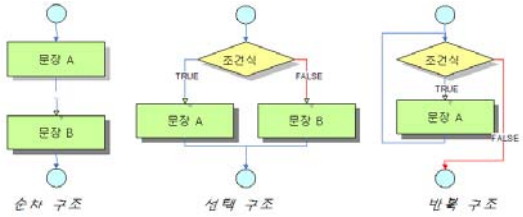
# 자바 기초문법
# 선택, 반복, 배열, 예외처리

514760-1
2017년 가을학기
9/11/2017
박경신

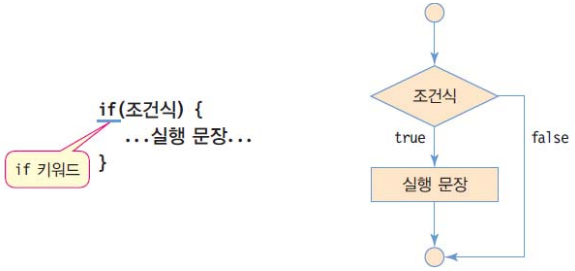
## Control Statement

- 제어문의 종류
  - 제어문이란 프로그램을 실행할 때는 논리적인 흐름이 필요한데, 이러한 문장의 논리적인 흐름을 통제해 주는것.
  - 조건문 - 조건식의 값에 따라 각각에 해당되는 명령문을 수행한다. 예) if 문, switch 문
  - 반복문 – 조건이 만족하는 동안 특정 명령문을 반복적으로 수행한다. 예) while 문, do 문, for 문, foreach 문
  - 점프문 – 제어권을 이동시킬 때 점프문을 사용한다. 예) label 문, break 문, continue 문




## If Statement

- 단순 if 문
  - if 다음의 괄호 안에는 조건식(논리형 변수나 논리 연산)
  - 조건식의 값
    - true인 경우, if문을 벗어나 다음 문장이 실행된다.
    - false의 경우에는 if 다음의 문장이 실행되지 않고 if 문을 빠져 나온다.
  - 실행문장이 단일 문장인 경우 둘러싸는 {, } 생략 가능




## 예제 : if 구문 사용

- 점수가 80점이 이상이면 합격을 판별

```
import java.util.Scanner;
public class SuccessOrFail {
  public static void main (String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("점수를 입력하시오: ");
    int score = in.nextInt();
    if (score >= 80)
      System.out.println("축하합니다! 합격입니다.");
  }
}
```

```
점수를 입력하시오: 95
축하합니다! 합격입니다.
```

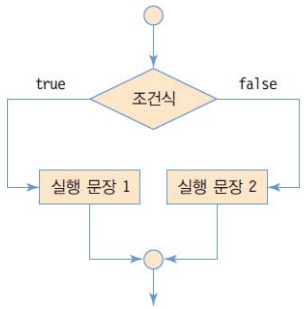
## If-Else Statement

- if-else 문
  - 조건식이 true면 실행문장1 실행 후 if-else문을 벗어남
  - false인 경우에 실행문장2 실행후, if-else문을 벗어남

```
if(조건식) {
    ...실행 문장 1...
}
else {
    ...실행 문장 2...
}
```

if 키워드

else 키워드




## 예제 : if-else 구문 사용

- 입력된 수가 3의 배수인지 판별

```
import java.util.Scanner;
public class MultipleOfThree {
  public static void main (String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("수를 입력하시오: ");
    int number = in.nextInt();
    if (number % 3 == 0)
      System.out.println("3의 배수입니다.");
    else
      System.out.println("3의 배수가 아닙니다.");
  }
}
```

```
수를 입력하시오: 129
3의 배수입니다.
```

## If-Else Statement

- 다중 if-else 문
  - 실행문장이 다시 if문 또는 if-else문을 포함
  - else 문은 바로 전의 if문과 짝을 이룬다.
  - 조건문이 너무 많은 경우, switch 문 사용 권장

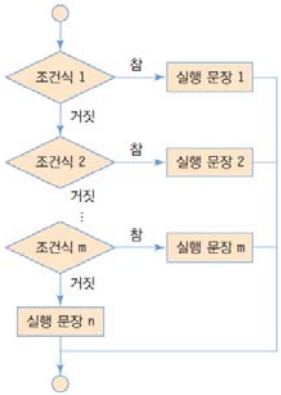
```
if(조건식 1) {
    실행 문장 1; // 조건식 1이 참인 경우
}
else if(조건식 2) {
    실행 문장 2; // 조건식 2가 참인 경우
}
else if(조건식 m) {
    ..... // 조건식 m이 참인 경우
}
else {
    실행 문장 n; // 앞의 모든 조건이 거짓인 경우
}
```

if 키워드

else if 키워드

else if 키워드

else 키워드




## 예제 : 다중 if-else 사용

- 키보드 입력된 성적에 학점을 부여하는 프로그램 작성

```
import java.util.Scanner;
public class Grading {
  public static void main (String[] args) {
    char grade;
    Scanner a = new Scanner(System.in);
    while (a.hasNext()) {
      int score = a.nextInt();
      if(score >= 90) // score가 90 이상인 경우
        grade = 'A';
      else if(score >= 80) // score가 80 이상이면서 90 미만인 경우
        grade = 'B';
      else if(score >= 70) // score가 70 이상이면서 80 이만인 경우
        grade = 'C';
      else if(score >= 60) // score가 60 이상이면서 70 이만인 경우
        grade = 'D';
      else // score가 60 이만인 경우
        grade = 'F';
      System.out.println("학점은 "+grade+"입니다");
    }
  }
}
```

키가 입력될 때까지 기다리며, 입력된 키가 있는 경우 true 리턴. 라인의 첫 문자로 ctrl-z 키가 입력되면 false 리턴

```
80
학점은
B입니다
90
학점은
A입니다
76
학점은
C입니다
```

## Ternary Conditional Operator ?:

- 조건 연산자 ?:는 if-else로 바꿀 수 있음

```
i = a>b?a-b:b-a;
```

⇨

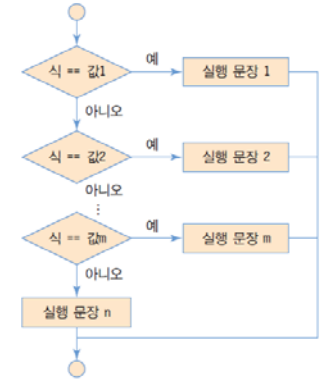
```
if (a>b)
    i = a – b;
else
    i = b – a;
```

## Switch Statement

- switch문은 식과 case 문의 값과 비교
  - case의 비교 값과 일치하면 해당 case문의 실행문장 수행
    - break를 만나면 switch문을 벗어남
  - case의 비교 값과 일치하는 것이 없으면 default 문 실행
- default문은 생략 가능

```
switch(식) {
  case 값1:
    실행 문장 1;
    break;
  case 값2:
    실행 문장 2;
    break;
  ...
  case 값m:
    실행 문장 m;
    break;
  default:
    실행 문장 n;
}
```

switch 키워드, case 키워드, break 키워드, default 키워드




## Switch Statement

- switch문 내의 break문
  - break 문장을 만나면 switch문을 벗어남
  - **만일 case 문에 break문이 없다면, 다음 case문의 실행** 문장으로 실행을 계속하게 되며, 언젠가 break를 만날 때까지 계속 내려감

```
char grade = 'A';
switch (grade) {
  case 'A':
    System.out.println("90 ~ 100점입니다.");
  case 'B':
    System.out.println("90 ~ 100점입니다.");
    break;
  case 'C':
    System.out.println("90 ~ 100점입니다.");
    break;
}
```

```
90 ~ 100점입니다.
80 ~ 89점입니다.
```

## 예제 : switch 사용

- switch구문을 사용하여 학점에 따른 출력

```
public class GradeSwitch {
  public static void main(String[] args) {
    char grade='C';
    switch (grade) {
      case 'A':
      case 'B':
        System.out.println("참 잘하였습니다.");
        break;
      case 'C':
      case 'D':
        System.out.println("좀 더 노력하세요.");
        break;
      case 'F':
        System.out.println("다음 학기에 다시 수강하세요.");
        break;
      default:
        System.out.println("잘못된 학점입니다.");
    }
  }
}
```

```
좀 더 노력하세요.
```

## Switch Statement

- case 문의 값의 특징
  - switch 문은 식의 결과 값을 case 문과 비교
  - 사용 가능한 case문의 비교 값
    - 정수 타입 리터럴, JDK 1.7부터는 문자열 리터럴도 허용

```
int a = 0;
int b = 1;
int c = 25;
switch(c%2) {
  case 1 : // 정수 리터럴
    ...;
    break;
  case 2: // 정수 리터럴
}
```

```
String s = "예";
switch(s) {
  case "예" : // 문자열 리터럴
    ...;
    break;
  case "아니요" : // 문자열 리터럴
    ...;
    break;
}
```

## Switch Statement

- 잘못된 case 문

```
switch(a) {
  case a :        // 오류. 변수 사용 안됨
  case a > 3 :    // 오류. 수식 안됨
  case a == 1 :   // 오류. 수식 안됨
}
```

## 예제 : switch 사용한 학점 분류

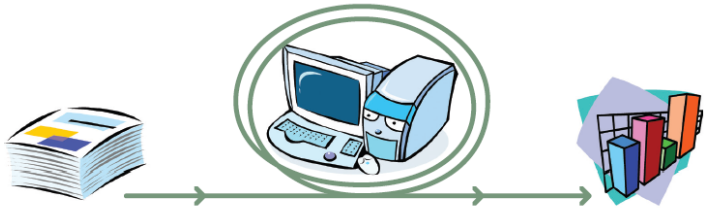
```
import java.util.Scanner;
public class Grading2 {
   public static void main (String[] args) {
      char grade;
      Scanner a = new Scanner(System.in);
      while (a.hasNext()) {
         int score = a.nextInt();
         switch (score/10) {
            case 10:
            case 9:
               grade = 'A';
               break;
            case 8:
               grade = 'B';
               break;
            case 7:
               grade = 'C';
               break;
            case 6:
               grade = 'D';
               break;
            default:
               grade = 'F';
         }
         System.out.println("학점은 "+grade+"입니다");
      }
   }
}
```

```
100
학점은 A입니다
55
학점은 F입니다
76
학점은 C입니다
```

## Loop

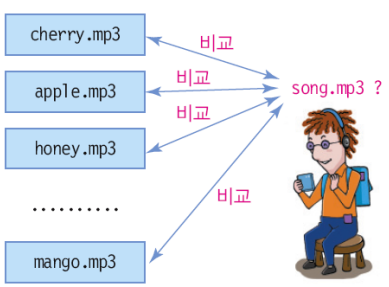
**Q) 반복 구조는 왜 필요한가?**

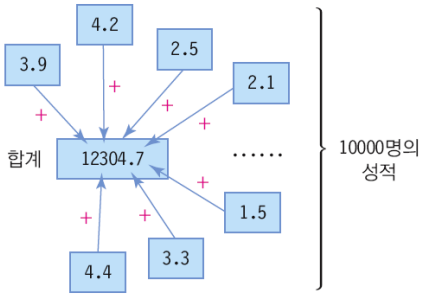
**A) 같은 처리 과정을 되풀이하는 것이 필요하기 때문이다. 학생 30명의 평균 성적을 구하려면 같은 과정을 30번 반복하여야 한다.**



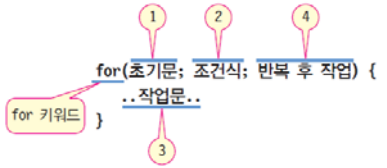
# Loop

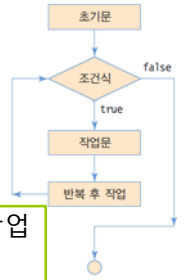
- 자바 반복문의 종류
  - for 문
  - while 문
  - do while 문




# For Statement



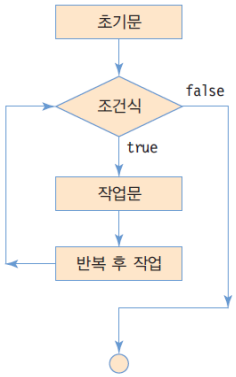

❶
- for 문이 실행한 후 오직 한번만 실행되는 초기화 작업
- 콤마(',')로 구분하여 여러 문장 나열 가능
- 초기화할 일 없으면 비어둘 수 있음

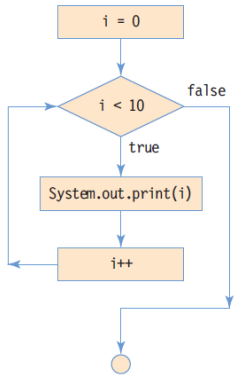
❷
- 논리형 변수나 논리 연산만 가능
- 반복 조건이 true이면 반복 계속, false이면 반복 종료
- 반복 조건이 true 상수인 경우, 무한 반복
- 반복 조건이 비어 있으면 true로 간주

❹
- 반복 작업 문장들의 실행 후 처리 작업
- 콤마(',')로 구분하여 여러 문장 나열 가능

# For Statement

- For 문의 실행 과정을 나타내는 순서도
  - 주로 정해진 횟수만큼 반복할 때 사용됨
  - 또는 정해진 범위를 반복할 때 활용됨




```
for(i=0; i<10; i++) {
    System.out.print(i);
}
```

# For Statement

- 0~9 까지 정수 출력

```
int i;
for(i = 0; i < 10; i++) {
  System.out.print(i);
}
```

```
int i;
for(i = 0; i < 10; i++)
  System.out.print(i);
```

- For-loop 안에 변수 선언 가능

```
for(int i = 0; i < 10; i++) // 변수 i는 for문을 벗어나서 사용할 수 없음
  System.out.print(i);
```

- 0~100까지 합 계산

```
int sum = 0;
for(int i = 0; i <= 100; i++)
  sum += i;
```

```
int sum;
for(int i = 0, sum=0; i <= 100; i++)
  sum += i;
```

```
int sum = 0;
for(int i = 100; i >= 0; i--)
  sum += i;
```

## For Statement

```
for(초기작업; true; 반복후작업) { // 반복 조건이 true이면 무한 반복
.....
}
```

```
for(초기작업; ; 반복후작업) { // 반복조건이 비어 있으면 true로 간주, 무한 반복
.....
}
```

```
// 초기 작업과 반복후작업은 ‘,’로 분리하여 여러 문장 나열 가능
for(i=0; i<10; i++, System.out.println(i)) {
.....
}
```

```
// for문 내에 변수 선언
for(int i=0; i<10; i++) { // 변수 i는 for문 내에서만 사용 가능
.....
}
```

## 예제 : for 구문 사용

- 1부터 10까지 덧셈을 표시하고 합 계산

```
public class ForSample {
  public static void main (String[] args) {
    int i, j;
    for (j=0,i=1; i <= 10; i++) {
      j += i;
      System.out.print(i);
      if(i==10) {
        System.out.print("=");
        System.out.print(j);
      }
      else
        System.out.print("+");
    }
  }
}
```

```
1+2+3+4+5+6+7+8+9+10=55
```

## 예제 : for 구문 사용

- 사용자로부터 정수를 입력 받아 factorial 계산

```
public class Factorial {
  public static void main (String[] args) {
    long fact = 1;
    int n;
    System.out.printf("정수를 입력하시오: ");
    Scanner scan = new Scanner(System.in);
    n = scan.nextInt();
    for (int i=1; i <= n; i++) {
      fact = fact * i;
    }
    System.out.printf("%d!는 %d입니다. ", n, fact);
  }
}
```

```
정수를 입력하시오: 5
5!는 120입니다.
```

## 예제 : for 구문 사용

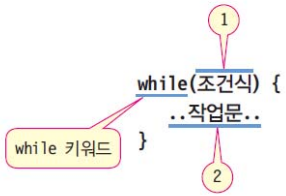
- 사용자로부터 양의 정수를 입력 받아 약수를 계산

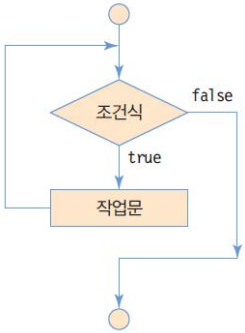
```
public class Divisor {
  public static void main (String[] args) {
    System.out.printf("양의 정수를 입력하시오: ");
    Scanner scan = new Scanner(System.in);
    int n = scan.nextInt();
    System.out.println(n + "의 약수는 다음과 같습니다.");
    for (int i=1; i <= n; ++i) {
      if (n % i == 0)
        System.out.print(" " + i);
    }
  }
}
```

```
양의 정수를 입력하시오: 100
100의 약수는 다음과 같습니다.
1 2 4 5 10 25 50 100
```

## While Statement

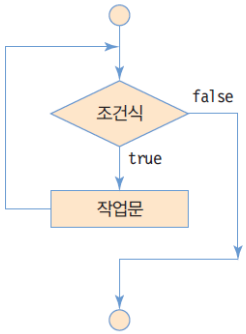
- 반복 조건이 true이면 반복, false 이면 반복 종료
- 반복 조건이 없으면 컴파일 오류
- 처음부터 반복 조건을 통과한 후 작업문 수행

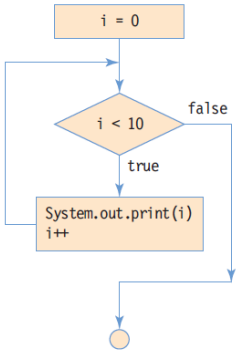



## While Statement

- While 문의 실행과정을 나타내는 순서도




```
i = 0;
while(i<10) {
    System.out.print(i);
    i++;
}
```

## 예제 : while 구문 사용

- 사용자로부터 숫자를 입력 받아 입력 받은 모든 수의 평균을 출력. 0이 입력되면 입력이 종료되고 평균 계산.

```
import java.util.Scanner;
public class WhileSample {
  public static void main (String[] args) {
    Scanner scan = new Scanner(System.in);
    int n = 0;
    double sum = 0;
    int i=0;
    while ((i = scan.nextInt()) != 0) {
      sum += i;
      n++;
    }
    System.out.println("입력된 수의 개수는 " + n + "개이며 평균은 "
                       + sum / n + "입니다.");
  }
}
```

```
10
20
30
40
0
```

마지막 입력

입력된 수의 개수는 4개이며 평균은 25.0입니다.

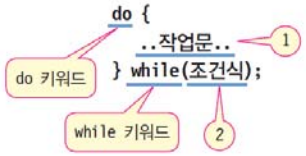
## 예제 : while 구문 사용

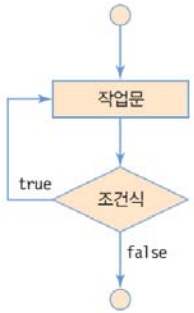
- 사용자로부터 두 수를 입력 받아 최대 공약수를 계산

```
public class Gcd {
  public static void main (String[] args) {
    int x, y, r;
    System.out.printf("두개의 정수를 입력하시오 (큰수, 작은수): ");
    Scanner scan = new Scanner(System.in);
    x = scan.nextInt();
    y = scan.nextInt();
    while (y !=0) {
      r = x % y;
      y = r;
    }
    System.out.println("최대공약수는 " + x);
  }
}
```

```
두개의 정수를 입력하시오 (큰수, 작은수) : 24 36
최대 공약수는 12
```

## Do-While Statement



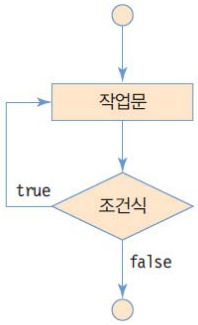



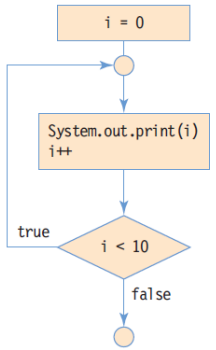

❶ • 무조건 최소 한번은 실행

❷ • 반복 조건이 true이면 반복, false이면 반복 종료
• 반복 조건이 없으며 컴파일 오류

## Do-While Statement

- do-while 문의 실행 과정을 나타내는 순서도




```
i = 0;
do {
    System.out.print(i);
    i++;
} while(i<10);
```

## 예제 : do-while 구문 사용

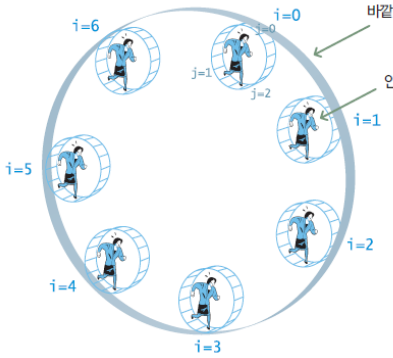
- 'a'부터 'z'까지 출력

```
public class DoWhileSample {
  public static void main (String[] args) {
    char a = 'a';
    do {
      System.out.print(a);
      a = (char) (a + 1);
    } while (a <= 'z');
  }
}
```

```
abcdefghijklmnopqrstuvwxyz
```

## Nested Loop

- 중첩 반복문(nested loop)은 반복문 안에 다른 반복문이 위치



## Nested Loop

- 중첩 반복
  - 반복문이 다른 반복문을 내포하는 구조
  - 이론적으로는 몇 번이고 중첩 반복 가능
  - 너무 많은 중첩 반복은 프로그램 구조를 복잡하게 하므로 2중 또는 3중 반복이 적당

```
for(i=0; i<100; i++) { // 100 개의 학교 성적을 모두 더한다.
    .....
    for(j=0; j<10000; j++) { // 10000 명의 학생 성적을 모두 더한다.
        .....
        .....
    }
    .....
}
```

10000명의 학생이 있는 100개 대학의 모든 학생 성적의 합을 구할 때, for 문을 이용한 이중 중첩 구조

## 예제 : Nested-Loop 사용

- 중첩 for문을 사용하여 구구단을 한 줄에 한 단씩 출력

```
public class NestedLoop {
  public static void main (String[] args) {
    int i, j;

    for (i = 1; i < 10; i++,System.out.println()) {
      for (j = 1; j < 10; j++,System.out.print('\t')) {
        System.out.print(i + "*" + j + "=" + i*j);
      }
    }
  }
}
```

```
1*1=1   1*2=2   1*3=3   1*4=4   1*5=5   1*6=6   1*7=7   1*8=8   1*9=9
2*1=2   2*2=4   2*3=6   2*4=8   2*5=10  2*6=12  2*7=14  2*8=16  2*9=18
3*1=3   3*2=6   3*3=9   3*4=12  3*5=15  3*6=18  3*7=21  3*8=24  3*9=27
4*1=4   4*2=8   4*3=12  4*4=16  4*5=20  4*6=24  4*7=28  4*8=32  4*9=36
5*1=5   5*2=10  5*3=15  5*4=20  5*5=25  5*6=30  5*7=35  5*8=40  5*9=45
6*1=6   6*2=12  6*3=18  6*4=24  6*5=30  6*6=36  6*7=42  6*8=48  6*9=54
7*1=7   7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49  7*8=56  7*9=63
8*1=8   8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64  8*9=72
9*1=9   9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

## Continue Statement

- continue 문
  - 반복문을 빠져 나가지 않으면서
  - 반복문 실행 도중 다음 반복을 진행

```
for (초기문; 조건식; 반복후작업) {
    ..............
    continue;
    ..............
}
```

분기

```
do {
    ..............
    continue;
    ..............
} while (조건식);
```

조건식으로 분기

```
while (조건식) {
    ..............
    continue;
    ..............
}
```

조건식으로분기

## 예제 : for와 continue 문 사용

- For와 continue문을 사용하여 1부터 100까지 짝수의 합을 계산

```
public class ContinueExample {
  public static void main (String[] args) {
    int sum = 0;
    for (int i = 1; i <= 100; i++) {
      if (i%2 == 1) // 홀수이면
        continue;
      else
        sum += i;
    }
    System.out.println("1부터 100까지 짝수의 합은 " + sum);
  }
}
```

```
1부터 100까지 짝수의 합은 2550
```

## Break Statement

- break 문
  - 반복문 하나를 완전히 빠져 나갈 때 사용
  - break문은 하나의 반복문만 벗어남
    - 중첩 반복의 경우 안쪽 반복문의 break 문이 실행되면 안쪽 반복문만 벗어남

## 예제 : while과 break문 사용

- while과 break문을 사용하여 -1이 입력될 때까지 입력된 숫자의 개수를 출력

```
import java.util.Scanner;
public class BreakExample {
  public static void main (String[] args) {
    Scanner in = new Scanner(System.in);
    int num = 0;

    while (true) {
      if (in.nextInt() == -1)
        break;
      num++;
    }
    System.out.println("입력된 숫자 개수는 " + num);
  }
}
```

```
10
8
9
5
-1
입력된 숫자 개수는 4
```

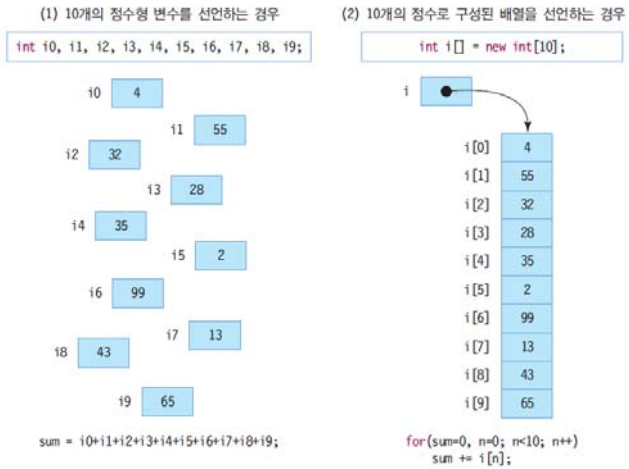
마지막 입력

## Array

- 배열(array)
  - 여러 개의 데이터를 같은 이름으로 활용할 수 있도록 해주는 자료 구조
    - 인덱스(Index, 순서 번호)와 인덱스에 대응하는 데이터들로 이루어진 자료 구조
    - 배열을 이용하면 한 번에 많은 메모리 공간 선언 가능
  - 배열은 같은 타입의 데이터들이 순차적으로 저장되는 공간
    - 원소 데이터들이 순차적으로 저장됨
    - 인덱스를 이용하여 원소 데이터 접근
    - 반복문을 이용하여 처리하기에 적합한 자료 구조(주로 for 또는 for-each 반복문과 많이 사용됨)
  - 배열 인덱스
    - 0부터 시작
    - 인덱스는 배열의 시작 위치에서부터 데이터가 있는 상대 위치

## Array

- 자바 배열의 필요성과 모양



# Array

- 배열 선언과 배열 생성의 두 단계 필요
  - 배열 선언

```
int     intArray[];
char    charArray[];
```

또는

```
int[]   intArray;
char[]  charArray;
```

  - 배열 생성

```
intArray = new int[10];
charArray = new char[20];
```

또는

```
int intArray[] = new int[10];
char charArray[] = new char[20];
```

  - 선언과 초기화
    - 배열 생성과 값 초기화

```
// 총 10개의 정수 배열 생성 및 값 초기화
int intArray[] = {0,1,2,3,4,5,6,7,8,9};
```

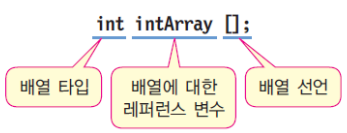
  - 잘못된 배열 선언

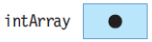
```
//int intArray[10]; // 컴파일 오류. 배열의 크기를 지정할 수 없음
```

# Array

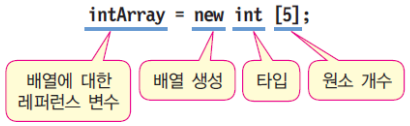
- 배열 선언과 생성

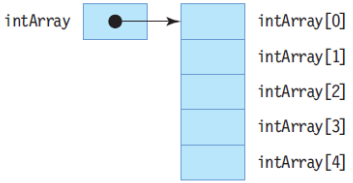
(1) 배열에 대한 레퍼런스 변수 intArray 선언




(2) 배열 생성

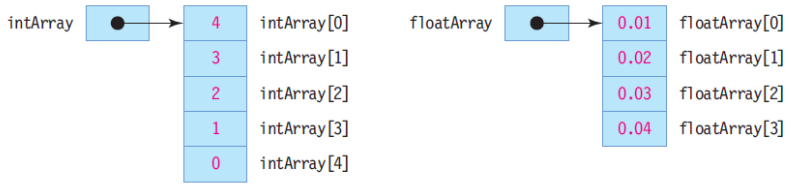



# Array

- 배열을 초기화하면서 생성한 결과

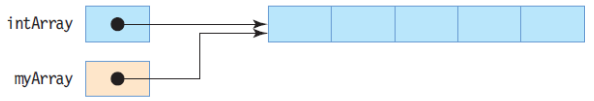
```
int intArray[] = {4, 3, 2, 1, 0};
float floatArray[] = {0.01, 0.02, 0.03, 0.04};
```



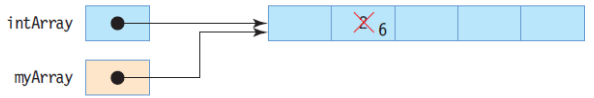

# Array

- 배열 참조
  - 생성된 1개의 배열을 다수의 레퍼런스가 참조 가능

```
int intArray[] = new int[5];
int myArray[] = intArray;
```




```
intArray[1] = 2;
myArray[1] = 6;
```



## Array

- 배열 원소 접근
  - 반드시 배열 생성 후 접근

```
int intArray []; // 배열 선언
intArray[4] = 8; // 오류, 아직 intArray 배열의 메모리가 할당되지 않았음
```

  - 배열 변수명과 [] 사이에 원소의 인덱스를 적어 접근
    - 배열의 인덱스는 0부터 시작
    - 배열의 마지막 항목의 인덱스는 (배열 크기 – 1)

```
int[] intArray; // 배열 선언
intArray = new int[10]; // 배열 생성

intArray[3]=6; // 배열에 값을 저장
int n = intArray[3]; // 배열로부터 값을 읽음
```

## 예제: Array 사용

- array를 사용하여 숫자를 입력받아 저장하고 입력된 숫자 중에 제일 큰 수를 화면에 출력

```
import java.util.Scanner;
public class ArrayAccess {
  public static void main (String[] args) {
    Scanner in = new Scanner(System.in);
    int intArray[] = new int[5];
    int max = 0;
    for (int i = 0; i < 5; i++) {
      System.out.print( " 숫자를 입력하
      intArray[i] = in.nextInt();
      if (intArray[i] > max)
        max = intArray[i];
    }
    System.out.print("입력된 수에서 가장 큰 수는 " + max + "입니다.");
  }
}
```

```
숫자를 입력하시오:1
숫자를 입력하시오:39
숫자를 입력하시오:78
숫자를 입력하시오:100
숫자를 입력하시오:99
입력된 수에서 가장 큰 수는
100입니다.
```

## 예제 : 문자열 Array 사용

- 문자열 array를 사용하여 5가지 피자 토핑을 출력

```
public class PizzaTopping {
    public static void main(String[] args) {
        String[] toppings = { "Pepperoni", "Mushrooms", "Onions", "Sausage", "Bacon" };
        for (int i = 0; i < toppings.length; i++) {
            System.out.print(toppings[i] + " ");
        }
    }
}
```

```
Pepperoni Mushrooms Onions Sausage Bacon
```

## Array

- 배열 인덱스
  - 인덱스는 0부터 시작하며 마지막 인덱스는 (배열 크기 -1)
  - 인덱스는 정수 타입만 가능

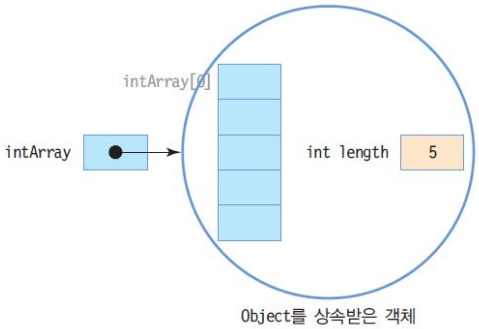
```
int intArray [] = new int[5]; // 인덱스는 0~4까지 가능
int n = intArray[-2];  // 실행 오류. -2는 인덱스로 적합하지 않음
int m = intArray[5];  // 실행 오류. 5는 인덱스의 범위(0~4)를 넘었음
```

- 배열의 크기
  - 배열의 크기는 배열 레퍼런스 변수를 선언할 때 결정되지 않음
    - 배열의 크기는 배열 생성 시에 결정되며, 나중에 바꿀 수 없음
  - 배열의 크기는 배열의 length 필드에 저장

```
int size = intArray.length;
```

## Array

```
int intArray[];
intArray = new int[5];

int size = intArray.length;
// size는 5
```




## 예제 : Array를 사용한 원소의 평균 구하기

- 성적을 5개 입력 받아 배열에 저장하고 평균 성적을 계산

```
import java.util.Scanner;
public class ArrayTest2 {
    public static void main(String[] args) {
        final int STUDENTS = 5;
        int total = 0;
        Scanner scan = new Scanner(Systen
        int[] scores = new int[STUDENTS];
        for (int i = 0; i < scores.length; i++) {
            System.out.print("성적을 입력하
            scores[i] = scan.nextInt();
        }
        for (int i = 0; i < scores.length; i++)
            total += scores[i];
        System.out.println("평균 성적은" + total / STUDENTS + "입니다");
    }
}
```

```
성적을 입력하시오:10
성적을 입력하시오:20
성적을 입력하시오:30
성적을 입력하시오:40
성적을 입력하시오:50
평균 성적은 30입니다.
```

## 예제 : 순차 탐색 (Sequential Search)

```
import java.util.Scanner;
public class SeqSearch {
    public static void main(String[] args) {
        int s[] = { 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };
        int value, index = -1;
        Scanner scan = new Scanner(System.in);
        System.out.print("탐색할 값을 입력하시오: ");
        value = scan.nextInt();
        for (int i = 0; i < s.length; i++) {
            if (s[i] == value)
                index = i;
        }
        if (index < s.length && index >= 0)
            System.out.println("" + value + "값은 " + index + "위치에 있습니다.");
    }
}
```

```
탐색할 값을 입력하시오:50
50값은 5위치에 있습니다.
```

## Array & For-each

- For-each 문
  - 배열이나 나열(enumeration)의 각 원소를 순차적으로 접근하는데 유용한 for 문

```
int[] num = { 1,2,3,4,5 };
int sum = 0;
// 반복될 때마다 k는 num[0], num[1], ..., num[4] 값으로 설정
for (int k : num)
  sum += k;
System.out.println("합은 " + sum);
```

```
합은 15
```

```
String names[] = { "사과", "배", "바나나", "체리", "딸기", "포도" } ;
// 반복할 때마다 s는 names[0], names[1], ..., names[5] 로 설정
for (String s : names)
  System.out.print(s + " ");
```

```
사과 배 바나나 체리 딸기 포도
```

## Anonymous Array

- 자바에서는 배열의 이름을 지정하지 않고 단순히 초기값만으로 배열을 생성시킬 수 있음
- 무명 배열 (Anonymous array)는 즉시 배열을 만들어서 함수의 인수로 전달하고자 할 때 많이 사용됨

**new int[] { 1, 2, 3, 4, 5 }; // 배열의 이름이 없다**
**// 초기값을 가지는 무명 배열 생성**

```
public static int sum(int[] numbers) {
    int total = 0;
    for (int i=0; i<numbers.length; i++) {
        total = total + numbers[i];
    }
    return total;
}
public static void main(String[] args) {
    System.out.println("합은 : " + sum(new int[] {1, 2, 3, 4}));
}
```

합은 10

## 2D Array

- 2차원 배열 선언

```
int     intArray[][];
char    charArray[][];
float   floatArray[][];
```

또는

```
int[][]    intArray;
char[][]   charArray;
float[][]  floatArray;
```

- 2차원 배열 생성

```
intArray = new int[2][5];
charArray = new char[5][5];
floatArray = new float[5][2];
```

또는

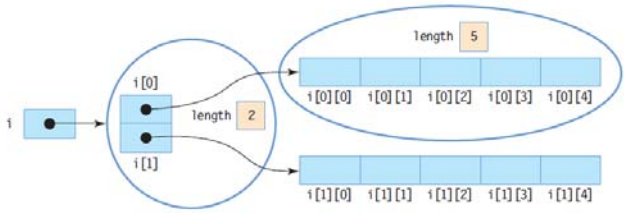
```
int     intArray[] = new int[2][5];
char    charArray[] = new char[5][5];
float   floatArray[] = new float[5][2];
```

- 2차원 배열 선언, 생성, 초기화

```
int intArray[][] = {{0,1,2},{3,4,5},{6,7,8}};
char charArray[][] = {{'a', 'b', 'c'},{'d', 'e', 'f'}};
float floatArray[][] = {{0.01, 0.02}, {0.03, 0.04}};
```

## 2D Array

```
int i[][] = new int[2][5];
int size1 = i.length; // 2
int size2 = i[0].length; // 5
int size3 = i[1].length; // 5
```




- 2차원 배열의 length
  - i.length -> 2차원 배열의 행의 개수로서 **2**
  - i[n].length는 n번째 행의 열의 개수
    - i[0].length -> 0번째 행의 열의 개수로서 **5**
    - i[1].length -> 1번째 행의 열의 개수로서 역시 **5**

## 예제 : 2D Array 사용

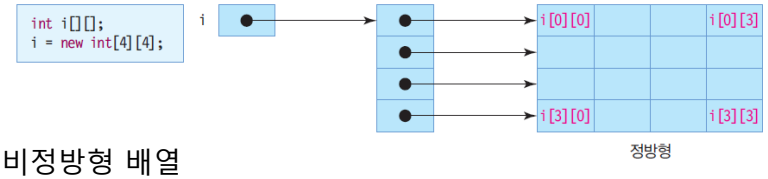
지난 3년간 매출 총액과 연평균 매출을 계산

```
public class SalesRevenue {
  public static void main (String[] args) {
    int intArray[][] = {{90, 90, 110, 110},
                        {120, 110, 100, 110},
                        {120, 140, 130, 150}} ;
    double sum = 0;

    for (int i = 0; i < intArray.length; i++) // intArray.length=3
      for (int j = 0; j < intArray[i].length; j++) // intArray[i].length=4
        sum += intArray[i][j];

    System.out.println("지난 3년간 매출 총액은 " + sum +
      "이며 연평균  매출은 " +  sum/intArray.length + "입니다.");
  }
}
```

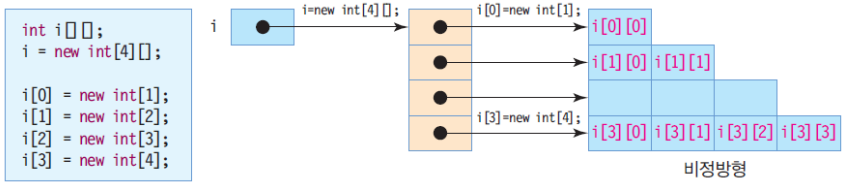
지난 3년간 매출 총액은 1380.0이며 연평균 매출은 460.0입니다.

# 비정방형 배열

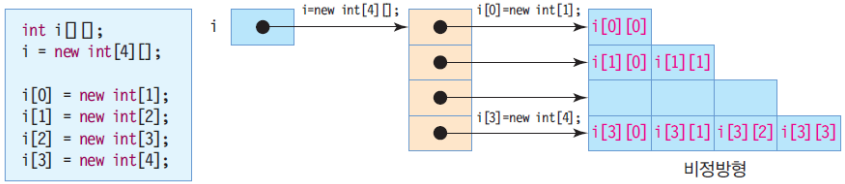
- 정방형 배열
  - 각 행의 열의 개수가 같은 배열




- 비정방형 배열
  - 각 행의 열의 개수가 다른 배열
  - 비정방형 배열의 생성



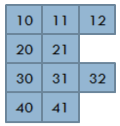

# 비정방형 배열의 length




- 비정방형 배열의 length
  - i.length -> 2차원 배열의 행의 개수로서 4
  - i[n].length는 n번째 행의 열의 개수
    - i[0].length -> 0번째 행의 열의 개수로서 1
    - i[1].length -> 1번째 행의 열의 개수로서 2
    - i[2].length -> 2번째 행의 열의 개수로서 3
    - i[3].length -> 3번째 행의 열의 개수로서 4

# 예제 : 비 정방형 배열의 생성과 접근

다음 그림과 같은 비정방형 배열을 만들어 값을 초기화하고 출력하시오.

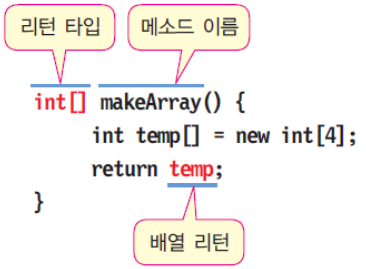



```
public class IrregularArray {
  public static void main (String[] args) {
    int intArray[][] = new int[4][];
    intArray[0] = new int[3];
    intArray[1] = new int[2];
    intArray[2] = new int[3];
    intArray[3] = new int[2];

    for (int i = 0; i < intArray.length; i++)
      for (int j = 0; j < intArray[i].length; j++)
        intArray[i][j] = (i+1)*10 + j;
    for (int i = 0; i < intArray.length; i++) {
      for (int j = 0; j < intArray[i].length; j++)
        System.out.print(intArray[i][j]+" ");
      System.out.println();
    }
  }
}
```

```
10 11 12
20 21
30 31 32
40 41
```

# 메소드에서 배열 리턴

- 메소드의 배열 리턴
  - 배열의 레퍼런스만 리턴
- 메소드의 리턴 타입
  - 메소드가 리턴하는 배열의 타입은 리턴 받는 배열 타입과 일치
  - 리턴 타입에 배열의 크기를 지정하지 않음



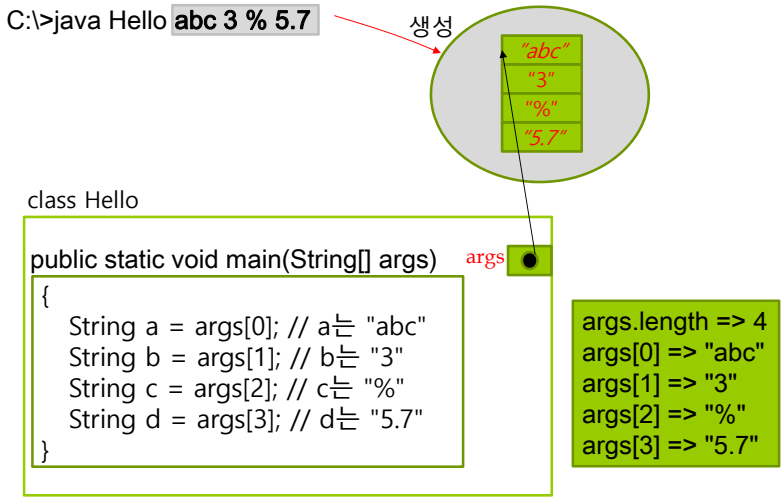
## 예제 : 배열 리턴 사용

배열을 생성하고 각 원소 값을 출력하는 프로그램을 작성하시오. 배열 생성은 배열을 생성하여 각 원소의 인덱스 값으로 초기화하여 반환하는 메소드를 이용한다.

```
public class ReturnArray {
  static int[] makeArray() {
    int temp[] = new int[4];
    for (int i=0;i<temp.length;i++)
      temp[i] = i;
    return temp;
  }
  public static void main(String[] args) {
    int intArray [];
    intArray = makeArray();
    for (int i = 0; i < intArray.length; i++)
      System.out.println(intArray[i]);
  }
}
```

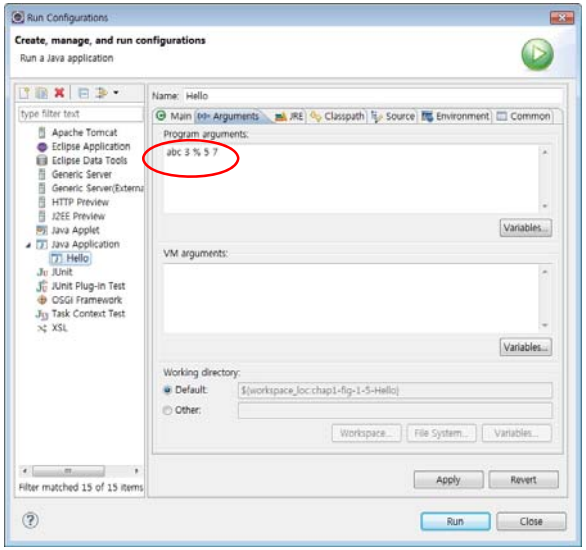
```
0
1
2
3
```

## main(string [] args) 메소드의 인자 전달




## 이클립스에서 main() 메소드의 인자전달

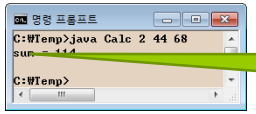
Run 메뉴의 Run Configurations 항목에서 main() 메소드의 인자 나열




## main()의 인자 이용 예

```
public class Calc {
  public static void main(String[] args) {
    int sum = 0;
    for(int i=0; i<args.length; i++) { // 명령행 인자의 개수만큼 반복
      int n = Integer.parseInt(args[i]); // 명령행 인자인 문자열을 정수로 변환
      sum += n; // 숫자를 합한다.
    }
    System.out.println("sum = " + sum);
  }
}
```

Integer.parseInt()는 매개변수로 주어진 문자열을 정수로 변환. Integer.parseInt("68") 은 정수 68 리턴



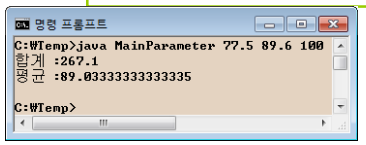

명령행 인자 2, 44, 68을 모두 합하여 114 출력

## 예제 : main()의 인자들을 받아서 평균값을 계산하는 예제

여러 개의 실수를 main() 메소드 인자로 전달받아 평균값을 구하는 프로그램을 작성하시오.

```
public class MainParameter {
  public static void main (String[] args) {
    double sum = 0.0;

    for (int i=0; i<args.length; i++)
      sum += Double.parseDouble(args[i]);
    System.out.println("합계 :" + sum);
    System.out.println("평균 :" +
sum/args.length);
  }
}
```

Double.parseDouble()는 매개변수로 주어진 문자열을 실수로 변환. Double.parseDouble("77.5") 은 실수 77.5 리턴




## 예제 : 정수가 아닌 문자열을 정수로 변환할 때 예외 발생

문자열을 정수로 변환할 때 발생하는 NumberFormatException을 처리하는 프로그램을 작성하라.

```
public class NumException {
  public static void main (String[] args) {
    String[] stringNumber = {"23", "12", "998", "3.141592"};
    try {
      for (int i = 0; i < stringNumber.length; i++) {
        int j = Integer.parseInt(stringNumber[i]);
        System.out.println("숫자로 변환된 값은 " + j);
      }
    }
    catch (NumberFormatException e) {
      System.out.println("정수로 변환할 수 없습니다.");
    }
  }
}
```

"3.141592"를 정수로 변환할 때 NumberFormatException 예외 발생

```
숫자로 변환된 값은 23
숫자로 변환된 값은 12
숫자로 변환된 값은 998
정수로 변환할 수 없습니다.
```

## 예외와 예외 클래스

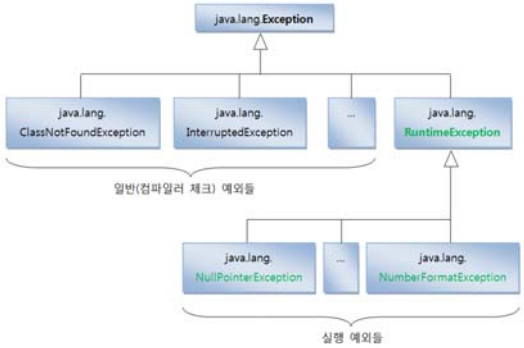
- 오류의 종류
  - 에러(Error)
    - 하드웨어의 잘못된 동작 또는 고장으로 인한 오류
    - 에러가 발생되면 JVM실행에 문제가 있으므로 프로그램 종료
    - 정상 실행 상태로 돌아갈 수 없음
  - 예외(Exception)
    - 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인한 오류
    - 예외가 발생되면 프로그램 종료
    - "예외 처리" 추가하면 정상 실행 상태로 돌아갈 수 있음

## 예외와 예외 클래스

- 예외의 종류
  - 일반 예외(컴파일 체크 Exception)
    - 컴파일하는 과정에서 예외 처리 코드가 필요한지 검사
    - 예외 처리 코드 없으면 컴파일 오류 발생
  - **실행 예외(RuntimeException)**
    - 예외 처리 코드를 생략하더라도 컴파일이 되는 예외
    - '경험'따라 예외 처리 코드 작성 필요

# 예외 클래스

- 예외 클래스
  - **Java는 예외를 클래스로 관리**
  - JVM이 프로그램을 실행하는 도중에 예외가 발생하면 해당 예외 클래스로 객체를 생성
    - 예외 처리코드에서 예외 객체를 이용




# 자주 발생하는 예외

| 예외 종류 | 예외 발생 경우 |
|---|---|
| ArithmeticException | 정수를 0으로 나눌 때 발생 |
| NullPointerException | null 레퍼런스를 참조할 때 발생 |
| ClassCastException | 변환할 수 없는 타입으로 객체를 변환할 때 발생 |
| OutOfMemoryError | 메모리가 부족한 경우 발생 |
| ArrayIndexOutOfBoundsException | 배열의 범위를 벗어난 접근 시 발생 |
| IllegalArgumentException | 잘못된 인자 전달 시 발생 |
| IOException | 입출력 동작 실패 또는 인터럽트 시 발생 |
| NumberFormatException | 문자열이 나타내는 숫자와 일치하지 않는 타입의 숫자로 변환 시 발생 |

# 실행 예외(RuntimeException)

- **NullPointerException**
  - 객체 참조가 없는 상태
    - null 값 갖는 참조변수로 객체 접근 연산자인 도트(.) 사용했을 때 발생

```
String data = null;
System.out.println(data.toString());
```

- **ArrayIndexOutOfBoundsException**
  - 배열에서 인덱스 범위 초과하여 사용할 경우 발생

```
public static void main(String[] args) {
  String data1 = args[0];
  String data2 = args[1];

  System.out.println("args[0]: " + data1);
  System.out.println("args[1]: " + data2);
}
```

실행시 매개값을 주지 않을 경우 예외 발생

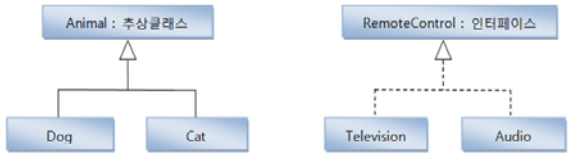
# 실행 예외(RuntimeException)

- **NumberFormatException**
  - 숫자로 변환될 수 없는 문자가 포함되어 있는 문자열을 숫자로 변경할 경우
    - Integer.parseInt(String s)
    - Double.parseDouble(String s)

```
public class NumberFormatExceptionExample {
    public static void main(String[] args) {
        String data1 = "100";
        String data2 = "a100";

        int value1 = Integer.parseInt(data1);
        int value2 = Integer.parseInt(data2);

        int result = value1 + value2;
        System.out.println(data1 + "+" + data2 + "=" + result);
    }
}
```

# 실행 예외(RuntimeException)

- **ClassCastException**
  - 타입 변환이 되지 않을 경우 발생




  - 정상 코드

```
Animal animal = new Dog();
Dog dog = (Dog) animal;
```

```
RemoteControl rc = new Television();
Television tv = (Television) rc;
```

  - 예외 발생 코드

```
Animal animal = new Dog();
Cat cat = (Cat) animal;
```

```
RemoteControl rc = new Television();
Audio audio = (Audio) rc;
```

# 실행 예외(RuntimeException)

- **ClassCastException**
  - 타입 변환이 되지 않을 경우 발생

```
public class NumberFormatExceptionExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        changeDog(dog);
        Cat cat = new Cat();
        changeDog(cat);
    }
    public static void changeDog(Animal animal) {
        //if(animal instanceof Dog) {
            Dog dog = (Dog) animal;     //ClassCastException 발생 가능
        //}
    }
}
class Animal {}
class Dog extends Animal {}
class Cat extends Animal {}
```

# 예제 : ArithmeticException 예외 처리

try-catch문을 이용하여 정수를 0으로 나누려고 할 때 "0으로 나룰 수 없습니다."라는 경고 메시지를 출력하도록 프로그램을 작성하시오.

```
import java.util.Scanner;
public class ExceptionExample2 {
   public static void main (String[] args) {
      Scanner rd = new Scanner(System.in);
      int divisor = 0;
      int dividend = 0;
      System.out.print("나뉨수를 입력하시오:");
      dividend = rd.nextInt();
      System.out.print("나눗수를 입력하시오:");
      divisor = rd.nextInt();
      try {
         System.out.println(dividend+"를 "+divisor+"로 나누면 몫은 "+
dividend/divisor+"입니다.");
      } catch (ArithmeticException e) {
         System.out.println("0으로 나눌 수 없습니다.");
      }
   }
}
```

ArithmeticException 예외 발생

```
나뉨수를 입력하시오:100
나눗수를 입력하시오:0
0으로 나눌 수 없습니다.
```

# 예제 : 범위를 벗어난 배열의 접근

배열의 인덱스가 범위를 벗어날 때 발생하는 ArrayIndexOutOfBoundsException을 처리하는 프로그램을 작성하시오.

```
public class ArrayException {
   public static void main (String[] args) {
      int[] intArray = new int[5];
      intArray[0] = 0;
      try {
         for (int i = 0; i < 5; i++) {
            intArray[i+1] = i+1 + intArray[i];
            System.out.println("intArray["+i+"]"+"="+intArray[i]);
         }
      }
      catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("배열의 인덱스가 범위를 벗어났습니다.");
      }
   }
}
```

i가 4일 때 ArrayIndexOutOfBoundsException 예외 발생

```
intArray[0]=0
intArray[1]=1
intArray[2]=3
intArray[3]=6
배열의 인덱스가 범위를 벗어났습니다.
```

## 예제 : 정수가 아닌 문자열을 정수로 변환할 때 예외 발생

문자열을 정수로 변환할 때 발생하는 NumberFormatException을 처리하는 프로그램을 작성하라.

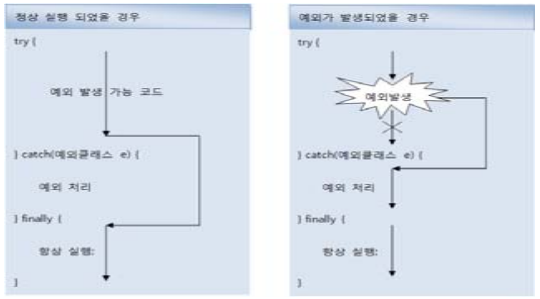
```
public class NumException {
  public static void main (String[] args) {
    String[] stringNumber = {"23", "12", "998", "3.14", "Hello"};
    try {
      for (int i = 0; i < stringNumber.length; i++) {
        int j = Integer.parseInt(stringNumber[i]);
        System.out.println("숫자로 변환된 값은 " + j);
      }
    }
    catch (NumberFormatException e) {
      System.out.println("정수로 변환할 수 없습니다.");
    }
  }
}
```

"3.141592"를 정수로 변환할 때 NumberFormatException 예외 발생

```
숫자로 변환된 값은 23
숫자로 변환된 값은 12
숫자로 변환된 값은 998
정수로 변환할 수 없습니다.
```

## 예외처리 코드 (try-catch-finally)

- 예외처리 코드
  - 예외 발생시 프로그램 종료 막고, 정상 실행 유지할 수 있도록 처리
    - 일반 예외: 반드시 작성해야 컴파일 가능
    - 실행 예외: 컴파일러가 체크해주지 않으며 개발자 경험 의해 작성
  - **try – catch – finally** 블록 이용해 예외 처리 코드 작성

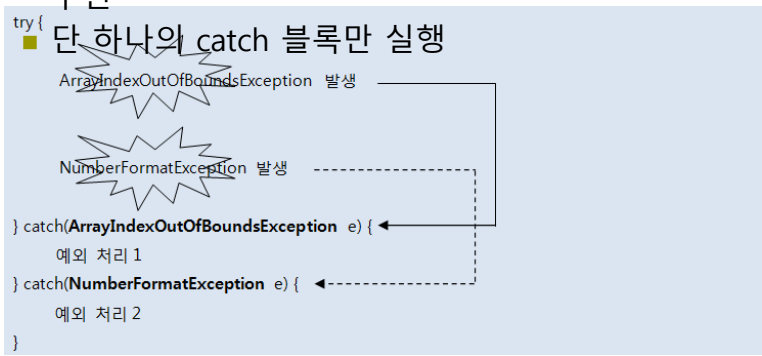



## 예외 처리 코드(try-catch-finally)

```
public class TryCatchFinallyRuntimeExceptionExample {
    public static void main(String[] args) {
        String data1 = null;
        String data2 = null;
        try {                                          실행시 매개값을 주지 않을 경우 예외 발생
            data1 = args[0];
            data2 = args[1];
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("실행 매개값의 수가 부족합니다.");
            System.out.println("[실행 방법]");
            System.out.println("java TryCatchFinallyRuntimeExceptionExample num1 num2");
            return;
        }
        try {
            int value1 = Integer.parseInt(data1);      실행시 매개값을 잘못 주었을 경우 예외 발생
            int value2 = Integer.parseInt(data2);
            int result = value1 + value2;
            System.out.println(data1 + "+" + data2 + "=" + result);
        } catch(NumberFormatException e) {
            System.out.println("숫자로 변환할 수 없습니다.");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

## 예외 종류에 따른 처리 코드

- **다중 catch**
  - 하나의 try 블록 내부에서 다양한 종류의 예외 발생시
  - 각 예외 별로 예외 처리 코드(catch 블록) 다르게 구현
  - 단 하나의 catch 블록만 실행



## 예외 종류에 따른 처리 코드

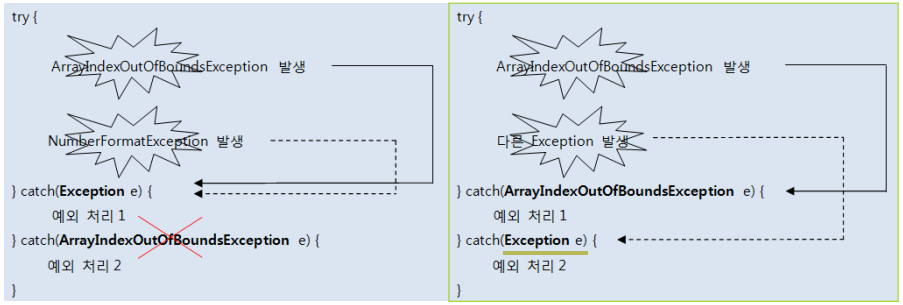
```
public class CatchByExceptionKindExample {
    public static void main(String[] args) {
        try {
            String data1 = args[0];
            String data2 = args[1];
            int value1 = Integer.parseInt(data1);
            int value2 = Integer.parseInt(data2);
            int result = value1 + value2;
            System.out.println(data1 + "+" + data2 + "=" + result);
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("실행 매개값의 수가 부족합니다.");
            System.out.println("[실행 방법]");
            System.out.println("java CatchByExceptionKindExample  num1  num2");
        } catch(NumberFormatException e) {
            System.out.println("숫자로 변환할 수 없습니다.");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

실행시 매개값을 주지 않을 경우 예외 발생

실행시 매개값을 잘못 주었을 경우 예외 발생

## 예외 종류에 따른 처리 코드

- 하위 예외는 상위 예외를 상속
  - 하위 예외는 상위 예외 타입도 됨
- **catch** 순서 – 상위 예외가 아래에 위치해야




상위 예외 Exception이 위에 있을 경우

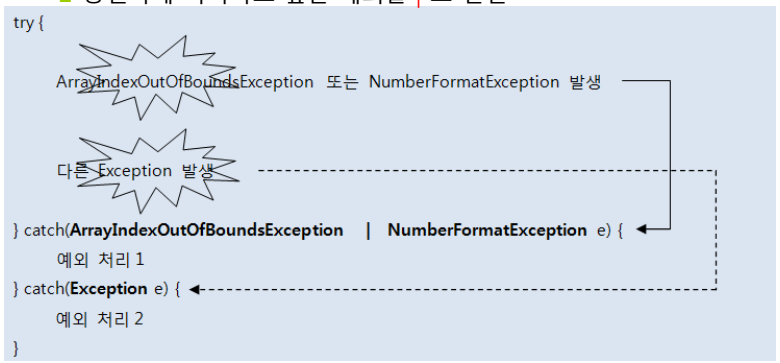
상위 예외 Exception이 아래에 있을 경우

## 예외 종류에 따른 처리 코드

```
public class CatchOrderExample {
    public static void main(String[] args) {
        try {
            String data1 = args[0];
            String data2 = args[1];
            int value1 = Integer.parseInt(data1);
            int value2 = Integer.parseInt(data2);
            int result = value1 + value2;
            System.out.println(data1 + "+" + data2 + "=" + result);
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("실행 매개값의 수가 부족합니다.");
        } catch(Exception e) {
            System.out.println("실행에 문제가 있습니다.");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

## 예외 종류에 따른 처리 코드

- **멀티(multi) catch**
  - 자바 7부터는 하나의 catch 블록에서 여러 개의 예외 처리 가능
    - 동일하게 처리하고 싶은 예외를 | 로 연결



## 예외 종류에 따른 처리 코드

```
public class MultiCatchExample {
	public static void main(String[] args) {
		try {
			String data1 = args[0];
			String data2 = args[1];
			int value1 = Integer.parseInt(data1);
			int value2 = Integer.parseInt(data2);
			int result = value1 + value2;
			System.out.println(data1 + "+" + data2 + "=" + result);
		} catch(ArrayIndexOutOfBoundsException | NumberFormatException e) {
			System.out.println("실행 매개값의 수가 부족하거나 숫자로 변환할 수
없습니다..");
		} catch(Exception e) {
			System.out.println("알수 없은 예외 발생");
		} finally {
			System.out.println("다시 실행하세요.");
		}
	}
}
```

실행시 매개값을 주지 않을 경우 예외 발생

실행시 매개값을 잘못 주었을 경우 예외 발생

## 자동 리소스 닫기

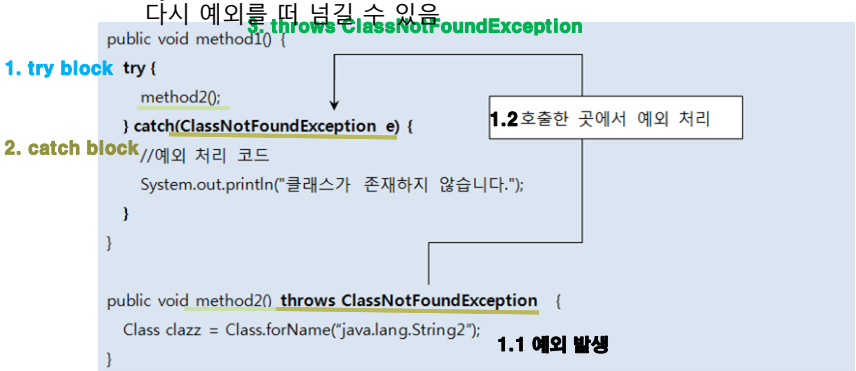
- try-with-resources
  - 예외 발생 여부와 상관 없음
  - 사용했던 리소스 객체의 close() 메소드 호출해 리소스 닫음
  - 리소스 객체
    - 각종 입출력스트림, 서버소켓, 소켓, 각종 채널
    - java.lang.AutoCloseable 인터페이스 구현하고 있어야 함

## 예외 떠 넘기기(throws)

- **throws**
  - 메소드 선언부 끝에 작성
  - 메소드내에서 처리하지 않은 예외를 메소드 호출한 곳(calling method)으로 떠 넘기는 역할

## 예외 떠 넘기기(throws)

- **throws**
  - throws 선언된 메소드를 호출하는 메소드(calling method)는
    1. 반드시 try 블록 내에서 호출
    2. catch 블록에서 떠 넘겨 받은 예외를 처리함
    3. try-catch 블록으로 예외처리를 하지 않고 throws 키워드로 자신도 다시 예외를 떠 넘길 수 있음



# 사용자 정의 예외와 예외 발생

- 사용자 정의 예외(user-defined exception) 클래스 선언
  - 자바 표준 API에서 제공하지 않는 예외
  - 애플리케이션 서비스와 관련된 예외, Application Exception
    - E.g. 잔고 부족 예외, 계좌 이체 실패 예외, 회원 가입 실패 예외....
  - 사용자 정의 예외 클래스 선언 방법
    1. 예외 클래스 상속
       - 일반 예외 : Exception class 상속
       - 실행 예외 : RuntimeException class 상속
    2. 생성자 정의
       - 매개변수 없는 기본 생성자, String 타입의 매개변수를 갖는 생성자

```
public class XXXException extends [ Exception | RuntimeException ] {
    public XXXException() { }
    public XXXException(String message) { super(message); }
}
```

# 사용자 정의 예외와 예외 발생

- 예외 발생 시키기(throw)
  - 코드에서 예외 발생시키는 법 - 예외 객체 생성

```
throw new XXXException()
throw new XXXException("메시지");
```

  - 호출한 곳에서 발생한 예외를 처리하도록

```
public void method() throws XXXException {
    throw new XXXException("메시지");
}
```

# 사용자 정의 예외와 예외 발생

```
public class BalanceInsufficientException extends Exception {
    public BalanceInsufficientException() { }
    public BalanceInsufficientException(String message) {
        super(message);
    }
}
```

1. Exception class 상속

2. constrctors 생성

```
public class Account {
    private long balance;

    public Account() { }

    public long getBalance() {
        return balance;
    }
    public void deposit(int money) {
        balance += money;
    }
    public void withdraw(int money) throws BalanceInsufficientException {
        if(balance < money) {
            throw new BalanceInsufficientException("잔고부족:"+(money-balance)+" 모자람");
        }
        balance -= money;
    }
}
```

4. 호출한 곳에서 발생한 예외를 처리하도록 함

3. 사용자 정의 예외 발생 – 예외 객체 생성

# 사용자 정의 예외와 예외 발생

```
public class AccountExample {
    public static void main(String[] args) {

        Account account = new Account();

        //예금하기
        account.deposit(10000);
        System.out.println("예금액: " + account.getBalance());

        //출금하기
        try {
            account.withdraw(30000);
        } catch(BalanceInsufficientException e) {
            String message = e.getMessage();
            System.out.println(message);
            System.out.println();
            e.printStackTrace();
        }
    }
}
```

5. try block에서 메소드 호출

6. catch block에서 사용자정의 예외 처리

7. 예외 정보 얻기(8절)

# 예외 정보 얻기

## getMessage()

- 예외 발생시킬 때 생성자 매개값으로 사용한 메시지 리턴

```
throw new XXXException("예외  메시지");
```

- 원인 세분화하기 위해 예외 코드 포함(예: 데이터베이스 예외 코드)

- catch() 절에서 활용

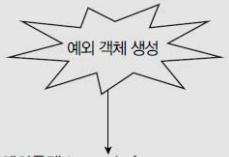
```
} catch(Exception  e) {
    String message  = e.getMessage();
}
```

# 예외 정보 얻기

## printStackTrace()

- 예외 발생 코드를 추적하여 모두 콘솔에 출력

```
try {
```




```
} catch(예외클래스    e) {
    //예외가 가지고 있는 Message 얻기
    String message = e.getMessage(
    //예외의 발생 경로를 추적
    e.printStackTrace();
}
```

Problems  Javadoc  Declaration  Console

<terminated> AccountExample [Java Application] C:₩Program Files₩Java₩jdk1.8.0_73₩bin₩javaw.exe (2

```
예금액: 10000
잔고부족:20000 모자람

BalanceInsufficientException: 잔고부족:20000 모자람
        at Account.withdraw(Account.java:14)
        at AccountExample.main(AccountExample.java:9)
```