

제어 구조 1. 선택

Control Structure : Selection

도경구

한양대학교 ERICA 소프트웨어학부



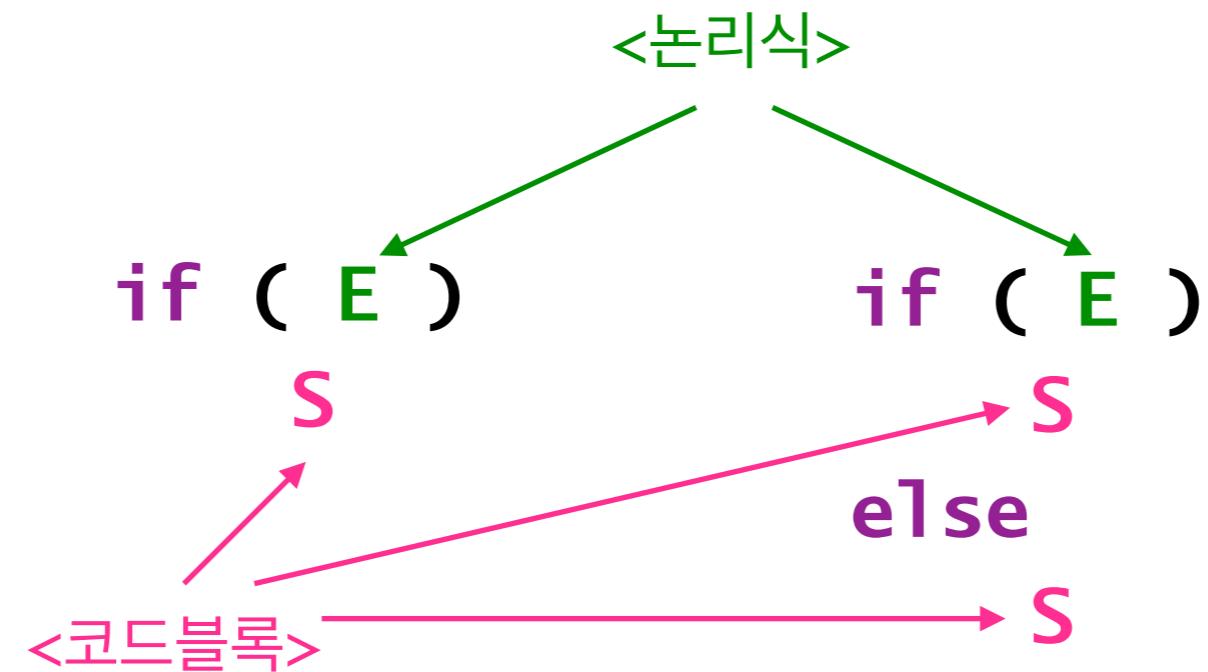
제어 구조

Control Structure

프로그램 실행 순서를 나타내주는 구조

- 기술한 순서대로 실행
- 메소드 호출

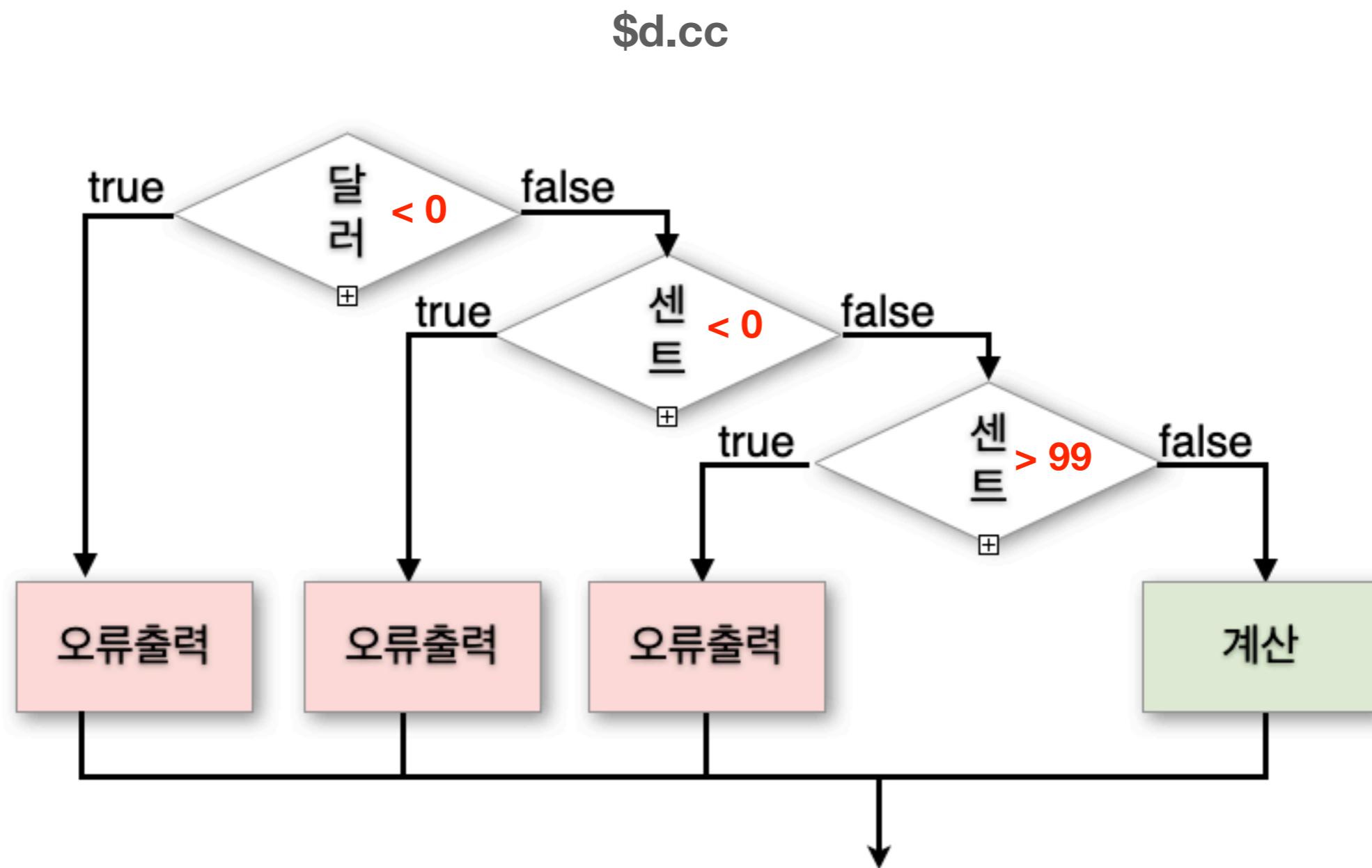
선택 구조



선택 적용 사례

```
1 import javax.swing.JOptionPane;
2
3 public class Conditional {
4
5     public static void main(String[] args) {
6         String input = JOptionPane.showInputDialog("나이를 알려주세요.");
7         int age = Integer.parseInt(input);
8         if (age < 19)
9             System.out.println(age + "세는 미성년입니다.");
10        else
11            System.out.println(age + "세는 성년입니다.");
12
13        int n = Integer.parseInt(JOptionPane.showInputDialog("역수를 계산해드립니다."));
14        if (n == 0)
15            System.out.println("0은 역수가 없습니다.");
16        else
17            System.out.println(n + "의 역수는 " + 1.0/n + " 입니다.");
18    }
19
20 }
21 }
```

동전 거슬러주기 문제



중첩 선택

```

1 public class MakeChange {
2
3     public static void main(String[] args) {
4         int dollars = Integer.parseInt(args[0]);
5         int cents = Integer.parseInt(args[1]);
6         if (dollars < 0) {
7             System.out.println("Error: negative dollars: " + dollars);
8         }
9         else {
10            if (cents < 0) {
11                System.out.println("Error: negative cents: " + cents);
12            }
13            else {
14                if (cents > 99) {
15                    System.out.println("Error: bad cents: " + cents);
16                }
17                else {
18                    int money = dollars * 100 + cents;
19                    System.out.println("quarters = " + (money / 25));
20                    money = money % 25;
21                    System.out.println("dimes = " + (money / 10));
22                    money = money % 10;
23                    System.out.println("nickels = " + (money / 5));
24                    money = money % 5;
25                    System.out.println("pennies = " + money);
26                }
27            }
28        }
29    }
30 }
```

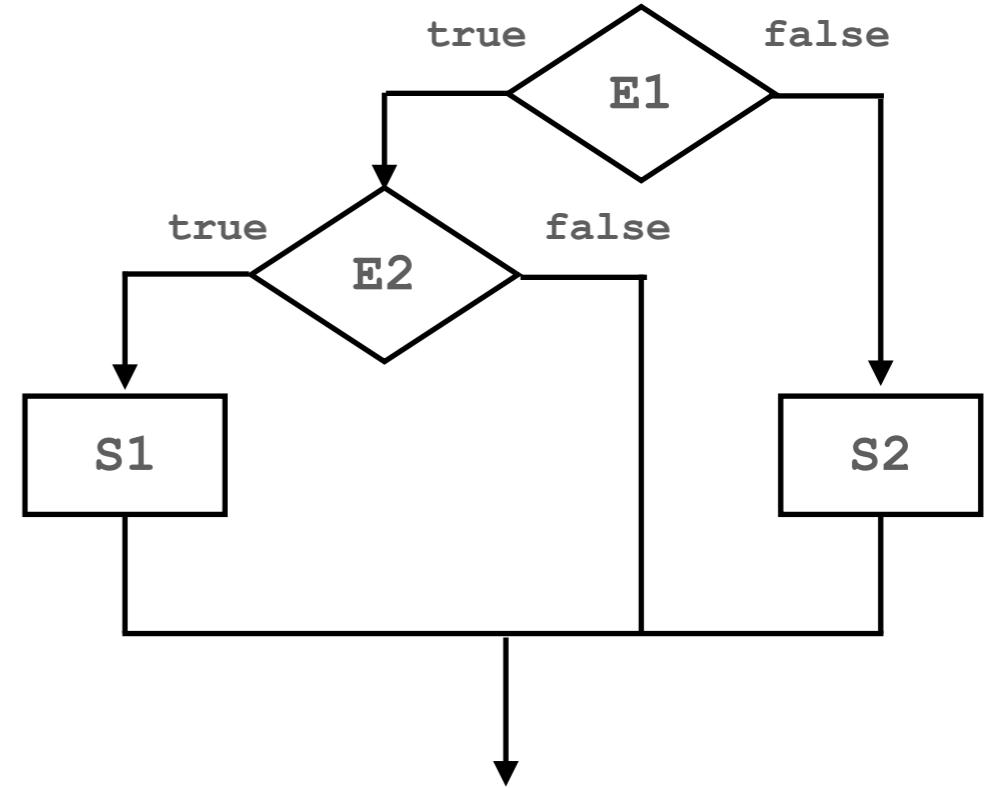
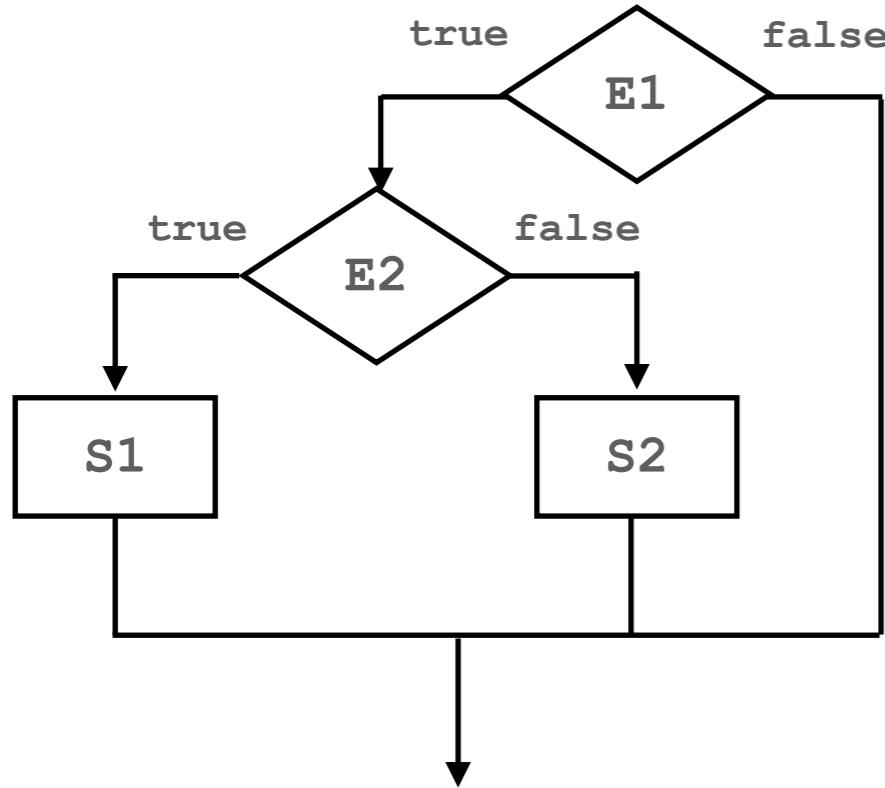
중첩 선택 - 펼치기

```
1 public class MakeChange {  
2  
3     public static void main(String[] args) {  
4         int dollars = Integer.parseInt(args[0]);  
5         int cents = Integer.parseInt(args[1]);  
6         if (dollars < 0)  
7             System.out.println("Error: negative dollars: " + dollars);  
8         else if (cents < 0)  
9             System.out.println("Error: negative cents: " + cents);  
10        else if (cents > 99)  
11            System.out.println("Error: bad cents: " + cents);  
12        else {  
13            int money = dollars * 100 + cents;  
14            System.out.println("quarters = " + (money / 25));  
15            money = money % 25;  
16            System.out.println("dimes = " + (money / 10));  
17            money = money % 10;  
18            System.out.println("nickels = " + (money / 5));  
19            money = money % 5;  
20            System.out.println("pennies = " + money);  
21        }  
22    }  
23 }
```

Dangling Else

```
if ( E1 )
if ( E2 )
S1
else S2
```

which?



논리 연산

연산	이름	의미
E1 && E2	논리곱 conjunction and	$\text{true} \quad \&\& \quad \text{true} \Rightarrow \text{true}$ $\text{false} \quad \&\& \quad \text{E2} \Rightarrow \text{false}$ $\text{E1} \quad \&\& \quad \text{false} \Rightarrow \text{false}$
E1 E2	논리합 disjunction or	$\text{false} \quad \mid\mid \quad \text{false} \Rightarrow \text{false}$ $\text{true} \quad \mid\mid \quad \text{E2} \Rightarrow \text{true}$ $\text{E1} \quad \mid\mid \quad \text{true} \Rightarrow \text{true}$
! E	논리역 negation not	$!\text{true} \Rightarrow \text{false}$ $!\text{false} \Rightarrow \text{true}$

단축 계산

Short-circuit Evaluation

논리 연산 적용 사례

```
1 public class MakeChange {  
2  
3     public static void main(String[] args) {  
4         int dollars = Integer.parseInt(args[0]);  
5         int cents = Integer.parseInt(args[1]);  
6         if (dollars < 0 || cents < 0 || cents > 99) {  
7             System.out.println("Error: bad input: " + dollars + " " + cents);  
8         }  
9         else {  
10             int money = dollars * 100 + cents;  
11             System.out.println("quarters = " + (money / 25));  
12             money = money % 25;  
13             System.out.println("dimes = " + (money / 10));  
14             money = money % 10;  
15             System.out.println("nickels = " + (money / 5));  
16             money = money % 5;  
17             System.out.println("pennies = " + money);  
18         }  
19     }  
20 }
```

예외를 발생시켜 오류처리하기

```
1 public class MakeChange {  
2  
3     public static void main(String[] args) {  
4         int dollars = Integer.parseInt(args[0]);  
5         int cents = Integer.parseInt(args[1]);  
6         if (dollars < 0 || cents < 0 || cents > 99) {  
7             throw new RuntimeException("Error: bad input: " + dollars + " " + cents);  
8         }  
9     else {  
10         int money = dollars * 100 + cents;  
11         System.out.println("quarters = " + (money / 25));  
12         money = money % 25;  
13         System.out.println("dimes = " + (money / 10));  
14         money = money % 10;  
15         System.out.println("nickels = " + (money / 5));  
16         money = money % 5;  
17         System.out.println("pennies = " + money);  
18     }  
19 }  
20 }
```

System.exit(0);

```
1 public class MakeChange {  
2  
3     public static void main(String[] args) {  
4         int dollars = Integer.parseInt(args[0]);  
5         int cents = Integer.parseInt(args[1]);  
6         if (dollars < 0 || cents < 0 || cents > 99) {  
7             System.exit(0); ← 프로그램 실행 끝!  
8         }  
9     else {  
10         int money = dollars * 100 + cents;  
11         System.out.println("quarters = " + (money / 25));  
12         money = money % 25;  
13         System.out.println("dimes = " + (money / 10));  
14         money = money % 10;  
15         System.out.println("nickels = " + (money / 5));  
16         money = money % 5;  
17         System.out.println("pennies = " + money);  
18     }  
19 }  
20 }
```

switch

```

if (i == 2)
    System.out.println("2");
else {
    if (i == 5) {
        System.out.println("5");
        j = j + 1;
    }
    else {
        if (i == 7) {
            System.out.println("7");
            j = j - 1;
        }
        else
            System.out.println("none");
    }
}

```

```

switch (i) {
    case 2: {
        System.out.println("2");
        break;
    }
    case 5: {
        System.out.println("5");
        j = j + 1;
        break;
    }
    case 7: {
        System.out.println("7");
        j = j - 1;
        break;
    }
    default:
        System.out.println("none");
}

```

switch

```

switch ( EXPRESSION ) {
    case VALUE : {
        STATEMENTS
        break;
    }
    case VALUE : {
        STATEMENTS
        break;
    }
    /* ... */
    case VALUE : {
        STATEMENTS
        break;
    }
    default:
        STATEMENTS
}

```

매우 제한적!

값만 사용 가능

break를 쓰지 않으면
끝으로 가지 않고
다음 **case**로 넘어감

권장하지 않음!!!!

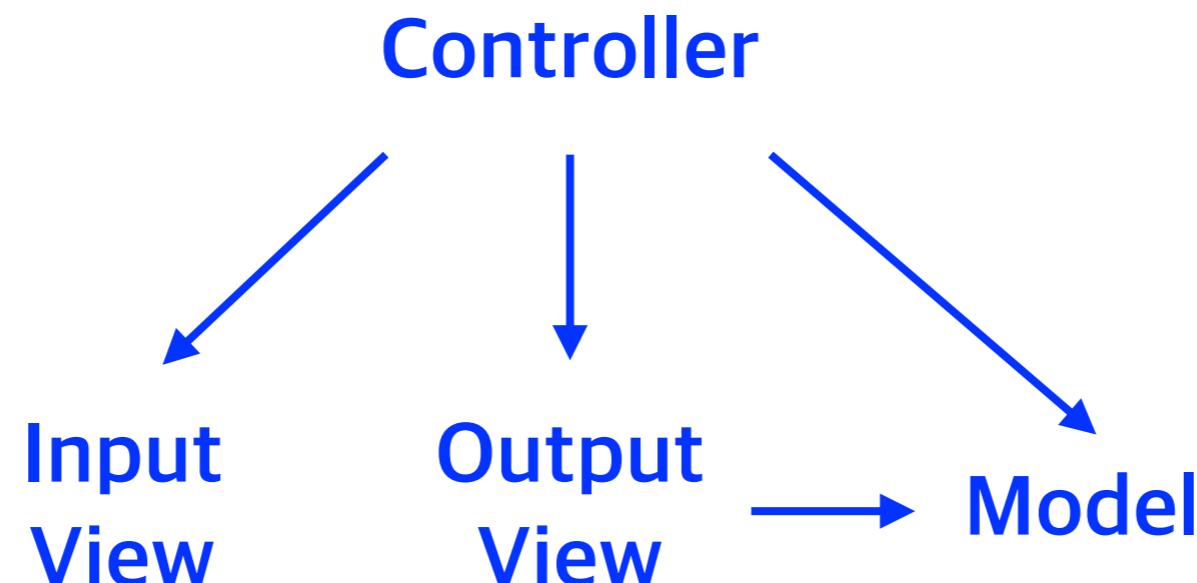
switch vs. if ... else if ... else

```
switch (i) {  
    case 2: {  
        System.out.println("2");  
        break;  
    }  
    case 5: {  
        System.out.println("5");  
        j = j + 1;  
        break;  
    }  
    case 7: {  
        System.out.println("7");  
        j = j - 1;  
        break;  
    }  
    default:  
        System.out.println("none");  
}
```

```
if (i == 2)  
    System.out.println("2");  
else if (i == 5) {  
    System.out.println("5");  
    j = j + 1;  
}  
else if (i == 7) {  
    System.out.println("7");  
    j = j - 1;  
}  
else  
    System.out.println("none");
```

MVC 아키텍처

전형 패턴



재사용
조립식
교체용이

애플리케이션 설계+구현 절차

[Step 1] 유즈케이스(use-case) 수집 => View(사용자 인터페이스) 설계

[Step 2] 유즈케이스를 실현할 소프트웨어 아키텍처 선정

[Step 3] 아키텍처 부품별로 클래스 명세 작성

[Step 4] 클래스 별로 코드 작성 및 테스트

[Step 5] 통합 테스트

사례 학습

은행 계좌 관리 애플리케이션

고객 계좌의 입출금을 처리하는 은행계좌 관리 애플리케이션

입금 (deposit)

출금 (withdraw)

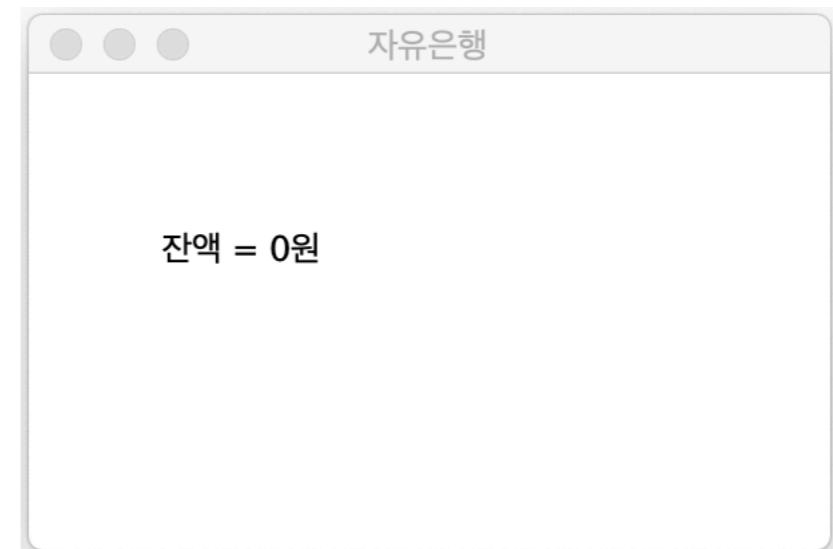
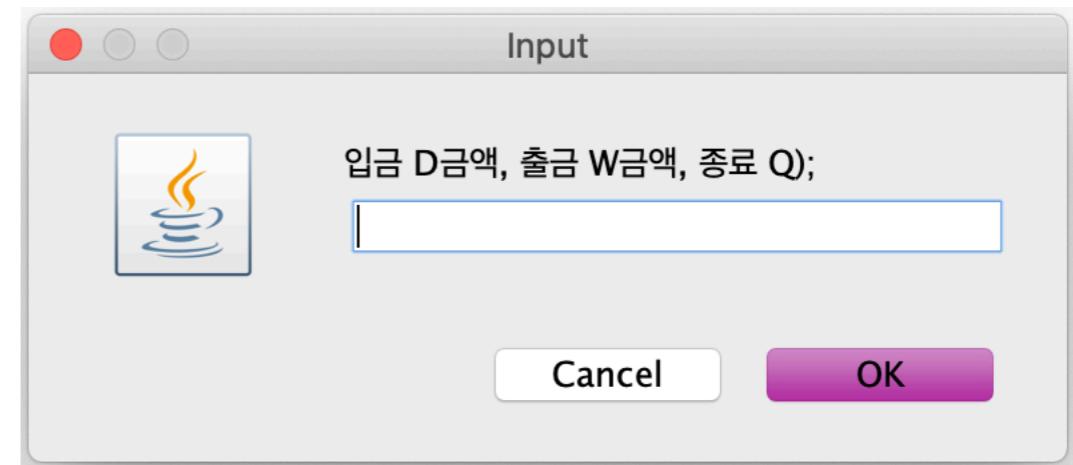
[1 단계] 유즈케이스 수집

- 사용자 입력 (d : 숫자)

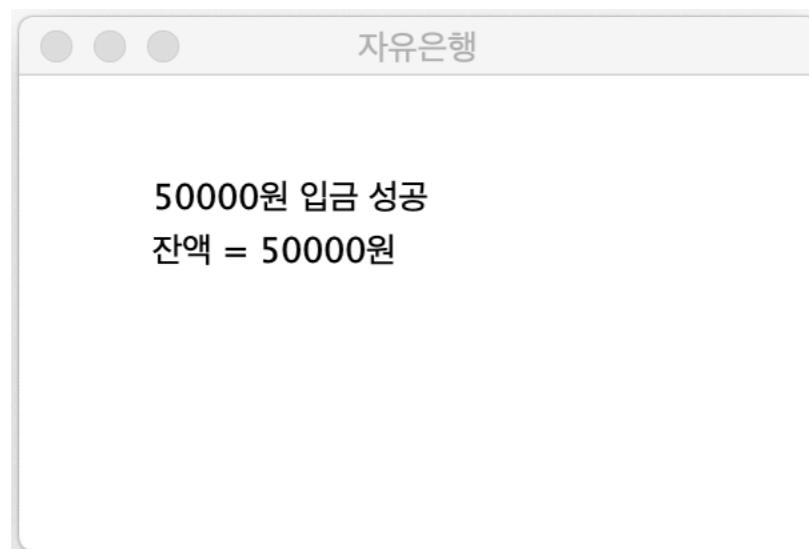
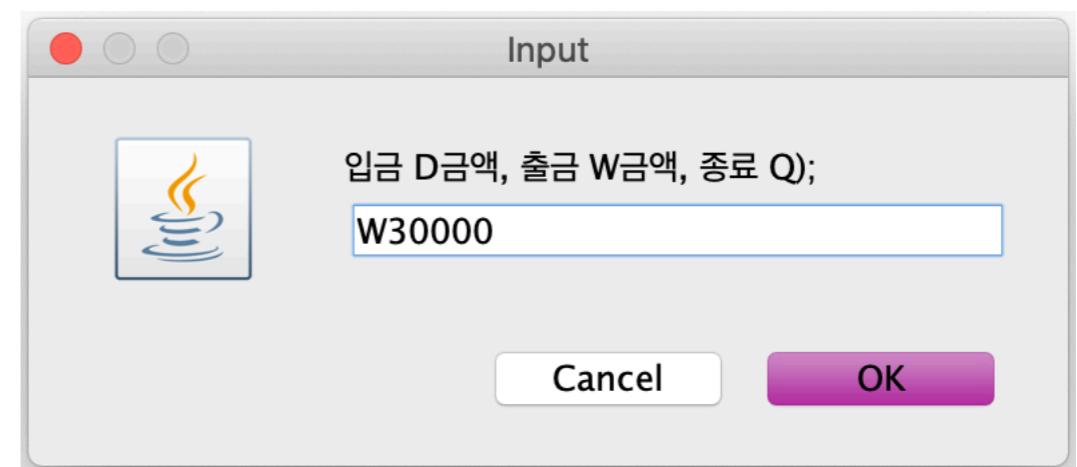
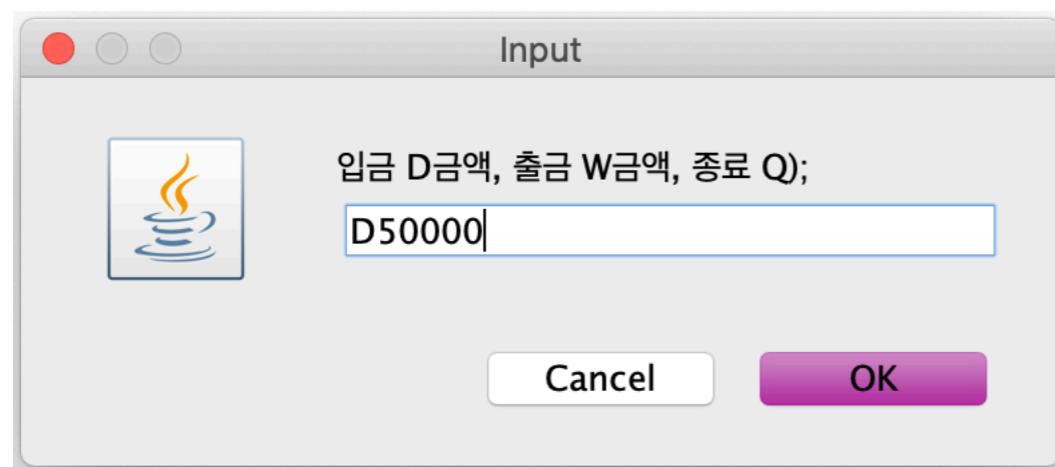
- 입금 (Deposit) : Dddd...d

- 출금 (Withdraw) : Wddd...d

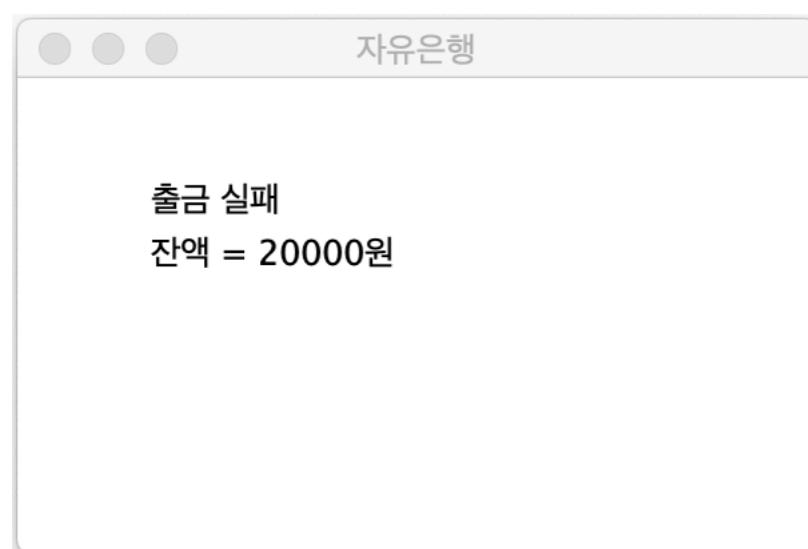
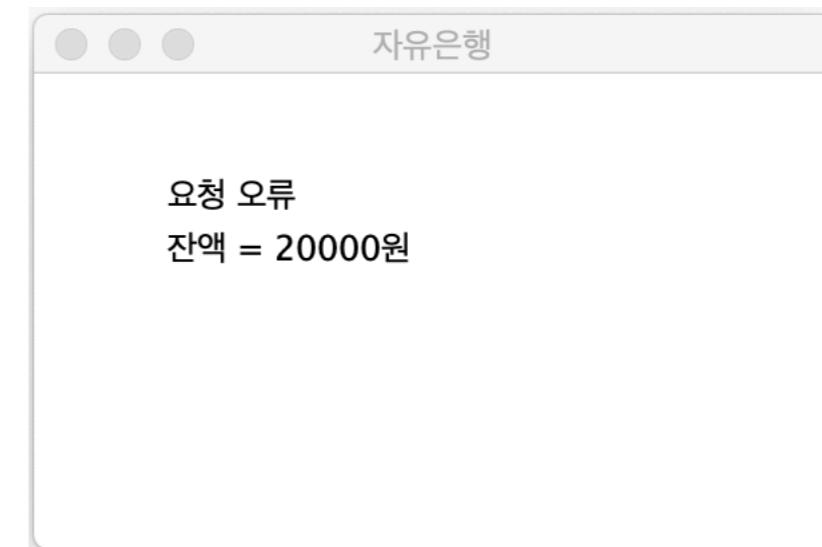
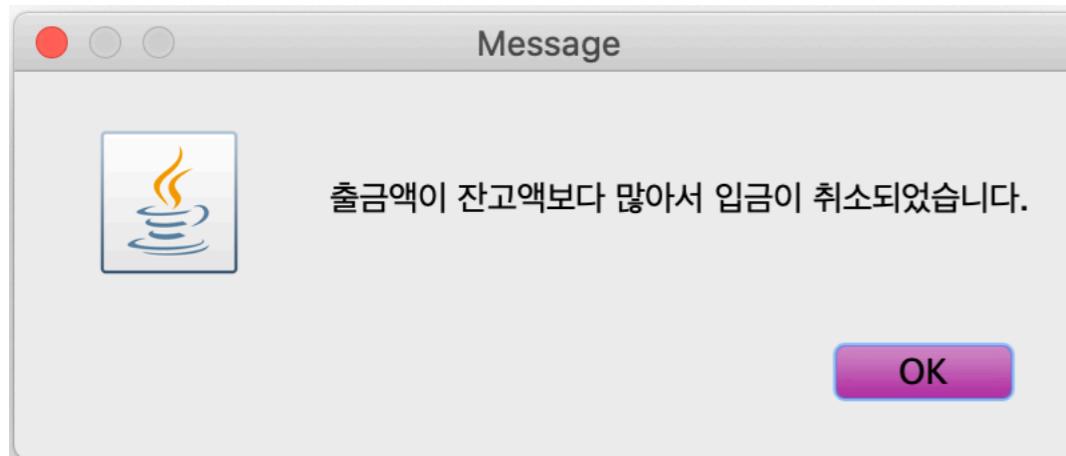
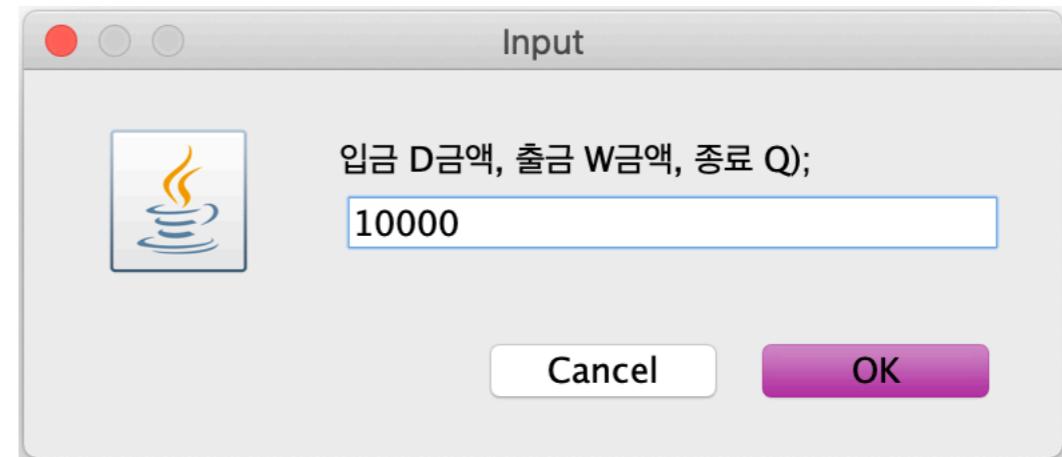
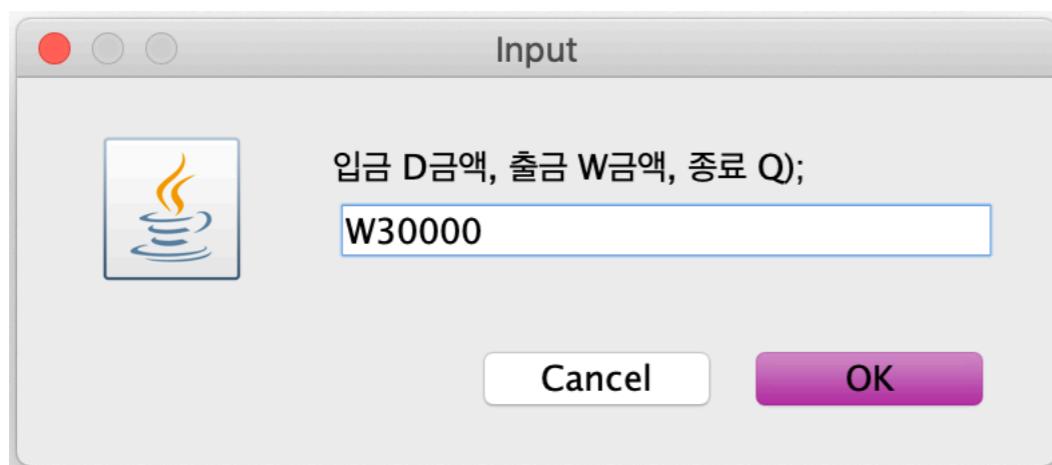
- 종료 (Quit) : Q



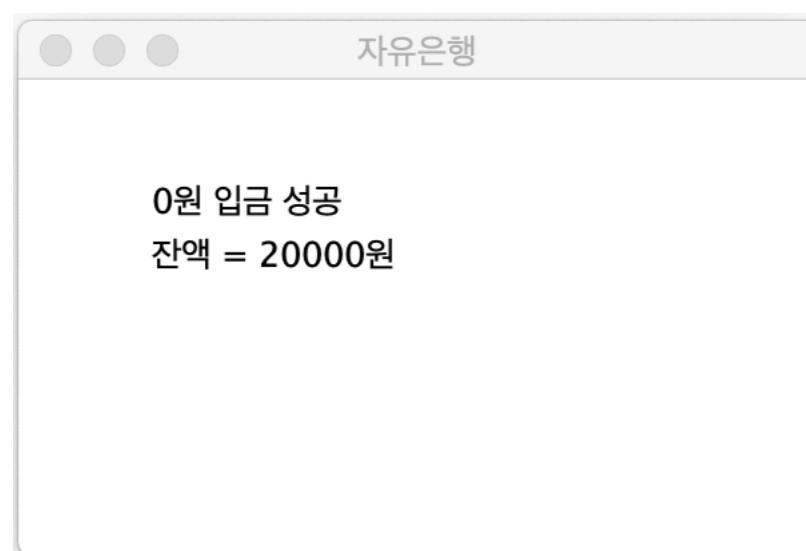
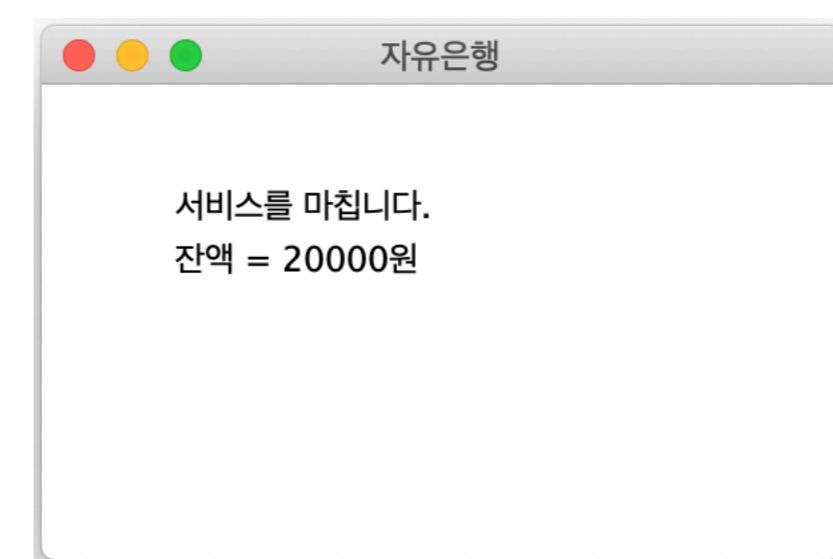
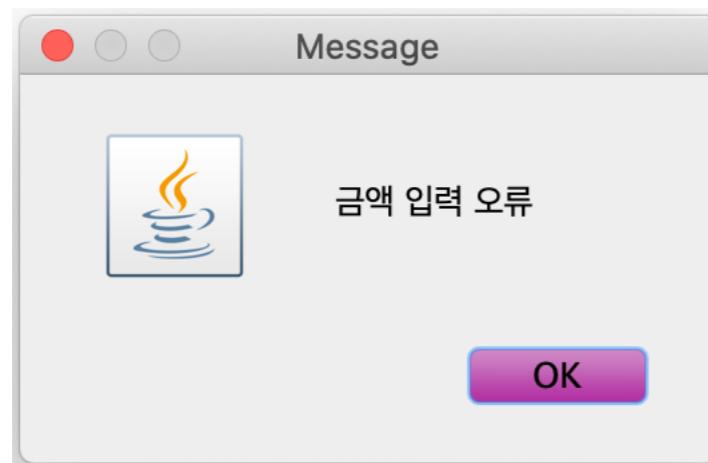
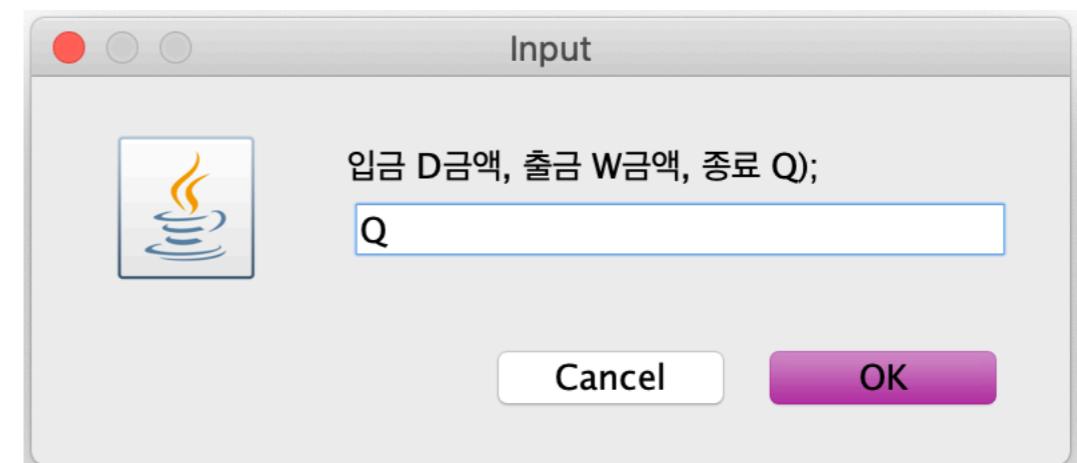
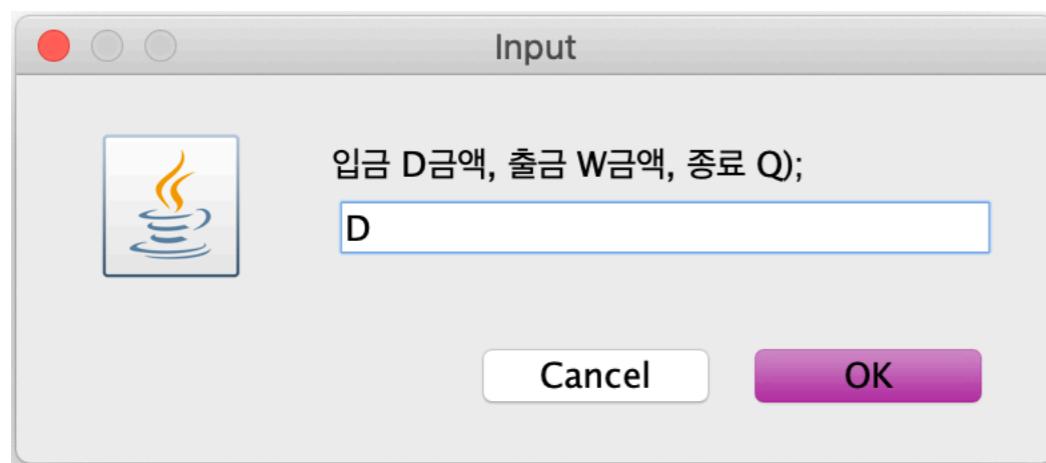
제어 구조 1. 선택



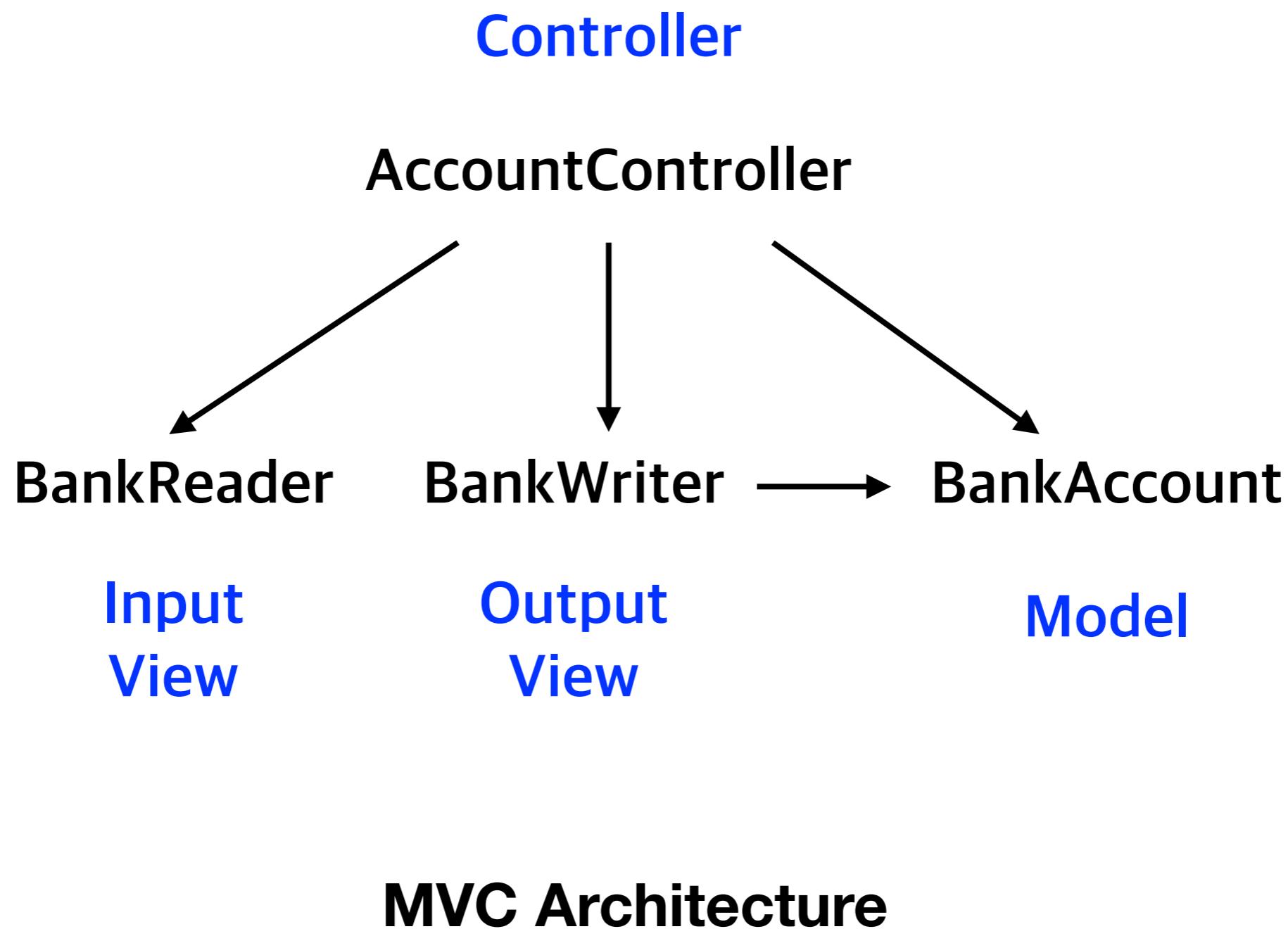
제어 구조 1. 선택



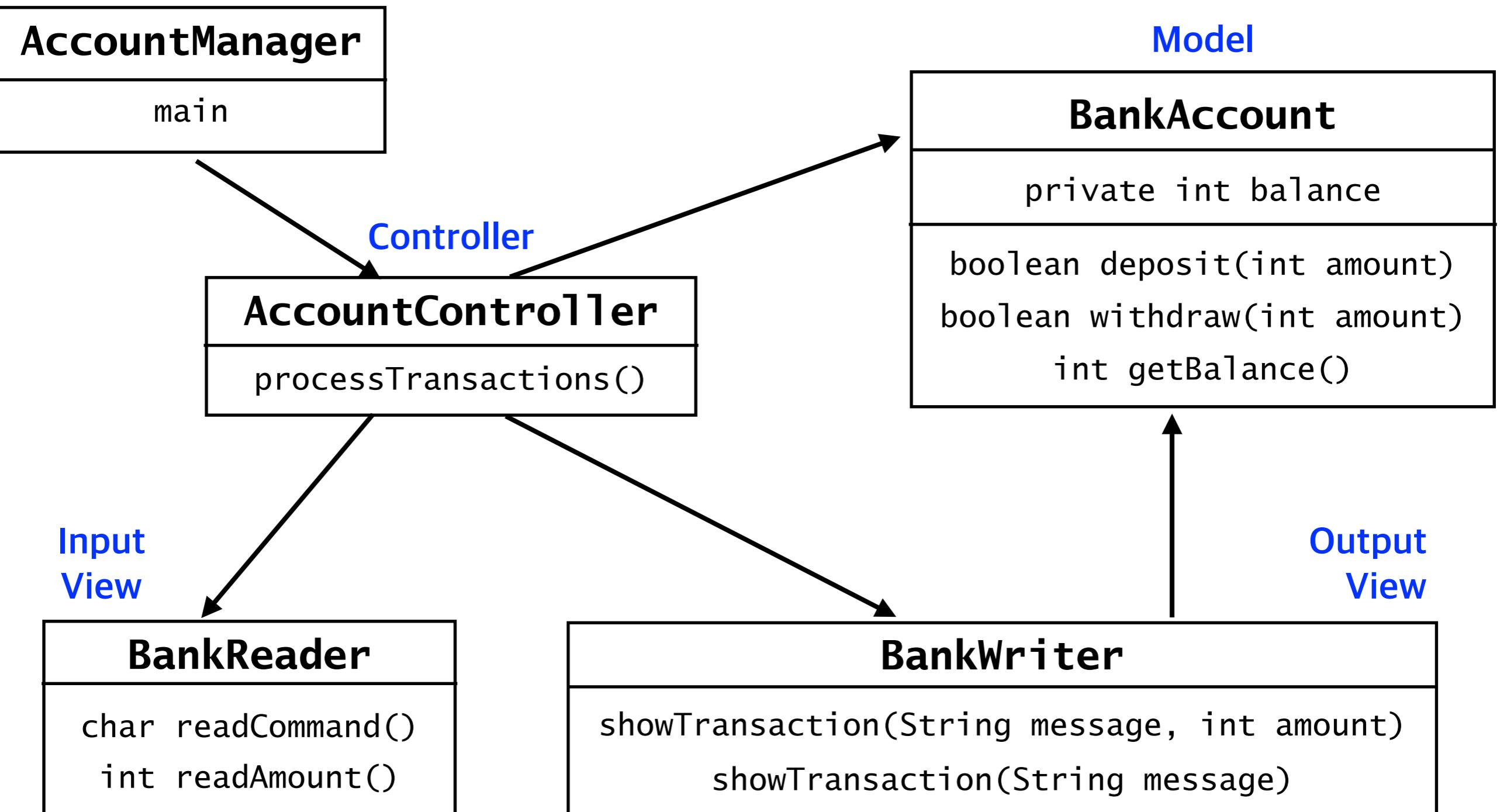
제어 구조 1. 선택



[2 단계] 소프트웨어 아키텍처 선정



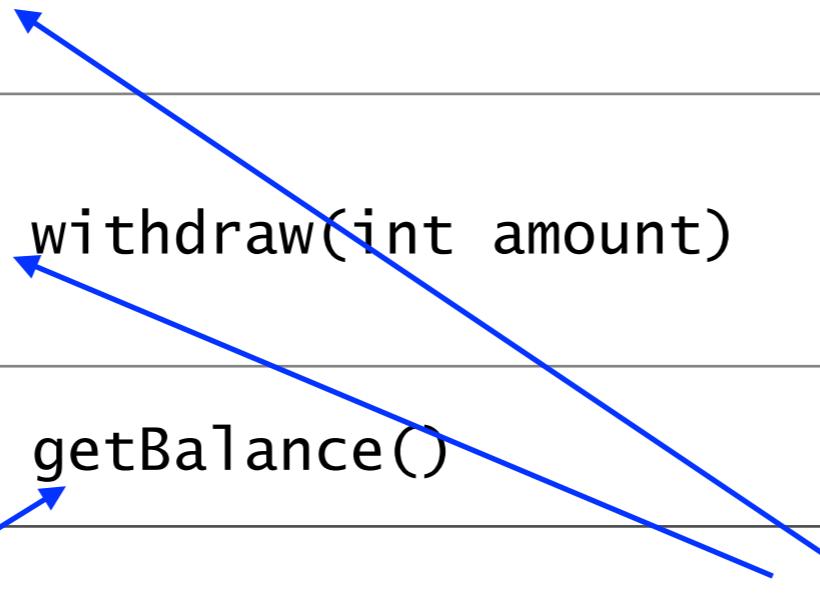
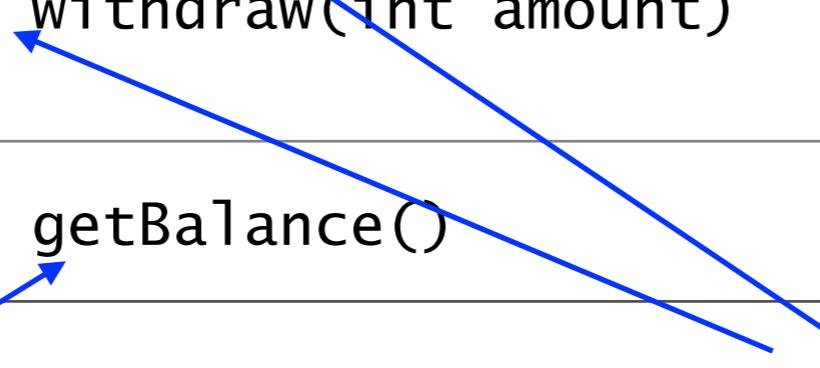
Starter



MVC Architecture

[3 단계] 부품별 클래스 명세 작성

Model

class	BankAccount	계좌 개설 및 운영
constructor	BankAccount(int initial_balance)	계좌 개설 initial_balance >= 0
field	private int balance	통장 잔액
method	<pre>boolean deposit(int amount)</pre> 	입금 amount >= 0 입금 성공하면 true, 실패하면 false
	<pre>boolean withdraw(int amount)</pre> 	출금 amount >= 0 출금 성공하면 true, 실패하면 false
	getBalance()	잔액 확인
accessor method		
mutator method		

[4 단계] 부품별 코드 작성 및 테스트

```
1 import javax.swing.*;
2
3 public class BankAccount {
4
5     private int balance; // invariant: balance >= 0
6
7     /* BankAccount - 계좌 개설
8      * @param initial_amount - 초기 입금 금액 (0 이상 양수) */
9     public BankAccount(int initial_amount) {
10        if (initial_amount >= 0)
11            balance = initial_amount;
12        else
13            balance = 0;
14    }
15
16     /* deposit - 입금
17      * @param amount - 입금액 (0 이상 양수)
18      * @return 입금 성공하면 true, 실패하면 false */
19     public boolean deposit(int amount) {
20        boolean result = false;
21        if (amount >= 0) {
22            balance = balance + amount;
23            result = true;
24        }
25        else
26            JOptionPane.showMessageDialog(null, "입금액에 문제가 있어서 입금이 취소되었습니다.");
27        return result;
28    }
29 }
```

[4 단계] 부품별 코드 작성 및 테스트

```
300  /* withdraw - 출금 (잔고가 있는 경우 한)
31   * @param amount - 출금액 (0 이상 양수)
32   * @return 출금 성공하면 true, 실패하면 false */
330  public boolean withdraw(int amount) {
34    boolean result = false;
35    if (amount < 0) {
36      JOptionPane.showMessageDialog(null, "출금액에 문제가 있어서 입금이 취소되었습니다.");
37    }
38    else if (amount > balance)
39      JOptionPane.showMessageDialog(null, "출금액이 잔고액보다 많아서 입금이 취소되었습니다.");
40    else {
41      balance = balance - amount;
42      result = true;
43    }
44    return result;
45  }
46
470  /* getBalance - 잔액 확인
48   * @return 잔액 */
490  public int getBalance() {
50    return balance;
51  }
52
53 }
```

[4 단계] 부품별 코드 작성 및 테스트

```

53④ public static void main(String[] args) {
54    BankAccount tester = new BankAccount(0);
55    System.out.println("잔액 = " + tester.getBalance());
56    int five = 50000;
57    int three = 30000;
58    if (tester.deposit(five))
59        System.out.println(five + "원 입금 성공 : 잔액 = " + tester.getBalance());
60    else
61        System.out.println(five + "원 입금 실패 : 잔액 = " + tester.getBalance());
62    if (tester.withdraw(three))
63        System.out.println(three + "원 출금 성공 : 잔액 = " + tester.getBalance());
64    else
65        System.out.println(three + "원 출금 실패 : 잔액 = " + tester.getBalance());
66    if (tester.withdraw(three))
67        System.out.println(three + "원 출금 성공 : 잔액 = " + tester.getBalance());
68    else
69        System.out.println(three + "원 출금 실패 : 잔액 = " + tester.getBalance());
70 }
```

클래스 테스트용 `main` 메소드
(임시용 - 검증 후 제거)

[3 단계] 부품별 클래스 명세 작성

View

class	BankReader	입력기
method	<pre>int readCommand()</pre>	요청 서비스 파악
	<pre>boolean readAmount(int amount)</pre>	요청 액수 파악

class	BankWriter	출력기
constructor	<pre>BankWriter(String title, BankAccount b)</pre>	출력기 초기화
method	<pre>showTransaction(String message, int amount)</pre>	거래 결과 표시
	<pre>showTransaction(String message)</pre>	거래 결과 표시

[4 단계] 부품별 코드 작성 및 테스트

```

1 import javax.swing.*;
2
3 public class BankReader {
4
5     private String input_line;
6
7     /* BankReader - 입력기 초기화 */
8     public BankReader() {
9         input_line = "";
10    }
11
12    /* readCommand - 요청 서비스 파악
13     * @param message - 안내 메시지
14     * @return 요청 문자열의 첫 문자 */
15    public char readCommand(String message) {
16        input_line = JOptionPane.showInputDialog(message).trim().toUpperCase();
17        return input_line.charAt(0);
18    }
19
20    /* readAmount - 요청 액수 파악
21     * @return 금액, 문제가 발생하면 알리고 0을 리턴 */
22    public int readAmount() {
23        int amount = 0;
24        String s = input_line.substring(1, input_line.length());
25        s = s.trim();
26        if (s.length() > 0)
27            amount = Integer.parseInt(s);
28        else
29            JOptionPane.showMessageDialog(null, "금액 입력 오류");
30        return amount;
31    }
32 }

```

[4 단계] 부품별 코드 작성 및 테스트

```

1 import java.awt.*;
2 import javax.swing.*;
3
4 public class BankWriter extends JPanel {
5     private int WIDTH = 300;
6     private int HEIGHT = 200;
7     private BankAccount bank;
8     private String last_transaction = "";
9
10    /* BankWriter - 출력기 초기화
11     * @param title - 창의 제목
12     * @param b - BankAccount 객체 */
13    public BankWriter(String title, BankAccount b) {
14        bank = b;
15        JFrame f = new JFrame();
16        f.getContentPane().add(this);
17        f.setTitle(title);
18        f.setSize(WIDTH, HEIGHT);
19        f.setVisible(true);
20        f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
21    }
22
23    public void paintComponent(Graphics g) {
24        g.setColor(Color.white);
25        g.fillRect(0, 0, WIDTH, HEIGHT+22);
26        g.setColor(Color.black);
27        int text_margin = 50;
28        int text_baseline = 50;
29        g.drawString(last_transaction, text_margin, text_baseline);
30        g.drawString("잔액 = " + bank.getBalance() + "원", text_margin, text_baseline + 20);
31    }
32

```

[4 단계] 부품별 코드 작성 및 테스트

```
33④  /* showTransaction - 거래 결과 표시
34    * @param message - 거래 메시지
35    * @param amount - 거래 금액 */
36⑤  public void showTransaction(int amount, String message) {
37    last_transaction = amount + "원 " + message;
38    this.repaint();
39 }
40
41⑥  /* showTransaction - 거래 결과 표시
42    * @param message - 거래 메시지 */
43⑦  public void showTransaction(String message) {
44    last_transaction = message;
45    this.repaint();
46 }
47 }
```

[3 단계] 부품별 클래스 명세 작성

Controller

1. 사용자 요청을 입력창에서 읽어온다.
2. 요청이 Q이면, 서비스를 종료한다.
3. 그렇지 않고, 요청이 D이면, 금액을 읽고 입금거래를 하고 거래 내역을 보여준다.
4. 그렇지 않고, 요청이 W이면, 금액을 읽고 출금거래를 하고 거래 내역을 보여준다.
5. 그렇지 않고. 요청이 부적절하면 문제점을 안내한다.

[4 단계] 부품별 코드 작성 및 테스트

```
1 public class AccountController {  
2  
3     private BankReader reader; // input-view  
4     private BankWriter writer; // output-view  
5     private BankAccount account; // model  
6  
7     /* AccountController 객체 초기  
8      * @param r - input-view 객체  
9      * @param w - output-view 객체  
10     * @param a - model 객체 */  
11    public AccountController(BankReader r, BankWriter w, BankAccount a) {  
12        reader = r;  
13        writer = w;  
14        account = a;  
15    }  
16}
```

[4 단계] 부품별 코드 작성 및 테스트

```

17  /* processTransactions - Q 요청을 할 때까지 고객의 요청을 처리 */
18  public void processTransactions() {
19      char command = reader.readCommand("입금 D금액, 출금 W금액, 종료 Q;");
20      if (command == 'Q') {
21          writer.showTransaction("서비스를 마칩니다.");
22          return;
23      }
24      else if (command == 'D') {
25          int amount = reader.readAmount();
26          if (account.deposit(amount))
27              writer.showTransaction(amount, "입금");
28          else
29              writer.showTransaction("입금 실패");
30      }
31      else if (command == 'W') {
32          int amount = reader.readAmount();
33          if (account.withdraw(amount))
34              writer.showTransaction(amount, "출금");
35          else
36              writer.showTransaction("출금 실패");
37      }
38      else
39          writer.showTransaction("요청 오류");
40      this.processTransactions();
41  }
42 }
```

[4 단계] 부품별 코드 작성 및 테스트

AccountController

테스트 전용

더미 클래스

```

1 // dummy class for test
2 public class BankAccount {
3
4     // no field variables
5
6     /* BankAccount - 계좌 개설
7      * @param initial_amount - 초기 입금 금액 (0 이상 양수) */
8     public BankAccount(int initial_amount) { }
9
10    /* deposit - 입금
11     * @param amount - 입금액 (0 이상 양수)
12     * @return 입금 성공하면 true, 실패하면 false */
13    public boolean deposit(int amount) {
14        System.out.println("입금 " + amount + "원");
15        return true;
16    }
17
18    /* withdraw - 출금 (잔고가 있는 경우 한)
19     * @param amount - 출금액 (0 이상 양수)
20     * @return 출금 성공하면 true, 실패하면 false */
21    public boolean withdraw(int amount) {
22        System.out.println("출금 " + amount + "원");
23        return true;
24    }
25
26    /* getBalance - 잔액 확인
27     * @return 잔액 */
28    public int getBalance() {
29        return 0;
30    }
31 }
```

Starter

애플리케이션 작동을 위한 무대 설치

```
1 /* 시동 클래스
2  * 입력 포맷
3  *   D 금액 - 입금
4  *   W 금액 - 출금
5  *   Q - 종료
6  * 출력
7  * 거래 결과 */
8 public class AccountManager {
9
10 public static void main(String[] args) {
11     BankReader reader = new BankReader();
12     BankAccount account = new BankAccount(0);
13     BankWriter writer = new BankWriter("자유은행", account);
14     AccountController controller = new AccountController(reader, writer, account);
15     controller.processTransactions();
16 }
17 }
```

[5 단계] 통합 테스트

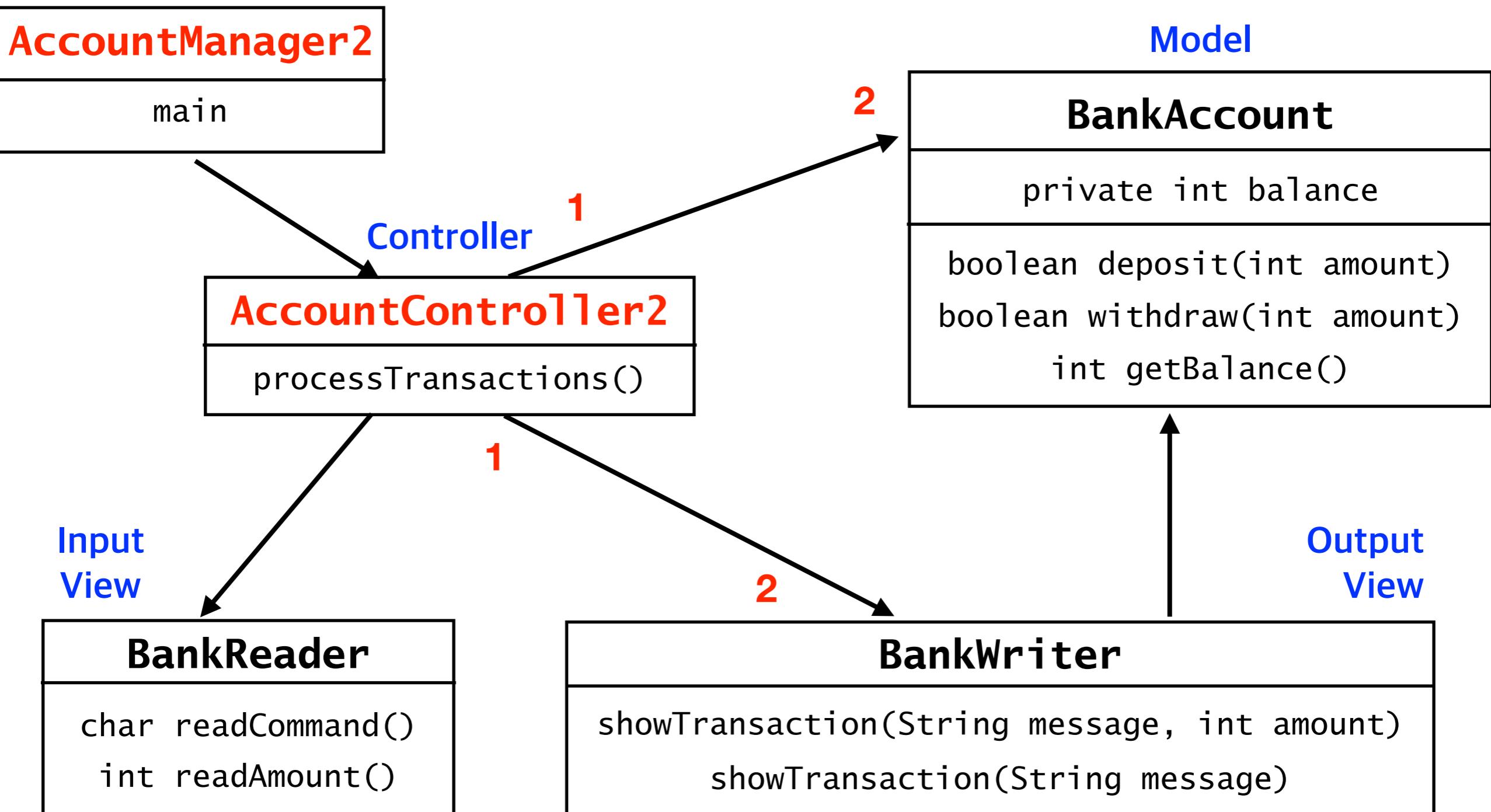
실습 #1

은행 계좌 관리 애플리케이션

2

2개의 별도 계좌 관리

Starter

**MVC Architecture**

```

1 public class AccountController2 {
2
3     private BankReader reader; // input-view
4     private BankWriter primary_writer; // output-view
5     private BankWriter secondary_writer;
6     private BankAccount primary_account; // model
7     private BankAccount secondary_account;
8     private BankAccount account; // remembers active account
9     private BankWriter writer; // remembers active writer
10
11    /* AccountController 객체 초기
12     * @param r - input-view 객체
13     * @param w - output-view 객체
14     * @param a - model 객체 */
15    public AccountController2(BankReader r, BankWriter w1, BankAccount a1,
16                               BankWriter w2, BankAccount a2) {
16
17        reader = r;
18        primary_writer = w1;
19        secondary_writer = w2;
20        primary_account = a1;
21        secondary_account = a2;
22        // 기본은 주계좌
23        account = primary_account;
24        writer = primary_writer;
25    }
26
27    /* resetAccount - 계좌 변경
28     * @param new_account - 변경할 계좌
29     * @param new_writer - 변경할 출력기 */
30    private void resetAccount(BankAccount new_account, BankWriter new_writer) {
31        writer.showTransaction("비활성");
32        account = new_account;
33        writer = new_writer;
34        writer.showTransaction("활성");
35    }

```

```
36
37     /* processTransactions - Q 요청을 할 때까지 고객의 요청을 처리 */
38     public void processTransactions() {
39         char command = reader.readCommand("계좌#1 P, 계좌#2 S, 입금 D금액, 출금 W금액, 종료 Q');");
40         if (command == 'Q') {
41             writer.showTransaction("서비스를 마칩니다.");
42             return;
43         }
44         else if (command == 'D') {
45             int amount = reader.readAmount();
46             if (account.deposit(amount))
47                 writer.showTransaction(amount, "입금");
48             else
49                 writer.showTransaction("입금 실패");
50         }
51         else if (command == 'W') {
52             int amount = reader.readAmount();
53             if (account.withdraw(amount))
54                 writer.showTransaction(amount, "출금");
55             else
56                 writer.showTransaction("출금 실패");
57         }
58         else if (command == 'P')
59             resetAccount(primary_account, primary_writer);
60         else if (command == 'S')
61             resetAccount(secondary_account, secondary_writer);
62         else
63             writer.showTransaction("요청 오류");
64         this.processTransactions();
65     }
66 }
```

```
1① /* 시동 클래스
2  * 입력 포맷
3  *   P      - 주계좌
4  *   S      - 보조계
5  *   D 금액 - 입금
6  *   W 금액 - 출금
7  *   Q      - 종료
8  * 출력
9  * 거래 결과 */
10 public class AccountManager2 {
11
12②     public static void main(String[] args) {
13         BankReader reader = new BankReader();
14         BankAccount primary_account = new BankAccount(0);
15         BankAccount secondary_account = new BankAccount(0);
16         BankWriter primary_writer = new BankWriter("자유은행 계좌#1", primary_account);
17         BankWriter secondary_writer = new BankWriter("자유은행 계좌#2", secondary_account);
18         AccountController2 controller = new AccountController2(reader, primary_writer, primary_account,
19                                     secondary_writer, secondary_account);
20         controller.processTransactions();
21     }
22 }
```

AccountManager2

...

a2 : BankAccount

int balance ==

0

```
public void deposit(...){...}
public boolean withdraw(...){...}
public int getBalance(){...}
```

a4 : BankAccount

int balance ==

0

```
public void deposit(...){...}
public boolean withdraw(...){...}
public int getBalance(){...}
```

a3 : BankWriter

BankAccount bank ==

a2

...

a1 : BankReader

...

a6 : AccountController2

BankAccount primary_account ==

a2

BankWriter primary_writer ==

a3

BankAccount secondary_account ==

a4

BankWriter secondary_writer ==

a5

BankReader reader ==

a1

BankAccount account ==

a2

BankWriter writer ==

a3

main

```
{ ...
    > processTransactions();
}
```

a5 : BankWriter

BankAccount bank ==

a4

...

실습 #2

BankAccountManager2 애플리케이션 - 계좌 이체 기능 추가

- 다음 사양의 계좌이체 기능 추가

method	send(int amount)	활성 계좌에서 비활성 계좌로 보내기
	receive(int amount)	비활성 계좌에서 활성 계좌로 보내기

- BankReader 클래스에 계좌이체 서비스 입력 메뉴 추가
 - > 금액 - send
 - < 금액 - receive