



# JAVA 프로그래밍

---

## 5. 결정문 및 반복문

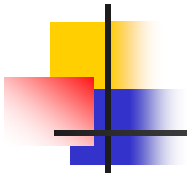
한 동 일



### 학습 목표

---

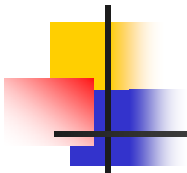
- To be able to implement decisions using if statements
- To understand how to group statements into blocks
- To learn how to compare integers, floating-point numbers, strings, and objects
- To recognize the correct ordering of decisions in multiple branches
- To program conditions using Boolean operators and variables



## 학습 목표

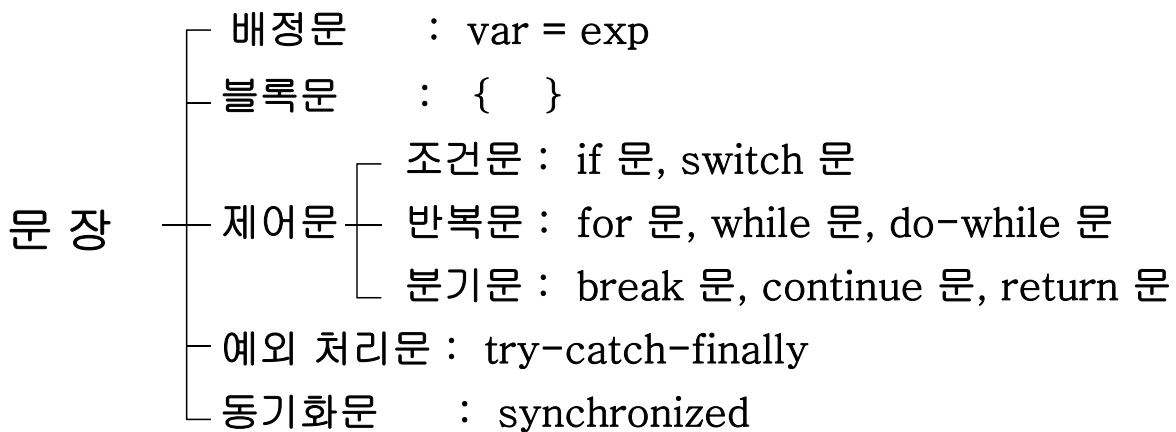
- To be able to program loops with the while, for, and do statements
- To avoid infinite loops and off-by-one errors
- To understand nested loops
- To implement simulations

3/45



## 제어 문장

- ANSI C 언어와 유사
- 문장의 종류



4/45



## 배정문(assignment statement)

- 값을 변수에 저장하는 데 사용
- 형태 : <변수> = <식> ;

```
remainder = dividend % divisor;  
i = j = k = 0;  
x *= y;
```

- 형 변환
  - 협소화(narrowing) 형 변환 : 캐스트(cast) 연산자 반드시 이용

5/45



## 복합문(compound statement)

- 단순문(simple statement)
  - 단순한 <변수> = <식> ; 형태

```
balance = balance - amount;
```

- 복합문(compound statement)
  - if, while, for 문 등

```
if (balance >= amount) balance = balance - amount;
```

6/45



## 블록문(block statement)

- 여러 문장을 한데 묶어 하나의 문장으로 나타냄
  - 주로 문장의 블록, 범위를 표시
- 형태
  - { <선언> 혹은 <문장> }

```
if (a > b) a--; b++; // if 조건이 변수 a에만 영향
```

```
if (a > b) { a--; b++; } // if 조건이 변수 a, b에 영향
```

- 지역변수(Local Variable)
  - 블록의 내부에서 선언된 변수
  - 선언된 블록 안에서만 참조 가능

7/45



## 제어문

- 프로그램의 실행 순서를 바꾸는 데 사용
- 실행 순서를 제어하는 방법에 따라
  - 조건문(결정문,판단문) : if 문, switch 문
  - 반복문 : for 문, while 문, do-while 문
  - 분기문 : break 문, continue 문, return 문

8/45

## 조건문 – if 문

- 조건에 따라 실행되는 부분이 다를 때 사용.
- if 문의 형태
  - if ( <조건식> ) <문장>
  - if ( <조건식> ) <문장1> else <문장2>
    - 조건식의 연산결과 : 논리형(true or false)
- 예 :

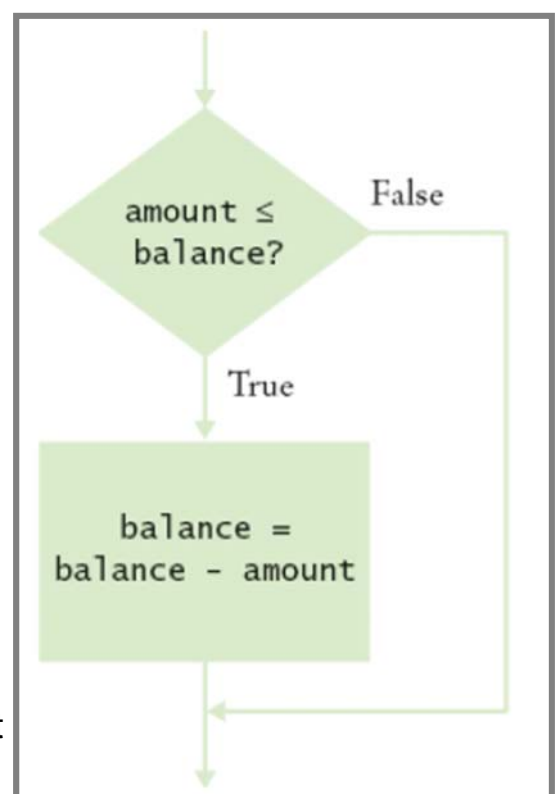
```
if (a < 0) a = -a;           // 절대값
if (a > b) m = a; else m = b; // 큰 값
```

9/45

## 조건문 – if 문

- if ( <조건식> ) <문장>

```
if (amount <= balance)
    balance = balance - amount;
```



Flowchart for an if statement

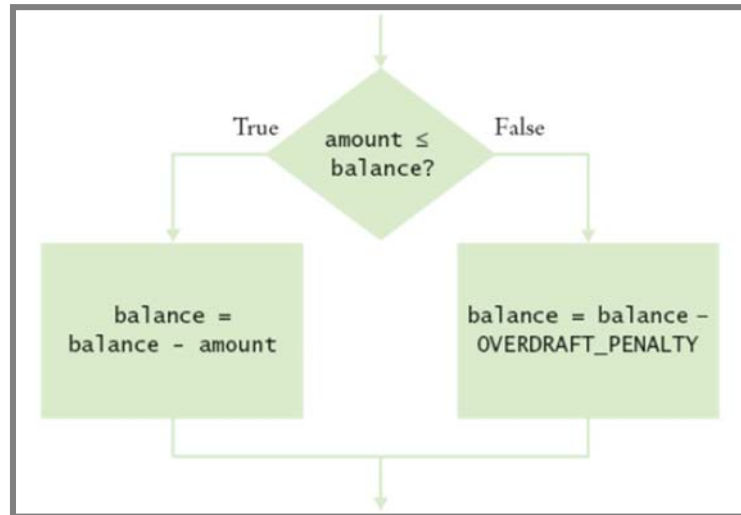
10/45

## 조건문 – if 문

- if ( <조건식> ) <문장> else <문장2>

```
if (amount <= balance)
    balance = balance - amount;
else
    balance = balance - OVERDRAFT_PENALTY;
```

Flowchart for an  
**if/else** statement



11/45

## 실수 비교

- 다음 코드의 출력은?

```
double r = Math.sqrt(2);
double d = r * r - 2;
if (d == 0)
    System.out.println("sqrt(2)squared minus 2 is 0");
else
    System.out.println("sqrt(2)squared minus 2 is not 0 but " + d);
```

- 출력 결과

sqrt(2)squared minus 2 is not 0 but 4.440892098500626E-16

12/45



## 실수 비교

- 실수의 경우 같은 값 비교는 에러 발생 가능성 큼
- 따라서 근사값 비교 구문 이용 필요
- 차이값의 절대값이 임계값 보다 적은지 비교

```
final double EPSILON = 1E-14;  
if (Math.abs(x - y) <= EPSILON)  
    // x is approximately equal to y
```

- $\epsilon$ (입실론, epsilon) : 보통  $10^{-14}$  사용

13/45



## 스트링 비교 - equals

- 스트링 비교 시 == 연산자 사용 금지

```
if (input == "Y") // WRONG!!!
```

- 대신 equals 메소드 사용

```
if (input.equals("Y"))
```

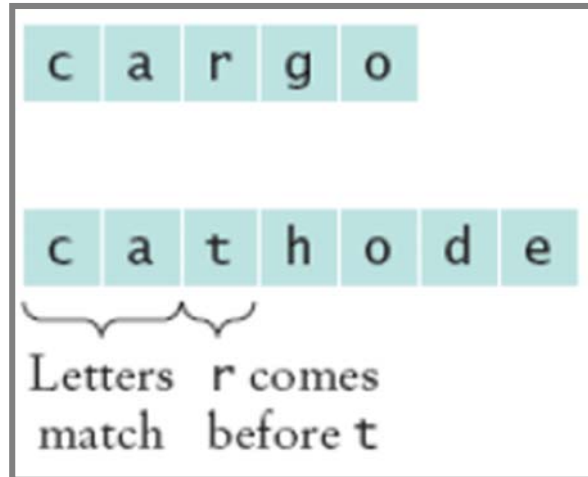
- 대소문자의 차이를 무시할 경우
  - 예를 들면 ("Y" or "y") 를 비교시

```
if (input.equalsIgnoreCase("Y"))
```

14/45

## 스트링 비교 - compareTo

- compareTo 메소드 : 대소문자를 고려한 사전식 비교
- `s.compareTo(t) < 0` : 사전에서 s가 t 앞에 올 의미
- “car” 는 “cargo” 앞에 위치
- 대문자는 소문자 앞에 위치
  - “Hello”는 “car” 앞에 위치



15/45

## 객체 비교

- 객체 비교시 `==` 연산자는 객체 참조의 동일성 여부, `equals` 메소드는 동일 내용 여부 검출

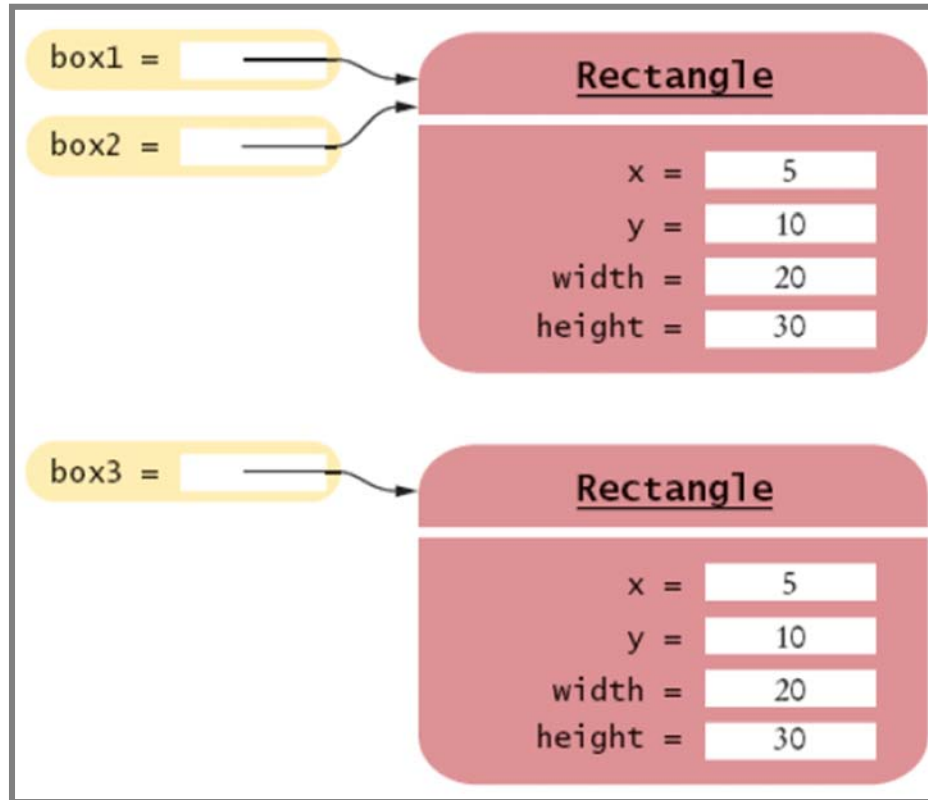
```
Rectangle box1 = new Rectangle(5, 10, 20, 30);  
Rectangle box2 = box1;  
Rectangle box3 = new Rectangle(5, 10, 20, 30);
```

- 위의 경우 다음 조건 만족
  - `box1 != box3`
  - `box1.equals(box3)`
  - `box1 == box2`
- 주의 : 클래스 구현 시 `equals` 메소드가 정의되어 있는 경우 사용 가능

16/45



## 객체 비교



17/45

## 널(null) 검사

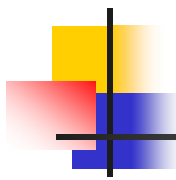
- null : 객체 참조 변수의 초기값
- 객체 참조 변수가 아무 객체도 참조하고 있지 않은 상태

```
String middleInitial = null; // Not set
if ( . . . )
    middleInitial = middleName.substring(0, 1);
```

- 대소문자의 객체 참조의 null 값 확인 예

```
if (middleInitial == null)
    System.out.println(firstName + " " + lastName);
else
    System.out.println(firstName + " " + middleInitial + ". "
        + lastName);
```

18/45



## 조건문 – if 문

- **내포된 if 문**

- 참 부분에서 if 문이 반복

```
if (<조건식>)  
    if (<조건식>)  
        // ...  
        <문장>
```

- else 부분에서 if 문이 반복

```
if (<조건식1>) <문장1>  
else if (<조건식2>) <문장2>  
...  
else if (<조건식n>) <문장n>  
else <문장>
```

19/45



## 조건문 – switch 문

- 조건에 따라 여러 경우로 처리해야 되는 경우
- **switch 문**의 형태

```
switch ( <식> ) {  
    case <상수식1> : <문장1>  
    case <상수식2> : <문장2>  
        :  
    case <상수식n> : <문장n>  
    default : <문장>  
}
```

- 여기서, **default**의 의미는 **otherwise**.
- **break** 문을 사용하여 탈출.

20/45

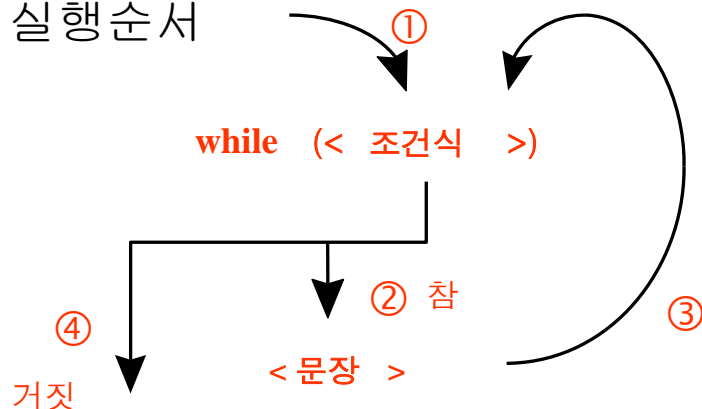
# 반복문 – while 문

- while 문의 형태

**while** ( 조건식 )  
<문장>

```
i = 1; s = 0;
while (i <= N) { // 1부터 N까지의 합
    s += i; ++i;
}
```

- 실행순서



21/45

# 반복문 – while 문

- 유의 사항 – 무한 루프

```
int years = 0;
while (years < 20) {
    double interest = balance * rate / 100;
    balance = balance + interest;
}
```

```
int years = 20;
while (years > 0) {
    years++; // Oops, should have been years--
    double interest = balance * rate / 100;
    balance = balance + interest;
}
```

- 프로그램을 끝거나 재부팅 필요

22/45

## 반복문 – do-while 문

- 반복되는 문장을 먼저 실행 한 후에 조건식을 검사
- do-while 문의 형태

do

<문장>

while (<조건식>);

조건식이 거짓이라도 <문장>  
부분이 적어도 한번은 실행

- precondition check     ---     for, while
- postcondition check    ---     do-while

23/45

## 반복문 – for 문

- 정해진 횟수만큼 일련의 문장을 반복
- for 문의 형태

for ( <식1> ; <식2> ; <식3> )

<문장>

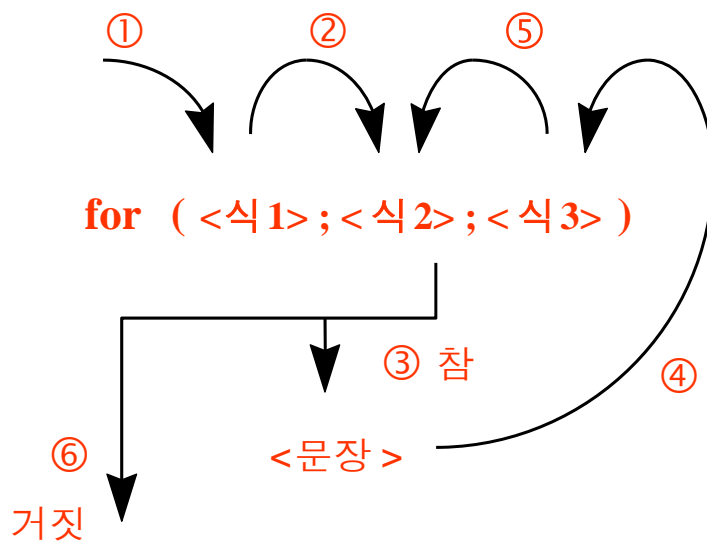
- <식1> : 제어 변수 초기화
- <식2> : 제어 변수를 검사하는 조건식
- <식3> : 제어 변수의 값을 수정

```
s = 0;
for (i=1; i<=N; i++) // 1부터 N까지의 합 : i 증가
    s += i;
```

24/45

## 반복문 – for 문

- for 문의 실행순서



25/45

## 반복문 – for 문

- 유의 사항 – 세미콜론의 사용

```
sum = 0;  
for (i = 1; i <= 10; i++); // 비어있는 for loop의 몸체  
    sum = sum + i;  
System.out.println(sum); // 출력은 11
```

- != 연산자의 사용 금지

```
for (i=1; i != n; i++) { ...}  
// n이 음수일 경우 의도하지 않은 무한 루프  
  
for (i=1; i <= n; i++) { ...} // OK
```

26/45

## 반복문 – for 문

- 무한 루프를 나타내는 for 문

```
for ( ; ; )  
    <문장>
```

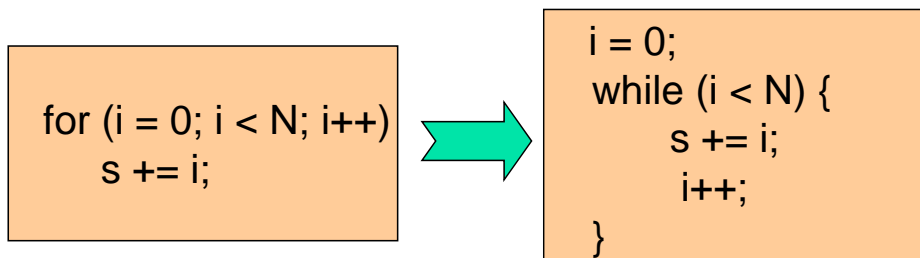
- 루프 종료 : break 문, return 문
- 내포된 for 문
  - for 문 안에 for 문이 있을 때.
  - 다차원 배열을 다룰 때.

```
for (i=0; i<N; i++)  
    for (j=0; j<M; j++)  
        matrix[ i ][ j ] = 0;
```

27/45

## 반복문 – for, while 문 비교

- for 문과 while 문의 비교



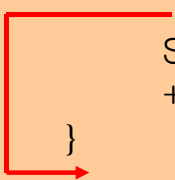
- for --- 주어진 횟수
- while --- 주어진 조건

28/45

## 분기문 – break 문

- 블록 밖으로 제어를 옮기는 역할
- break 문의 형태  
**break [레이블] ;**
- 레이블이 없는 경우
  - C/C++와 동일

```
int i = 1;
while (true) {
    if (i == 3)
        break;
    System.out.println("This is a " + i + " iteration");
    ++i;
}
```

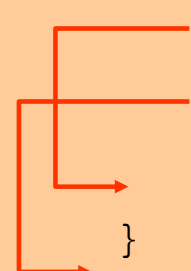


29/45

## 분기문 – break 문

- 레이블 break 문
  - goto 문 대용으로 사용 가능
  - 사용형태

```
labelName :
반복문1 {
    반복문2 {
        // ...
        break;
        // ...
        break labelName;
    }
    // ...
}
```



30/45

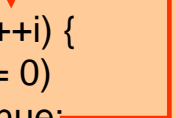
## 분기문 – continue 문

- 다음 반복이 시작되는 곳으로 제어를 옮기는 기능
- continue 문의 형태

**continue [레이블] ;**

- for 문 안에서 사용될 때

```
for (i=0; i<=5; ++i) {  
    if (i % 2 == 0)  
        continue;  
    System.out.println("This is a " + i + " iteration");  
}
```

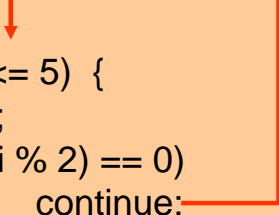


31/45

## 분기문 – continue 문

- while 문 안에서 사용될 때
  - 조건 검사 부분으로 이동

```
i = 0;  
while (i <= 5) {  
    ++i;  
    if ((i % 2) == 0)  
        continue;  
    System.out.println("This is a odd iteration - " + i);  
}
```

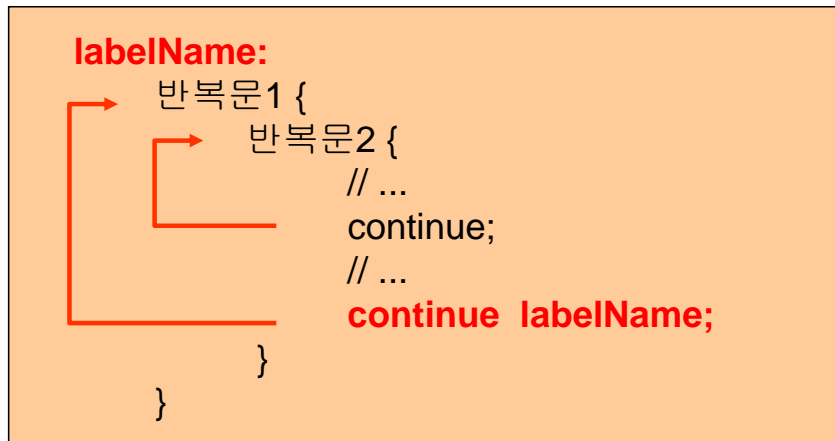


32/45



## 분기문 – continue 문

- 레이블 continue 문
  - 레이블 break와 유사



33/45

## 분기문 – return 문

- 메소드의 실행을 종료하고 호출한 메소드에게 제어를 넘겨주는 문장
- return 문의 형태
  - **return;**
  - **return <식>;**

return;	// return 값이 없을 때
return 100;	// return 값이 있을 때
return a;	// 변수 a의 값을 return 할 때
return (a + "," + b);	// 연산식의 결과를 return 할 때

34/45



# 난수와 시뮬레이션

- 시뮬레이션
  - 임의로 이벤트를 발생시키고 그 효과를 예측하기 위한 방법
  - 먼저 난수를 생성하고 이를 이용하여 행동 모사를 반복적으로 진행
- 난수 발생기: java.util package의 Random 클래스 사용

```
Random generator = new Random();  
int n = generator.nextInt(a); // 0 <= n < a  
double x = generator.nextDouble(); // 0 <= x < 1
```

- 주사위의 경우(1 ~ 6 사이의 정수)

```
int d = 1 + generator.nextInt(6);
```

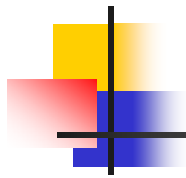
35/45



## [실습1] - 실수, 스트링, 객체비교

- 강의 홈페이지의 ComparisionTest.java code를 이해하라.
- 코드의 실행 이전에 그 결과를 예측하라.
- 예측결과와 실행결과가 다를 경우 그 이유를 파악하라.

36/45



## [실습2] - if 문

- 첨부된 Grade.java를 이해하고 GradeTest.java의 실행결과를 예상하라.
- Grade.java를 다음과 같이 동작하게 수정하라.
  - 90점 이상 A, 80점 이상 B, 70점 이상 C, 60점 이상 D, 나머지 F

37/45



## [실습3] - case 문

- 첨부된 LocalCode.java, LocalCodeTest.java 코드는 전화번호 코드의 지역정보를 출력하는 동작을 수행한다. 이를 이해하고 지역 코드 입력에 대한 결과를 예측하라.
  - 2: 서울, 31: 경기도, 32: 인천
- 각각의 입력에 대해서 출력을 확인하고 예측과 다른 경우 그 이유를 파악하고 이를 디버깅하라.
- switch문으로 되어있는 메소드를 if문으로 수정하고 같은 결과가 나오는지 테스트하라.

38/45



## [실습4] – print 메소드의 사용

- 첨부된 CodePrint.java, CodePrintTest.java 코드는 전화번호 코드의 지역정보를 잘못 출력하고 있다. 지역 코드 입력에 대한 결과를 예측하라.
  - 2: 서울, 31: 경기도, 32: 인천
- 실습 3과 같은 방법으로 이를 디버깅하라.
- main method가 아닌 메소드에서 print 메소드를 사용하는 것이 바람직한가?

39/45



## [실습5] – break, continue 문

- 첨부된 Sum.java를 통해 break문과 continue문의 용법에 대해 이해하고 SumTest.java의 결과를 예상하라.
- Sum.java를 수정하여 다음과 같이 동작하게 하라.
  - 200이상의 수는 계산하지 않게, 3의 배수의 합

40/45



## [실습6] - 중첩 루프 활용

- 강의 홈페이지의 Triangle.java code를 이해하라.
- 다양한 입력 값을 이용하여 실행 결과를 확인하라.
- 삼각형 대신에 정사각형을 출력하도록 중첩루프를 수정하라.
- 다양한 입력 값을 이용하여 실행 결과를 확인하라.

41/45



## [실습7] - 중첩 루프 활용 2

- 크기(홀수)를 입력 받아 아래 예와 같은 이등변 삼각형을 출력하는 프로그램을 작성하라.

예) 크기 = 7

```
  *
 ***
*****
*****
```

- 다음과 같은 마름모를 출력하는 프로그램을 작성하라.

예) 크기 = 7

```
  *
 ***
*****
*****
 *****
  ***
   *
```

42/45

## [실습8] - 중첩 루프 활용 3

- 크기(양의 정수)를 입력 받아 아래 예와 같은 오른쪽으로 치우친 역삼각형을 출력하는 프로그램을 작성하라.

예) 크기 = 4

```
****
***
**
*
```

- 별 대신 아래와 같이 수를 출력하도록 프로그램을 수정하라.(각 수는 3칸을 차지)

예) 크기 = 5

```
□□1 2 3 4 5
      6 7 8 9
        10 11 12
          13 14
            15
```

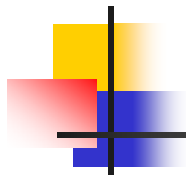
43/45

## [실습9] - 소수 출력

- 아래의 출력 결과는 1 ~ 200 사이의 소수를 구한 결과이다. 아래와 같이 소수와 그 개수를 출력 결과로 나타내는 프로그램을 작성하라.
  - 힌트 : 소수는 1과 자기 자신으로 밖에 나누어지지 않음

2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199				
Total : 46									

44/45



## [실습10] – 난수의 발생

- 강의 홈페이지의 Die.java, DieTester.java code 를 이해하라.
- 여러 번 테스트를 수행하고 각각의 실행 결과를 확인하라.
- 다음의 확률 게임에 응하는 것이 좋은지 판단하고 DieTester.java 코드를 수정하여 이를 확인하라.
  - 주사위를 던졌을 때 1이 나오면 이기고 1800원을 얻음
  - 주사위를 던졌을 때 1이 아닌 수가 나오면 지게되며 300원을 잃음
  - 초기값으로 10,000원을 가지고 시뮬레이션을 통해 보여라.
  - 여러 번 실행하여 잃은 횟수와 진 횟수를 비교하면서 예상 결과와 비교하라.
  - 필요 시 테스트 횟수를 10에서 1000 정도로 변경한 후 확인하라.