause-Effect Analysis



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test case  Condition | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Buy productX, productY, productZ | Y | Y | Y | Y | Y | Y | Y | Y | N | N | N | N | N | N | N | N |
| Buy 100 units of productX | Y | Y | Y | Y | N | N | N | N | Y | Y | Y | Y | N | N | N | N |
| Brought in the last year more than $100K | Y | Y | N | N | Y | Y | N | N | Y | Y | N | N | Y | Y | N | N |
| Buy 2 units of ProductX on national holidays | Y | N | Y | N | N | Y | Y | N | Y | N | Y | N | Y | N | Y | N |
| Count | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Discount of 5% | x | x | x | x | x | x | x | x |  |  |  |  |  |  |  |  |
| Discount of 15% | x | x | x | x |  |  |  |  | x | x | x | x |  |  |  |  |
| Discount of 20% | x | x |  |  | x | x |  |  | x | x |  |  | x | x |  |  |
| Get 1 unit of ProductX (or Product Y)free | x |  | x |  |  | x | x |  | x |  | x |  | x |  | x |  |
| Result (a customer gets discount) | 64.6%  +free | 64.6% | 80.75%  +free | 80.75% | 76% | 76%  +free | 95%  +free | 95% | 68%  +free | 68% | 85%  +free | 85% | 80%  +free | 80% | free | - |

We use method calculate() as the analysis object, there are 16 results.

As shown above, if consumers have more than one discount at the same time, they will have discount price by pricing, and there are have 16 kinds of results, each of results will be different.

## White Box Test

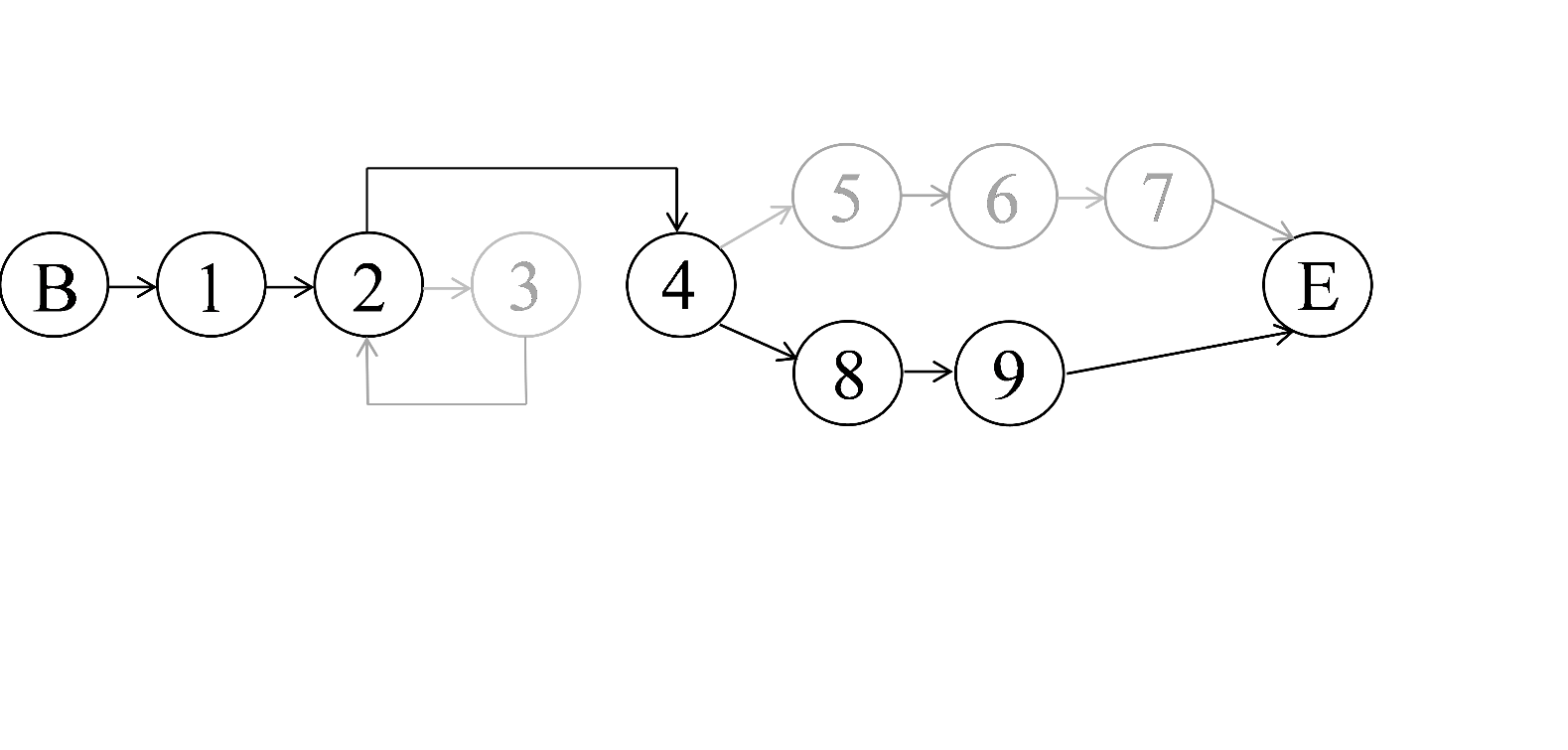
Basis Path Testing

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Override  **public** **boolean** check\_product\_Num(Object[] data) **throws** SQLException {  Product product = **new** Product(data);  String ask = "SELECT Inventory FROM product WHERE (Name='" + data[3] + "' && Product ID='" + data[8] + "')";  ResultSet resultset = dbmgr.queryDB(ask);  **int** num = 1;  **while** (resultset.next()) {  num = resultset.getInt(7);  }  **if** (num == 0) {//This product’s number equal to zero.  String query = "DELETE FROM notify WHERE (Product Name='" + data[3] + "' AND Product ID='" + data[8] + "')";  dbmgr.modify(query); // Delete the message that has been notified.  **return** **true**;  } **else** {// This product’s number doesn’t equal to zero.  product.setData(data); //Notify all observers.  **return** **false**;  }  } |

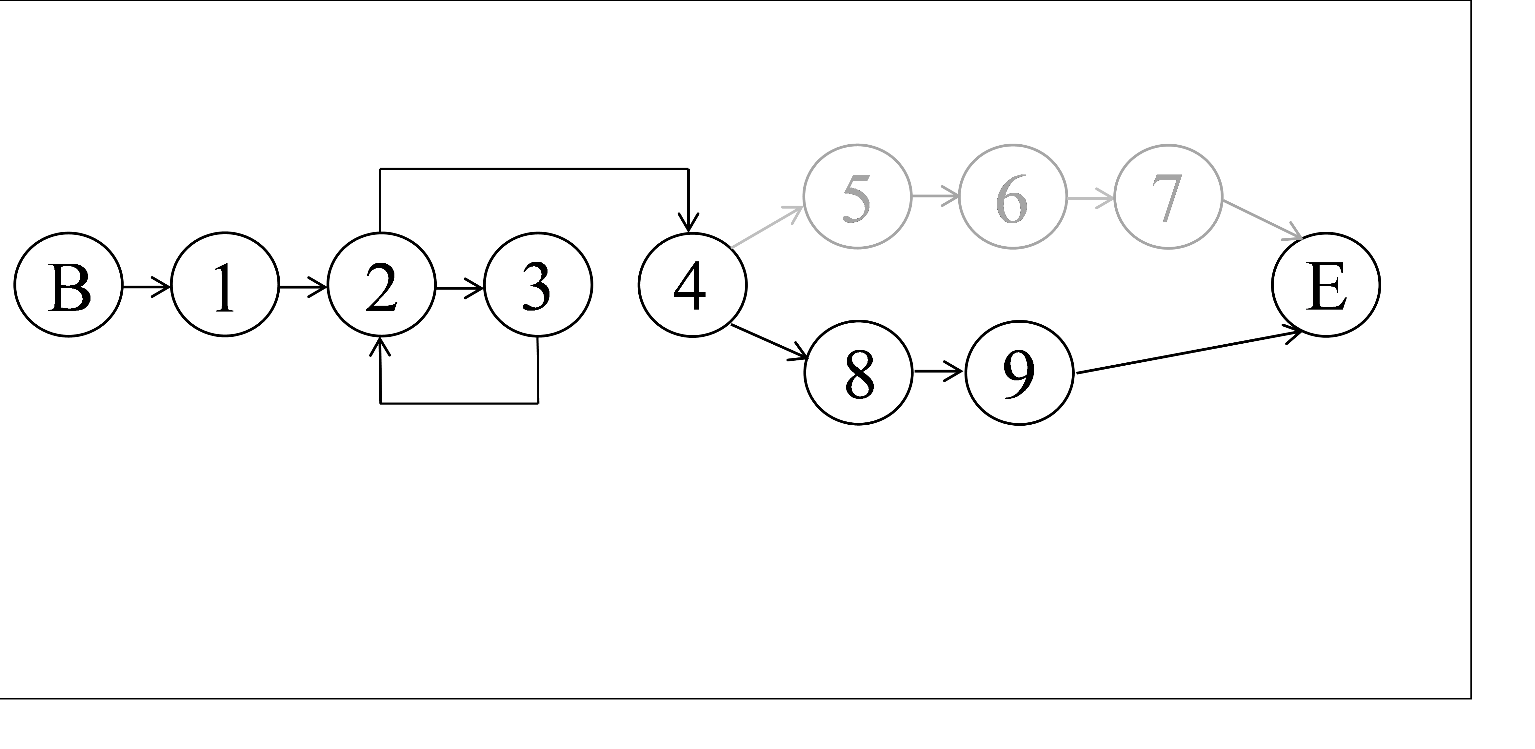
Path**:**

Cyclomatic Complexity : 12 – 11 + 2 = 3

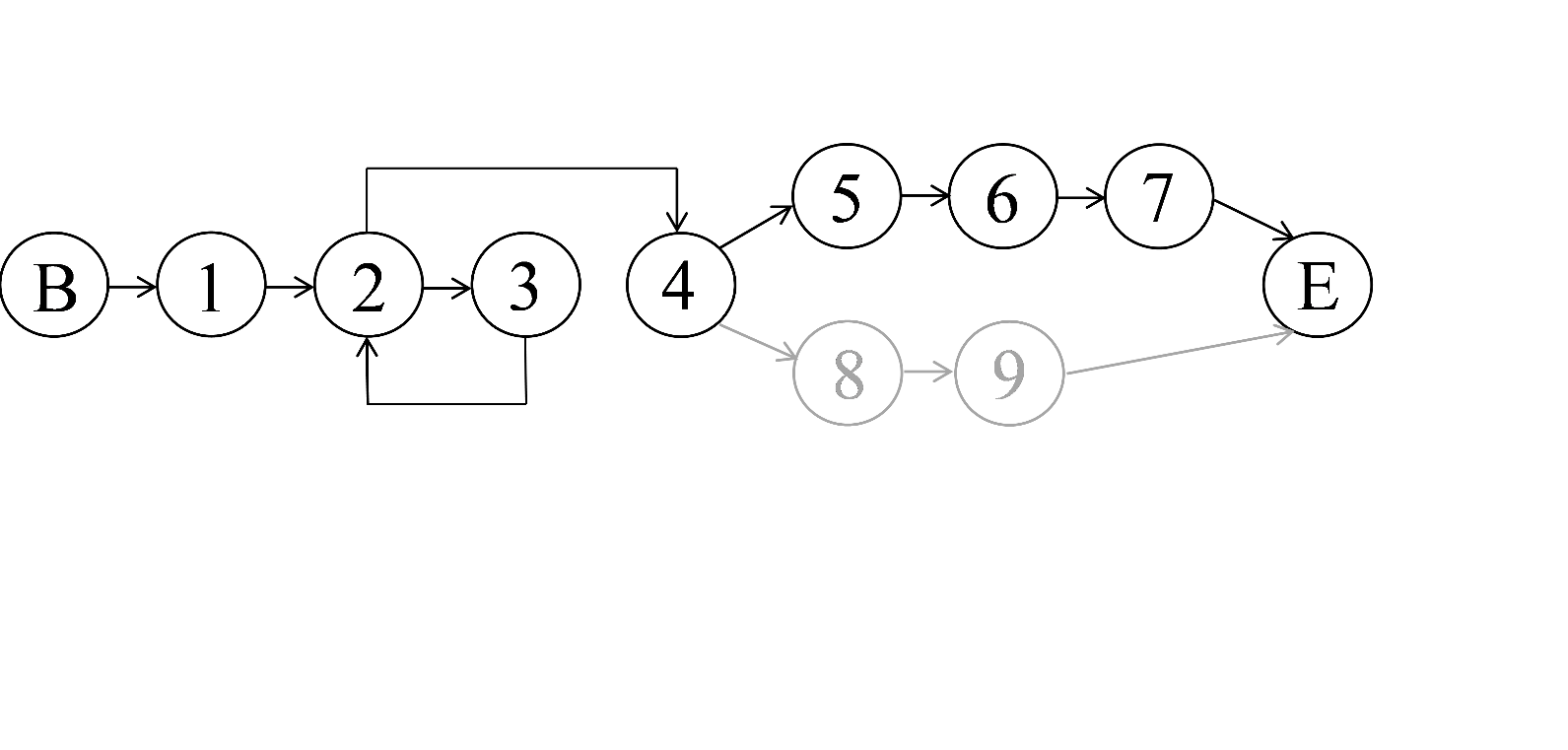
First Path: B,1,2,4,8,9,E



Second Path: B,1,2,3,2,4,8,9,E



Third Path: B,1,2,3,2,4,5,6,7,E



# **Invocation chain of our design**

|  |  |  |  |
| --- | --- | --- | --- |
| Invocation chain length | 1 | 2 | 3 |
| Number of chains | 1 | 3 | 3 |

* Add\_subscription(…) has 1 length of invocation chain.
* Product(…), Register\_Observer(…), User(…) have 2 length of invocation chain.
* getName(), modify(…), printStackTrace() have 3 length of invocation chain.

Our invocation chain is use method Add\_subscription() to analysis, you can see the highest length just have three call process, so if this method is changed, it will not affect to many methods which call this method, and it will easy to know our program’s relationship between Observer and other classes. The lower invocation chain we have, the better program we design.

# Participate In Assignments

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Name | Percentage (%) | Responsibility |
| B10523001 | Carol | 100% | 1. Design(Interface) 2. Document 3. PowerPoint 4. Participation 80% |
| B10523002 | Ivan | 65% | 1. Design(Interface) 2. Class Diagram 3. Participation 40% |
| B10523003 | Hank | 55% | 1. Design(Interface) 2. Document 3. Participation 20% |
| B10523004 | Rick | 100% | 1. Code(Interface) 2. Class Diagram 3. Participation 100% |
| B10523005 | Aliss | 100% | 1. Design(Interface) 2. Document 3. PowerPoint 4. Participation 100% |
| B10523006 | Peggy | 100% | 1. Design(Interface) 2. Document 3. PowerPoint 4. Participation 90% |
| B10523007 | Bess | 100% | 1. Code(Patterns) 2. Participation 100% |
| B10423019 | Ivan | 30% | 1. Participation 10% |
| B10423051 | Ginobili | 0% | 1. Nothing 2. Participation 0% |
| B10423052 | Linda | 100% | 1. Design(Interface) 2. Document 3. Class Diagram 4. Participation 100% |
| B10423054 | John | 0% | 1. Nothing 2. Participation 0% |