

Hong Kong Institute of Vocational Education (Tsing Yi)
Department of Information Technology
HD in Software Engineering (2016/2017)

Assignment Report

Module Name: Contemporary Topics in Software Engineering

Module Number: ITP4507

Course Code: IT114105 Group: 2D

Hand-in: 14th November 2016
(On or before 4:30 PM to Collection Boxes outside Room C440
and Moodle)

Student Name: CHAN Xuan

Student ID: 150212355

Table of Contents

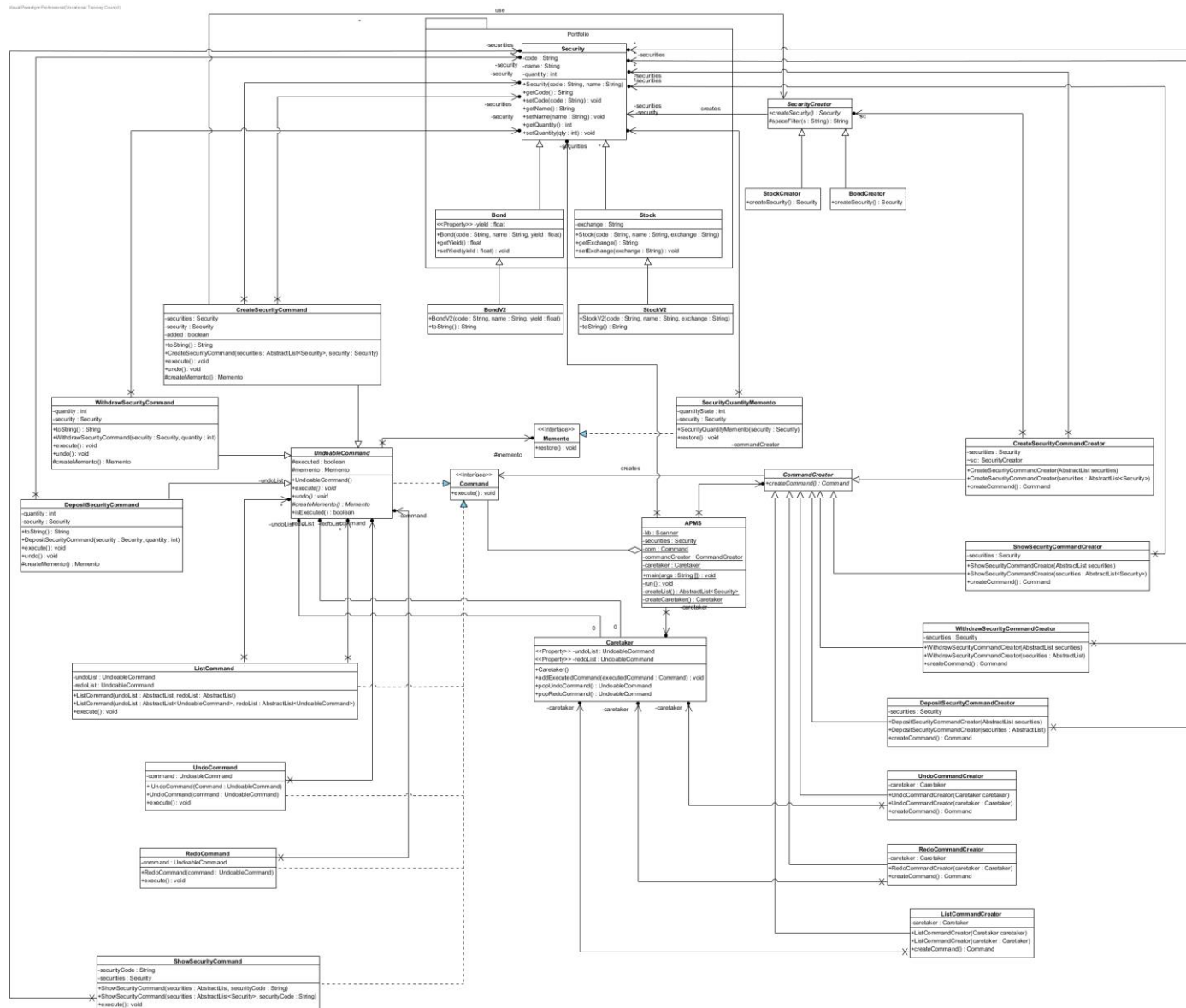
1.	Assumptions regarding the problem context	4
2.	Application design with class diagram	5
3.	Discussion and explanation on each of the design patterns applied to the application	6
3.1	Factory Pattern	6
3.2	Command Pattern	6
3.3	Memento Pattern	7
4.	User Guide.....	8
4.1	Create Security (n).....	8
4.2	Show Security (s).....	9
4.3	Deposit Security (s)	10
4.4	Withdraw Security (s).....	11
4.5	Undo and Redo (u & r)	12
4.6	List undo/redo action (l).....	12
4.7	Exit System (q)	13
5.	Test Plan and Test Cases	14
5.1	Test Plan and Result	14
5.2	Screen dumps of result.....	16
5.2.1	Test Create Command (1)	16
5.2.2	Test Create Command (2)	16
5.2.3	Test Create Command (3)	16
5.2.4	Test Create Command (4)	17
5.2.5	Test Create Command (5)	17
5.2.6	Test Create Command (6)	17
5.2.7	Test show Command (1)	18
5.2.8	Test show Command (2)	18
5.2.9	Test show Command (3)	18
5.2.10	Test show Command (3)	19
5.2.11	Test Deposit Security Command (1)	19
5.2.12	Test Deposit Security Command (2)	19
5.2.13	Test Withdraw Security Command (1).....	20
5.2.14	Test Withdraw Security Command (2).....	20
5.2.15	Test Withdraw Security Command (3).....	20
5.2.16	Test Undo Command (1)	21
5.2.17	Test Redo Command (1).....	21
5.2.18	Test Redo Command (2).....	21
5.2.19	Test Undo Command (2)	22
5.2.20	Test List Command	22
6.	Well documented Source Code	23
6.1	Main program (APMS.java).....	23
6.2	Command interface (Command.java).....	24
6.3	Abstract Undoable Command (UndoableCommand.java).....	24
6.4	Abstract Command Factory (CommandCreator.java).....	24
6.5	Abstract Security Factory (SecurityCreator.java).....	24
6.6	Memento interface (Memento.java).....	24
6.7	Bond version 2 (BondV2.java)	25
6.8	Stock version 2 (StockV2.java)	25
6.9	Stock Factory (StockCreator.java)	26

6.10	Bond Factory (Bond Creator.java).....	26
6.11	Caretaker (Caretaker.java).....	27
6.12	Create Security Command (CreateSecurityCommand.java)	28
6.13	Show Security Command (ShowSecurityCommand.java)	29
6.14	Deposit Security Command (DepositSecurityCommand.java)	30
6.15	Withdraw Security Command (WithdrawSecurityCommand.java)	31
6.16	Undo Command (UndoCommand.java)	32
6.17	Redo Command (RedoCommand.java).....	32
6.18	List Command (ListCommand.java)	33
6.19	Create Security Command Factory(CreateSecurityCommandCreator.java)	34
6.20	Show Security Command Factory (ShowSecurityCommandCreator.java).....	35
6.21	Security Quantity Memento (SecurityQuantityMemento.java).....	35
6.22	Deposit Security Command Factory (DepositSecurityCommandCreator.java).....	36
6.23	Withdraw Security Command Factory (WithdrawSecurityCommandCreator.java)	37
6.24	Undo Command Factory (UndoCommandCreator.java)	38
6.25	Redo Command Factory (UndoCommandCreator.java).....	38
6.26	List Command Factory (ListCommandCreator.java).....	38

1. Assumptions regarding the problem context

- The class of Portfolio can be extent to fulfil
- The Security code will not duplicate.

2. Application design with class diagram



3. Discussion and explanation on each of the design patterns applied to the application

3.1 Factory Pattern

To create different Command object, Abstract Factory is applied to the application. Firstly, the Command interface(Product) and CommandCreator(Creator) abstract class need to be declared. Then, other command (Concrete Product) (e.g.: ShowSecurityCommand, DepositSecurityCommand) which implements the Command interface. After then, the CommandCreator declares the factory method, which returns an object of type Product. Next, the ConcreteCreators(e.g.: ShowSecurityCommandCreator, DepositSecurityCommandCreator) just overrides the factory method to return an instance of a ConcreteProduct. Finally, Creator is declared in the main program(APMS), user will input a command to choose different subclass(ConcreteCreator) that implement the factory method, it will return an instance of the Command(ConcreteProduct).

To create different Security is same. Security is the Product, Bond and Stock are the ConcreteProducts. SecurityCreator is the Creator. BondCreator and StockCreator are the ConcreteCreators.

Moreover, to create Memento object, Factory Method(createMemento) is implemented into UndoableCommand class. This is to fulfil each different command needs since consider different UndoableCommand need different Memento object.

In summary, use Factory Pattern which can hide how object are created and help make the overall system independent of how its object are created and composed. Also, Abstract Factory can contain other code such as method and another object instance to help create the new Product. In the future, when the requirements of the application change such as new Command(Product), just need to create new ConcreteCreator and ConcreteProduct which override the old Create Method, and change the associated factory class.

3.2 Command Pattern

To encapsulate a request as an object, Command Pattern is applied to the application. Firstly, declared a Command interface that contain an execute() abstract method. Additionally, consider the case of undoing or redoing some action, to make the application supporting it, the UndoableCommand that is an abstract class implemented Command interface is declared. The UndoableCommand have Memento object and undo() method. Then, when the user sends the request, it is stored in the object which is created by the CommandCreator. Next, the Invoker(APMS) that holds the command will execute the request by calling the execute() method. After then, the Receiver will act by each request.

In the application, when the user send a deposit request, the DepositCommandCreator will return a DepositCommand which contains the deposit quantity and the security(receiver) to the invoker(APMS). Then, invoker(APMS) will call the execute() method to add the quantity of the security(receiver) by calling setQuantity() Method. To support undoable operation, before the action need to create new Memento object that own by command, to store the quantity. After the calling execute() method, the Command object will store in Caretaker(push to Command Stack), when user in case of undoing action, then pop out the Command and call the undo() method.

In summary, the intent of command pattern is to encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

3.3 Memento Pattern

To support undoable operation, Memento Pattern is applied to the application. Firstly, declared a memento interface that contain a `restore()` method. For each undoable operation, the command object will save the state of the originator to Memento object. Then, push the command into Caretaker. When an undoable operation need to be undone, the main program will call `popUndoCommand()` and call the `undo()` undoableCommand to undo. In addition, for redo operation just need to store the state again before undo operation.

In the application, each command before execute will create a memento object to store the state. After executing, the command will push into `undoCommand` stack of Caretaker. For example, when the user send an undo request to undo deposit, the `UndoCommandCreator` will call `caretaker pop` the deposit command and return a new `UndoCommand` (instance of `Command`), as the same time the caretaker will push the deposit command into redo command stack. Next, the main program(APMS) that holds the command will call the `execute()` method, than to call `undo()` method that into the deposit Command. In redo case, steps are same, but this time will pop the command object to undo command stack. Regarding the roles of Memento Pattern in this case, Security is the originator.

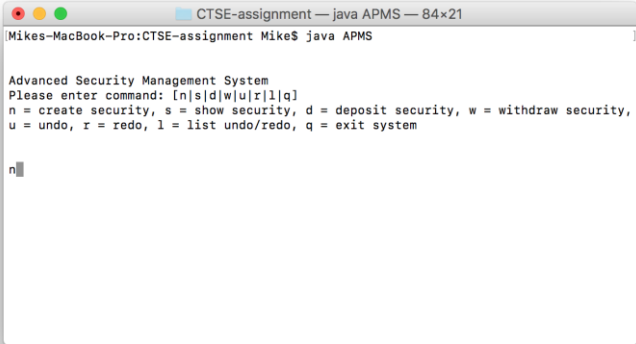
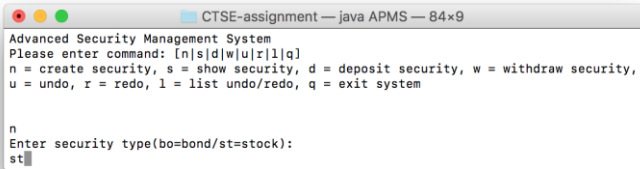
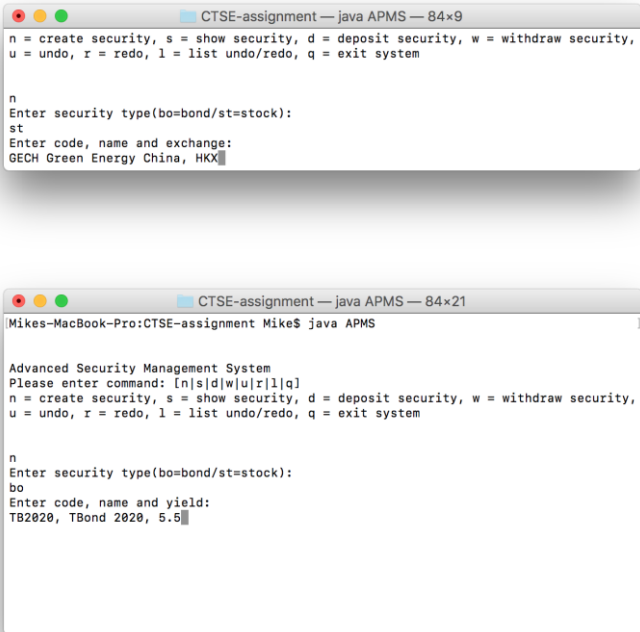
However, the `CreateSecurityCommand` do not need to store the state for undo or redo operation, so that do not be applied memento pattern.

In summary, the intent of this pattern is to capture the internal state of an object without violating encapsulation and thus providing a mean for restoring the object into initial state when needed.

4. User Guide

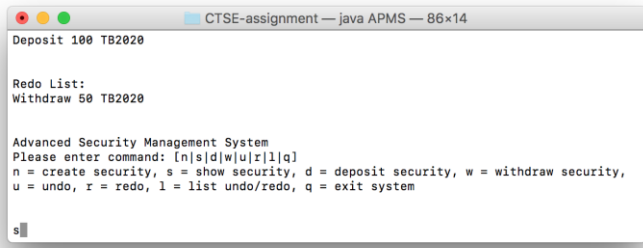
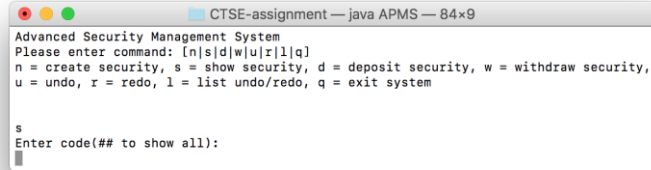
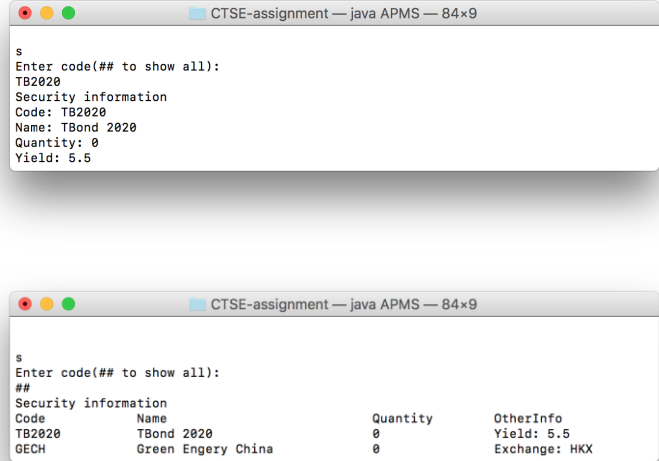
4.1 Create Security (n)

This function is to create a new create security.

1.		Enter “n” to start create security section, then press enter.
2.		Enter security type to select the type, then press enter. Enter “bo” to select bond, “st” to select stock.
3.		Enter the code, name and other info. The format must be: “Code, Name, Other Info” For the bond, the other info is yield. (yield is a number) For the stock, the other info is stock. Finally, Press enter to create.

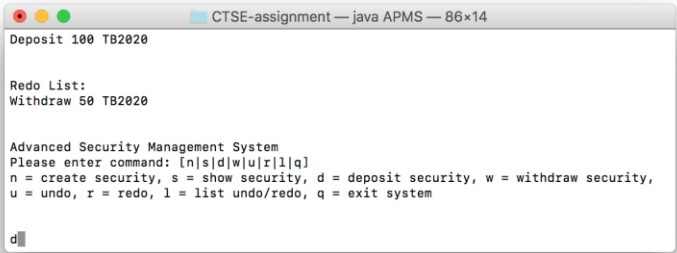
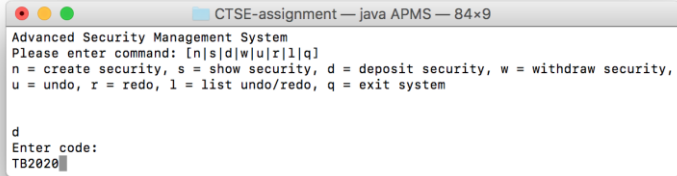
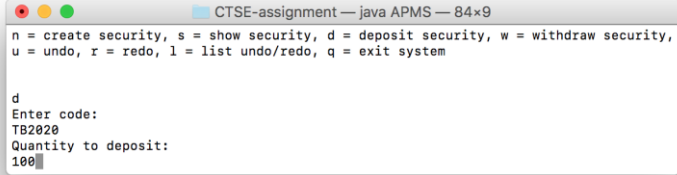
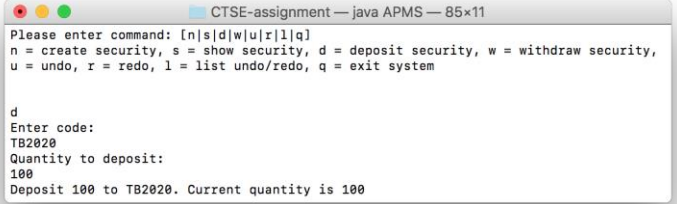
4.2 Show Security (s)

This function is to show security information.

1.		Enter “s” to start show security section, then press enter.												
2		Enter the code of the security which you want, then press enter. (Enter ## to show all)												
3.	 <table><tr><th>Code</th><th>Name</th><th>Quantity</th><th>OtherInfo</th></tr><tr><td>TB2020</td><td>TBond 2020</td><td>0</td><td>Yield: 5.5</td></tr><tr><td>GECH</td><td>Green Engery China</td><td>0</td><td>Exchange: HKX</td></tr></table>	Code	Name	Quantity	OtherInfo	TB2020	TBond 2020	0	Yield: 5.5	GECH	Green Engery China	0	Exchange: HKX	The system will show you the information of security.
Code	Name	Quantity	OtherInfo											
TB2020	TBond 2020	0	Yield: 5.5											
GECH	Green Engery China	0	Exchange: HKX											

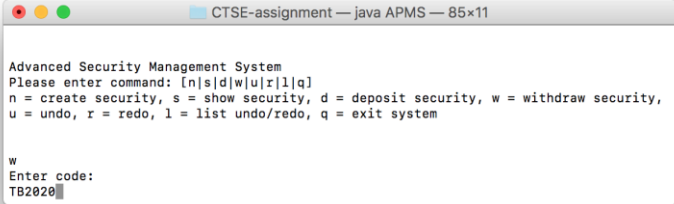
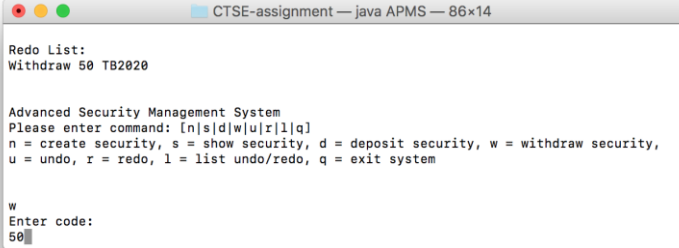
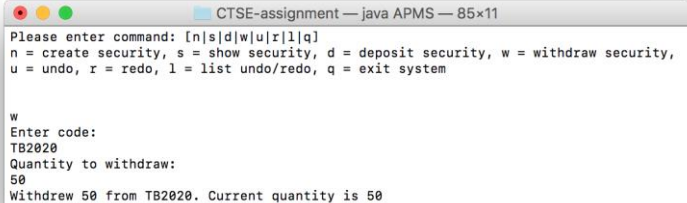
4.3 Deposit Security (s)

This function is to deposit security.

1.		Enter “d” to start the deposit section, then press enter.
2.		Enter the code of the security which you want, then press enter.
3.		Enter the quantity which you want to deposit, then press enter.
4.		Completed.

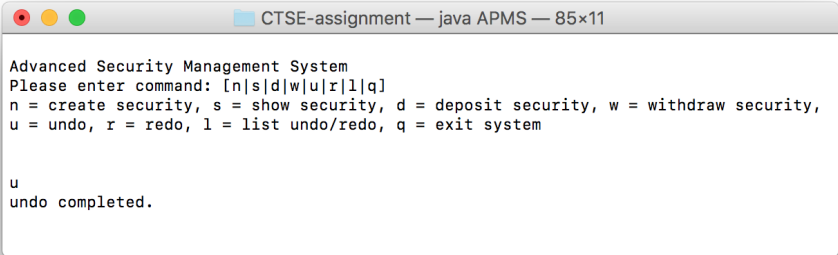
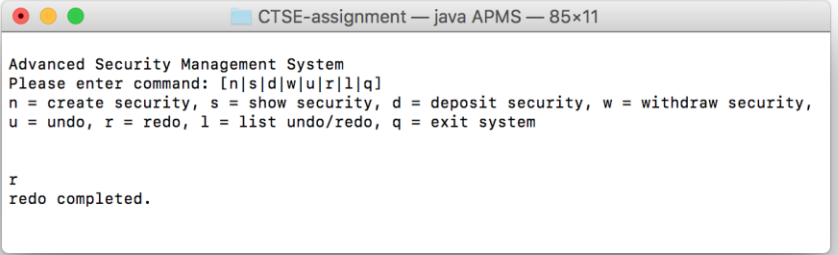
4.4 Withdraw Security (s)

This function is to withdraw security.

1.		Enter “d” to start the withdraw section, then press enter.
2.		Enter the code of the security which you want, then press enter.
3.		Enter the quantity which you want to withdraw, then press enter. (Make sure the current quantity greater than or equal withdrawal quantity)
4.		Completed.

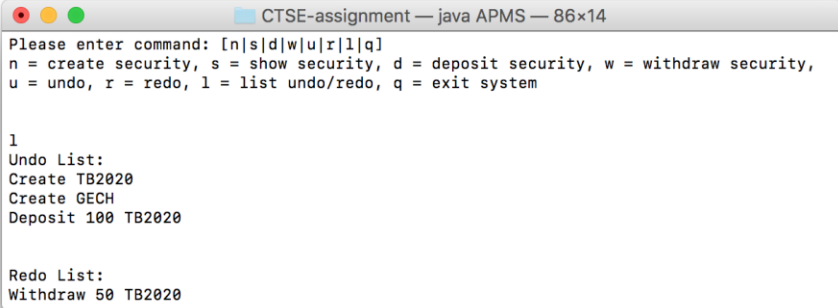
4.5 Undo and Redo (u & r)

This system allows user undo and redo action.

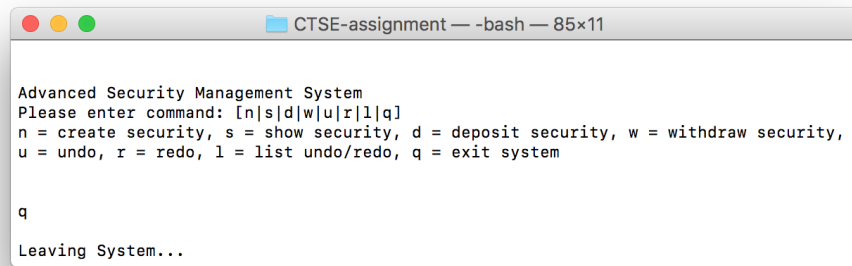
 <pre>CTSE-assignment — java APMS — 85x11 Advanced Security Management System Please enter command: [n s d w u r l q] n = create security, s = show security, d = deposit security, w = withdraw security, u = undo, r = redo, l = list undo/redo, q = exit system u undo completed.</pre>	<p>Input “u”, then press enter to Undo last action.</p>
 <pre>CTSE-assignment — java APMS — 85x11 Advanced Security Management System Please enter command: [n s d w u r l q] n = create security, s = show security, d = deposit security, w = withdraw security, u = undo, r = redo, l = list undo/redo, q = exit system r redo completed.</pre>	<p>Input “r”, then press enter to Redo last undid action.</p>

4.6 List undo/redo action (l)

This system allows user list undo and redo action.

 <pre>CTSE-assignment — java APMS — 86x14 Please enter command: [n s d w u r l q] n = create security, s = show security, d = deposit security, w = withdraw security, u = undo, r = redo, l = list undo/redo, q = exit system l Undo List: Create TB2020 Create GECH Deposit 100 TB2020 Redo List: Withdraw 50 TB2020</pre>	<p>Input “l”, then press enter to list all the actions that can be undone or redone.</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------

4.7 Exit System (q)



```
CTSE-assignment — -bash — 85x11

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

q

Leaving System...
```

Input “q”, then press enter to quit the system.

5. Test Plan and Test Cases

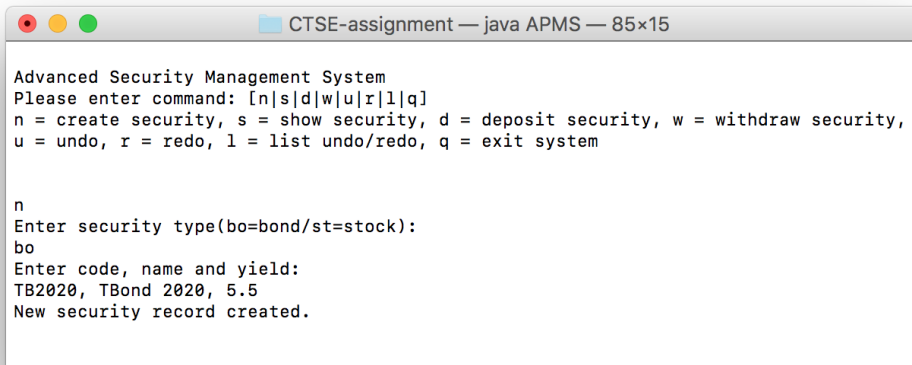
5.1 Test Plan and Result

Test plan	Excepted result	Result
1. Test Create Command Input n→press enter Input bo→press enter Input TB2020, TBond 2020, 5.5 →press enter	Create successful message	Create successful message
2. Test Create Command Input n→press enter Input st→press enter Input GECH, Green Energy China, HKX →press enter	Create successful message	Create successful message
3. Test Create Command Input n→press enter Input 123→press enter	Invalid input message	Invalid input message
4. Test Create Command Input n→press enter Input bo→press enter Input GECH1, Green Energy China, HKX →press enter	Invalid input message	Invalid input message
5. Test Create Command Input n→press enter Input bo→press enter Input ,, →press enter	Invalid input message	Invalid input message
6. Test Create Command Input n→press enter Input bo→press enter Input TB2020, TBond 2020, 5.5 →press enter (same code is existing)	Security already existed message	Security already existed message
7. Test show Command Input s→press enter Input ##→press enter	Show all security info	Show all security info
8. Test show Command Input s→press enter Input TB2020→press enter	Show TB2020 security info	Show TB2020 security info
9. Test show Command (no security be created) Input s→press enter	No Security Created message	No Security Created message
10. Test show Command Input s→press enter Input 123→press enter	Security not found message	Security not found message
11. Test Deposit Security Command Input d→press enter Input TB2020→press enter Input 100→press enter	Deposit successful message	Deposit successful message
12. Test Deposit Security Command Input d→press enter Input 123→press enter	Security not found message	Security not found message

13. Test Withdraw Security Command Input w→press enter Input TB2020→press enter Input 101→press enter	Invalid quantity message	Invalid quantity message
14. Test Withdraw Security Command Input w→press enter Input TB2020→press enter Input 50→press enter	Withdraw successful message	Withdraw successful message
15. Test Withdraw Security Command Input w→press enter Input 123→press enter	Security not found message	Security not found message
16. Test Undo Command Input u→press enter	Undo complete message	Undo complete message
17. Test Redo Command Input r→press enter	Redo complete message	Redo complete message
18. Test Redo Command (No action to be redone) Input r→press enter	No action to be redone message	No action to be redone message
19. Test Undo Command (No action to be undone) Input u→press enter	No action to be undone message	No action to be undone message
20. Test List Command Input l→press enter	List all command executed	List all command executed

5.2 Screen dumps of result

5.2.1 Test Create Command (1)

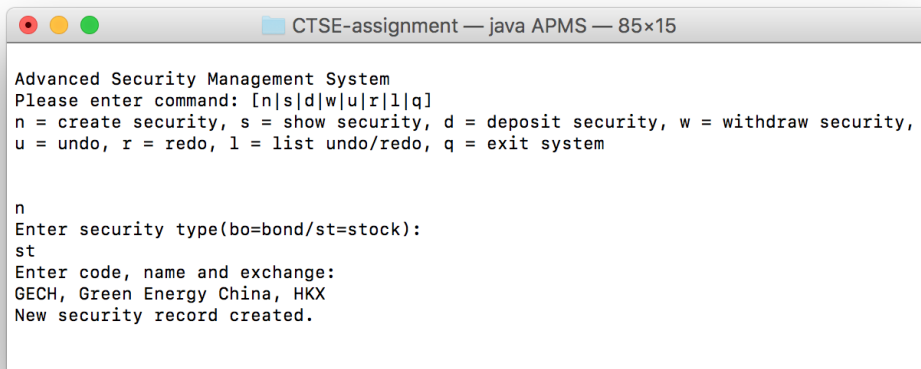


```
CTSE-assignment — java APMS — 85x15

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

n
Enter security type(bo=bond/st=stock):
bo
Enter code, name and yield:
TB2020, TBond 2020, 5.5
New security record created.
```

5.2.2 Test Create Command (2)

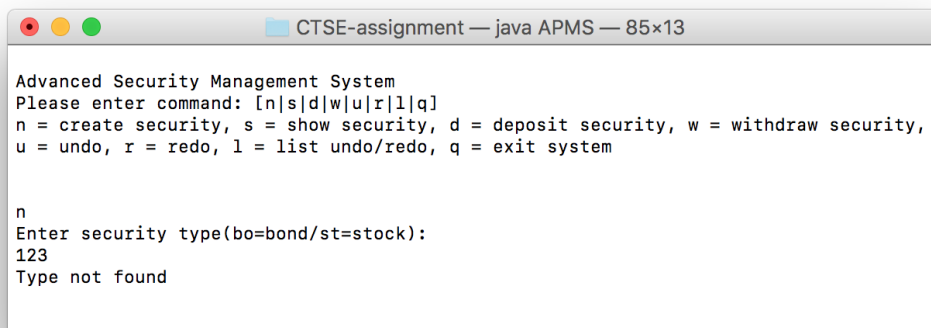


```
CTSE-assignment — java APMS — 85x15

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

n
Enter security type(bo=bond/st=stock):
st
Enter code, name and exchange:
GECH, Green Energy China, HKX
New security record created.
```

5.2.3 Test Create Command (3)

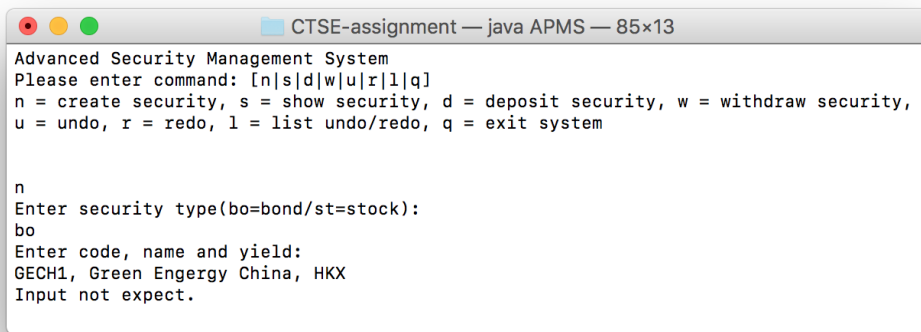


```
CTSE-assignment — java APMS — 85x13

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

n
Enter security type(bo=bond/st=stock):
123
Type not found
```

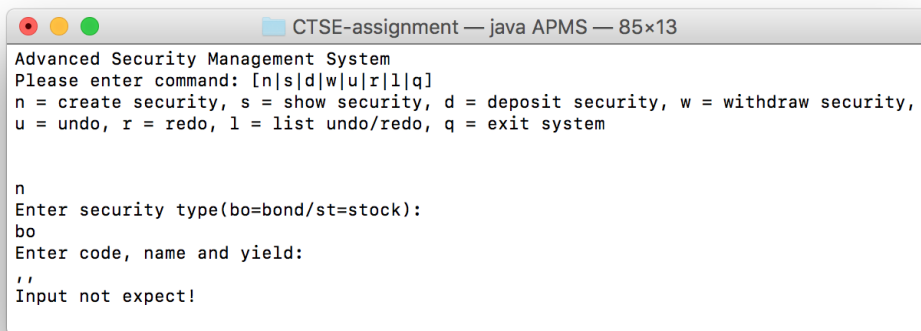

5.2.4 Test Create Command (4)



```
CTSE-assignment — java APMS — 85x13
Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

n
Enter security type(bo=bond/st=stock):
bo
Enter code, name and yield:
GECH1, Green Engergy China, HKX
Input not expect.
```

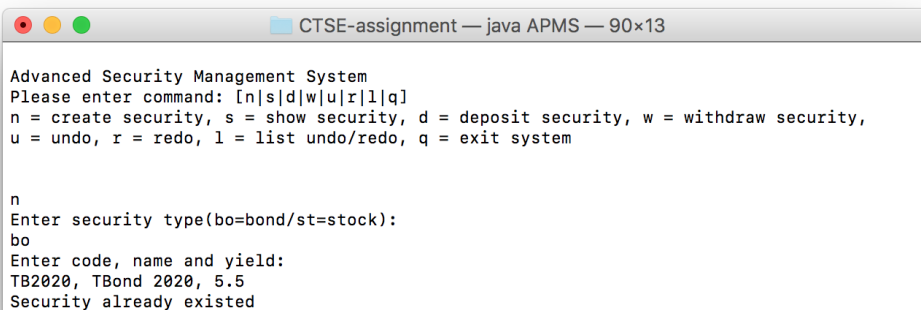
5.2.5 Test Create Command (5)



```
CTSE-assignment — java APMS — 85x13
Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

n
Enter security type(bo=bond/st=stock):
bo
Enter code, name and yield:
''
Input not expect!
```

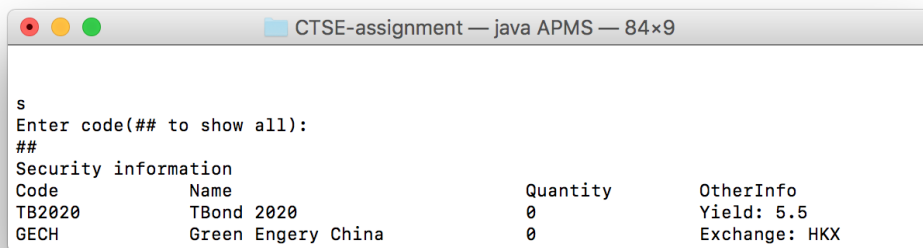
5.2.6 Test Create Command (6)



```
CTSE-assignment — java APMS — 90x13
Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

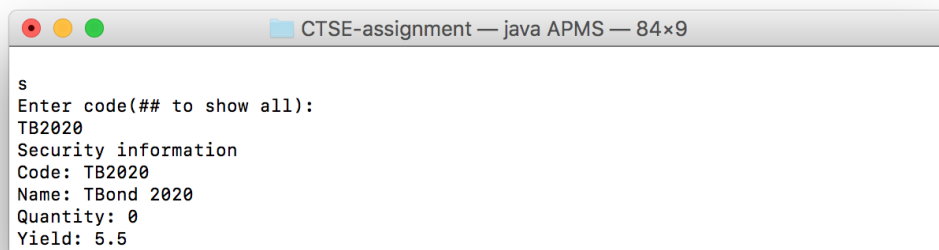
n
Enter security type(bo=bond/st=stock):
bo
Enter code, name and yield:
TB2020, TBond 2020, 5.5
Security already existed
```

5.2.7 Test show Command (1)



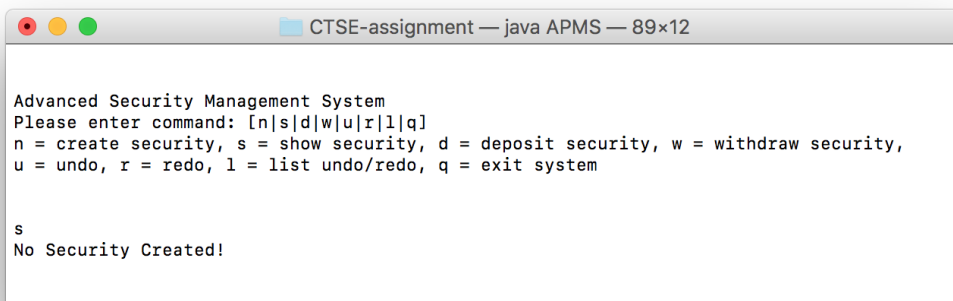
```
s
Enter code(## to show all):
##
Security information
Code      Name      Quantity  OtherInfo
TB2020    TBond 2020  0         Yield: 5.5
GECH      Green Engery China  0         Exchange: HKX
```

5.2.8 Test show Command (2)



```
s
Enter code(## to show all):
TB2020
Security information
Code: TB2020
Name: TBond 2020
Quantity: 0
Yield: 5.5
```

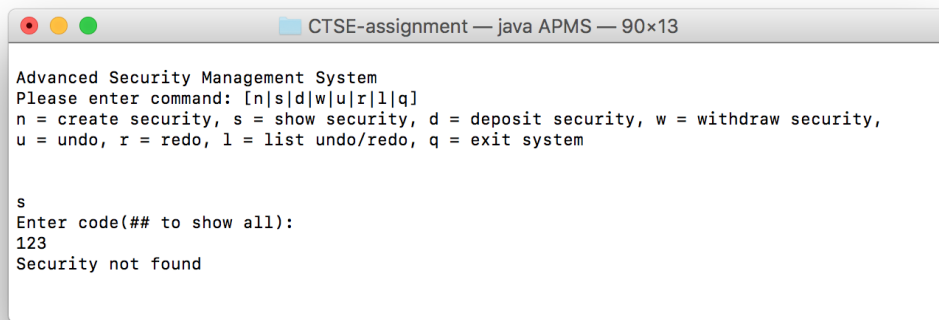
5.2.9 Test show Command (3)



```
Advanced Security Management System
Please enter command: [n|s|d|w|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

s
No Security Created!
```

5.2.10 Test show Command (3)

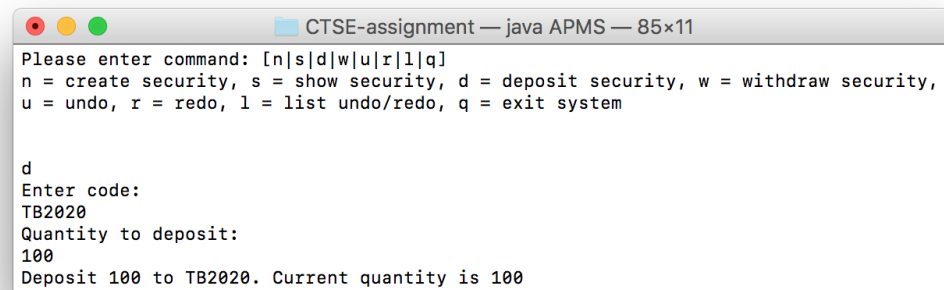


```
CTSE-assignment — java APMS — 90x13

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

s
Enter code(## to show all):
123
Security not found
```

5.2.11 Test Deposit Security Command (1)

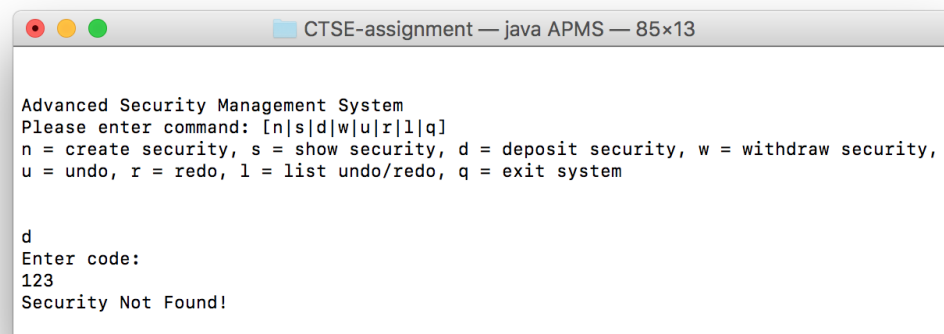


```
CTSE-assignment — java APMS — 85x11

Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

d
Enter code:
TB2020
Quantity to deposit:
100
Deposit 100 to TB2020. Current quantity is 100
```

5.2.12 Test Deposit Security Command (2)

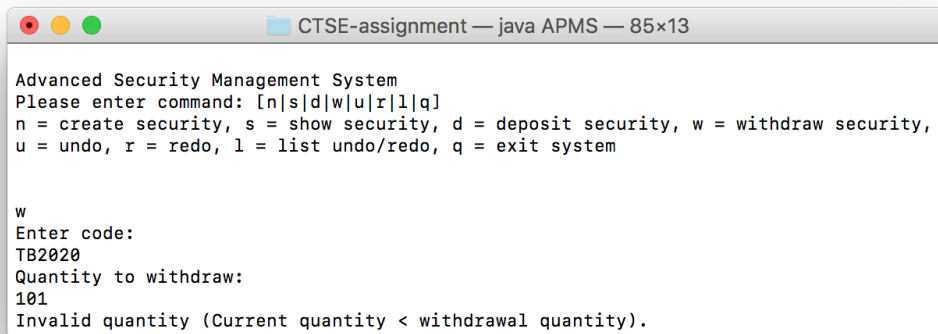


```
CTSE-assignment — java APMS — 85x13

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

d
Enter code:
123
Security Not Found!
```

5.2.13 Test Withdraw Security Command (1)

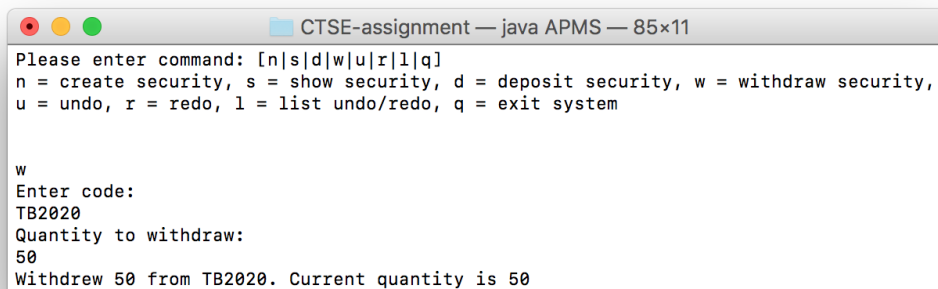


```
CTSE-assignment — java APMS — 85x13

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

w
Enter code:
TB2020
Quantity to withdraw:
101
Invalid quantity (Current quantity < withdrawal quantity).
```

5.2.14 Test Withdraw Security Command (2)

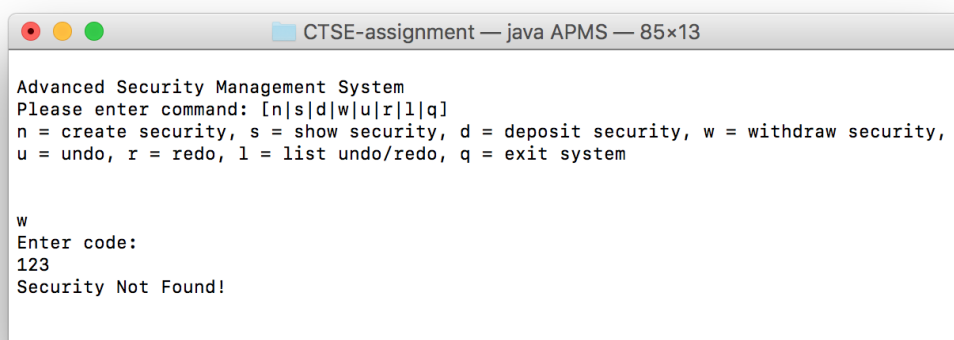


```
CTSE-assignment — java APMS — 85x11

Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

w
Enter code:
TB2020
Quantity to withdraw:
50
Withdrew 50 from TB2020. Current quantity is 50
```

5.2.15 Test Withdraw Security Command (3)

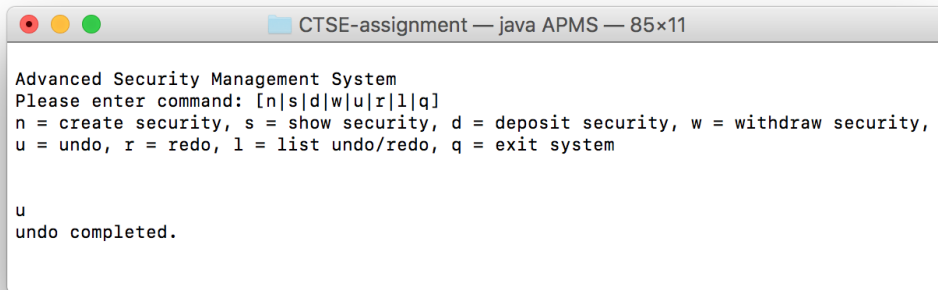


```
CTSE-assignment — java APMS — 85x13

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

w
Enter code:
123
Security Not Found!
```

5.2.16 Test Undo Command (1)

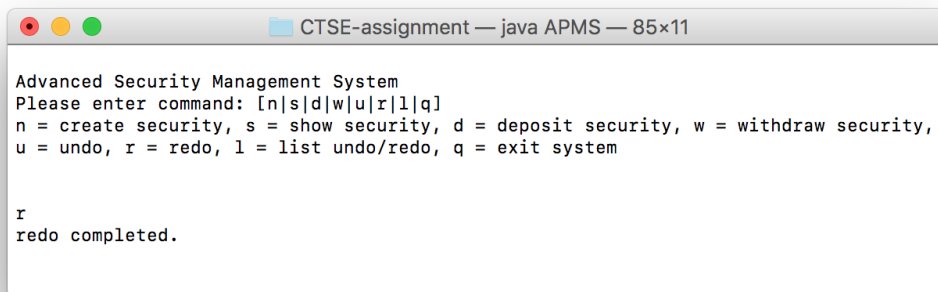


```
CTSE-assignment — java APMS — 85x11

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

u
undo completed.
```

5.2.17 Test Redo Command (1)

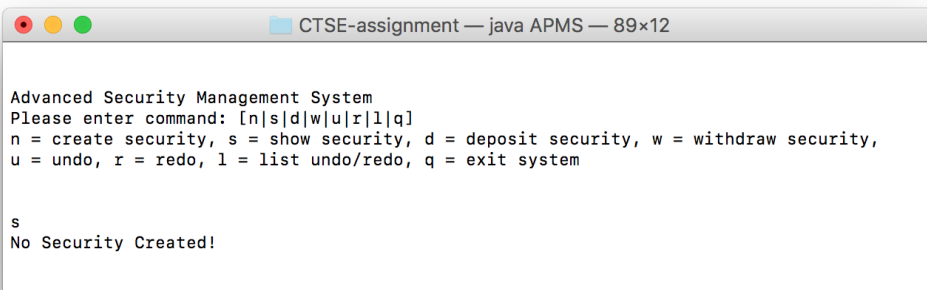


```
CTSE-assignment — java APMS — 85x11

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

r
redo completed.
```

5.2.18 Test Redo Command (2)

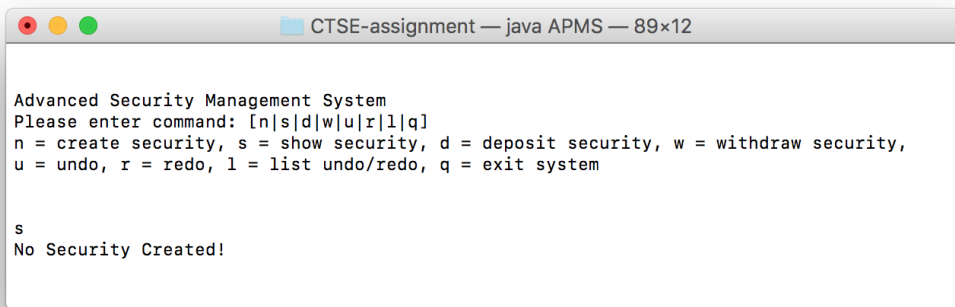


```
CTSE-assignment — java APMS — 89x12

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

s
No Security Created!
```

5.2.19 Test Undo Command (2)

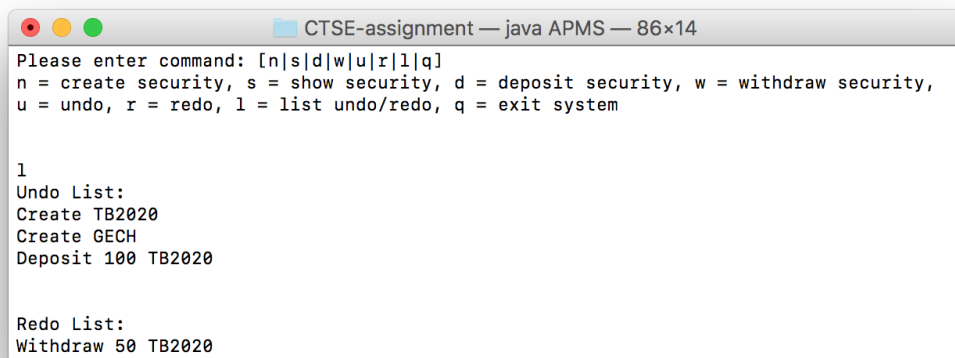


```
CTSE-assignment — java APMS — 89x12

Advanced Security Management System
Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

s
No Security Created!
```

5.2.20 Test List Command



```
CTSE-assignment — java APMS — 86x14

Please enter command: [n|s|d|w|u|r|l|q]
n = create security, s = show security, d = deposit security, w = withdraw security,
u = undo, r = redo, l = list undo/redo, q = exit system

l
Undo List:
Create TB2020
Create GECH
Deposit 100 TB2020

Redo List:
Withdraw 50 TB2020
```

6. Well documented Source Code

6.1 Main program (APMS.java)

```
import Portfolio.Security;

import java.util.ArrayList;

public class ShowSecurityCommand implements Command {

    private ArrayList<Security> securities;
    private String securityCode;

    public ShowSecurityCommand(ArrayList<Security> securities, String securityCode) {
        this.securities = securities;
        this.securityCode = securityCode;
    }

    public void execute() {
        if (!securities.isEmpty()) {
            //input ## to show all security created
            if (securityCode.equals("##")) {
                System.out.println("Security information");
                System.out.printf("%-15s %-30s %-15s %-20s\n", "Code", "Name",
"Quantity", "OtherInfo");
                for (Security security : securities) {
                    System.out.printf("%-15s %-30s %-15s %-20s\n",
                        security.getCode(), security.getName(),
security.getQuantity(), security);
                }
            } else {
                Security security = null;
                for (Security s : securities) {
                    if (s.getCode().equals(securityCode)) {
                        security = s;
                        break;
                    }
                }
                if (security != null) {
                    System.out.println("Security information");
                    System.out.println("Code: " + security.getCode()
                        + "\nName: " + security.getName()
                        + "\nQuantity: " + security.getQuantity()
                        + "\n" + security);
                } else {
                    System.out.println("Security not found");
                }
            }
        } else {
            System.out.println("No Security Created!");
        }
    }
}
```

6.2 Command interface (Command.java)

```
public interface Command {  
    void execute();  
}
```

6.3 Abstract Undoable Command (UndoableCommand.java)

```
public abstract class UndoableCommand implements Command {  
    protected Memento memento;  
    // when the command executed it will turn true  
    protected boolean executed;  
  
    public UndoableCommand() {  
        executed = false;  
    }  
  
    public abstract void execute();  
  
    public abstract void undo();  
  
    protected abstract Memento createMemento();  
  
    public boolean isExecuted() {  
        return executed;  
    }  
}
```

6.4 Abstract Command Factory (CommandCreator.java)

```
public abstract class CommandCreator {  
    public abstract Command createCommand();  
}
```

6.5 Abstract Security Factory (SecurityCreator.java)

```
import Portfolio.Security;  
//AbstractFactory  
public abstract class SecurityCreator {  
    public abstract Security createSecurity();  
  
    protected String spaceFilter(String s) {  
        if (s.charAt(0) == ' ') {  
            s = s.replaceFirst(" ", "");  
            return spaceFilter(s);  
        } else {  
            return s;  
        }  
    }  
}
```

6.6 Memento interface (Memento.java)

```
public interface Memento {  
    void restore();  
}
```


6.7 Bond version 2 (BondV2.java)

```
import Portfolio.Bond;

/**
 * This class is for providing show security(Other Info) function.
 * through override toString() method.
 */

public class BondV2 extends Bond {

    public BondV2(String code, String name, float yield) {
        super(code, name, yield);
    }

    @Override
    public String toString() {
        return "Yield: " + super.getYield();
    }
}
```

6.8 Stock version 2 (StockV2.java)

```
import Portfolio.Stock;

/**
 * This class is for providing show security(Other Info) function.
 * through override toString() method.
 */

public class StockV2 extends Stock {

    public StockV2(String code, String name, String exchange) {
        super(code, name, exchange);
    }

    @Override
    public String toString() {
        return "Exchange: " + super.getExchange();
    }
}
```

6.9 Stock Factory (StockCreator.java)

```
import Portfolio.Security;

import java.util.Scanner;

public class StockCreator extends SecurityCreator {

    public Security createSecurity() {
        Scanner kb = new Scanner(System.in);
        System.out.println("Enter code, name and exchange:");
        String input = kb.nextLine();
        String code;
        String name;
        String exchange;
        String[] result = input.split(",");
        if (result.length == 3) {
            code = result[0];
            name = spaceFilter(result[1]);
            exchange = spaceFilter(result[2]);
        } else {
            throw new RuntimeException("Input not expect!");
        }
        return new StockV2(code, name, exchange);
    }
}
```

6.10 Bond Factory (Bond Creator.java)

```
import Portfolio.Security;

import java.util.Scanner;
//ConcreteFactory
public class BondCreator extends SecurityCreator {
    @Override
    public Security createSecurity() {
        Scanner kb = new Scanner(System.in);
        System.out.println("Enter code, name and yield:");
        String input = kb.nextLine();
        String code;
        String name;
        float yield;
        String[] result = input.split(",");
        // the result format must equals 3
        if (result.length == 3) {
            code = result[0];
            name = spaceFilter(result[1]);
            yield = Float.parseFloat(result[2]);
        } else {
            throw new RuntimeException("Input not expect!");
        }
        return new BondV2(code, name, yield);
    }
}
```

6.11 Caretaker (Caretaker.java)

```
import java.util.Stack;

/**
 * Created by Mike on 2/11/2016.
 */
public class Caretaker {

    private Stack<UndoableCommand> undoList, redoList;

    public Caretaker() {
        undoList = new Stack();
        redoList = new Stack();
    }

    //this method for adding executed command
    public void addExecutedCommand(Command executedCommand) {
        if (executedCommand instanceof UndoableCommand) {
            if (((UndoableCommand) executedCommand).isExecuted()) {
                undoList.push((UndoableCommand) executedCommand);
                // when a command added the redoList must clear
                redoList.clear();
            }
        }
    }

    public UndoableCommand popUndoCommand() {
        if (!undoList.empty()) {
            //undo command at the same time push the command to redo.
            redoList.push(undoList.peek());
            return undoList.pop();
        } else {
            throw new NullPointerException("No Action to be undone");
        }
    }

    public UndoableCommand popRedoCommand() {
        if (!redoList.empty()) {
            //redo command at the same time push the command to undo.
            undoList.push(redoList.peek());
            return redoList.pop();
        } else {
            throw new NullPointerException("No Action to be redone");
        }
    }

    public Stack<UndoableCommand> getUndoList() {
        return undoList;
    }

    public Stack<UndoableCommand> getRedoList() {
        return redoList;
    }
}
```

6.12 Create Security Command (CreateSecurityCommand.java)

```
import Portfolio.Security;

import java.util.ArrayList;

public class CreateSecurityCommand extends UndoableCommand {

    private ArrayList<Security> securities;
    private Security security;
    private boolean added;

    public CreateSecurityCommand(ArrayList<Security> securities, Security security) {
        this.securities = securities;
        this.security = security;
        added = false;
    }

    @Override
    public void execute() {
        for (Security s : securities) {
            if (security.getCode().equals(s.getCode())) {
                System.out.println("Security already existed");
                return;
            }
        }
        securities.add(security);
        System.out.println("New security record created.");
        added = true;
        executed = true;
    }

    @Override
    public void undo() {
        if (added) {
            securities.remove(security);
            added = false;
        } else {
            securities.add(security);
            added = true;
        }
    }

    @Override
    protected Memento createMemento() {
        return null;
    }

    @Override
    public String toString() {
        return "Create " + security.getCode();
    }
}
```

6.13 Show Security Command (ShowSecurityCommand.java)

```
import Portfolio.Security;

import java.util.ArrayList;

public class ShowSecurityCommand implements Command {

    private ArrayList<Security> securities;
    private String securityCode;

    public ShowSecurityCommand(ArrayList<Security> securities, String securityCode) {
        this.securities = securities;
        this.securityCode = securityCode;
    }

    public void execute() {
        if (!securities.isEmpty()) {
            //input ## to show all security created
            if (securityCode.equals("##")) {
                System.out.println("Security information");
                System.out.printf("%-15s %-30s %-15s %-20s\n", "Code", "Name",
"Quantity", "OtherInfo");
                for (Security security : securities) {
                    System.out.printf("%-15s %-30s %-15s %-20s\n",
security.getCode(), security.getName(),
security.getQuantity(), security);
                }
            } else {
                Security security = null;
                for (Security s : securities) {
                    if (s.getCode().equals(securityCode)) {
                        security = s;
                        break;
                    }
                }
                if (security != null) {
                    System.out.println("Security information");
                    System.out.println("Code: " + security.getCode()
+ "\nName: " + security.getName()
+ "\nQuantity: " + security.getQuantity()
+ "\n" + security);
                } else {
                    System.out.println("Security not found");
                }
            }
        } else {
            System.out.println("No Security Created!");
        }
    }
}
```

6.14 Deposit Security Command (DepositSecurityCommand.java)

```
import Portfolio.Security;

public class DepositSecurityCommand extends UndoableCommand {

    private Security security;
    private int quantity;

    public DepositSecurityCommand(Security security, int quantity) {
        this.security = security;
        this.quantity = quantity;
    }

    @Override
    public void execute() {
        memento = createMemento();
        security.setQuantity(security.getQuantity() + quantity);
        System.out.println("Deposit " + quantity + " to " + security.getCode()
            + ". Current quantity is " + security.getQuantity());
        executed = true;
    }

    @Override
    public void undo() {
        //save the state before undo and redo
        Memento temp = createMemento();
        memento.restore();
        memento = temp;
    }

    @Override
    protected Memento createMemento() {
        return new SecurityQuantityMemento(security);
    }

    @Override
    public String toString() {
        return "Deposit " + quantity + " " + security.getCode();
    }
}
```

6.15 Withdraw Security Command (WithdrawSecurityCommand.java)

```
import Portfolio.Security;

public class WithdrawSecurityCommand extends UndoableCommand {

    private Security security;
    private int quantity;

    public WithdrawSecurityCommand(Security security, int quantity) {
        this.security = security;
        this.quantity = quantity;
    }

    public void execute() {
        memento = createMemento();
        security.setQuantity(security.getQuantity() - quantity);
        System.out.println("Withdrew " + quantity + " from " + security.getCode()
            + ". Current quantity is " + security.getQuantity());
        executed = true;
    }

    @Override
    public void undo() {
        //save the state before undo and redo
        Memento temp = createMemento();
        memento.restore();
        memento = temp;
    }

    @Override
    protected Memento createMemento() {
        return new SecurityQuantityMemento(security);
    }

    @Override
    public String toString() {
        return "Withdraw " + quantity + " " + security.getCode();
    }
}
```

6.16 Undo Command (UndoCommand.java)

```
public class UndoCommand implements Command {
    private UndoableCommand command;

    public UndoCommand(UndoableCommand command) {
        this.command = command;
    }

    @Override
    public void execute() {
        command.undo();
        System.out.println("undo completed.");
    }
}
```

6.17 Redo Command (RedoCommand.java)

```
public class RedoCommand implements Command {
    private UndoableCommand command;

    public RedoCommand(UndoableCommand command) {
        this.command = command;
    }

    @Override
    public void execute() {
        command.undo();
        System.out.println("redo completed.");
    }
}
```


6.18 List Command (ListCommand.java)

```
import java.util.ArrayList;

/**
 * Created by Xuan on 3/11/2016.
 */
public class ListCommand implements Command {
    private ArrayList<UndoableCommand> undoList, redoList;

    public ListCommand(ArrayList<UndoableCommand> undoList,
        ArrayList<UndoableCommand> redoList) {
        this.redoList = redoList;
        this.undoList = undoList;
    }

    @Override
    public void execute() {
        System.out.println("Undo List:");
        if (!undoList.isEmpty()) {
            for (UndoableCommand command : undoList) {
                System.out.println(command);
            }
        } else {
            System.out.println("Empty");
        }
        System.out.println("\n\nRedo List:");
        if (!redoList.isEmpty()) {
            for (UndoableCommand command : redoList) {
                System.out.println(command);
            }
        } else {
            System.out.println("Empty");
        }
    }
}
```

6.19 Create Security Command Factory(CreateSecurityCommandCreator.java)

```
import Portfolio.Security;

import java.util.ArrayList;
import java.util.Scanner;
//ConcreteFactory
public class CreateSecurityCommandCreator extends CommandCreator {

    private ArrayList<Security> securities;
    SecurityCreator sc;

    public CreateSecurityCommandCreator(ArrayList<Security> securities) {
        this.securities = securities;
    }

    @Override
    public Command createCommand() {
        System.out.println("Enter security type(bo=bond/st=stock):");
        Scanner kb = new Scanner(System.in);
        String type = kb.nextLine();
        switch (type) {
            case "bo":
                sc = new BondCreator();
                break;
            case "st":
                sc = new StockCreator();
                break;
            default:
                throw new RuntimeException("Type not found");
        }
        Security security = sc.createSecurity();
        return new CreateSecurityCommand(securities, security);
    }
}
```

6.20 Show Security Command Factory (ShowSecurityCommandCreator.java)

```
import Portfolio.Security;

import java.util.ArrayList;
import java.util.Scanner;

public class ShowSecurityCommandCreator extends CommandCreator {

    private ArrayList<Security> securities;

    public ShowSecurityCommandCreator(ArrayList<Security> securities) {
        this.securities = securities;
    }

    @Override
    public Command createCommand() {
        if (!securities.isEmpty()) {
            System.out.println("Enter code(## to show all):");
            Scanner kb = new Scanner(System.in);
            String code = kb.nextLine();
            return new ShowSecurityCommand(securities, code);
        } else {
            throw new RuntimeException("No Security Created!");
        }
    }
}
```

6.21 Security Quantity Memento (SecurityQuantityMemento.java)

```
import Portfolio.Security;

public class SecurityQuantityMemento implements Memento {
    private Security security;
    // state
    private int quantityState;

    public SecurityQuantityMemento(Security security) {
        this.security = security;
        quantityState = security.getQuantity();
    }

    @Override
    public void restore() {
        security.setQuantity(quantityState);
    }
}
```

6.22 Deposit Security Command Factory (DepositSecurityCommandCreator.java)

```
import Portfolio.Security;

import java.util.ArrayList;
import java.util.Scanner;
//ConcreteFactory
public class DepositSecurityCommandCreator extends CommandCreator {

    private ArrayList<Security> securities;

    public DepositSecurityCommandCreator(ArrayList securities) {
        this.securities = securities;
    }

    @Override
    public Command createCommand() {
        Scanner kb = new Scanner(System.in);
        Security security = null;
        boolean isFound = false;
        System.out.println("Enter code:");
        String code = kb.nextLine();
        for (Security s : securities) {
            if (s.getCode().equals(code)) {
                security = s;
                isFound = true;
                break;
            }
        }
        if (!isFound) {
            throw new RuntimeException("Security Not Found!");
        }
        System.out.println("Quantity to deposit:");
        int quantity = kb.nextInt();
        return new DepositSecurityCommand(security, quantity);
    }
}
```

6.23 Withdraw Security Command Factory (WithdrawSecurityCommandCreator.java)

```
import Portfolio.Security;

import java.util.ArrayList;
import java.util.Scanner;

public class WithdrawSecurityCommandCreator extends CommandCreator {

    private ArrayList<Security> securities;

    public WithdrawSecurityCommandCreator(ArrayList securities) {
        this.securities = securities;
    }

    public Command createCommand() {
        Scanner kb = new Scanner(System.in);
        Security security = null;
        boolean isFound = false;
        System.out.println("Enter code:");
        String code = kb.nextLine();
        for (Security s : securities) {
            if (s.getCode().equals(code)) {
                security = s;
                isFound = true;
                break;
            }
        }
        if (!isFound) {
            throw new RuntimeException("Security Not Found!");
        }
        System.out.println("Quantity to withdraw:");
        int quantity = kb.nextInt();
        if ((security.getQuantity() - quantity) < 0)
            throw new RuntimeException("Invalid quantity (Current quantity < withdrawal quantity).");
        return new WithdrawSecurityCommand(security, quantity);
    }
}
```

6.24 Undo Command Factory (UndoCommandCreator.java)

```
public class UndoCommandCreator extends CommandCreator {
    private Caretaker caretaker;

    public UndoCommandCreator(Caretaker caretaker) {
        this.caretaker = caretaker;
    }

    @Override
    public Command createCommand() {
        return new UndoCommand(caretaker.popUndoCommand());
    }
}
```

6.25 Redo Command Factory (UndoCommandCreator.java)

```
public class RedoCommandCreator extends CommandCreator {
    private Caretaker caretaker;

    public RedoCommandCreator(Caretaker caretaker) {
        this.caretaker = caretaker;
    }

    @Override
    public Command createCommand() {
        return new RedoCommand(caretaker.popRedoCommand());
    }
}
```

6.26 List Command Factory (ListCommandCreator.java)

```
public class ListCommandCreator extends CommandCreator {
    private Caretaker caretaker;

    public ListCommandCreator(Caretaker caretaker) {
        this.caretaker = caretaker;
    }

    @Override
    public Command createCommand() {
        return new ListCommand(caretaker.getUndoList(), caretaker.getRedoList());
    }
}
```