

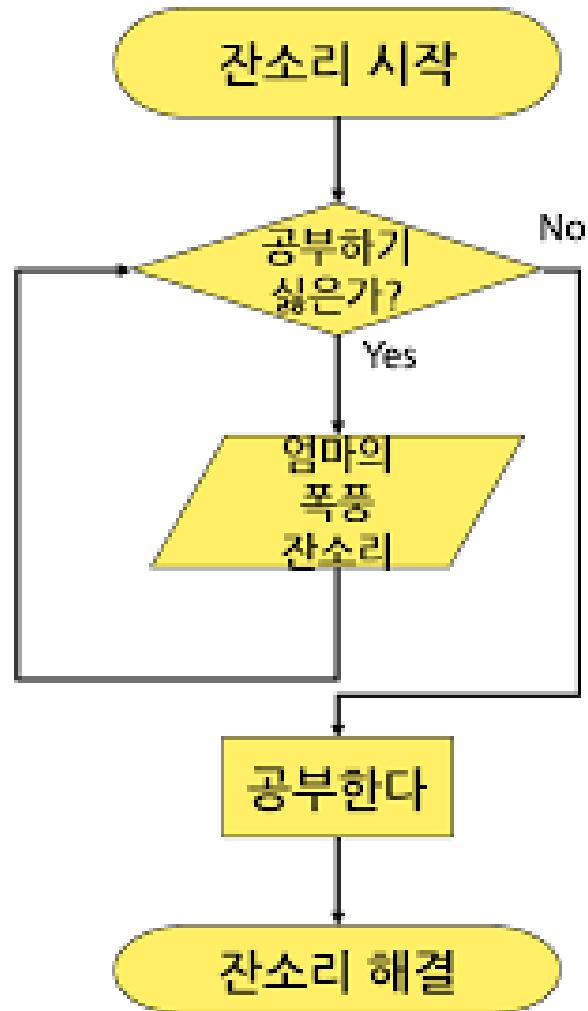
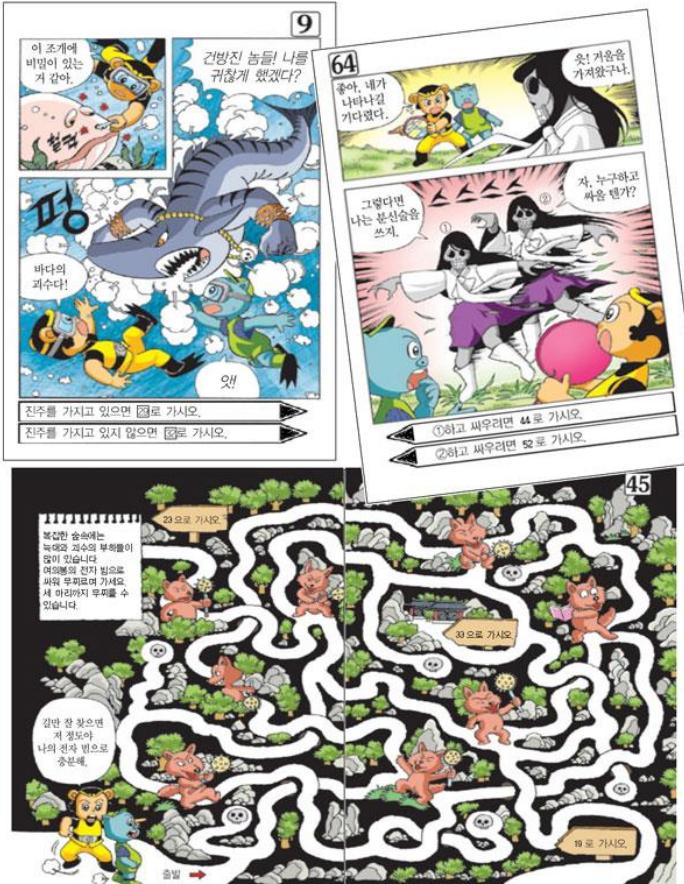
1강 Java란?

- 
- 1-1 프로그래밍이란?
 - 1-2 Java 의 역사
 - 1-3 Java 언어의 특징
 - 1-4 무엇을 할 것인가?
 - 1-5 Java 기본 설치

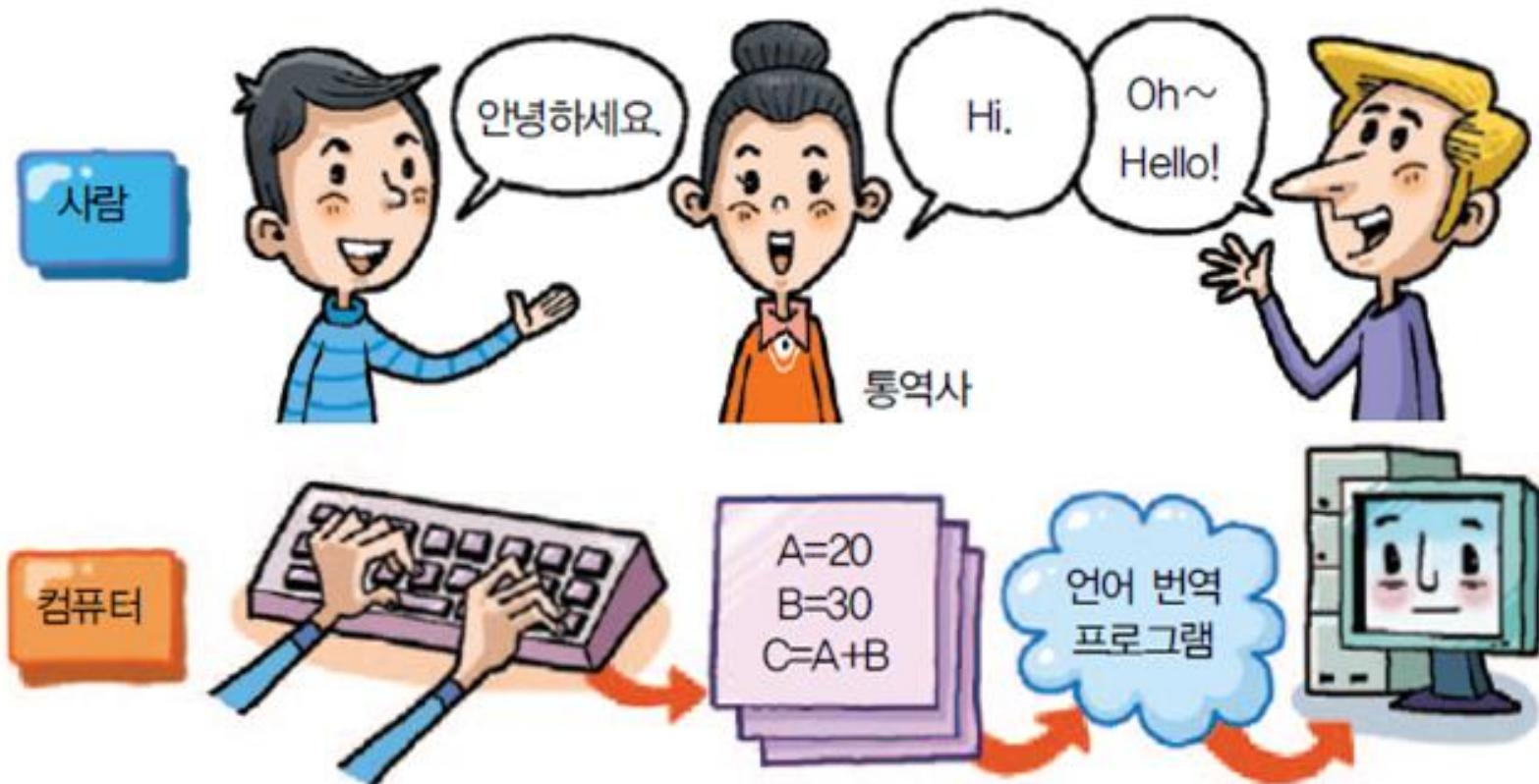
1-1 프로그래밍란?



1-1 프로그래밍란?



1-1 프로그래밍란?

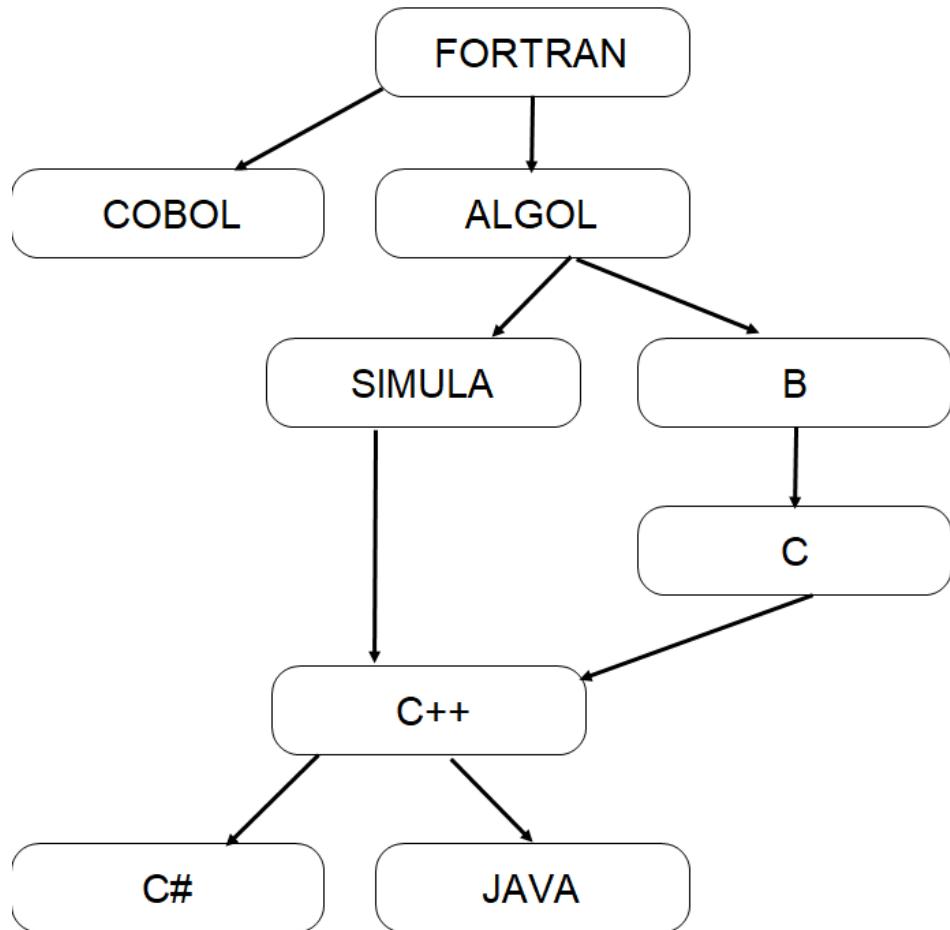


1-2 Java의 역사



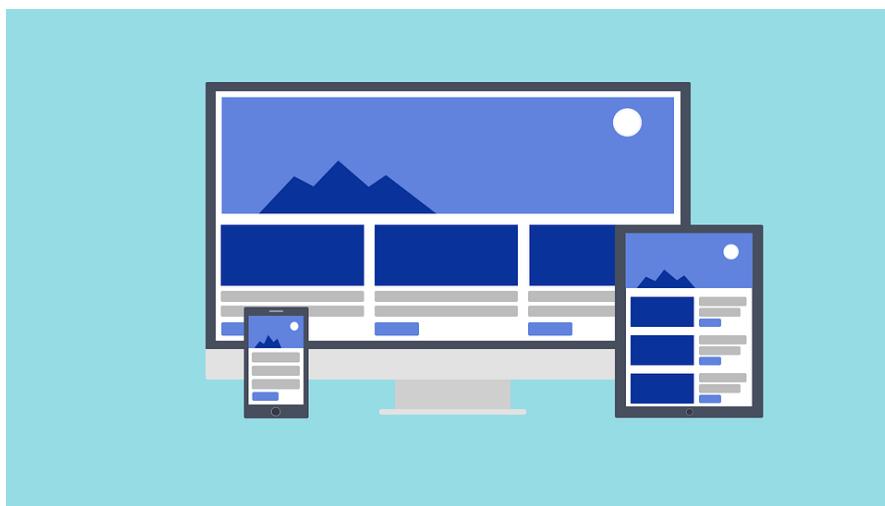
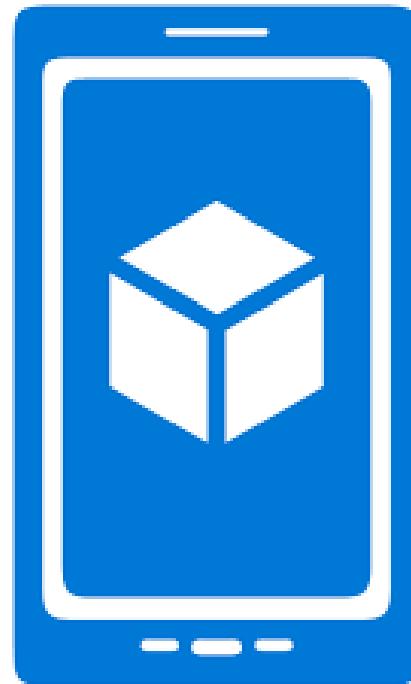
- 1995년 제임스 고슬링에 의해서 탄생
- Sun microsystems에서 발표
- Oak 언어에서 Java
- 가전제품에 탑재할 수 있는 프로그램을 개발하기 위한 목적

1-3 Java의 특징

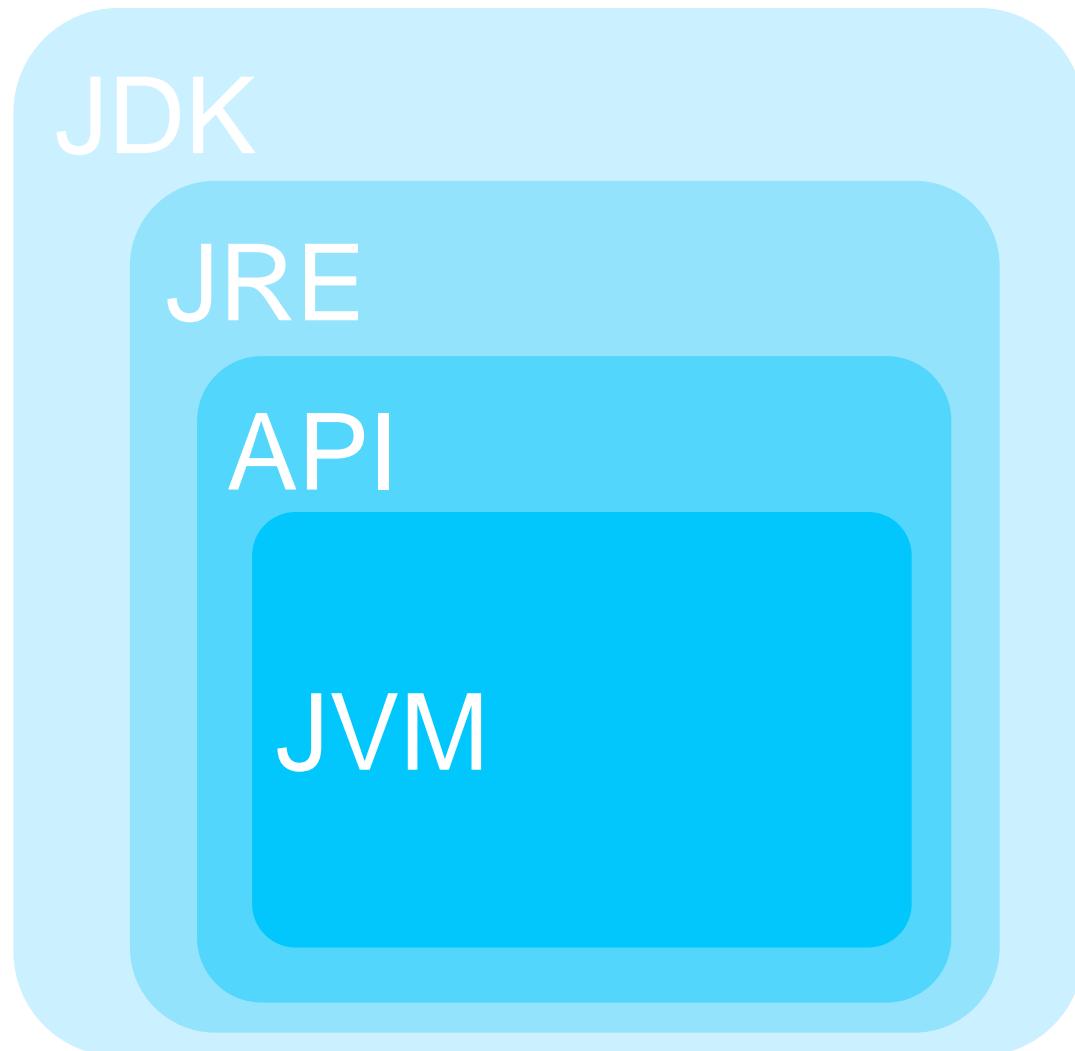


- ▣ 객체 지향 언어로 기능을 부풀화
- ▣ JRE를 이용해서 OS로부터 자유
- ▣ WEB 및 Mobile 개발이 쉬움
- ▣ GC를 통한 자동 메모리 관리
- ▣ 실행 속도 개선으로 빠른 속도

1-4 무엇을 할 것인가?



1-5 Java 설치



```
package org.tutorial.java.eclipse;           //디렉토리

public class HelloWorld {                   //파일의 이름

    public static void main(String[] args) { //프로그램의 시작
        System.out.print("Hello World\n");
        System.out.println("Hello World");   //출력문
        System.out.printf("%s", "Hello World");
    }
}
```

변수(Variable)

변하는 수?

하나의 값을 저장할 수 있는 기억공간

메모리상의 공간

“값을 담을 수 있는 그릇”

변수(Variable)

변수를 사용하려면 먼저 선언을 해야 합니다. 변수의 선언 형식은 다음과 같습니다.

```
int a;
```

① 데이터 타입 ② 변수명

```
int a = 1;
```

대입 연산자 초기 값

- ① 데이터 타입 : 변수에 기억시킬 데이터의 형태를 의미합니다.
- ② 변수명 : 기억장소 주소를 대신하여 사용할 이름입니다.
- ③ 초기 값 : 변수를 선언한 후 기억시킬 값입니다.

변수(Variable)

규칙

- 반드시 대소문자 구분
- 예약어는 사용불가
- 첫 글자는 반드시 A~Z, a~z, \$, _ 으로 시작
- \$, _를 제외한 어떤 특수문자도 사용할수 없음

변수(Variable)

변수명을 짓는 방법에는 헝가리안, 카멜, 파스칼 표기법 등이 있습니다.

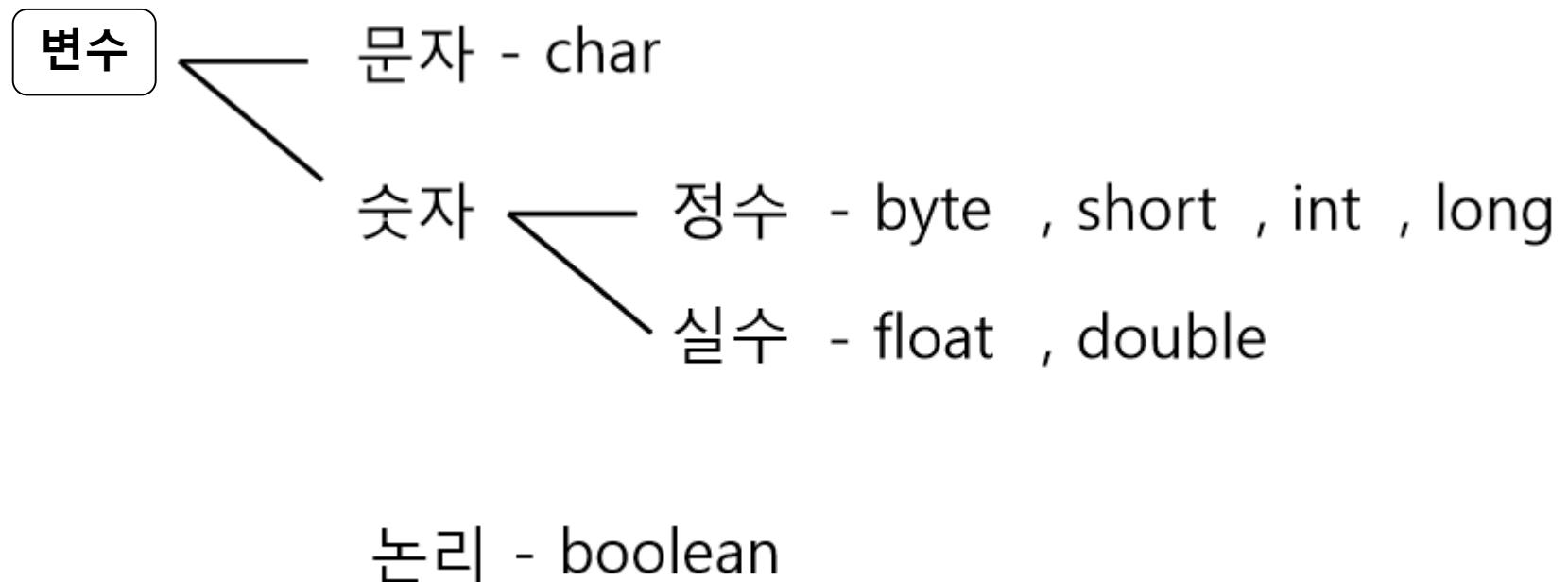
- 헝가리안 표기법 : 변수에 자료형을 구분 지을 수 있도록 접두사를 달아서 표기하는 방법입니다.

자료형	접두사	예시
int	i	int iNum;
float	f	float fEpsilon;
bool	b	bool bPonet;

- 카멜 표기법 : 두 개 이상의 단어가 사용된 경우 새로운 단어가 나타나면 대문자를 사용하여 표기하는 방법이며 대체로 함수명과 변수명을 지을 때 사용됩니다. ex) int myAge;, int newPeople;
- 파스칼 표기법 : 모든 단어의 첫 글자를 대문자로 표기하는 방법이며 클래스 명을 지을 때 사용됩니다.
ex) class GameEngine{};

변수(Variable)

Data Type



변수(Variable)

구분	데이터형	바이트 수	데이터 범위
정수	byte	1Byte	$-2^7 \sim 2^7 - 1$ (-128 ~ 127)
	short	2Byte	$-2^{15} \sim 2^{15} - 1$ (-32768 ~ 32767)
	int	4Byte	$-2^{31} \sim 2^{31} - 1$
	long	8Byte	$-2^{63} \sim 2^{63} - 1$
실수	float	4Byte	$-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$
	double	8Byte	$-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$
문자	char	2Byte	0 ~ $2^{16} - 1$ (Unicode 문자)
논리	boolean	-	true 또는 false

변수(Variable)

형변환이란?

변수와 상수의 데이터 타입을 다른 타입으로 변경

자동 형변환/명시적 형변환

자동 형변환은 별도의 작업 없이 알아서 타입변경
명시적 형변환은 변수의 자료형을 강제로 변경

형변환(Casting)

```
public class dataType {
```

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

```
        // TODO Auto-generated method stub
```

```
        byte a = 127; // byte가 가질 수 있는 최댓값
```

```
        int b = a; // 자동형변환( byte → int )
```

```
        System.out.println(b);
```

```
        float c = b; // 자동형변환( int → float )
```

```
        System.out.println(c);
```

```
}
```

```
}
```

```
public class dataType {
```

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

```
        // TODO Auto-generated method stub
```

```
        int a = 263;
```

```
        System.out.println(a);
```

```
        byte b = (byte) a; // 명시적 형변환
```

```
        System.out.println(b);
```

```
}
```

```
}
```

상수

상수란?

상수는 프로그램 중에
변경할 수 없는 고정된
값

EX)final datatype 상수명 = 값;

```
public class Prca {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        int a = 3;  
        a = 4;  
  
        final double PI = 3.14;  
        PI = 3.15; // 에러발생  
    }  
}
```

연산자

연산자?

특정한 연산을 하기 위해 사용하는 기호

종류	연산자	설명
산술 연산자	+ - * / %	사칙연산 및 나머지 연산
단항 연산자	-	부호 연산자
	!	논리 부정 연산자
	++ --	증감 연산자
비교 연산자	> < >= <= = !=	양변을 비교하는 연산
논리 연산자	&& ^	AND, OR, XOR 연산
쉬프트 연산자	<< >> >>>	비트를 이동하는 연산
기타 연산자	(조건) ? 참 : 거짓	삼항 연산자
	= += -= *=	대입 연산자

연산자

연산자 우선순위	종류	연산자
1	최우선 연산자	[], (), .
2	단항 연산자	!, ~, +, -, ++, --, ()
3	산술 연산자	+,-,*,/,%
4	쉬프트 연산자	<< >> >>>
5	비교 연산자	<, >, >=, <=, ==, !=
6	논리 연산자	&&,
7	삼항 연산자	(조건) ? 참: 거짓
8	대입 연산자	=, +=, -=, *= 등 (산술과 조합)

TIPS_ 연산의 방향

연산의 대부분은 왼쪽에서 오른쪽으로 실행하지만, 단항 연산자(!, ~, +, -, ++, --)와 대입 연산자(=, +=, -=, *=) 등은 오른쪽에서 왼쪽으로 실행합니다.

연산자

산술연산자?

변수 또는 상수와 함께 사용하여 기본적인 계산을 할 수 있다

연산자	의미	사용방법	설명
+	더하기	$c = a + b$	변수 c에 a와 b의 더한 결과 값을 대입
-	빼기	$c = a - b$	변수 c에 a에서 b를 뺀 결과 값을 대입
*	곱하기	$c = a * b$	변수 c에 a와 b를 곱한 결과 값을 대입
/	나누기	$c = a / b$	변수 c에 a를 b로 나눈 결과 값을 대입
%	나머지	$c = a \% b$	변수 c에 a를 b로 나눈 나머지를 대입

TIPS__ 나머지 연산자는 정수만 사용할 수 있으며 연산의 결과가 몫이 아닌 나머지 값을 구할 때 사용합니다.

연산자

단항연산자? 항이 오로지 하나인 연산자

연산자	의미	사용방법	설명
+, -	부호연산자	-a	변수 a의 부호를 바꿉니다.
!	부정연산자	!a	논리 부정 연산자는 참(true)을 거짓(false)으로, 거짓(false)을 참(true)으로 바꾸는 연산자입니다.
++	증가연산자	++a; 또는 a++;	변수 a에 1을 더하여 a에 기억시킨다.
--	감소연산자	--a; 또는 a--;	변수 a에 1을 뺀 후 a에 기억시킨다.

연산자

```
package java_practice;

public class Operator {
    public static void main(String[] args) {
        int a = 10; // a에 10을 대입
        int b = 3; // b에 3을 대입
        System.out.println(a + b);
        System.out.println(a - b);
        System.out.println(a * b);
        System.out.println(a / b);
        System.out.println(a % b);
    }
}
```

```
package java_practice;

public class Sign_operator {
    public static void main(String[] args) {
        int a = -1; // a에 -1을 대입합니다.
        int b = 2; // b에 2를 대입합니다.
        System.out.println(a); // a를 출력합니다.
        System.out.println(-b); // b의 부호를 바꿔 출력합니다.
    }
}
```

연산자

```
package java_practice;
```

```
public class Logic_operator {
```

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

```
        boolean a = true;
```

```
        boolean b = false;
```

```
        boolean c = !b; // b의 값을 반대로 바꾸어 대입합니다.
```

```
        System.out.println(a); // true가 출력됩니다.
```

```
        System.out.println(!a); // 값을 바꾸어 출력합니다.
```

```
        System.out.println(b); // false가 출력됩니다.
```

```
        System.out.println(c); // true가 출력됩니다.
```

```
}
```

```
package java_practice;
```

```
public class operator_ch3 {
```

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

```
        int a = 1;
```

```
        System.out.println(a); // 현재 a는 1
```

```
        a++; // a의 값을 1 증가
```

```
        System.out.println(a); // 증가되었으므로 2
```

```
        System.out.println(++a); // 증가시킨 다음에 출력했으므로 3
```

```
        System.out.println(a++); // 먼저 3을 출력한 다음에 1증가
```

```
        System.out.println(a); // 현재 a는 4
```

```
}
```

```
}
```

연산자

비교연산자?

양변을 비교 같으면 참 또는 거짓을 반환하는 연산

종류	연산자	연산자 사용방법	설명
같다	=	$a = b$	a와 b가 같으면 참
작다	<	$a < b$	a가 b보다 작으면 참
크다	>	$a > b$	a가 b보다 크면 참
작거나 같다	\leq	$a \leq b$	a가 b보다 작거나 같으면 참
크거나 같다	\geq	$a \geq b$	a가 b보다 크거나 같으면 참
같지 않다	\neq	$a \neq b$	a와 b가 같지 않으면 참

연산자

```
package java_practice;

public class Logic_operator2 {

    public static void main(String[] args) {

        int a = 10;
        System.out.println(5 < a && a < 15); // 5< a <15
        System.out.println((5 < a && a < 15)&& a % 2==0);

        a = 4;
        System.out.println((5 < a && a < 15)&& a % 2==0);
        System.out.println((5 < a && a < 15)|| a % 2==0);

    }
}
```

연산자

논리연산자?

양변을 비교 같으면 참 또는 거짓을 반환하는 연산

종류	연산자	사용 예	설명
AND	<code>&&</code>	<code>a && b</code>	<code>a</code> 가 참이고 <code>b</code> 이면 참이 된다.
OR	<code> </code>	<code>a b</code>	<code>a</code> 또는 <code>b</code> 중 하나라도 참이면 참이 된다.
NOT	<code>!</code>	<code>!a</code>	<code>a</code> 가 참이면 거짓이 되고, 거짓이면 참이 된다.

연산자

```
package java_practice;

public class Logic_operator2 {

    public static void main(String[] args) {

        int a = 10;
        System.out.println(5 < a && a < 15); // 5< a <15
        System.out.println((5 < a && a < 15)&& a % 2==0);

        a = 4;
        System.out.println((5 < a && a < 15)&& a % 2==0);
        System.out.println((5 < a && a < 15)|| a % 2==0);

    }
}
```

연산자

비트연산자?

개발자가 집적 비트를 조작할 수 있는 연산자

종류	연산자	사용방법	설명
비트 AND	&	$a \& b$	변수 a와 변수 b의 비트 단위의 AND
비트 OR		$a b$	변수 a와 변수 b의 비트 단위의 OR
비트 XOR	\wedge	$a \wedge b$	변수 a와 변수 b의 비트단위의 XOR
비트 NOT	\sim	$\sim a$	변수 a의 단위의 부정

연산자

삼항연산자?

조건식의 참/거짓 여부에 따라서 각각 다른 결과를 반환

Ex) 조건?항목₁:항목₂

조건 : 참과 거짓으로 구분할 수 있는 비교 연산, 논리 연산 등입니다

항목₁ : 조건이 참인 경우 실행할 연산이나 함수 등입니다.

항목₂ : 조건이 거짓인 경우 실행할 연산이나 함수 등입니다.

연산자

```
package java_practice;
```

```
public class xor_practice {
```

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

```
        int a = 15;
```

```
        int b = 5;
```

```
        System.out.println(a|b);
```

```
        System.out.println(a&b);
```

```
        System.out.println(a^b);
```

```
        System.out.println(a>>2);
```

```
        System.out.println(b<<4);
```

```
}
```

```
package java_practice;
```

```
public class Three_operator {
```

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

```
        int age = 17;
```

```
        System.out.println(age > 19 ? "성인입니다": "청소년입니다");
```

```
}
```

```
}
```

```
}
```

연산자

대입연산자?

값을 할당하는 데 쓰이는 연산자

연산자	예	설명	풀어쓴 형식
=	$a = 1$	변수 a에 1을 기억	$a = 1$
$+=$ (더하기)	$a += 2$	변수 a에 2를 더하여 다시 a에 기억	$a = a + 2$
$-=$ (빼기)	$a -= 4$	변수 a에서 4를 뺀 후 다시 a에 기억	$a = a - 4$
$*=$ (곱하기)	$a *= 7$	변수 a에 7을 곱한 후 다시 a에 기억	$a = a * 7$
$/=$ (나누기)	$a /= 3$	변수 a를 3으로 나눈 몫을 다시 a에 기억	$a = a / 3$
$\%=$ (나머지)	$a \%= 5$	변수 a를 5로 나눈 나머지를 a에 기억	$a = a \% 5$

연산자

```
package java_practice;

public class Assign_operator {

    public static void main(String[] args) {
        int a = 3;
        int b = 5;

        b = a;
        System.out.println(b);

        a += 1;
        System.out.println(a);
        a /= 2;
        System.out.println(a);
        a *= a;
        System.out.println(a);
    }

}
```

조건문

IF

- if 뒤의 () 안에는(조건문) 오직 **boolean** 만 쓸수 있다

```
IF( 조건문 ) 실행문1 ; //조건이 참이면 실행1이 실행, 거짓이면 실행되지 않는다.
```

```
실행문2; //건문에 상관없이 실행
```

조건문

```
package java_practice;

public class If_sentence {
    public static void main(String[] args) {
        int a = 5;
        if(a > 3) {
            System.out.println("a는 3보다 큽니다.");
        }
        System.out.println("검사가 끝났습니다.");
    }
}
```

조건문

IF ~ Else

```
if( 조건문 ) {
```

```
    실행문1; //조건이 참일때 실행
```

```
} else {
```

```
    실행문2; //조건이 거짓일때
```

```
}
```

```
    실행문3; //조건에 상관없이 실행
```

조건문

```
package java_practice;

public class If_else {

    public static void main(String[] args) {
        int age = 15;

        if(age > 19) { // 조건식은 false입니다.
            System.out.println("성인입니다.");
            System.out.println("성인요금이 적용됩니다.");
        }
        else{
            System.out.println("청소년입니다");
            System.out.println("청소년 요금이 적용됩니다.");
        }
        System.out.println("결제를 진행해주세요.");
    }
}
```

조건문

IF ~ Else If ~ Else

- 조건이 여러 개인 경우

```
if(조건문 ){
```

```
    실행문 1;
```

```
} else if( 조건문 ){
```

```
    실행문 2;
```

```
} else {
```

```
    실행문 3;
```

```
}
```

조건문

import java.util.*; //Scanner 클래스를 사용하기 위해서 추가한 부분

```
public class If_else_if{
```

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

```
        int age;
```

```
        int age = input.nextInt();
```

```
        age = Integer.parseInt(tmp);
```

```
        if(age > 19) {
```

```
            System.out.println("성인입니다.");
```

```
            System.out.println("성인요금이 적용됩니다.");
```

```
        }
```

```
        // 19 >= age가 14~19살인 경우
```

```
        else if(age > 13) {
```

```
            System.out.println("청소년 입니다.");
```

```
            System.out.println("청소년 요금이 적용됩니다.");
```

```
    }
```

```
    // 13 >= age가 9~13살인 경우
```

```
    else if(age > 8) {
```

```
        System.out.println("어린이입니다.");
```

```
        System.out.println("어린이 요금이 적용됩니다.");
```

```
    }
```

```
    // age가 9보다 작은 경우
```

```
    else{
```

```
        System.out.println("유아입니다.");
```

```
        System.out.println("유아 요금이 적용됩니다.");
```

```
}
```

```
System.out.println("결제를 진행해 주세요.");
```

```
}
```

조건문

import java.util.*; 는 Scanner 클래스를 사용하기 위해 추가한 부분입니다. Scanner 클래스는 사용자로부터 입력을 받기 원할 때 사용하는 클래스입니다.

```
Scanner input = new Scanner(System.in);
String tmp = input.nextLine();
```

Scanner 클래스를 사용하기 위해서는 먼저 Scanner 클래스를 참조하는 변수를 생성합니다. 그리고 참조변수를 이용해서 input.nextLine() 부분에서 사용자에게 입력을 받습니다. 입력을 받고 나면 tmp이라는 문자형 변수에 입력내용이 저장됩니다.

TIPS_

double d = input.nextDouble(); 실수타입 입력
char c = input.next().charAt(0); 1개 문자 입력

조건문

```
public class If_if {  
    public static void main(String[] args) {  
        String id,password;  
        Scanner input = new Scanner(System.in);  
        System.out.printf("아이디를 입력해주세요 : ");  
        id = input.nextLine( );  
  
        if(id.equals("java")) {  
            System.out.println("id 일치");  
            System.out.printf("비밀번호를 입력해주세요 : ");  
            password = input.nextLine( );  
            if(password.equals("abc123")) {  
                System.out.println("password 일치");  
                System.out.println("로그인 성공!");  
            }  
            else  
                System.out.println("password 불일치");  
        }  
        else  
            System.out.println("id 불일치");  
    }  
}
```

조건문

Switch?

하나의 조건식으로 많은 경우 수를 처리하는 조건문

```
switch(조건식) {  
    case 값1:  
        // 조건식의 결과와 값1이 같은 경우 실행  
        break;  
    case 값2:  
        // 조건식의 결과와 값2가 같은 경우 실행  
        break;  
    default:  
        // 조건식의 결과와 일치하는 값이 없을 때 실행  
}
```

조건문

```
public class Switch {  
    public static void main(String[] args) {  
        // 일 년 동안 읽은 책 수에 따라 멘트를 출력합니다.  
        int book = 2;  
        book = book / 10 ;  
        switch(book) {  
            case 0:// 10권 미만  
                System.out.println("조금 더 노력하세요!");  
                break;  
            case 1:// 10권 이상 20권 미만  
                System.out.println("책 읽는 것을 즐기는 분이시네요!");  
                break;  
            case 2:// 20권 이상 30권 미만  
                System.out.println("책을 사랑하는 분이시네요!");  
                break;  
            default:// 30권 이상  
                System.out.println("당신은 다독왕입니다!!");  
        }  
    }  
}
```

반복문

FOR문?

특정한 횟수동안 작업을 반복하고 싶을 때 사용

```
for( 초기화 ; 조건식; 증감식 ){  
    System.out.println("조건식이 참일 때 실행되는 부분");  
}
```

- ① 초기화 : 반복문을 위해 사용할 변수를 초기화합니다.
- ② 조건식 : 반복을 계속하기 위한 조건을 정합니다. 조건식의 값이 true이면 반복을 계속 진행하고 false이면 중단하고 for문을 벗어납니다.
- ③ 증감식 : 변수의 값을 증가 또는 감소합니다.
- ④ 실행부분 : 조건식이 참일 때 실행되는 부분입니다.

반복문

```
package java_practice;

public class For2 {

    public static void main(String[] args) {
        int sum = 0; // 총 합을 담을 변수
        for(int i = 1; i <= 10; i++) {
            System.out.printf(" i= %d sum = %d \n", i, sum+=i);
        }
    }
}
```

반복문

While문?

조건식을 검사하고 조건식이 참이면 반복, false 탈출

```
while(조건식) {  
    // 실행되는 부분  
}
```

반복문

```
import java.util.*;  
  
public class While3 {  
  
    public static void main(String[] args) {  
  
        Scanner sc = new Scanner(System.in);  
  
        String answer = "Y"; // while이 시작될 수 있도록 응답 값을 초기화  
        int count = 0;  
  
        while(answer.equals("Y")) { // 사용자의 응답을 검사합니다.  
            System.out.println("음악을 재생하시겠습니까?(Y)");  
            answer = sc.nextLine(); // 사용자의 응답을 받습니다.  
  
            if(answer.equals("Y")) {  
                System.out.printf("음악을 %d번 재생했습니다 %n", ++count);  
            }  
            System.out.println("재생 종료");  
        }  
    }  
}
```

반복문

Do~While문?

한번은 문장들을 수행하고 조건식을 검사해서 반복
무조건 한번은 실행

```
do{  
    // 실행되는 부분  
} while(조건식);
```

반복문

```
public class Do_while {  
  
    public static void main(String[] args) {  
        int i = 100;  
        do{  
            System.out.println("적어도 한번은 출력되는 문장");  
        } while(i<10);  
    }  
}
```

반복문

```
public class Break {  
  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        int sum = 0;  
        int i;  
  
        while(true) {  
            System.out.println("더할 숫자를 입력하세요 : (종료하려면 0입력)");  
            i = sc.nextInt(); // 정수 입력받아서 i에 저장  
            if(i == 0) { // 만약 0을 입력하였다면 종료  
                break;  
            }  
            sum += i; // 입력받은 값 더해주기  
        }  
        System.out.println("현재까지의 총합 = " + sum);  
    }  
}
```

배열

같은 타입의 변수를 하나의 묶음으로 관리.
여러 개의 데이터를 저장하고 효율적으로 관리 가능

```
// 배열을 사용하지 않을 경우
```

```
int student1, student2, student3, ... student50,... student100;
```

```
// 배열을 사용 할 경우
```

```
int[] student = new int[100];
```

배열

선언과 생성?

선언을 하려면 변수 선언에서 타입 뒤에 대괄호([]) 추가

타입[] 배열 이름;

ex) int[] student;

String[] name;

타입[] 배열이름 = new 타입[길이];

ex) int[] student = new int[5];

String[] name = new String[3];

배열

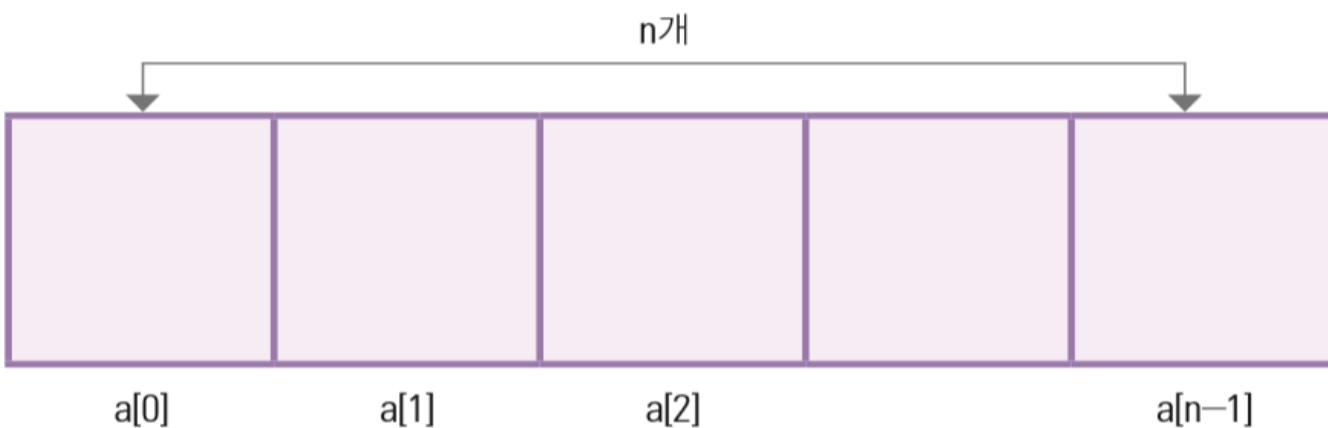
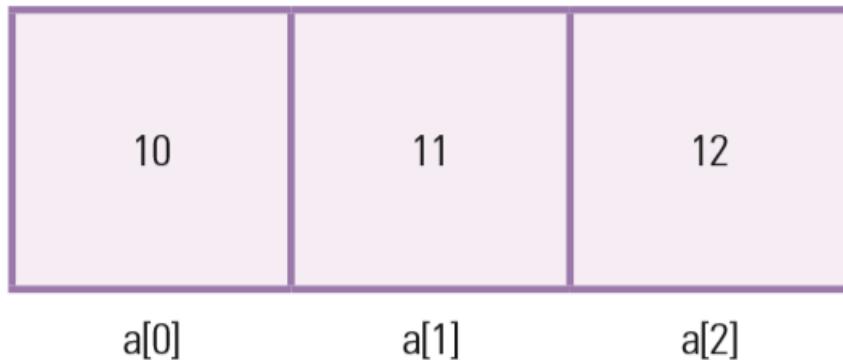
배열의 길이와 인덱스?

배열의 다룰땐 인덱스 사용, 인덱스란 배열의 공간 번호

$a[0] = 10;$

$a[1] = 11;$

$a[2] = 12;$



배열

초기화와 출력?

```
public static void main(String[] args) {  
  
    int[] student = new int[3]; // 길이가 3인 배열 생성  
    System.out.println("현재 자동으로 초기화된 값 : " + student[0]);  
  
    student[0]=30; // 배열의 첫 번째 요소에 30을 저장  
    student[1]=20; // 배열의 두 번째 요소에 20을 저장  
    student[2]=10; // 배열의 세 번째 요소에 10을 저장  
  
    System.out.println("현재 첫 번째 요소의 값 = " + student[0]);  
  
}
```

배열

2차원 배열 선언과 생성

타입[][] 배열이름 ;

ex) int[][] student;

String[][] name ;

타입[][] 배열이름 = new 타입[][];

ex) int[][] student = new int[3][3];

String[][] name = new String[3][2];

배열

2차원 배열 인덱스

```
int[][] student = new int[3][4];
```

student[0][0]	student[0][1]	student[0][2]	student[0][3]
student[1][0]	student[1][1]	student[1][2]	student[1][3]
student[2][0]	student[2][1]	student[2][2]	student[2][3]

2행 3열에 있는 요소를 가지고 작업을 하고 싶다면 행과 열에서 1씩 빼 인덱스인 student[1][2]를 사용합니다.

```
student[1][2] = 30; // 2행 3열에 30 저장
```

```
System.out.println(student[1][2]); // 2행 3열 출력
```

위와 같은 방법으로 모든 요소에 접근합니다.

배열

2차원 배열 초기화

```
int[][] student = { {100, 200, 300} , {400, 500, 600} , {700, 800, 900} ,  
{200,400,500} } ;
```

```
int[][] student = {  
    {100, 200, 300} ,  
    {400, 500, 600} ,  
    {700, 800, 900} ,  
    {200,400,500}  
};
```

배열

배열 복사

반복문 사용으로도 복사가 가능하지만 제공해주는
메소드를 사용하면 간편

```
System.arraycopy( src, srcPos, dest, destPos, length);
```

- ① src는 복사할 배열이고 srcPos는 복사를 하기 시작할 인덱스
- ② dest는 덮어쓸 배열이고 destPos는 덮어쓰기 시작할 인덱스
- ③ length는 복사할 길이입니다.

배열

```
import java.util.*;
public class Array_4 {

    public static void main(String[] args) {
        int[] a = {1,2,3,4,5,6};
        int[] b = {0,0,0,0,0,0};
        System.out.println(Arrays.toString(a));
        System.out.println(Arrays.toString(b));

        System.arraycopy(a,2,b,3,4); // a를 b로 복사
        System.out.println(Arrays.toString(b));
    }
}
```

배열

Foreach

```
for(배열의 타입 변수이름 : 배열이름) {  
    // 실행 구문  
}
```

```
public class Array_5 {  
  
    public static void main(String[] args) {  
        String[] a = {"Java", "Hello", "Programming"} ;  
  
        for(String i : a) { // foreach 구문  
            System.out.println(i);  
        }  
  
        int[] b = {1,2,3,4,5};  
  
        for(int i:b) { // foreach 구문  
            System.out.print(i);  
        }  
  
    }  
}
```

객체지향언어

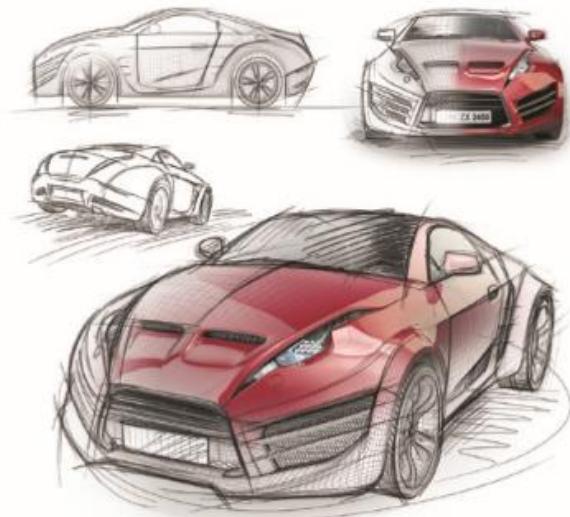
객체지향 언어란 현실에 존재하는 사물과 개념들을 소프트웨어적으로 구현하고, 그 구현된 객체들이 상호작용하여 데이터를 처리하는 방식입니다. 객체지향 언어는 다음과 같은 특징을 가집니다.

- ① 코드의 재사용성 : 새로운 코드를 작성할 때 기존의 코드를 이용합니다. 이는 코드의 수를 크게 줄일 수 있고 유지 보수하기에도 편리합니다.
- ② 신뢰성 높은 프로그래밍 : 제어자와 메서드를 사용해서 데이터를 보호합니다. 또 코드의 중복을 제거하여 오동작을 방지합니다.
- ③ 코드 관리의 편리함 : 객체지향 프로그래밍은 한 부분만 변경하면 관련된 모든 부분이 변경됩니다. 그렇기 때문에 코드의 관리가 편리합니다.

클래스

클래스란

자동차를 만들기 위해 설계도가 필요하듯 객체를 만들기 위한 설계도



객체

자동차 인스턴스

클래스

인스턴스란

클래스란 설계도를 통해 실질적으로 만들어진 형태



자동차 설계도



자동차 인스턴스

클래스

클래스의 사용

구현할려는 객체의 속성과 기능들의 정의하는 설계도
속성을 변수 기능을 메서드로 만들어 사용

```
class 클래스명{  
    /* 속성(변수) 작성 */  
    /* 기능( 메서드) 작성 */  
}
```

자동차의 색, 바퀴의 수, 자동차의 속력이 속성
시동, 액셀, 브레이크, 와이퍼 동작이 기능

클래스

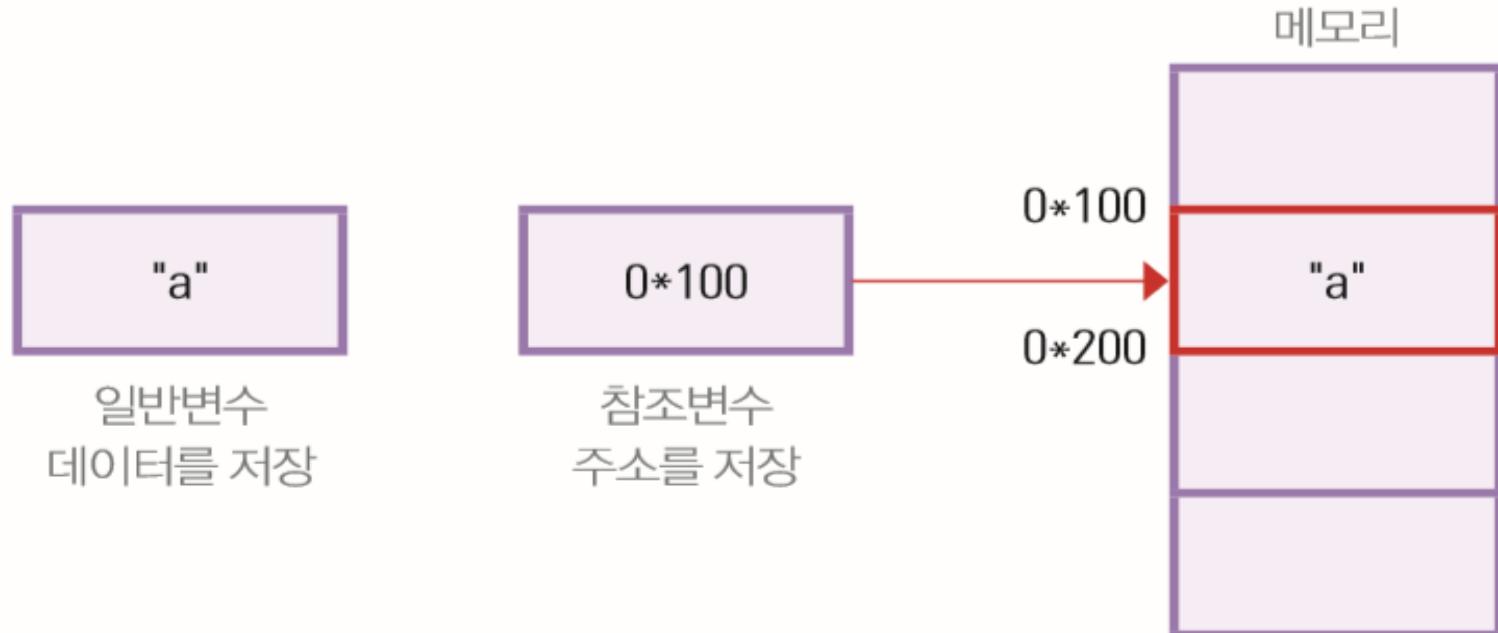
```
class Car{  
    boolean powerOn;//시동  
    String color; // 차량의 색상  
    int wheel; // 바퀴의 수  
    int speed; // 속력  
    boolean wiperOn;// 와이퍼  
  
    void power( ) { powerOn = !powerOn;} // 시동 메서드  
    void speedUp( ) { speed++; } // 액셀 메서드  
    void speedDown( ) { speed--; } // 브레이크 메서드  
    void wiper( ) { wiperOn = !wiperOn; } // 와이퍼 메서드  
}
```

클래스

클래스명 변수명; // 클래스의 객체를 참조할 수 있는 참조변수 선언

변수명 = new 클래스명(); // 클래스의 객체를 생성하고 객체의 주소를 참조변수에 저장

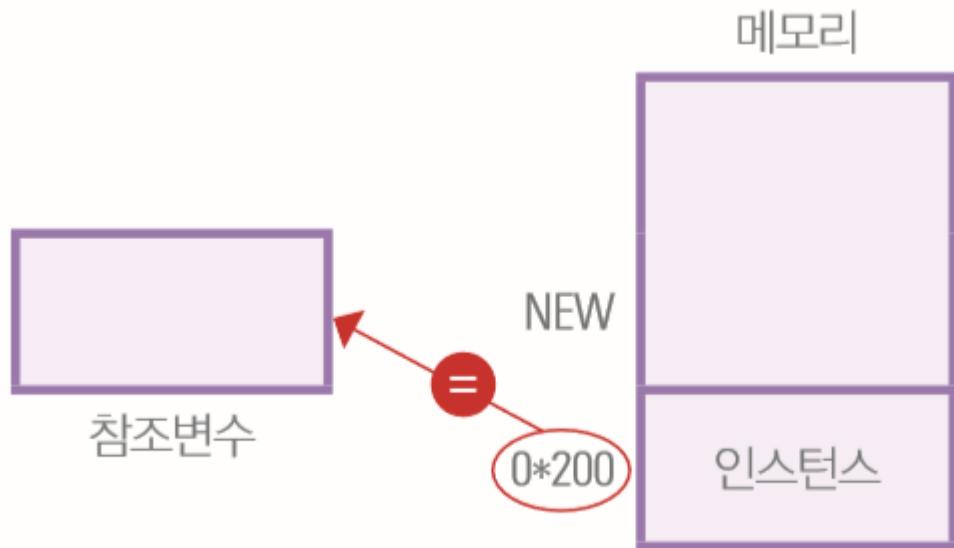
클래스명 변수명 = new 클래스명(); // 한 번에 선언하는 경우



클래스

변수에 접근할 경우 : 참조변수.변수명

메서드를 실행시킬 경우 : 참조변수.메서드명();



클래스

```
public class Class_practice {  
  
    public static void main(String[] args) {  
        Car mycar = new Car();  
        System.out.println("시동 처음 초기화:"+mycar.powerOn);  
        System.out.println("차의 색상 초기화:"+mycar.color);  
        System.out.println("바퀴의 수 초기화:"+mycar.wheel);  
        System.out.println("속력 초기화:"+mycar.speed);  
        System.out.println("와이퍼 작동 초기화:"+mycar.wiperOn);  
  
        mycar.power(); // 시동 메서드 실행  
        System.out.println("시동 메서드 동작:"+mycar.powerOn);  
        mycar.power(); // 시동 메서드 실행  
        System.out.println("시동 메서드 다시 동작:"+mycar.powerOn);  
  
        mycar.color = "black"; // 색상 변수에 black 대입  
        System.out.println("현재 차의 색상:"+mycar.color);  
    }  
}
```

클래스

인스턴스 변수

액체마다 가지는 고유 변수

인스턴스를 생성할 때 만들어짐

인스턴스마다 고유의 저장공간을 가진다.

인스턴스 생성 시 개별마다 특별한 값을 가져야 할 경우

```
class Cars{  
    int speed; // 인스턴스 변수 선언  
                // 기존의 방식과 동일!  
}
```

클래스

클래스 변수

모든 인스턴스가 공통된 내용을 공유하는 변수

모든 인스턴스가 똑같은 저장 공간을 공유

클래스가 처음 메모리에 로딩될 때 생성

인스턴스 생성하지 않아도 사용 가능

```
class Cars{  
    static int wheel; // 클래스 변수 선언  
}
```

클래스

```
public class Class_practice_3 {  
  
    public static void main(String[] args) {  
        System.out.println(Cars.wheel); // 클래스 변수 접근 가능  
        // System.out.println(Cars.speed); // 에러발생! 인스턴스 변수 접근 불가  
  
        cars myCar1 = new Cars();  
  
        System.out.println(Cars.wheel);  
        System.out.println(myCar1.speed); // 인스턴스 생성 후에는 접근 가능  
  
        Cars myCar2 = new Cars();
```

클래스

```
System.out.println("<변경 전>");  
System.out.println("myCar1.speed : "+ myCar1.speed);  
System.out.println("myCar2.speed : "+ myCar2.speed);  
System.out.println("myCar1.wheel : "+ myCar1.wheel);  
System.out.println("myCar2.wheel : "+ myCar2.wheel);
```

```
myCar2.speed = 100;  
myCar2.wheel = 5;  
System.out.println("<변경 후>");  
System.out.println("myCar1.speed : "+ myCar1.speed);  
System.out.println("myCar2.speed : "+ myCar2.speed);  
System.out.println("myCar1.wheel : "+ myCar1.wheel);  
System.out.println("myCar2.wheel : "+ myCar2.wheel);  
  
}  
}
```

```
class Cars{  
  
    static int wheel = 4; // 클래스변수  
    int speed; // 인스턴스 변수
```

클래스

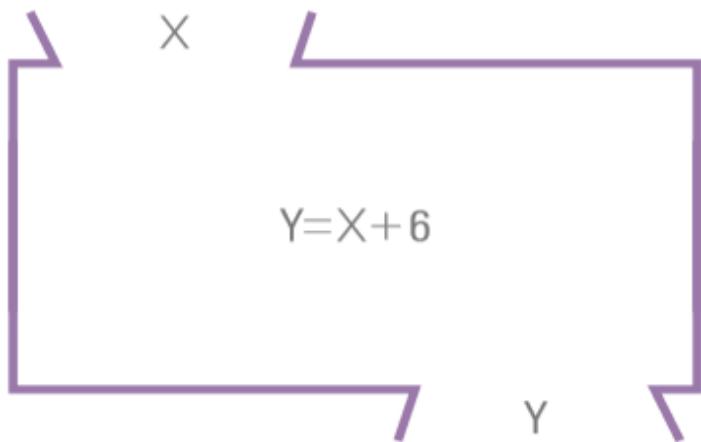
메서드

다른 언어에서의 함수와 같은 개념

특정한 작업이나 논리를 구성하는 코드를 괄호로 묶음

한번 정하면 언제든지 호출 가능

효율적이며 코드의 중복을 회피



클래스

메서드 사용

반환타입 메서드명(타입 변수명, 타입 변수명, ...) {

①

②

③

// 메서드 내부의 동작

return 값;

④

}

class cars{

 boolean powerOn() { }

 // 인스턴스 메서드

 static boolean wiperoOn() { }

 // 클래스 메서드

}

클래스

```
class Area{  
    static void manual( ) { // static이 있으므로 클래스 메서드  
        System.out.println("현재 사용 가능한 함수 목록");  
        System.out.println("triangle : 삼각형 넓이");  
        System.out.println("rectangle : 사각형 넓이");  
        System.out.println("입니다");  
    }  
  
    double triangle(int a, int b) { // 삼각형 넓이를 반환하는 메서드  
        return (double)a*b/2;  
    }  
  
    int rectangle(int a, int b) { // 사각형 넓이 반환하는 메서드  
        return a*b;  
    }  
}
```

클래스

```
public class Method2 {  
    public static void main(String[] args) {  
        Area.manual( ); // 클래스 메서드 접근 가능  
        // Area.triangle(3,5); 에러발생  
        // Area.rectangle(3,4);에러발생  
  
        Area cal = new Area( );  
  
        cal.manual( );  
        System.out.println(cal.triangle(3,5));  
        System.out.println(cal.rectangle(3,4));  
    }  
}
```

클래스

클래스 멤버와 인스턴스 멤버간의 참조 호출

클래스 멤버 = 클래스 변수와 클래스 메서드

인스턴스 멤버 = 인스턴스 변수와 인스턴스 메소드

인스턴스 멤버의 클래스 멤버 사용 → 가능

클래스 멤버의 인스턴스 멤버 사용 → 에러

인스턴스 멤버의 인스턴스 멤버 사용 → 가능

클래스 멤버의 클래스 멤버 사용 → 가능

클래스

오버로딩

매개변수의 개수와 타입은 다르지만 이름이 같은 메서드

```
int sum(int a, int b) {  
    return a+b;  
}  
  
int sum(int a, int b, int c) {  
    return a+b+c;  
}
```

클래스

```
public class Overloading1 {  
    static int sum(int a,int b) {  
        System.out.println("인자가 둘일 경우 호출됨");  
        return a+b;  
    }  
    static int sum(int a, int b, int c) {  
        System.out.println("인자가 셋일 경우 호출됨");  
        return a + b + c;  
    }  
  
    static double sum(double a,double b, double c) {  
        System.out.println("double 타입일 경우 호출됨");  
        return a + b + c;  
    }  
    public static void main(String[] args) {  
        System.out.println(sum(3,2));  
        System.out.println(sum(2,3,4));  
        System.out.println(sum(2.5,3.5,4.5));  
    }  
}
```

클래스

함수	설명
println()	Terminates the current by writing the line separator string
println(boolean x)	Prints a boolean and then terminate the line.
println(char x)	Prints a character and then terminate the line.
println(char[] x)	Prints an array characters and then terminate the line.
println(double x)	Prints a double and then terminate the line.
println(float x)	Prints a float and then terminate the line.
println(int x)	Prints a integer and then terminate the line.
println(long x)	Prints a long and then terminate the line.
println(Object x)	Prints a Object and then terminate the line.
println(String x)	Prints a String and then terminate the line.

클래스

생성자

모든 클래스에는 반드시 하나 이상의 생성자가 존재

〈기존의 방식 코드〉

1. myphone1.color = "gold";
2. myphone1.capacity = 32;
3. myphone1.model = "Galaxy";
4. myphone1.price = 640,000;



```
class Cellphone{  
    String model;  
    String color;  
    int capacity;  
    Cellphone(){} // 컴파일시 컴파일러가 자동으로 추가  
}
```

클래스

```
class Cellphone{  
    String model = "Galaxy 8";  
    String color;  
    int capacity;  
  
    Cellphone(String color, int capacity) { // 매개변수가 있는 생성자  
        this.color = color;  
        this.capacity = capacity;  
    }  
}
```

클래스

```
public class Constructor1 {  
    public static void main(String[] args) {  
        //Cellphone myphone1 = new Cellphone( ); // 에러발생  
  
        Cellphone myphone = new Cellphone("Silver",64); //생성자 호출  
  
        System.out.println(myphone.model);  
        System.out.println(myphone.color);  
        System.out.println(myphone.capacity);  
    }  
}
```

클래스

생성자(매개변수)

매개변수가 있는 생성자가 작성되면 기본 생성자는 무시
기본생성을 통한 작업이 필요하면 직접 작성

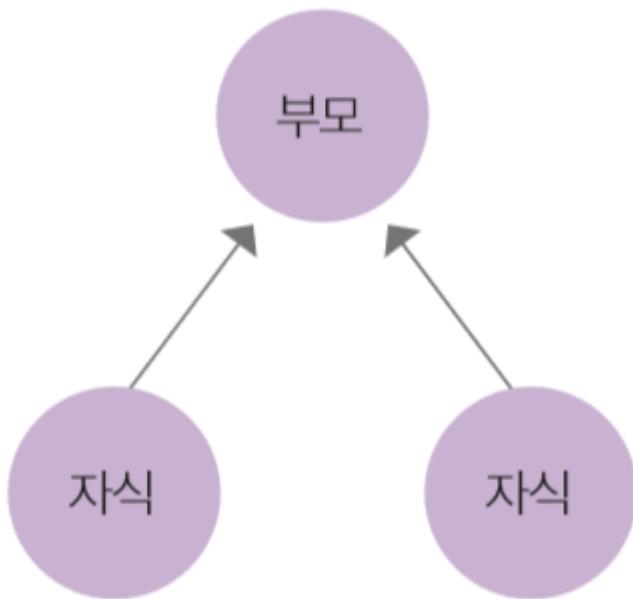
```
Cellphone(String color, int capacity) {  
    this.color = color;  
    pacity = capacity;  
}
```

```
Cellphone() {  
    this.color = 'black';  
    this.capacity = 32;  
}
```

상속

상속

새로운 클래스를 작성 시 기존에 존재하는 클래스 이용
기존 클래스 멤버를 물려 받으므로 코드의 양이 줄어듦



상속

상속의 사용이유는 코드를 재사용하고 중복을 줄임
생성자는 상속되지 않는다.

```
class Parents{ }  
class Child extends Parents{ }  
// Parents 클래스의 멤버들을 상속받음
```

상속

```
class Person{ // 사람 클래스  
    void breath( ) {  
        System.out.println("숨쉬기");  
    }  
    void eat( ) {  
        System.out.println("밥먹기");  
    }  
    void say( ) {  
        System.out.println("말하기");  
    }  
}
```

```
class Teacher extends Person{ // 사람 클래스를 상속한 선생 클래스  
    void teach( ) {  
        System.out.println("가르치기");  
    }  
  
public class Inheritance1 {  
    public static void main(String[] args) {  
        Student s1 = new Student( ); // 학생 인스턴스 s1 생성  
        s1.breath( ); // 사람 클래스의 breath 메서드를 상속받았음  
        s1.learn( );  
  
        Teacher t1 = new Teacher( ); // 선생 인스턴스 t1 생성  
        t1.eat( ); // 사람 클래스의 eat 메서드를 상속받았음  
        t1.teach( );  
    }  
}
```

상속

자바는 단일 상속만을 허용 다중상속을 지원하지 않음
클래스 앞에 final이 붙은 경우 상속이 불가능

```
class Test extends B, extends C{  
} // 에러발생, 다중상속 불가.
```

```
final class Parent{}  
class Child extends Parent{} // 에러발생
```

오버라이딩(Overriding)

자식클래스가 부모클래스로 부터 받은 메소드를 재정의
인스턴스 호출시 자식클래스에서 재작성한 메서드 호출

```
class Parents{  
    void method1( ) {  
        // 부모 클래스의 메서드  
    }  
}  
  
class Child extends Parents{  
    void method1( ) {  
        // 자손 클래스에서 메서드 내용을 재정의  
    }  
}
```

오버라이딩(Overriding)

```
class Child extends Parents{  
  
    @Override  
    void method1(){  
        // TODO Auto-generated method stub  
        super.method1();  
    }  
  
    @Override  
    void method2(){  
        // TODO Auto-generated method stub  
        super.method2();  
    }  
}
```

-자동 추가 시
이클립스 메뉴의
[source]-[Implement
Methods]

-직접 작성해도 된다.

-@Override 는
annotation 주석기능

-부모의 메소드 동작시
super 사용

오버라이딩(Overriding)

TIPS_

- `this` : 인스턴스들이 메서드를 호출할 때, 메서드는 공용으로 사용하게 된다. 이때 각 인스턴스의 정보를 저장하는 변수를 `this`라고 합니다

- `this()` : 클래스에서 생성자의 임무가 겹칠 때, 한 생성자로 임무를 집중시킬 수 있습니다.

```
생성자(){  
    this(10);  인수 한 개짜리 생성자 a 변수에 10을 전달한다.
```

```
}
```

```
생성자(int a){  
}
```

- `super` : 상속을 하면 자식의 멤버(변수, 메서드)와 이름이 겹치게 될 때가 있는데 이를 구분하기 위해 `super.변수` `super.메서드()`를 사용하면 자식의 멤버가 호출되지 않고 상속받은 부모의 멤버를 호출할 수 있게 됩니다.

- `super()` : 자식 생성자에서 부모 생성자를 호출할 때 사용합니다.

```
super(값, 값..) 하게 되면 인자 있는 부모 생성자를 명시적으로 호출합니다.
```

상속과 생성자

Super

- 부모클래스의 생성자 호출은 super()로 호출 가능
- 자식클래스 생성자 작업시 부모클래스 생성자 반드시 호출
- 만약 미작성시 컴파일러가 자식클래스의 생성자 첫 줄에 super() 추가

Object 클래스

- 모든 클래스의 조상 클래스
- 가장 상위 클래스로서 누구의 상속도 받지 않는다.
- 아무런 상속도 받지 않는 독자적인 클래스 생성시
자동으로 object 클래스를 상속하는 문구 추가
- Ex)toString(),equals()

```
class Tree extends Object{  
    // 내부 멤버  
}
```

Object 클래스

상속관계에 있는 클래스에서 반드시 상위 클래스의 생성자를 호출해야 하는 규칙도 결국 object 클래스까지 거슬러 올라간다.

```
Class SportsCar extends Car2{
```

```
Class Car2{
```

```
Class Object{
```

```
    SportsCar() {
```

```
        super();
```

```
    Car2() {
```

```
        super();
```

```
    Object
```

```
    ...
```

```
}
```

```
}
```

```
    ...
```

```
}
```

```
    ...
```

```
}
```

접근제어자

클래스나 멤버의 사용을 제어하기 위해 사용.

- public : 접근 제한이 없음
- protected : 같은 패키지 내에서와 자손 클래스에서 접근 가능
- default : 같은 패키지 내에서만 접근 가능
- private : 같은 클래스 내에서만 접근 가능

접근제어자

Static

- 변수나 메소드 앞에 붙어 클래스 멤버임을 의미
- 처음 클래스 로드시 생성으로 인스턴스 생성없이 사용

```
package java_practice;
class StaticPractice{
    public static int number = 3;
    public static void say( ) {
        System.out.println("인스턴스 생성하지 않고도 사용 가능합니다");
    }
}
public class StaticModifier {
    public static void main(String[] args) {
        System.out.println(StaticPractice.number);
        StaticPractice.say( );
    }
}
```

접근제어자

Final

말 그대로 ‘종결의’ 라는 의미
내용이나 값을 변경하지 못하게 된다,

final 변수 : 값을 더 이상 변경할 수 없는 상수

final 메서드 : 내용을 더 이상 변경할 수 없는 메서드. 오버라이딩 불가

final 클래스 : 내용을 더 이상 변경할 수 없는 클래스. 상속 불가

```
final class FinalClass{
    final int number = 4; // 상수
}

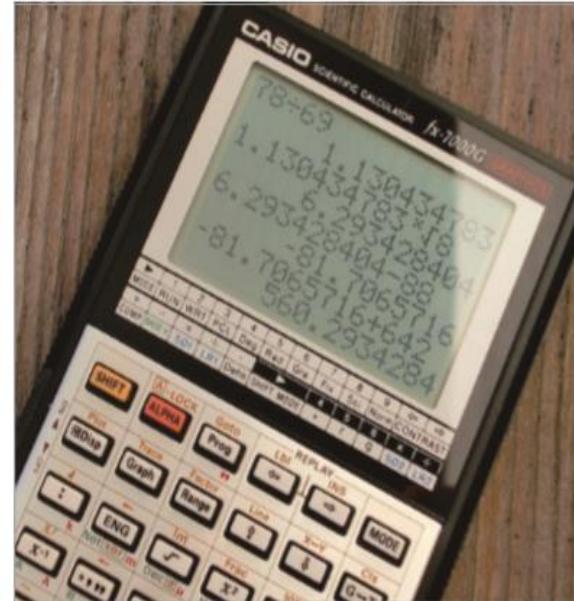
class Parents //extends FinalClass // 에러발생!
{
    final void finalMethod() { // 오버라이딩 불가한 메서드
        System.out.println("상속불가한 메서드");
    }
}

final class Son extends Parents{
    // void finalMethod() {} // 에러발생!
}

public class FinalModifier {
    public static void main(String[] args) {
        FinalClass member1 = new FinalClass();
        System.out.println(member1.number);
        // member1.number = 4; // 상수는 변경 불가
    }
}
```

다형성 (polymorphism)

‘한가지 타입이 여러가지 형태의 인스턴스 가능’ ,



▲ 계산기

▲ 공학용 계산기

▲ 프로그래머용 계산기

다형성 (polymorphism)

지금까지는 인스턴스를 생성하고 참조변수 할당시
인스턴스와 참조변수의 클래스를 동일하게 작성

```
A obj = new A();
```

다형성의 정의에 따라 부모 클래스 타입의 참조변수로
자식 클래스 타입의 객체 참조 가능

```
A obj = new B();
```

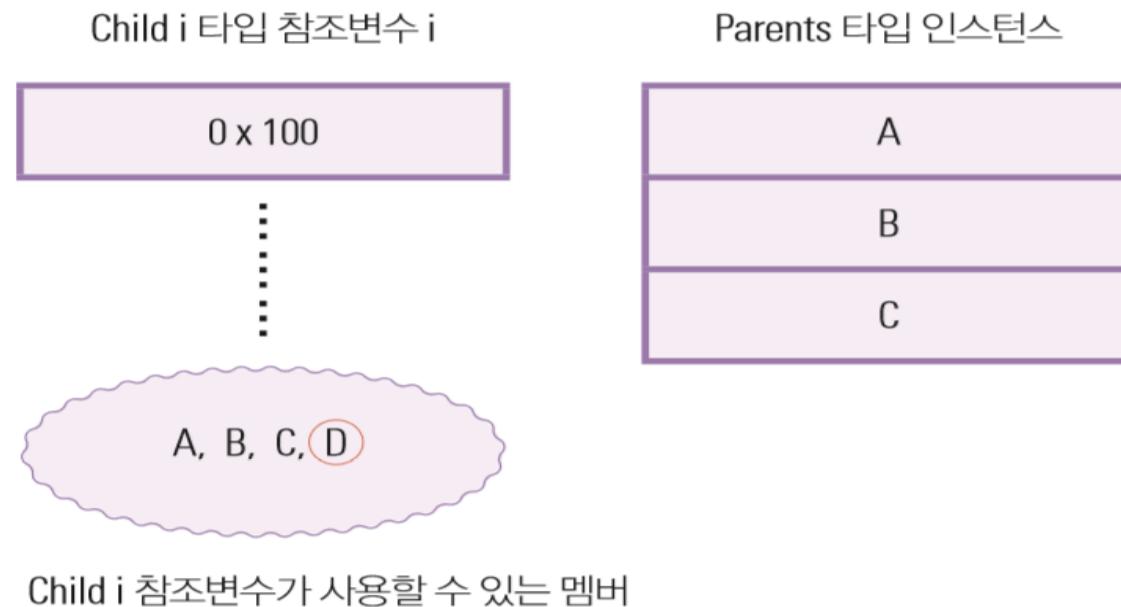
(클래스 B가 A를 상속할 때)

참조변수 하나로 A타입과 B타입의 인스턴스를 참조 할
수 있게 된다.

다형성 (polymorphism)

새로운 멤버 선언 시 자식 타입의 참조변수는 부모
클래스 참조 가능

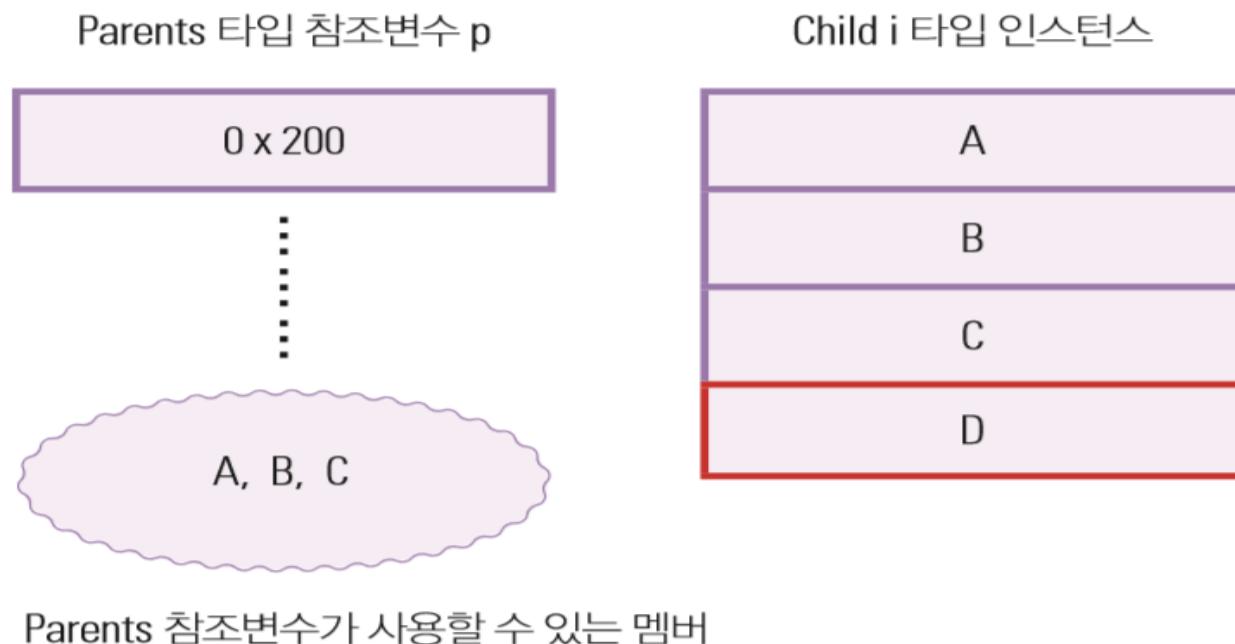
- Child i = new Parents();



다형성(polymorphism)

부모클래스의 인스턴스를 참조하면 새로운 멤버를 참조할 수도 있으므로 에러가 발생

- Parents p = new Child();



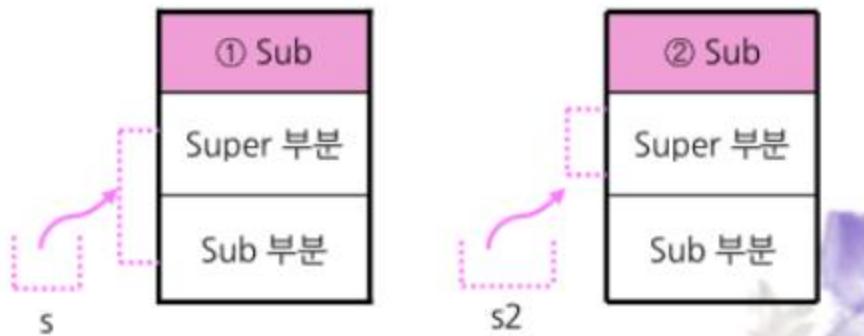
다형성 (polymorphism)

```
class Super{  
    속성;  
    메소드  
}
```

```
class Sub extends Super{  
    속성;  
    메소드  
}
```

① Sub s = new Sub();

② Super s2 = new Sub();



- s2변수를 이용하여 자식 부분에 접근 불가(단, 재정의된 메소드만 자식부분 접근 가능)

Sub s3 = (Sub)s2 ; => ObjectCasing

다형성 (polymorphism)

```
class A {                                public class Polymorphism1 {  
    void methodA( ) {  
        System.out.println("methodA");  
    } ;  
}  
  
class B extends A{  
    void methodB( ) {  
        System.out.println("methodB");  
    }  
}
```

```
                                public static void main(String[] args) {  
    A obj = new B();  
    obj.methodA();  
    //obj.methodB(); // 에러발생  
}
```

다형성 (polymorphism)

```
class Animal{ //동물 클래스  
    void breath( ){  
        System.out.println("숨쉬기");  
    }  
}
```

```
class Lion extends Animal{ //동물 클래스를 상속한 사자 클래스  
    public String toString( ){  
        return "사자";  
    }  
}
```

```
class ZooKeeper{ //사육사 클래스  
    void feed(Lion lion) { //사자에게 먹이주는 클래스  
        System.out.println(lion+"에게 고기 주기");  
    }  
}
```

```
public class Polymorphism3 {  
    public static void main(String[] args) {  
        Lion lion1 = new Lion(); // Lion 인스턴스 생성  
        ZooKeeper james = new ZooKeeper(); // james 이름의 사육사 인스턴스 생성  
        james.feed(lion1); // james가 lion1에게 먹이를 줌  
    }  
}
```

추상메서드

- 선언부만 정의하고 구체적인 내용은 비워놓은 메서드
- 구체적인 내용은 적지 않았으므로 자식클래스에서
쓸려면 반드시 구현하라는 의미

```
abstract void methodA();
```

추상클래스

- 추상메서드를 멤버로 가지는 클래스(하나라도)
- 자손클래스를 완전히 작성하기 위한 틀로 사용

```
abstract class Cellphone{  
  
    abstract void methodA();  
  
    // 일반 멤버들 생략  
  
}
```

추상클래스

추상클래스를 사용함으로 자식클래스에 문법적 제한부여

```
abstract class Cellphone{  
  
    abstract void methodA( );  
  
    // 일반 멤버들 생략  
  
}  
  
class MyPhone extends CellPhone{  
  
    //추가적인 메서드  
  
}
```

에러발생!

```
class Cellphone{  
  
    void methodA( );// ** 에러발생  
  
    // 일반 멤버들 생략  
  
}  
  
class MyPhone extends CellPhone{  
  
    //추가적인 메서드  
  
}
```

에러발생하지 않음

인터페이스

- 클래스가 설계도라면 인터페이스는 설계할 때 필요한 목록을 써놓은 종이
- 상수와 추상 메서드를 멤버로 가진다.

```
interface 인터페이스이름{  
    public static final 타입 이름 = 값;  
    public abstract 반환타입 메서드이름(매개변수);  
}
```

인터페이스

- 제어자 생략시
컴파일러가
자동으로 추가
- 인터페이스를
구현한 자식
클래스는
오버라이딩을 통해
구체적으로 구현

```
interface A{  
    int a = 4; // 제어자 생략가능  
    void methodA( ); // 제어자 생략가능  
    void methodB( );  
}  
  
class B implements A{  
    public void methodA( ) {  
        /*구체적인 구현, 오버라이딩*/  
    }  
  
    public void methodB( ) {  
        /*구체적인 구현, 오버라이딩*/  
    }  
}
```

인터페이스

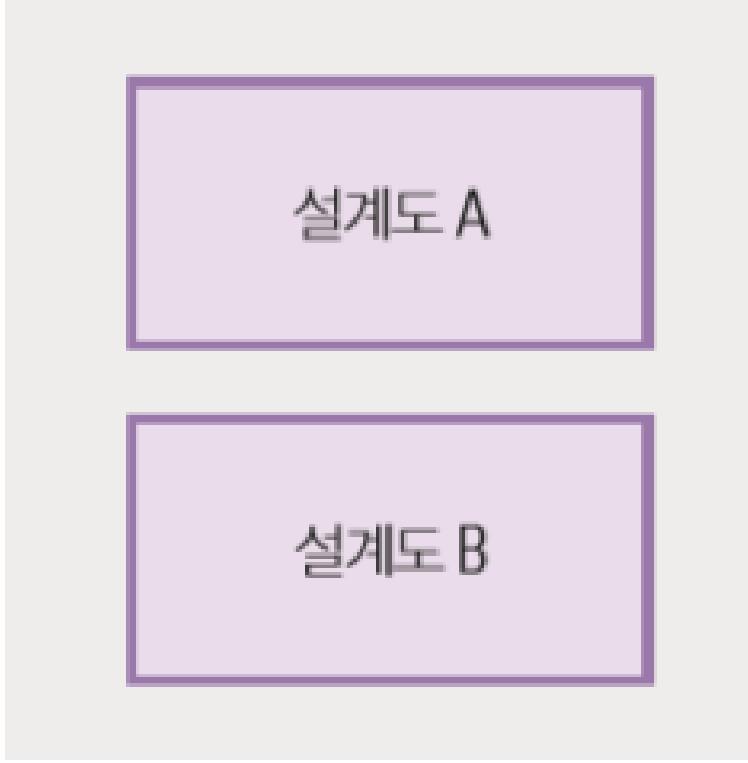
```
class B extends C implements A{    interface A{  
  
    /*                                void methodA( );  
  
    C의 멤버                            }  
  
    A의 멤버                            interface B{  
  
    */                                void methodB( );  
  
    }  
  
    interface C extends A, B{  
  
    }  
}
```

- 오버라이딩 시 부모의 접근자보다 넓거나 같은 범위 사용
- 상속을 받으면서 구현 동시 가능
- 인터페이스 간에도 상속이 가능
- 인터페이스는 다수의 인터페이스 구현 가능

내부클래스

클래스 내부에 생성

큰 틀의 설계도를 완성하기 위한 내부 부품 설계도



설계도 A

설계도 B

내부클래스

내부 클래스의 사용은 클래스 간의 관계가 긴밀할 때 사용하며 코드가 간결해지는 효과가 있따.

```
class OuterClass{ // 외부 클래스
    class InnerClass{ // 내부 클래스
        //...
    }
    //...
}
```

내부클래스

- 인스턴스 클래스

```
class OuterClass{  
    class Innerinstance{ }  
}
```

- 스태틱 클래스

```
class OuterClass{  
    static class InnerStatic{ }  
}
```

- 지역 클래스

```
class OuterClass{  
    void A( ){  
        class InnerLocal{ }  
        //...  
    }  
}
```

내부클래스

```
class OuterClass{  
  
    private class Innerinstance1{  
        //...  
    }  
  
    protected class Innerinstance2{  
        //...  
    }  
}
```

- 내부클래스도 클래스이므로 선언부에 제어자 사용
- 내부 클래스는 멤버 변수와 같이 다루어 지므로 접근제어자 사용

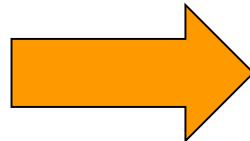
익명 클래스

기존 클래스

```
class Some{  
    private int a = 3;  
    int getter( ) {  
        return this.a;  
    }  
    void setter(int a) {  
        this.a = a;  
    }  
}  
  
Some s1 = new Some( );
```

이름이 없는 클래스

```
Some anonym = new Some( ) {  
    private int a = 10;  
    int getter( ) {  
        return this.a;  
    }  
    void setter(int a) {  
        this.a = a;  
    }  
}; // 세마콜론 주의
```



익명클래스

```
class OuterClass1 // a와 b라는 메서드를 가진 클래스
{
    void a( )
    {
        System.out.println("method a");
    }
    void b( ) { }

}

class Anonymous2
{
    public static void main(String[] args)
    {
        OuterClass1 o = new OuterClass1( )
        {
            void a( ) {
                //익명 클래스 안에 있는 메서드는 객체 o를 통해서만 가능하다
                System.out.println("새롭게 정의한 익명 클래스의 메서드입니다");
            }
        };
        // 익명 클래스는 반드시 마지막에 ;를 붙여야 한다.

        o.a( ); //익명 클래스로 오버라이드한 메서드를 출력
        OuterClass1 ok=new OuterClass1( );
        ok.a( ); // 익명 클래스는 일회용이므로 다시 기존의 메서드가 출력
    }
}
```

예외처리

프로그램을 작동시 종종 예기치 못한 상황을 만난다
개발자가 조치 할 수 없는 수준의 오류를 Error
수습 가능한 덜 심각한 오류를 예외

- 에러 : 개발자가 조치를 취할 수 없는 수준 ex) 메모리 부족, JVM 동작이상
- 컴파일 에러 : 컴파일 시에 발생하는 에러 ex) 오타, 잘못된 자료형 등
- 런타임 에러 : 프로그램이 실행하는 도중에 발생하는 에러
- 로직 에러 : 실행은 되지만 의도와는 다르게 동작하는 에러
- 예외 : 다른 방식으로 처리 가능한 오류 ex) 입력 값 오류, 네트워크 문제

예외처리

간단한 예외 예시

```
package exceptionPackage;  
public class exception1 {  
    public static void main(String[] args) {  
        int a = 0;  
        int b = 2;  
        int c = b/a; // 0으로 나눌 수 없어서 예외 발생  
    }  
}
```

예외가 발생했을 때 이를 적절히 처리하여 프로그램이 비정상적으로 종료 되는 것을 방지

예외처리

Try - catch/finally

```
try{  
    // 예외 발생할 가능성이 있는 문장들  
} catch(예외 1) {  
    // 예외 1에 알맞은 처리  
} catch(예외 2) {  
    // 예외 2에 알맞은 처리  
}
```

예외처리

```
package exceptionpackage;
public class Exception1 {
    public static void main(String[] args) {
        try{
            int[] a = {2,0} ;
            int b = 4;
            int c = b/a[2];
            System.out.println(c);
        } catch(ArithmeticException e) {
            System.out.println("산술 오류발생");
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("배열 길이 오류발생");
        }
        System.out.println("예외 처리 공부 중");
    }
}
```

예외처리

Finally

오류발생 여부에 상관없이 무조건 실행되는 구문

```
try{
```

```
    // 예외 발생할 가능성이 있는 문장들
```

```
} catch(예외 1) {
```

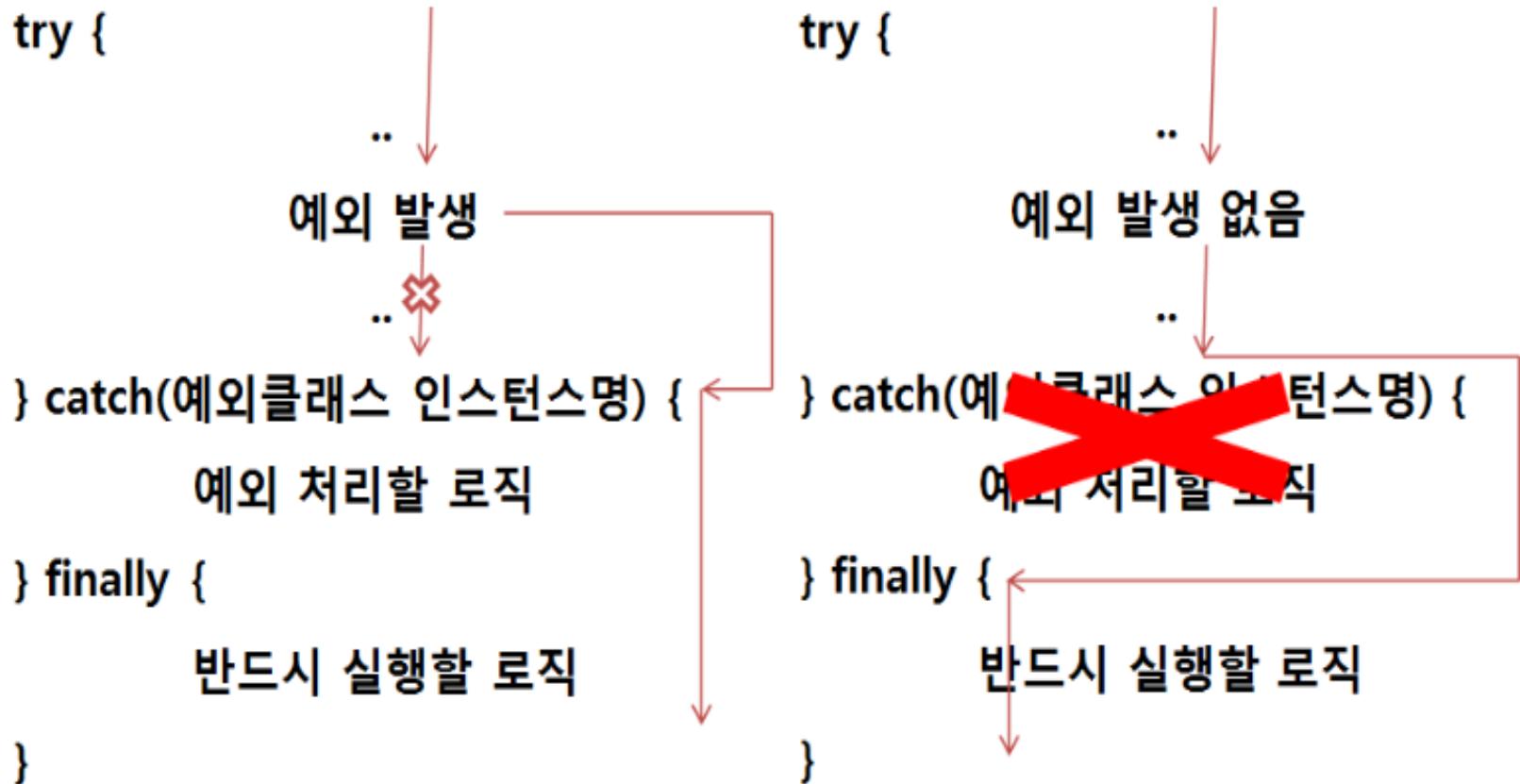
```
    // 예외 1에 알맞은 처리
```

```
} finally{
```

```
    // 예외 여부와 관계없이 실행되는 부분
```

```
}
```

예외처리



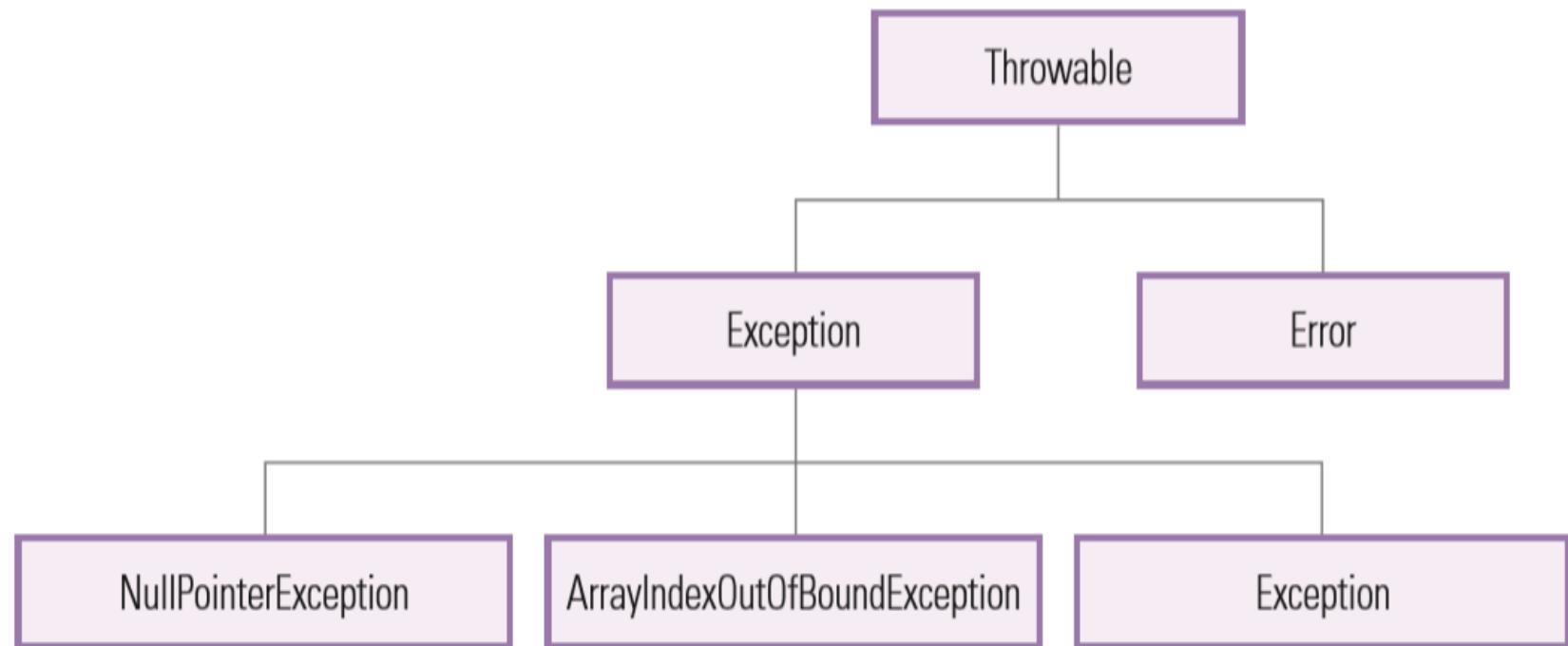
예외처리

```
try{                                public class Finally {  
    System.out.println("외부로 접속");  
    int c = b/a;  
    System.out.println("연결 해제");  
} catch(ArithmeticException e){  
    System.out.println("오류 발생하였습니다.");  
    System.out.println("연결 해제");  
}  
  
public static void main(String[] args) {  
    // 외부로 접근  
    int a = 0;  
    int b = 2;  
    try{  
        System.out.println("외부로 접속");  
        int c = b/a;  
    } catch(ArithmeticException e){  
        System.out.println("오류가 발생했습니다.");  
    } finally{  
        System.out.println("무조건 연결 해제");  
    }  
}
```

예외처리

예외도 객체로 처리

Arithmet icException/ArrayIndexOutOfBoundsException
예외 클래스도 서로 상속의 관계



예외처리

모든 예외 클래스들을 Exception 형의 참조변수로 참조

```
try{  
    // 예외 발생할 가능성이 있는 문장들  
} catch(Exception e) {  
    // 알맞은 처리  
}
```

try{
 ● 예외 1 발생
 ● 예외 2 발생 //실행
 ● 예외 3 발생
} catch(Exception e) {
 // 처리
}

예외처리

Throw

고의로 발생시키는
예외

키워드 throw를
이용해 집적 예외
발생

```
Exception e = new Exception("Exception");
throw e; // 예외 발생
```

```
public class Exception2 {
    public static void main(String[] args) {
        try{
            Exception e = new Exception("고의 예외");
            throw e;
        } catch(Exception e) {
            System.out.println("예외 발생");
            System.out.println(e.getMessage());
        }
    }
}
```

예외처리

예외 던지기

예외가 발생 시 현재 메서드가 예외를 처리 하지 않고
자신을 호출한 쪽에 처리에 대한 책임 전가

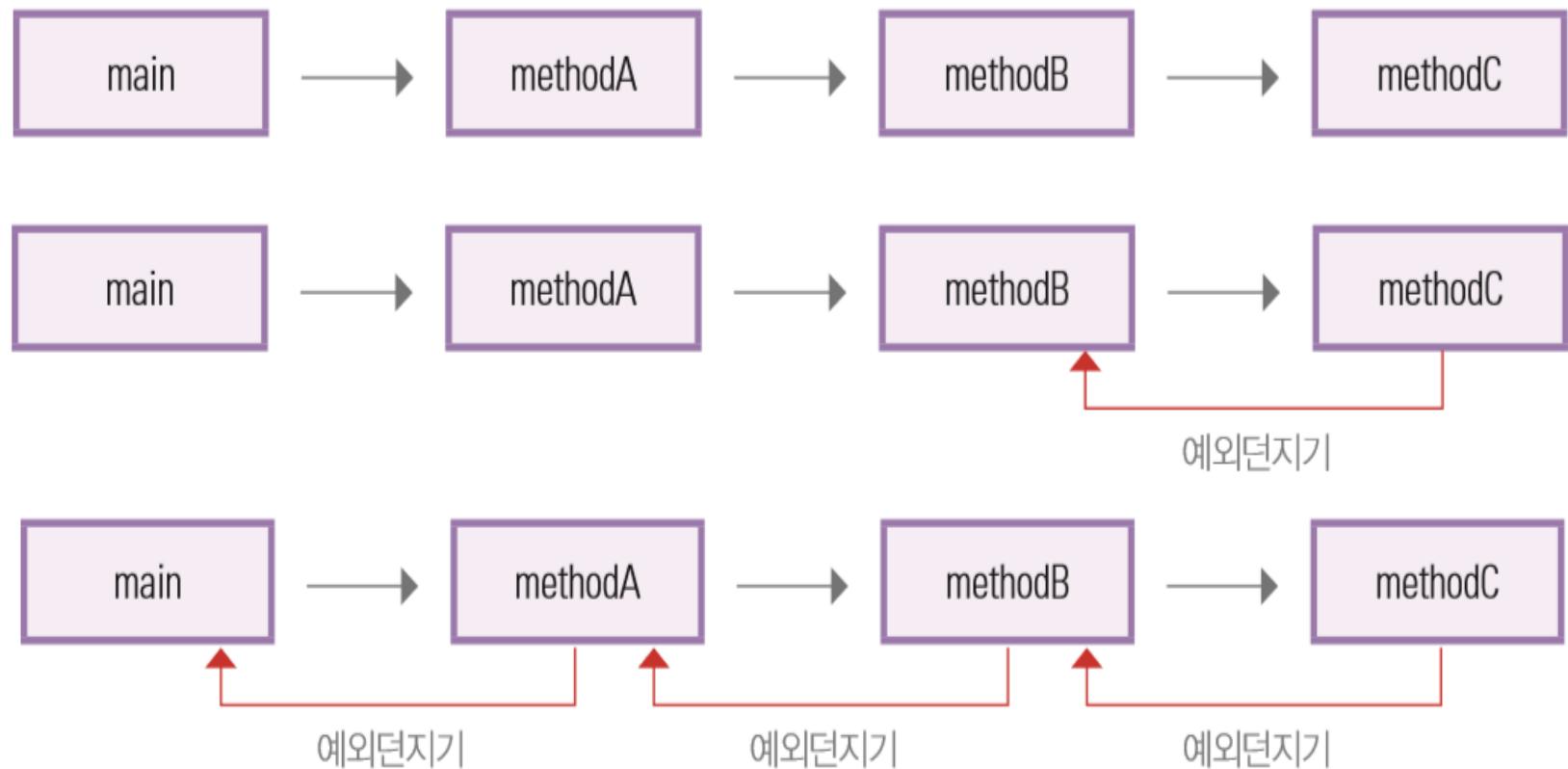
```
void method( ) throws Exception{  
    // 생략  
}
```

호출한 쪽으로 책임을 전가하기에 호출한 쪽에 대해
문법적 강제성이 발생
넘겨받은 쪽은 반드시 집적처리 또는 자신도 예외를
던져야 한다.

예외처리

```
public class Exception3 {  
  
    public static void methodA( ) throws Exception{  
        methodB( );  
    }  
  
    public static void methodB( ) throws Exception{  
        methodC( );  
    }  
  
    public static void methodC( ) throws Exception{  
        Exception e = new Exception( );  
        throw e;  
    }  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        try{  
            methodA( );  
        } catch(Exception e) {  
            System.out.println("메인에서 처리");  
        }  
    }  
}
```

예외처리



예외처리

```
public class RethrowExample {  
    /** Main method */  
    public static void main(String[] args) {  
        // Try some code  
        try {  
            System.out.println("외부 try ....");  
  
            try {  
                System.out.println("내부 try ....");  
                Exception e = new Exception();  
                throw e;  
            }  
        }  
    }  
}
```

```
    catch (Exception e) {  
        System.out.println("(내부 try-catch) exception : " + e);  
        System.out.println("예외 던지기 한번더 : ");  
        throw e;  
    }  
    finally {  
        System.out.println("finally 구문 출력");  
    }  
}  
}  
catch (Exception e) {  
    System.out.println("(외부 try-catch) Catch exception : " + e);  
}  
System.out.println("종료");  
}
```

예외처리

```
class AgeException extends Exception{  
    public AgeException() {}  
    public AgeException(String message){  
        super(message);  
    }  
}  
  
public class Exception5 {  
    public static void ticketing(int age) throws AgeException{  
        if(age < 0){  
            throw new AgeException("나이 | 입력이 잘못되었습니다");  
        }  
    }  
}  
  
public static void main(String[] args) {  
    int age = -19;  
    try {  
        ticketing(age);  
    } catch (AgeException e) {  
        e.printStackTrace();  
    }  
}
```

API

자바에서 개발자들에게 기본적으로 제공하는 클래스

<http://docs.oracle.com/javase/8/docs/api/index.html>

The screenshot shows the Java™ Platform, Standard Edition 8 API Specification page. The top navigation bar includes links for OVERVIEW, PACKAGE, CLASS, USE, TREE, DEPRECATED, INDEX, and HELP. Below the navigation is a toolbar with PREV, NEXT, FRAMES, and NO FRAMES buttons. The main content area features a title 'Java™ Platform, Standard Edition 8 API Specification' and a subtitle 'This document is the API specification for the Java™ Platform, Standard Edition.' It includes sections for 'Profiles' (compact1, compact2, compact3) and 'Packages'. The 'Packages' section is currently selected, displaying a table with columns for Package and Description.

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.

API

Object 클래스

Modifier and Type	메서드	설명
protected Object	clone()	객체의 복사본을 반환합니다.
boolean	equals(Object obj)	obj와 같은 객체인지 아닌지 boolean 값으로 반환합니다.
protected void	finalize()	객체가 소멸할 때 호출됩니다.
Class<?>	getClass()	클래스의 정보를 담고 있는 'Class' 객체를 반환합니다.
int	hashCode()	해시 코드를 반환합니다.
void	notify()	대기 중인 하나의 스레드를 깨웁니다.
void	notifyAll()	대기 중인 모든 스레드를 깨웁니다.
String	toString()	객체의 정보를 문자열로 반환합니다.
void	wait()	현재의 스레드를 멈추고 대기합니다.
void	wait(long timeout)	timeout 만큼 스레드를 멈추고 대기합니다.
void	wait(long timeout, int nanos)	실제로 일어난 시간만큼 스레드를 멈추고 대기합니다.

API

① clone()

clone() 메서드는 인스턴스를 복사할 때 사용됩니다. 자신과 똑같은 인스턴스를 생성할 때 사용합니다.

② equals(Object obj)

equals() 메서드는 객체의 참조변수를 비교하여 같은지 다른지 boolean 값으로 반환하는 메서드입니다.

③ getClass()

getClass() 메서드는 해당 객체의 클래스 정보를 전달하는 메서드입니다.

④ hashCode()

hashCode() 메서드는 객체의 해시 코드 값을 반환하는 메서드입니다. 해시 코드란 해시함수(Hahs Function)를 통해 반환되는 객체가 저장된 위치 값입니다. 즉 서로 다른 객체는 서로 다른 해시 코드 값을 가집니다.

⑤ toString()

toString() 메서드는 객체에 대한 기본적인 정보를 문자열로 반환합니다. Object 클래스에 정의된 toString 메서드는 클래스의 정보와 해시 코드를 반환합니다.

API (String 클래스)

문자열 클래스

메서드	의미
concat(문자열)	문자열을 연결합니다.
substring(인덱스, 잘라낼 길이)	문자열을 잘라냅니다.
length()	문자열의 길이를 반환합니다.
toUpperCase()	문자열을 대문자로 변경합니다.
toLowerCase()	문자열을 소문자로 변경합니다.
charAt(인덱스)	인덱스에 해당하는 글자를 반환합니다.
indexOf(문자열)	문자열의 위치를 반환합니다.
equals(문자열)	문자열이 같은지 boolean 값으로 반환합니다.
trim()	문자열 앞, 뒤에 있는 공백을 제거합니다.
replace(바꾸고 싶은 문자열, 바꿀 문자열)	문자열 내의 특정 부분을 다른 문자열로 변경합니다.
replaceAll(변환할 문자열, 변환될 문자열)	문자열 내의 특정 부분을 다른 문자열로 변경합니다.

API (String클래스)

① concat

concat() 메서드를 이용하면 원하는 문자열을 결합합니다. 하지만 실제로는 문자열 외에 다른 자료형도 결합할 수 있는 + 연산자를 더 많이 이용합니다.

② charAt()

charAt(index) 메서드는 지정한 인덱스의 위치에 있는 문자를 반환해주는 메서드입니다.

③ length()

length() 메서드는 해당 문자열의 길이를 반환하는 메서드입니다. 문자열의 길이를 쉽게 알 수 있기 때문에 문자열 전체에 대해 작업하는 반복문에 사용합니다.

④ indexOf()

indexOf() 메서드는 문자나 문자열이 어디에 위치하는지 알려주는 메서드입니다. 만약 문자열 내에 없다면 -1을 반환합니다.

⑤ substring()

substring() 메서드는 문자열을 잘라낼 때 사용합니다. substring의 파라미터로 인덱스 2개를 사용할 경우에는 첫 번째 인덱스부터 두 번째 인덱스의 전까지 잘라냅니다.

⑥ equals()

equals() 메서드는 문자열의 내용을 비교하는 메서드입니다. 문자열의 내용이 일치하면 true, 일치하지 않으면 false를 반환합니다.

API (String클래스)

⑦ equalsIgnoreCase()

equalsIgnoreCase() 메서드는 영문으로 된 문자열을 대소문자 구분없이 비교하는 메서드입니다. 대소문자를 구분하지 않고 문자열이 일치하면 true, 일치하지 않으면 false를 반환합니다.

⑧ join()

join 메서드는 배열을 특정한 문자들 사이사이에 넣어서 문자열로 결합하는 메서드입니다.

⑨ trim()

trim() 메서드는 문자열의 앞, 뒤 공백을 제거하는 메서드입니다.

⑩ valueOf()

valueOf() 메서드는 기본형을 String으로 변환시키는 메서드입니다.

⑪ format()

String은 문자열에 적절한 포맷을 줄 수 있습니다. 활용방법은 printf() 메서드와 같습니다. 문자열 내에 정수가 들어갈 자리에는 %d, 실수의 자리에는 %f, 다른 문자열의 자리에는 %s를 작성해놓고 순서대로 파라미터를 추가합니다.

⑫ StringBuilder / StringBuffer

String은 immutable(변하지 않는)한 객체이기 때문에 자주 사용할수록 속도가 느려집니다. 왜냐하면 문자열을 작업할 때마다 인스턴스를 만들기 때문입니다. 이러한 단점을 보완하는 클래스가 StringBuilder / StringBuffer 클래스입니다. 이 클래스들은 객체 안의 데이터를 내부적으로 변경할 수 있으므로 새로운 객체를 만들지 않습니다. 즉 mutable(변할 수 있는) 한 객체입니다.

API (String Buffer 클래스)

- StringBuffer 클래스의 메서드

메서드	의미
append()	매개변수로 입력된 값을 문자열로 바꾸어서 더해주는 메서드
reverse()	문자열의 순서를 반대로 나열하는 메서드
insert(int pos, Object obj)	두 번째 매개변수의 값을 문자열로 바꾸어서 pos인 위치에 추가하는 메서드
deleteCharAt(int index)	start의 위치부터 end 직전의 위치의 문자열을 제거하는 메서드
deleteCharAt(int index)	index위치에 있는 문자를 제거하는 메서드

- StringBuffer

StringBuffer는 StringBuilder와 사용하는 방법과 메서드가 똑같습니다. 두 클래스의 차이는 StringBuffer가 나중에 배울 멀티 스레드에서 동기화 처리를 지원한다는 사실입니다. 다만 이 동기화 처리가 StringBuffer의 성능을 저하하므로 지금과 같이 단일 스레드로 작성할 경우에는 StringBuilder를 사용하는 것이 유리하고 멀티스레드의 경우에는 동기화 처리를 지원하는 StringBuffer를 사용하는 것이 좋습니다.

API (String Buffer 클래스)

사용 용도

- String: 문자열 변경이 적을 때, + 연산으로 구성하는 부분이 적을 때
- StringBuffer: 문자열 변경이 많을 때 & 스레드에 안전한 부분 개발 시
- StringBuilder: 문자열 변경이 많을 때 & 스레드와 무관한 부분 개발 시

연산속도

- 느림 < String <<< StringBuffer < StringBuilder < 빠름

메모리 사용량

- 많음 > String >>> StringBuffer = StringBuilder > 적음

API (wrapper 클래스)

Wrapper 클래스

객체로 다루어지지 않는 기본자료형을 객체로 사용

기본 자료형	Wrapper 클래스
boolean	Boolean 클래스
char	Character 클래스
byte	Byte 클래스
short	Short 클래스
int	Integer 클래스
long	Long 클래스
double	Double 클래스
float	Float 클래스

API (wrapper 클래스)

```
Integer i = new Integer(100);
Double d = new Double(3.14);
```

래퍼 클래스 -> 기본자료형

```
Integer i = new Integer(100);
int a = i.intValue(); // int형으로 변환
double b = i.doubleValue(); // double형으로 변환
```

문자열 -> 숫자

```
int a = Integer.parseInt("100"); // 문자 100이 int형 100으로 변환
double b = Double.parseDouble("3.14");
```

API (wrapper 클래스)

- 숫자 → 문자열

```
int number = 36;  
String s = Integer.toString(number);  
System.out.println(s+36); // 3636으로 출력됨
```

- 문자열 → 래퍼 클래스

```
Integer i = Integer.valueOf("100");  
Float f = Float.valueOf("2.65")
```

API (wrapper 클래스)

래퍼 클래스는 Immutable(변하지 않는)한 성격을 지닙니다. 즉 연산할 때마다 새로운 객체가 만들어집니다.

오토박싱은 JDK 1.5버전 이상부터 도입된 개념으로 기본 자료형을 자동으로 객체자료형으로 변환시켜주는 것을 말합니다.

```
Integer i = 100; // 오토박싱 Integer i = new Integer(100)
```

언박싱은 반대로 객체자료형을 자동으로 기본 자료형으로 변환시켜주는 것을 말합니다.

```
Integer i = new Integer(100);
int a = 3;
a = a + i // 언박싱 a = a + i.intValue(); //생략 가능
```

API (wrapper 클래스)

박싱(boxing), 언박싱(un-boxing)

-박싱(boxing)은 기본 자료형의 데이터를 래퍼(wrapper) 클래스의 객체로 만드는 과정을 의미하며,

-언박싱(un boxing)은 래퍼(wrapper) 클래스의 데이터를 기본 자료형으로 얻어내는 과정을 말한다.

(예 : int → Integer 박싱(boxing) / Integer → int 언박싱(un boxing))

API(랜덤 클래스)

무작위 값을 얻고 싶을 때 사용

무작위의 값을 추출하기 위한 알고리즘 소유

기본 종자값은 현재의 시간(currentTimeMillis())

Random()	현재 시간을 종자 값으로 인스턴스 생성
Random(long seed)	seed를 종자 값으로 하는 인스턴스 생성
int nextInt()	무작위의 int 값 반환
int nextInt(int n)	0보다 크고 n보다 작은 int 값을 반환
boolean nextBoolean()	무작위의 boolean 값 반환
long nextLong()	무작위의 long 값 반환
double nextDouble()	무작위의 double 값 반환
float nextFloat()	무작위의 float 값 반환

싱글톤

하나의 해당 클래스에서 단 하나의 인스턴스만 만들도록 보장하는 방법

하나의 인스턴스만은 재사용

불필요한 자원 낭비나 오버헤드 등을 막을 수 있다.

```
class 클래스 {  
    private static 클래스 객체명 = new 클래스(); // 자신의 인스턴스 생성  
    private 클래스( ){ .. } // 생성자 호출 제한  
    public static 클래스 getInstance( ){ // getInstance 메서드 정의  
        return 객체명; // 생성한 인스턴스를 리턴  
    }  
}
```

Java.util

java.util 패키지는 다음과 같은 특징이 있습니다.

- 언어를 인코딩 해주는 기능이 있습니다. 예를 들어, 윈도우에서 한글과 영어를 제외한 중국어, 일본어 등은 출력 시 깨지는 현상이 발생되는데, 이를 정상적으로 출력되도록 하는 기능이 있습니다.
- 배열 대신 활용할 수 있다. 배열이 메모리 공간의 크기를 정해 놓고 사용한다면, 컬렉션 시스템은 무한대로 데이터를 넣을 수 있어 배열보다 수정이나 관리가 편리합니다.
- 날짜를 관리할 수 있는 있습니다. 특히, Date, Calender, GregorianCalendar과 같은 기능이 많이 사용됩니다.
- List 인터페이스를 구현할 수 있습니다. List인터페이스를 구현하는 것들은 전부 다 순서를 가지고 있으며, 초기 용량을 고려하지 않아도 됩니다. 다시 말해, List 인터페이스는 사이즈 변경이 가능한 배열의 구현이라고 정의할 수 있습니다.

Java.util(시간처리)

```
public class Date1 {
```

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

```
        long start = System.currentTimeMillis();
```

```
        System.out.println("시작시간: " + start);
```

```
        int a = 0;
```

```
        for(int i = 1; i < 10000000; i++) {
```

```
            a++;
```

```
}
```

```
        long end = System.currentTimeMillis();
```

```
        System.out.println("종료시간: " + end);
```

```
        System.out.println("걸린 작업 시간: " + (end - start));
```

```
}
```

System.currentTimeMillis

-현 운영체제의 시작을
Long으로 반환

-보통 작업소요시간 측정

Java.util(시간처리)

```
import java.util.Calendar;  
import java.util.GregorianCalendar;  
  
public class Date2 {  
  
    public static void main(String[] args) {  
        Calendar a = Calendar.getInstance(); // 싱글턴 패턴  
        Calendar b = new GregorianCalendar();  
        System.out.println(a.toString());  
        System.out.println(b.toString());  
    }  
}
```

Java.util.Calendar

- 날짜를 다루기 위한 클래스
- 단, 추상클래스이므로 getInstance 메서드를 통해 인스턴스 생성(싱글톤)

Java.util(시간처리)

- **get()** 메서드로 정보 가져오기

Calendar 클래스에서 데이터를 얻어오려면 get() 메서드를 사용합니다. get() 메서드의 반환타입은 int이며 한 가지 주의할 점은 월(MONTH) 반환 시 실제 월보다 1이 작은 값이 반환됩니다. (1월 => 0월, 12월 => 11월) 또한 반환타입이 int이므로 요일을 반환할 때도 정수의 형태로 반환하는데 일요일이 시작인 1이고 월요일이 2, 그리고 토요일이 7입니다.

- **set()**으로 날짜 지정하기

set() 메서드는 Calendar 인스턴스의 값을 변경할 때 사용합니다. 두 개의 파라미터를 필요로 하며 첫 번째 파라미터는 바꿔야 할 항목, 두 번째 파라미터는 바꿀 값을 대입합니다.

```
set(바꿔야할 항목, 바꿀 값);  
set(년,월,일);
```

Java.util(시간처리)

- long 타입을 일(日)로 변경하는 방법

① 1000으로 나누어서 ‘초’단위로 만듭니다. (long 타입은 1/1000까지 보여줌)

$$26179200039 / 1000 = 26179200 \text{ 초}$$

② 초를 분으로 바꾸기 위해 60으로 나누어줍니다. (60초 = 1분)

$$26179200 / 60 = 436320 \text{ 분}$$

③ 분을 시간으로 바꾸어줍니다. (60분 = 1시간)

$$436320 / 60 = 7272 \text{ 시간}$$

④ 시간을 다시 일로 바꾸어줍니다. (24시간 = 1일)

$$7272 / 24 = 303 \text{ 일}$$

Java.util(시간처리)

```
import java.util.Date;  
  
public class Date4 {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        Date today = new Date();  
        System.out.println(today);  
        long a = System.currentTimeMillis();  
        Date today2 = new Date(a);  
        System.out.println(today2);  
    }  
}
```

Java.util.Date

JDK 1.1 부터 문제로 인해
대부분의 메서드와 생성자
사용 X

Java.util(시간처리)

Java.util.SimpleDateFormat

날짜를 형식화하는 클래스

Data와 Calendar을 원하는 형태로 출력하기 위함

기호	의미	기호	의미
y	년	a	오전/오후
M	월	H	시간
d	일	m	분
E	요일	s	초

Java.util(시간처리)

Java.util.Scanner

문자데이터를 읽어오는데 도움을 주는 클래스

Scanner(File/InputStream/String/Readable)

```
Scanner sc = new Scanner(System.in);
String input = sc.nextLine(); // 문자열을 입력받을 땐 nextLine()
```

```
byte nextByte();
```

```
short nextShort();
```

```
long nextLong();
```

Java.util(시간처리)

```
import java.util.Scanner;  
  
public class Scanner1 {  
  
    public static void main(String[] args) {  
  
        Scanner sc = new Scanner(System.in);  
  
        System.out.println("이름을 입력하세요.");  
  
        String name = sc.nextLine();  
  
        System.out.println(name+"님 환영합니다.");  
    }  
}
```

```
import java.io.FileInputStream;  
  
import java.util.Scanner;  
  
public class Prac {  
  
    public static void main( String args[] ) {  
  
        try {  
  
            FileInputStream fis = new FileInputStream("src\efefe\sample");  
  
            Scanner s = new Scanner( fis );  
  
            while ( s.hasNext( ) ) { // 값이 존재하면 계속해서 반복.  
                System.out.println( s.nextLine( ) );  
            }  
  
            catch ( Exception e ) {  
                e.printStackTrace( );  
            }  
        }  
    }  
}
```

Java.util(시간처리)

Java.util.time

Jdk 1.8에 추가 Date/Calendar 보완

날짜와 시간 변경, 실제로 변경X 새로운 객체 반환

패키지	설명
java.time	날짜와 시간을 나타내는 LocalDate, LocalTime 등을 포함한 패키지
java.time.format	날짜와 시간을 파싱하고 포맷팅하는 API 포함
java.time.chrono	여러 가지 달력 시스템을 사용할 수 있는 API 포함
java.time.temporal	날짜와 시간을 연산하기 위한 API 포함
java.time.zone	타임존을 지원하는 API 포함

Java.util(시간처리)

- **LocalDate** 클래스

LocalDate 클래스에는 날짜 정보를 저장합니다. LocalDate 클래스에 날짜정보를 저장하는 방법은 현재 날짜를 지정하는 now() 메서드를 쓰는 방법과 특정 날짜를 지정하는 of() 메서드를 사용하는 방법이 있습니다.

```
LocalDate ld = LocalDate.now();
```

```
LocalDate mld = LocalDate.of(int year, int month, int dayOfMonth);
```

Java.util(시간처리)

● Localtime 클래스

LocalTime 클래스에는 시간 정보를 저장합니다. LocalTime 클래스에 시간정보를 저장하는 방법은 현재 시간을 지정하는 now() 메서드를 쓰는 방법과, 특정 시간을 지정하는 of() 메서드를 사용하는 방법이 있습니다.

```
LocalTime lt = LocalTime.now();
LocalTime mlt = LocalTime.of(int hour, int minute, int second, int nanoOfSecond);
```

of() 메서드의 경우에는 파라미터의 개수나 종류에 따라 다양하게 사용합니다.

```
of(int hour, int minute);
of(int hour, int minute, int second);
of(int hour, int minute, int second, int nanoOfSecond);
```

Java.util(시간처리)

● LocalDateTime 클래스

LocalDateTime 클래스는 LocalDate 클래스와 LocalTime 클래스를 결합한 클래스입니다. 날짜 정보와 시간 정보 모두 저장합니다. LocalDateTime 클래스에 날짜와 시간정보를 저장하는 방법은 현재 날짜와 시간정보를 지정하는 now() 메서드를 쓰는 방법과, 특정 날짜와 시간정보를 지정하는 of() 메서드를 사용하는 방법이 있습니다.

```
LocalDateTime ldt = LocalDateTime.now();
```

```
LocalDateTime mldt = LocalDateTime.of(int year, int month, int dayOfMonth, int hour, int minute, int second, int nanoOfSecond);
```

Java.util(시간처리)

- **ZonedDateTime** 클래스

ZoneDateTime 클래스는 ISO-8601 달력 시스템에서 정의하는 TimeZone에 따라 날짜와 시간을 저장하는 클래스입니다.

```
2016-01-08T12:56:09.017+09:00[Asia/Seoul]
```

```
ZonedDateTime zdt = ZonedDateTime.now(ZoneId.of("UTC"));
```

Java.util(시간처리)

● Instant

Instant는 특정 시점의 타임스탬프 객체입니다. 1970년 1월 1일부터 현재까지의 시간을 세는 객체이며 사람이 사용하기보다는 Machine time에 유리합니다.

```
Instant i1 = Instant.now();
```

Java.util(시간처리)

LocalDate/LocalTime

월요일 = 1, 화요일 = 2 … 일요일 = 7

isLeapYear() → toLocalDate() 클래스 변환

클래스	리턴타입	메서드	설명
LocalDate	int	getYear()	년도
	Month	getMonth()	Month의 열거값
	int	getMonthValue()	월
	int	getDayOfYear()	1년의 몇 번째 일
	int	getDayOfMonth()	월의 몇 번째 일
	DayOfWeek	getDayOfWeek()	요일
	boolean	isLeapYear()	윤년 여부
LocalTime	int	getHour()	시간
	int	getMinute()	분
	int	getSecond()	초
	int	getNano()	나노초

Java.util(시간처리)

클래스들의 정보 더하기/빼기

메서드	설명	메서드	설명
minusYears(long)	년도 빼기	plusWeeks(long)	주 더하기
minusMonths(long)	월 빼기	minusHours(long)	시간 빼기
minusDays(long)	일 빼기	minusMinutes(long)	분 빼기
minusWeeks(long)	주 빼기	minusSeconds(long)	초 빼기
plusYears(long)	년도 더하기	minusNanos(long)	나노초 빼기
plusMonths(long)	월 더하기	plusHours(long)	시간 더하기
plusDays(long)	일 더하기	plusMinutes(long)	분 더하기

Java.util(시간처리)

클래스들의 정보 값 변경

메서드	설명	메서드	설명
withYear(int)	년 변경	withHour(int)	시간 변경
withMonth(int)	월 변경	withMinute(int)	분 변경
withDayOfMonth(int)	월의 일 변경	withSecond(int)	초 변경
withDayOfYear(int)	년의 일 변경	withNano(int)	나노초 변경

Java.util(시간처리)

특정한 날짜를 리턴

메서드	설명
firstDayOfYear()	년도의 첫 번째 일
lastDayOfYear()	년도의 마지막 일
firstDayOfMonth()	달의 첫 번째 일
lastDayOfMonth()	달의 마지막 일
firstInMonth(DayofWeek dayOfWeek)	달의 첫 번째 요일
lastInMonth(DayOfWeek dayOfWeek)	달의 마지막 요일
next(DayOfWeek dayOfWeek)	돌아오는 요일
nextOrSame(DayOfWeek dayOfWeek)	오늘을 포함한 돌아오는 요일
previous(DayOfWeek dayOfWeek)	지난 요일
previousOrSame(DayOfWeek dayOfWeek)	오늘을 포함한 지난 요일

Java.util(시간처리)

Time 패키지에서는 각 필드 값을 비교하는 메서드 사용

메서드	설명
isAfter()	이전의 날짜인지 비교하여 boolean값 반환
isBefore()	지나간 날짜인지 비교하여 boolean값 반환
isEqual()	동일 날짜인지 비교하여 boolean값 반환
until()	날짜나 시간의 차이를 반환
between()	전체 날짜나 시간의 차이를 반환

Java.util(Timer/TimerTask)

특정한 시간에 코드를 실행하거나 특정시간 간격에 반복 작업

```
Class Work1 extends TimerTask{  
    // run 추상 메서드 오버라이딩  
    public void run( ) {  
        System.out.println("work1 실행");  
    }  
}
```

```
Timer t = new Timer(true);  
TimerTask w1 = new Work1( );  
t.schedule(w1,5000); // 5초 뒤에 실행
```

Java.util(BigDecimal)

정확한 소수점

연산시

double/long으로

소수점 오차가

나는 부분을

보완

```
import java.math.BigDecimal;  
public class BigDecimal1 {  
    public static void main(String[] args) {  
        double a = 24.3953;  
        double b = 50.343998;  
        System.out.println(a+b);  
    }  
}
```

```
BigDecimal number = new BigDecimal(String.valueOf(a));
```

```
BigDecimal number2 = new BigDecimal(String.valueOf(b));
```

```
System.out.println(number.add(number2));
```

```
}
```

```
}
```

Java.util(DecimalFormat)

데이터를 볼 때 일정한 형식으로 통일해서 보는게 편함
이런 형식을 지정해주는 클래스

기호	의미	기호	의미
#	있으면 출력	,	콤마를 넣음
0	없어도 0으로 채움	E	지수 기호
.	소수점	%	퍼센트
-	음수 기호를 붙임		

Collection Framework

프로그래밍에서 많은 데이터를 저장할 때는 어떤 형태로 저장하는 것이 효율적일지 고려해야 합니다. 그러한 고민을 통해 만들어진 효율적인 구조를 자료구조(Data Structure)라고 하며, 자바에서는 이를 클래스로 제공합니다. 이 클래스들의 집합을 컬렉션 프레임워크(Collection Framework)라고 합니다.

컬렉션 프레임워크는 Collection 인터페이스를 구현하는 Set, List, Queue 그리고 독자적인 Map 인터페이스가 있습니다. Collection 인터페이스의 중요한 메서드는 다음과 같습니다. Collection 인터페이스를 구현하는 클래스들은 같은 조작방법을 가지게 됩니다.

메서드	설명
boolean add(E e)	파라미터로 지정된 요소를 컬렉션에 추가
boolean contains(Object obj)	객체 obj가 컬렉션에 존재하는지 여부
boolean isEmpty()	컬렉션이 비어있는지 아닌지 여부
Iterator iterator()	해당 컬렉션 요소의 iterator 객체를 반환
boolean remove(Object obj)	객체 obj를 제거
int size()	요소의 개수를 반환

Collection Framework

컬렉션 프레임워크 구성

Collection : 인스턴스 단위의 데이터 저장 기능 제공(배열과 같이 단순 인스턴스 참조값 저장)

Set – 중복없는 객체를 모아둘 때 사용, 순서X (SortedSet)

List – 순서가 있는 객체를 모아둘 때 사용, 중복O (Vector, Stack, LinkedList, ArrayList)

※ List와 Set 차이점 – 중복 가능 유무, 순서 유지 유무

Map 항상 Key와 Value 쌍(pair)으로 인스턴스를 저장하는 자료구조를 제공 (HashTable, HashMap)
사전과 구조가 비슷한 객체를 저장하는 클래스, key – value

Sorted가 붙은 클래스

순서가 있는 데이터를 관리하는 데 사용되는 클래스

컬렉션 프레임워크의 인터페이스를 사용하려면 실제 구현 클래스의 인스턴스를 생성해야 함.

컬렉션 프레임워크 특징

1. import java.util.*;
2. Data 를 저장하는 방식에 따라 class 이미 정의 되어 있음.
3. 여러 가지 Type 을 여러 개 저장 가능
4. 검색/수정/삭제/삽입이 편리함 => method 가 존재함
5. 객체만 저장 가능함. (기본형(primitive Type)은 안됨, 단 Wraper 클래스 이용)

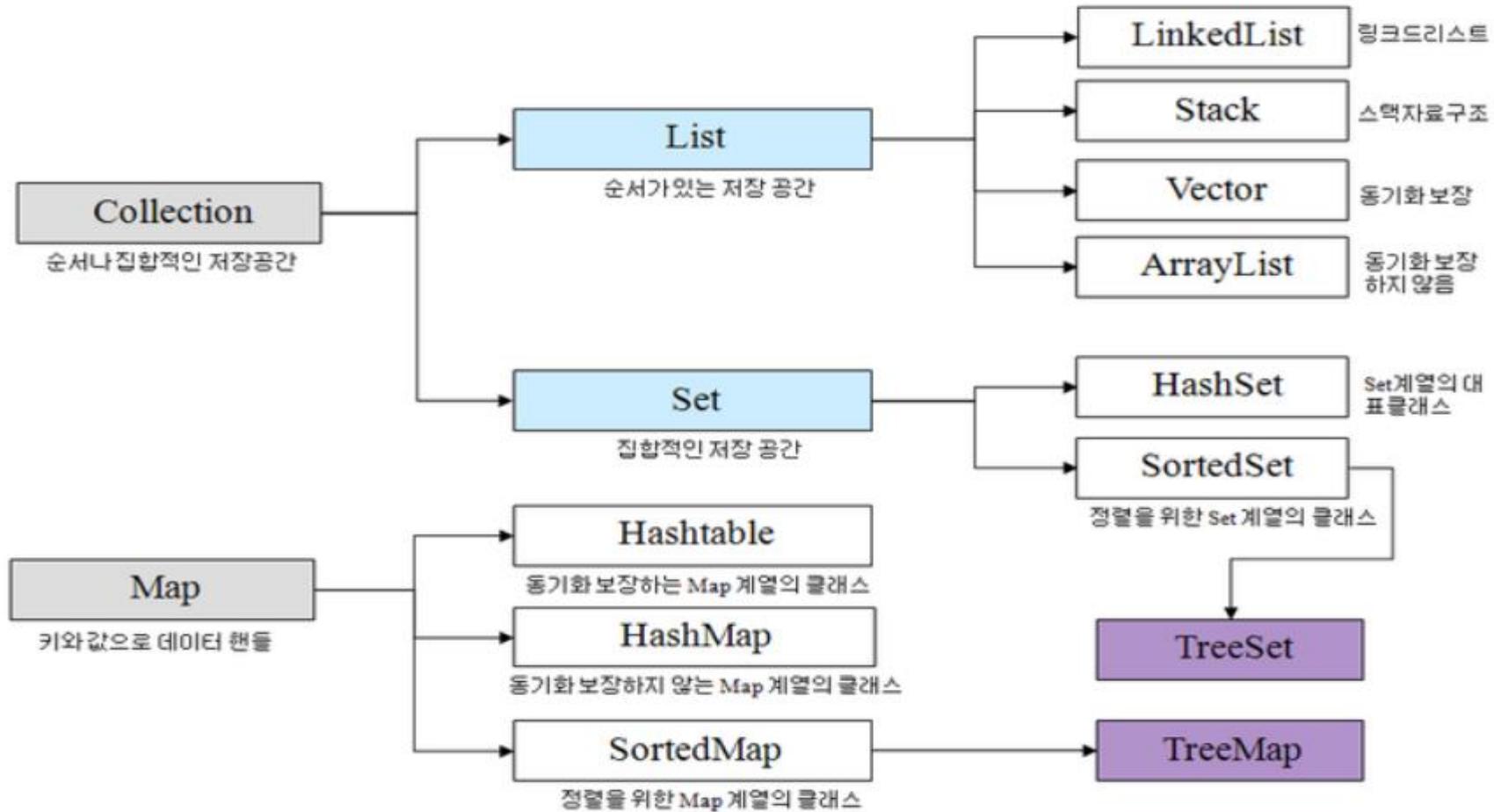
Collection Framework

컬렉션 프레임워크의 핵심 인터페이스 – List, Set, Map

인터페이스	특징
List	순서가 있는데 데이터집합, 데이터의 중복을 허용 예) 대기자 명단
	구현 클래스 : ArrayList, LinkedList, Stack, Vector 등
Set	한마디로 수학에서 집합의 개념, 순서를 유지하지 않는 데이터의 집합, 데이터의 중복을 허용하지 않음 예) 양의 정수집합, 소수의 집합
	구현 클래스 : HashSet, TreeSet; 데이터를 오름차순정렬 등
Map	키(key)와 값(value)의 쌍(pair)으로 이루어진 데이터의 집합 순서는 유지되지 않으며, 키는 중복을 허용하지 않고, 값은 중복을 허용 예) 우편번호, 지역번호(전화번호)
	구현 클래스 : HashMap, TreeMap, Hashtable, Properties 등

※ 배열에서 좀 더 사용하기 편리하게 변경된 내용이 **컬렉션**이며, 컬렉션을 특정 타입에 맞게 형식화 해 놓은 개념이 **제네릭**입니다

Collection Framework



Collection Framework

Iterator 객체의 메서드	설명
hasNext()	다음 요소가 있는지 없는지 판단
next()	다음 요소를 반환

while문의 조건으로 it.hasNext()를 사용했습니다. 이렇게 작성하면 다음 요소가 있으면 계속해서 작업하고 없으면 멈추게 되어 모든 요소에 대해 작업이 가능합니다. it.next() 메서드는 다음 요소를 반환하기 때문에 while문과 더불어서 원하는 작업을 합니다.

```
while(it.hasNext( )) {  
    System.out.println(it.next( ))  
}
```

List-ArrayList

여러 변수를 효과적으로 관리하기 위해 배열형태의 변수에 데이터를 집어넣어 처리
리스트의 자료에는 순서 존재, 데이터 같아도 중복 허용

객체생성

배열형태<자료의속성> 변수명=new 배열형태<자료속성>();

삽입

메서드	설명
void add(int index, E elem)	index 위치에 요소 elem 추가
E get(int index)	index 위치에 있는 요소를 반환
int indexOf(Object o)	요소 o가 있는 위치를 반환
ListIterator listIterator()	ListIterator 객체를 반환
E remove(int index)	index 위치에 있는 요소 삭제하고 삭제한 요소를 반환
E set(int index, E elem)	지정한 index의 요소를 elem으로 변경

Collection Framework

[ArrayList 사용방법]

리스트 생성

```
ArrayList<String> list = new ArrayList<String>();
```

데이터 추가 방법

```
list.add("값"); //타입 파라미터와 맞는 데이터 값을 넘겨주어야함, ArrayList에 순서대로 데이터가 저장
```

데이터를 가져오는 방법

```
String str = list.get(2); // 인덱스 2 위치에 있는 데이터를 리턴
```

데이터의 수를 가져오는 방법

```
int num = list.size(); //리스트에 있는 데이터의 수를 리턴
```

데이터를 중간에 삽입하는 방법

```
list.add(2, "데이터"); //인덱스 2위치에 데이터를 삽입
```

기존 데이터를 교체하는 방법

```
list.set(0, "데이터"); //인덱스 0위치에 있는 데이터를 새 데이터로 교체.
```

데이터를 삭제하는 방법

```
list.remove(1); //인덱스 1 위치에 있는 데이터를 삭제
```

```
list.remove("데이터") //리스트에서 해당데이터를 찾아서 삭제
```

데이터를 검색하는 방법

```
int index = list.indexOf("데이터 1"); //첫번째 데이터값의 위치를 리턴 (내부적으로 equals()메서드 사용)
```

데이터를 뒤에서부터 검색하는 방법

```
int index = list.lastIndexOf("데이터 1"); //마지막 데이터값의 위치를 리턴 (내부적으로 equals()메서드 사용)
```

제네릭(Generic)

클래스가 다룰 객체를 명시
ArrayList는 데이터 타입 자동형변환

```
ArrayList al = new ArrayList();
al.add("apple");
al.add(1);

// String i = al.get(0); // 캐스팅 하지 않을 경우 오류 발생
String s = (String)al.get(0);
int a = (int)al.get(1);
```

제네릭을 사용 데이터 타입을 선언
다른 타입은 넣지 못하게 강제함

```
ArrayList<String> al = new ArrayList<String>();
al.add("apple");
// al.add(1); // 오류발생
```

Set - HashSet

요소들을 집합적으로 모아놓은 자료구조
중복된 데이터 X 저장 순서를 유지하지 않는다.
HashSet, TreeSet

Set 인터페이스

1. 중복된 요소를 가지지 않는다. // 중복된 요소를 추가하면 false 반환
2. 저장 순서를 유지하지 않는다.

생성자	설명
HashSet()	HashSet 클래스의 기본 생성자
HashSet(Collection collection)	컬렉션의 요소로 HashSet 객체 생성
HashSet(int capacity)	capacity의 용량을 가진 객체 생성

Set-TreeSet

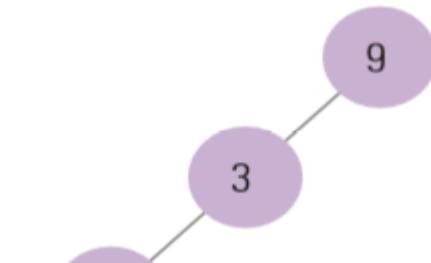
기존의 set 인터페이스에서 정렬기능이 추가

- TreeSet에 $\langle 9, 3, 2, 5, 10 \rangle$ 의 데이터를 저장할 경우

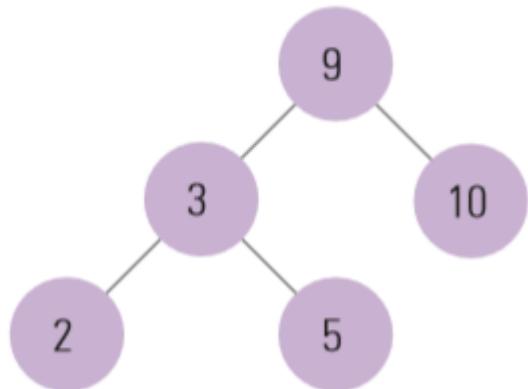
① 9 저장



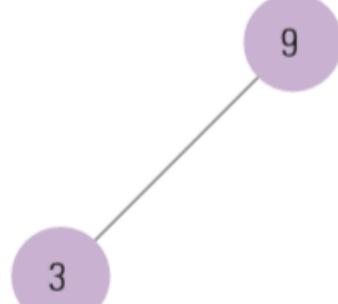
③ 2 저장



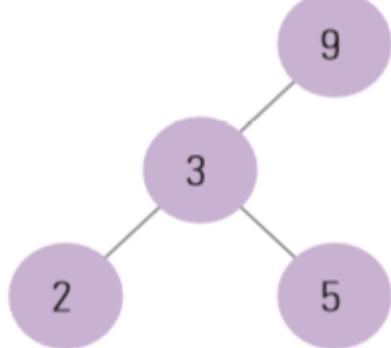
⑤ 10 저장



② 3 저장



④ 5 저장



List-ArrayList(Vector)

List 파생 클래스

삽입과 출력은 ArrayList와 동일

Map

키와 값을 쌍으로 저장하는 구조

키를 통해서 값을 참조 수학의 함수와 비슷한 구조

HashMap, TreeMap

메서드	설명
boolean containsKey(Object key)	해당 키가 있는지 여부 반환
boolean containsValue(Object value)	해당 값이 있는지 여부 반환
V get(Object key)	해당 key와 쌍인 값을 반환
boolean isEmpty()	맵 객체에 요소가 없는지 판단
Set<key> keySet()	키들을 Set 형태로 반환
V put(K key, V value)	키와 값을 요소로 추가
V remove(Object key)	이 키를 가진 요소를 제거
int size()	전체 요소의 개수를 반환

Map-HashMap

HashMap이란 Map 인터페이스의 한 종류로 Key와 Value 값으로 데이터를 저장하는 형태를 가지고 있습니다. HashMap은 Map의 속성을 모두 가지고 있으며 저장 방식도 동일합니다. 그리고 해싱(hashing)이란 검색 방법을 사용하기 때문에 많은 양의 데이터를 검색하는데 있어서 뛰어난 성능을 보여줍니다. HashMap에서 주의할 점은 Map에 데이터를 등록할 때, key 값은 중복이 허용되지 않지만, value 값은 중복이 허용된다는 점입니다. 다시 말해 key 값을 컬렉션 내의 유일한 key이어야 하고, value 값은 같은 중복이 있어도 상관없습니다.

Thread

프로그램을 실행시키면 메모리가 할당되어 실행되는데 이 실행중인 프로그램을 프로세서라 한다.

프로세스에서 작업을 수행하는 것이 Thread이다.

두가지 이상 작업시에는 두 개 이상의 스레드가 필요

① 시간분할 방식

시간분할 방식은 모든 프로세스에게 동일한 시간을 할당하고 골고루 실행합니다.

② 선점형 방식

선점형 방식은 각각의 프로세스에게 우선순위를 부여하고 우선순위가 높은 순으로 실행되는 방식입니다. 우선순위가 높은 프로세스가 잠시 멈추면 그다음 순위의 프로세스가 동작합니다.

Thread

Thread 클래스를 상속받고 run() 메서드를 오버라이딩

```
class Th1 extends Thread{  
    public void run( ) {  
        // 작업할 내용  
    }  
}
```

실행할 때는 인스턴스를 생성하고 start() 메서드로 실행

```
Th1 t1 = new Th1( ); // 인스턴스 생성  
t1.start( ); // 스레드 실행
```

Thread

Runnable 인터페이스를 구현하고 run() 오버라이딩

```
class Th2 implements Runnable{  
    public void run( ) {  
        // 작업할 내용 작성  
    }  
}
```

Runnable 인터페이스를 구현한 클래스의 인스턴스를 먼저 생성한 다음 그 인수로 Thread 클래스에 전달하여 스레드 생성

```
Th2 t2 = new Th2( );  
Thread t = new Thread(t2);  
t.start( );
```

Thread

<Thread 메소드>

static void sleep(long msec) throws InterruptedException	msec에 지정된 밀리초 동안 대기
String getName()	스레드의 이름을 s로 설정
void setName(String s)	스레드의 이름을 s로 설정
void start()	스레드를 시작 run() 메소드 호출
int getPriority()	스레드의 우선 순위를 반환
void setPriority(int p)	스레드의 우선순위를 p값으로
boolean isAlive()	스레드가 시작되었고 아직 끝나지 않았으면 true 끝났으면 false 반환
void join() throws InterruptedException	스레드가 끝날 때 까지 대기
void run()	스레드가 실행할 부분 기술 (오버라이딩 사용)
void suspend()	스레드가 일시정지 resume()에 의해 다시시작 할 수 있다.
void resume()	일시 정지된 스레드를 다시 시작.
void yield()	다른 스레드에게 실행 상태를 양보하고 자신은 준비 상태로

Thread(우선순위)

시분할 방식으로 CPU의 시간을 분배하여 실행 다만 사용자가 직접 스레드의 우선순위를 지정해서 더 많은 시간 할애 가능.

void setPriority(int newPriority);	우선순위를 설정합니다.
int getPriority();	우선순위를 반환합니다.

Thread(라이프사이클)

① new

스레드가 키워드 `new`를 통해서 인스턴스화된 상태입니다. `Runnable`이 될 수 있는 상태이며 아직 대기열에 올라 가지 않은 상태입니다.

② Runnable

`start()` 메서드가 호출되면 `new` 상태에서 `Runnable` 상태가 됩니다. `Runnable` 상태가 되면 실행할 수 있는 상태로 대기하게 됩니다. 스케줄러에 의해 선택이 되면 `run()` 메서드를 바로 수행합니다.

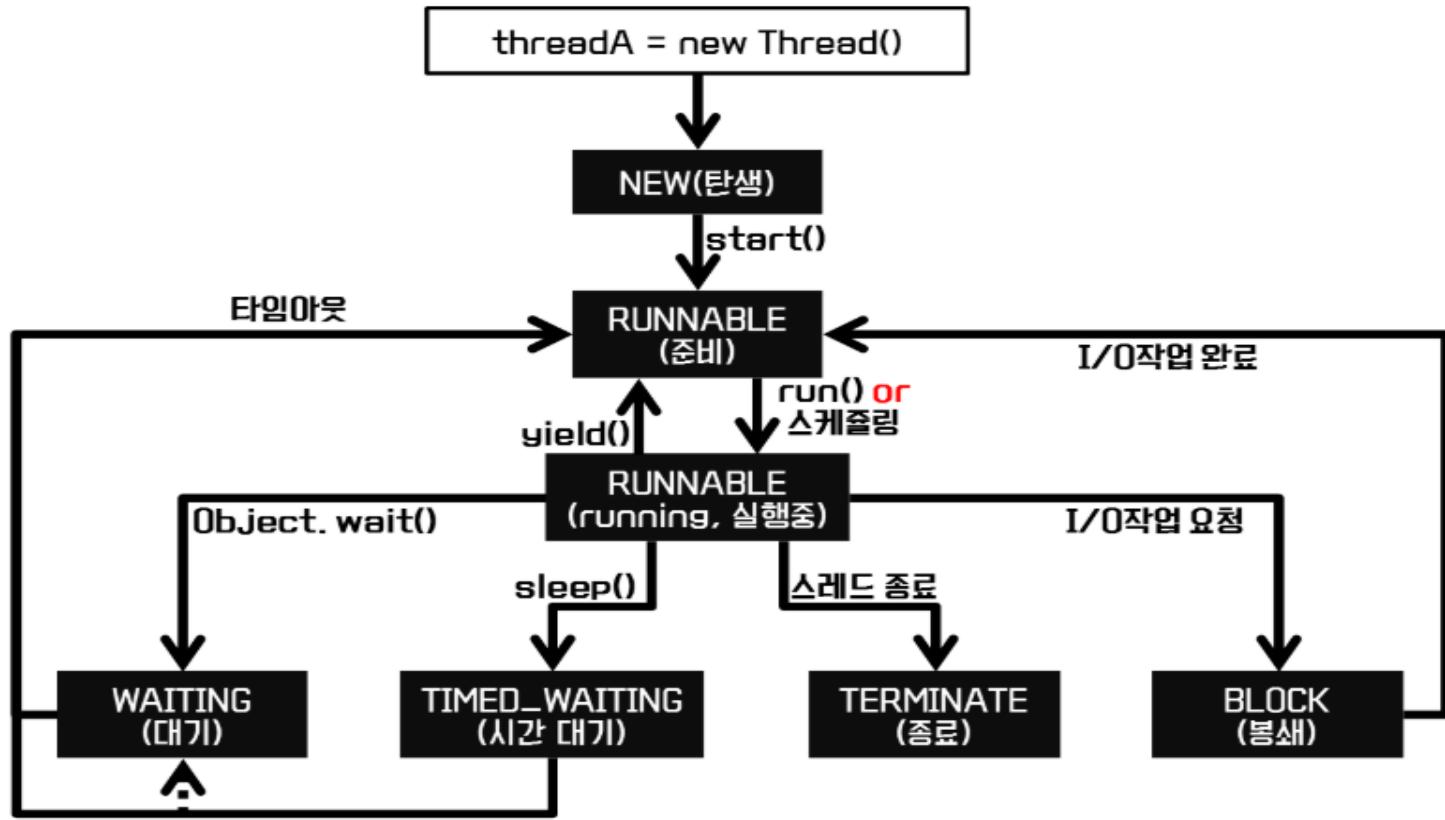
③ Blocked

실행중인 스레드가 `sleep()`, `join()` 메서드를 호출하게 되면 `Blocked` 상태가 됩니다. `Blocked` 상태가 되면 스케줄러에 의해서 선택받을 수 없습니다.

④ Dead

`run()` 메서드의 실행을 모두 완료하게 되면 스레드는 `Dead` 상태가 됩니다. 할당받은 메모리와 정보가 삭제됩니다.

Thread(라이프사이클)



```
< threadB >
Object.notify();
Object.notifyAll();
```

Thread(라이프사이클)

Sleep()

스레드를 지정된 시간 동안 Block 상태로 만듬

Yield()

자신의 시간을 양보하는 메서드

Join()

특정한 스레드가 작업을 먼저 수행할 때 사용
실행순서를 지켜야 하는 스레드를 제어

```
join()
```

```
join(long millis)
```

```
join(long millis, int nanos)
```

Thread(동기화)

멀티스레드로 작업을 할 때 스레드간 작업 간섭 방지

단일 스레드로 작업할 때는 프로세스의 자원을 사용하는데 아무런 문제가 없습니다. 하지만 멀티스레드 프로세스의 경우 여러 스레드가 같은 자원을 공유하게 되기 때문에 예기치 못한 결과를 불러올 수도 있습니다. 예를 들어 A와 B가 같은 메서드를 사용할 경우 A가 메서드의 사용을 완료하고 그 결과를 반영하기 전에 B가 메서드를 사용한다면 B의 메서드 사용이 A의 결과에 영향을 미치게 됩니다. 이것은 마치 화장실 한 칸을 두고 여러 사람이 이용할 때 앞사람이 나오기 전에 뒤에 사람이 들어가는 것과 같은 오류입니다.

Thread

스레드를 대기시키는 `wait()` 메서드 대기 중인 스레드
깨우는 `notify()` 메서드를 통해 스레드 제어

동기화를 통해 객체의 Lock을 한 스레드의 작업이 완료될
때까지 제공함으로 데이터를 보호 특정한 조건으로
완료되지 못하는 상태라면 Lock이 해지 되지 않는다.

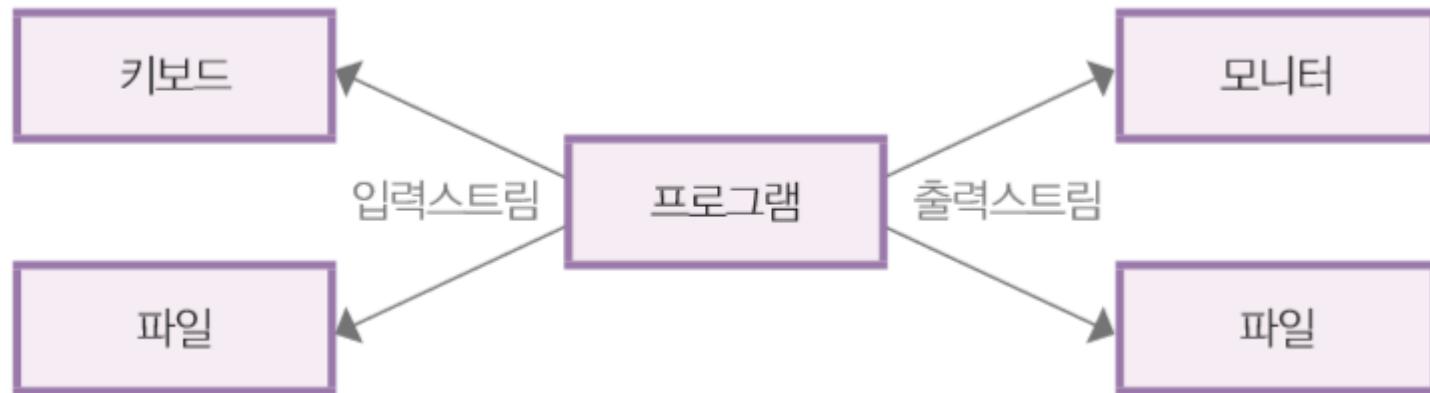
이런 세부적인 문제들을 제어하는 것이 `Wait()`,`Notify()`

```
void wait()
void wait(long timeout)
void wait(long timeout, int nanos)
void notify()
void notifyAll()
```

I/O와 스트림(Stream)

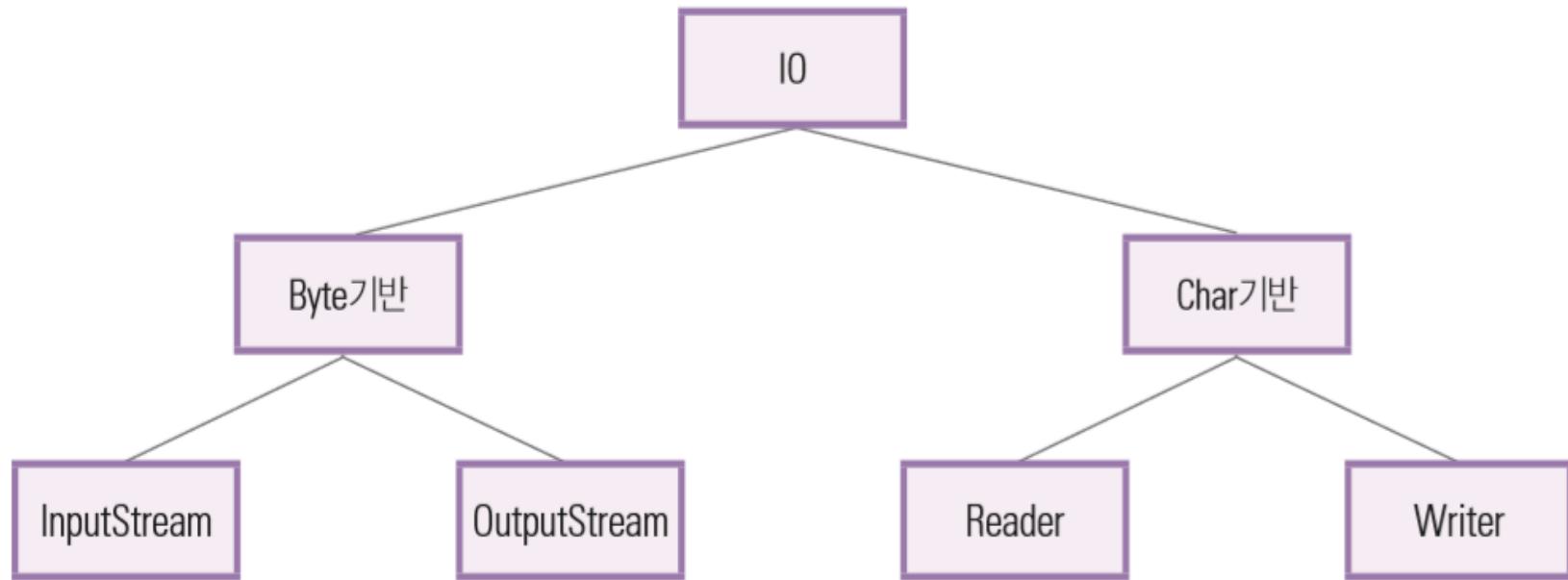
데이터를 읽고 기록하는 것을 데이터의 input, output 라
하고 데이터를 주고 받는 작업을 도와주는 역할을 스트림

자바에서는 데이터의 흐름을 의미



I/O와 스트림(Stream)

데이터의 통신은 한 쪽 방향으로만 가능
입력과 출력 스트림은 각각 따로 사용(FIFO 구조)



Byte 기반 스트림(Stream)

1byte씩 읽어서 1byte씩 출력

InputStream ->read() 추상메서드 구현

OutputStream ->write() 추상메서드 구현

InputStream	OutputStream
abstract int read()	abstract void write(int b)
int read(byte[] b)	void write(byte[] b)
int read(byte[] b, int off, int len)	void write(byte[] b, int off, int len)

입력 스트림	출력스트림	대상
FileInputStream	FileOuputStream	파일
PipedInputStream	PipedOuputStream	메모리
AudioInputStream	AudioOuputStream	오디오
ByteArrayInputStream	ByteArrayOuputStream	프로세스

Byte 기반 스트림(Stream)

스트림 사용을 위해서는 인스턴스 생성시 생성자의 인자로 데이터를 입력 받거나 입력 받을 곳을 넘겨준 다음 적절한 메소드 사용

```
FileInputStream fis = null;  
fis = new FileInputStream("읽어올 파일의 위치");  
fis.read();
```

Byte 기반 스트림(Stream)

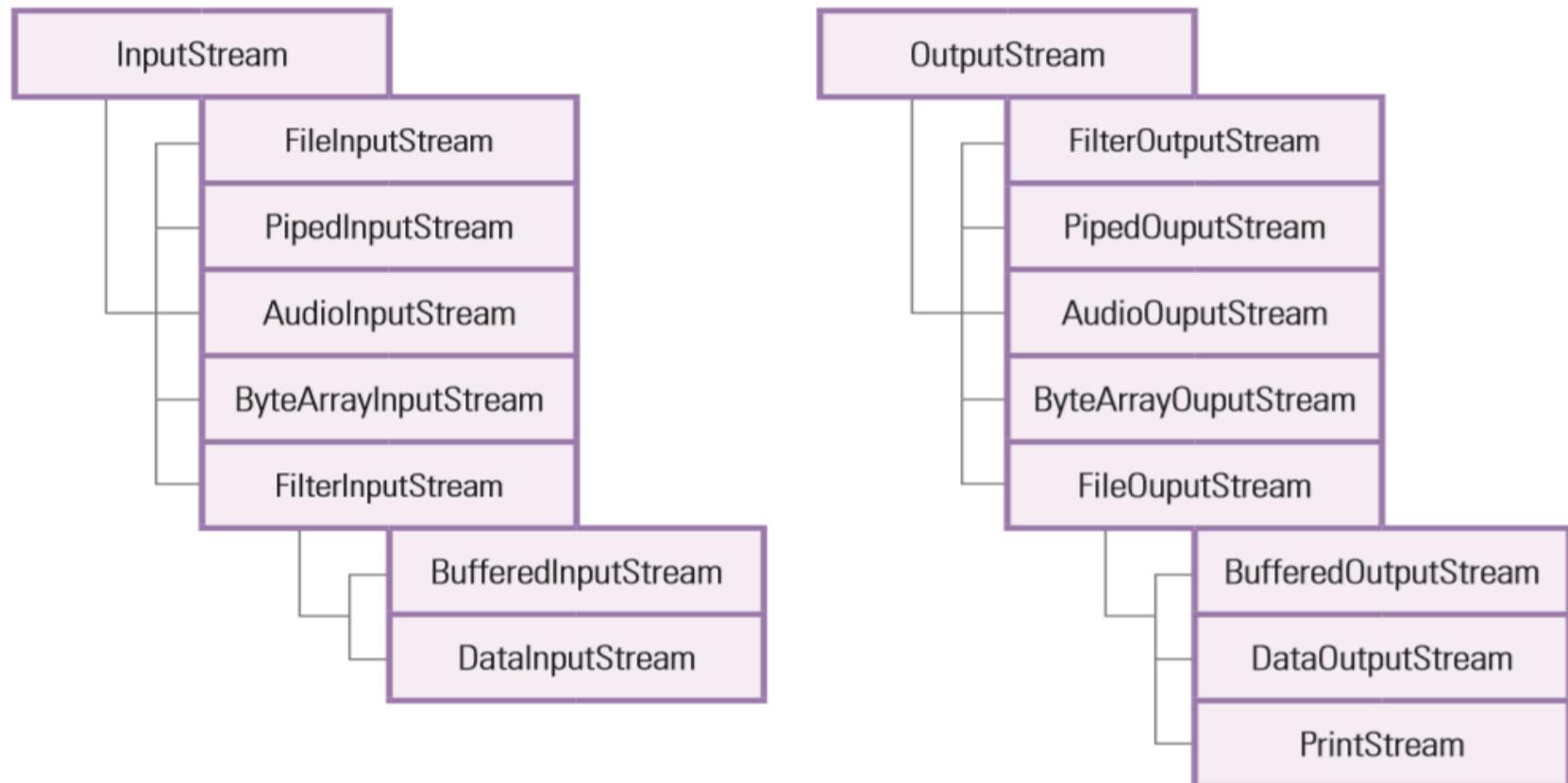
보조 스트림은 실제 입력 스트림을 도와 성능 향상

입력 보조 스트림	출력 보조 스트림	사용
FilterInputStream	FilterOutputStream	필터를 이용한 입출력
BufferedInputStream	BufferedOutputStream	버퍼를 통해 입출력
DataInputStream	DataOutputStream	기본형 단위로 데이터 처리
없음	PrintStream	print, printf, println

```
FileInputStream fis = null;  
fis = new FileInputStream("파일 위치");  
BufferedInputStream bis = new BufferedInputStream(fis);
```

Byte 기반 스트림(Stream)

모든 보조스트림은 InputStream과 OutputStream의 자손으로
입출력 방법이 같으므로 보조스트림의 참조변수로 입출력



문자 기반 스트림(Stream)

16비트의 문자나 문자열을 읽고 쓰는 스트림

바이트 기반스트림으로는 문자 입출력 처리 불편

Reader	Writer
int read()	void write(int c)
int read(char[] cbuf)	void write(char[] cbuf)
abstract int read(char[] cbuf, int off, int len)	abstract void write(char[] cbuf, int off, int len) void write(String str) void write(String str,int off, int len)

입력 스트림	출력스트림	대상
FileInputStream	FileWriter	파일
PipedInputStream	PipedWriter	메모리
CharArrayReader	CharArrayWriter	프로세스

Byte 기반 스트림(Stream)

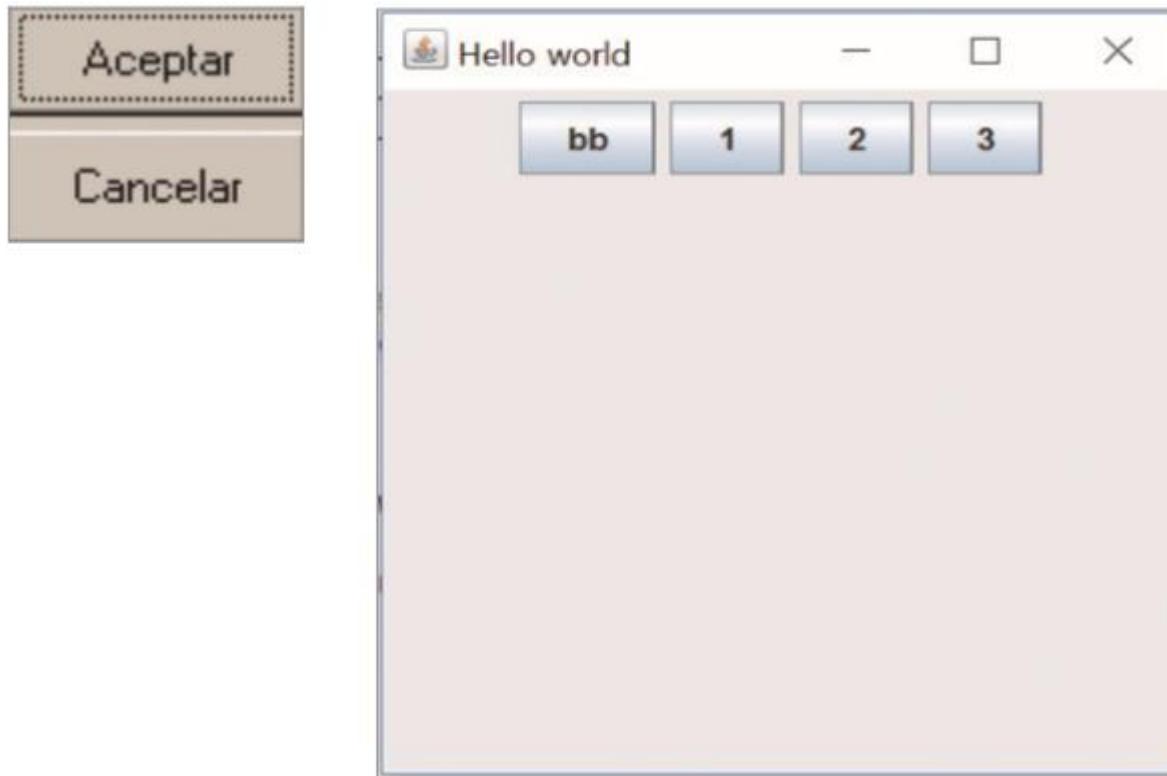
보조 스트림은 실제 입력 스트림을 도와 성능 향상

입력 보조 스트림	출력 보조 스트림	사용
FilterReader	FilterWriter	필터를 이용한 문자 입출력
BufferedReader	BufferedWriter	버퍼를 통해 문자 입출력

```
FileReader fr = null;  
fr = new FileReader("파일 위치");  
BufferedReader br = new BufferedReader(fr);
```

Java GUI 프로그래밍

사용자가 이해하기 쉬운 모양으로 정보를 제공하는 요소들을 프로그래밍하는 것



AWT/Swing

AWT

- 자바가 처음 나왔을 때부터 함께 배포된 GUI 라이브러리
- Frame, Panel, Button, Label 등
- 프로그램이 구동 되고 있는 OS가 제공하는 자원 이용하여 컴포넌트를 형성하므로 OS마다 다른 모양이 그려짐

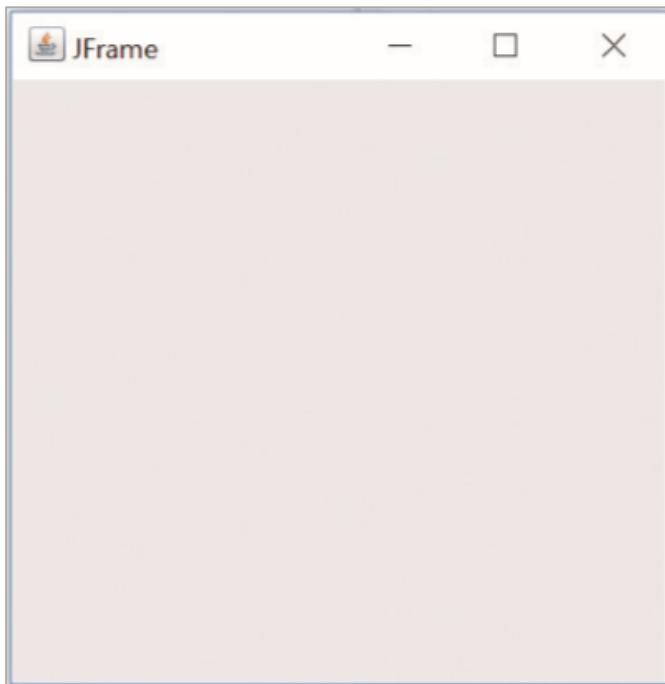
Swing

- AWT를 대체하기 위한 GUI 객체
- JFrame, JWindow, JPanel, JButton, 등 J가 붙은 특징
- OS에 관계 없이 동일한 모양, AWT다는 느낌

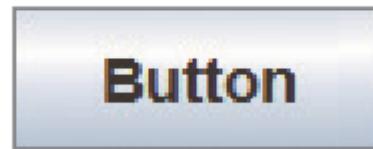
Componet

Java에서 GUI를 구성하는 일련의 블록 요소
GUI 객체로서 속성, 메서드, 이벤트를 가지며 사용자와
상호작용하는 기본 요소

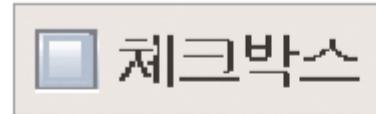
① JFrame



② JButton



③ JCheckBox



④ JSlider



⑤ JTextField



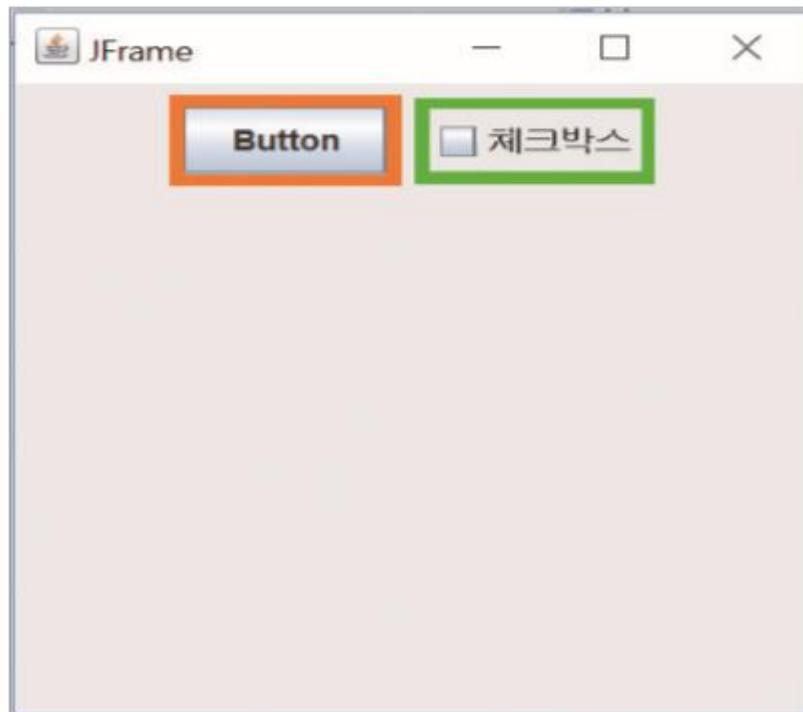
Container

다른 Component를 포함할 수 있는 Component

모든 Component는 컨테이너에 포함되어야만 화면에 출력 가능

그 중 Frame은 다른 컨테이너에 포함될 수 없는 최상위

컨테이너이며 Panel이나 Pane 같은 일반 컨테이너



Swing GUI Programming

프레임을 만드는 방법은 두 가지가 있습니다.

- ① 직접 JFrame f = new JFrame() 와 같이 JFrame 객체를 생성하는 방법
- ② JFrame을 상속받아 새로운 프레임 클래스를 작성하는 방법

setLocation(x ,y)	프레임의 위치를 지정합니다.
setSize(width ,height)	프레임의 크기를 지정합니다.
setIconImage(IconImage)	프레임의 타이틀 바에 보여질 아이콘을 설정합니다.
setTitle()	타이틀바의 제목을 설정합니다.
setVisible(boolean)	화면에 표시여부를 설정합니다.

Swing GUI Programming

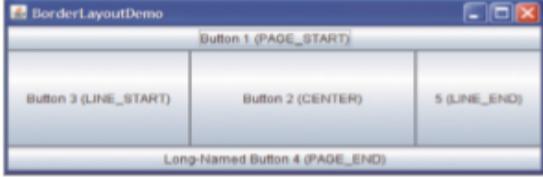
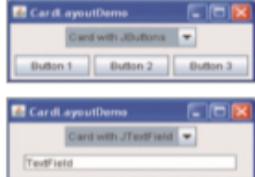
- ① 필요한 요소의 객체를 생성합니다.

```
JButton button = new JButton("Button");
JCheckBox box = new JCheckBox("체크박스");
JSlider slide = new JSlider();
 JTextField tf = new JTextField("Text입력",20);
```

- ② 이 요소들을 하나씩 add 메서드를 통해 추가합니다.

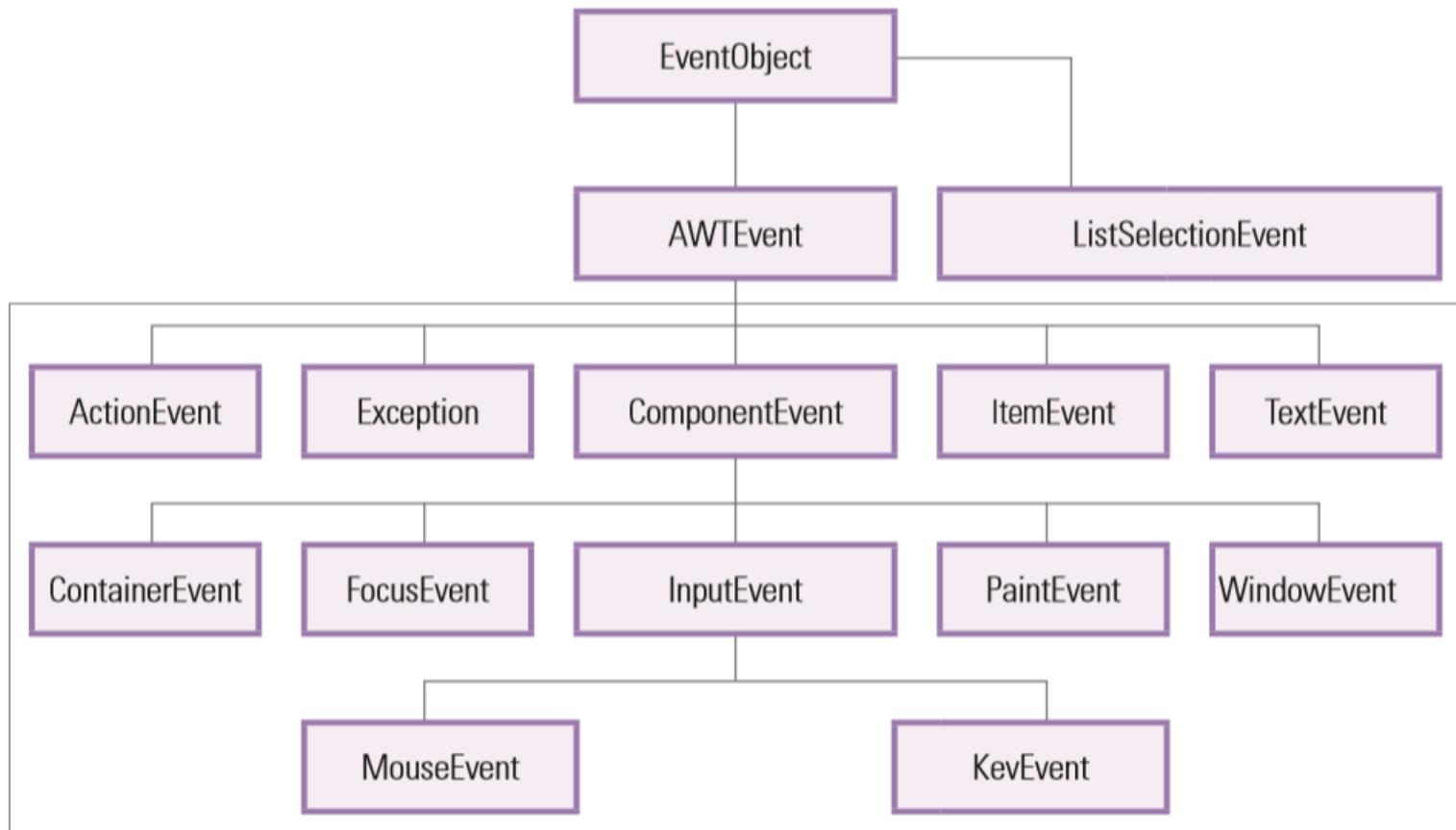
```
this.add(button);
this.add(box);
this.add(slide);
this.add(tf);
```

Swing GUI(배치관리자)

배치관리자	특징	예시
FlowLayout	<p>컨테이너에 컴포넌트가 들어오면 순서대로 왼쪽에서 오른쪽으로 배치한다.</p> <p>Panel, Applet은 이 배치관리자를 기본값으로 갖는다.</p>	
BoardLayout	<p>컨테이너의 공간을 동서남북중앙으로 나누고 지정한 영역에 컴포넌트를 배치한다.</p> <p>Window, Frame, Dialogue는 이 배치관리자를 기본값으로 갖는다.</p>	
GridLayout	<p>컨테이너의 공간을 동일 크기의 2차원 그리드로 나누고 들어오는 순서대로 왼쪽에서 오른쪽으로 배치한다.</p>	
CardLayout	<p>컨테이너의 공간에 카드를 쌓아놓은 듯이 컴포넌트를 쪼개어 배치한다.</p>	

Swing GUI(이벤트 객체)

이벤트가 발생할 때 그 이벤트에 대한 정보를 가진 객체



Swing GUI(이벤트 리스너)

발생한 이벤트 객체에 의해서 호출되어 해당 이벤트를 처리하는 객체
사용을 위해서는 인터페이스 내부의 추상메서드 구현

이벤트 종류	리스너 인터페이스	추상 메서드	발생 상황
Action	ActionListener	void actionPerformed(ActionEvent)	Action이 발생할 때
Key	KeyListener	void keyPressed(KeyEvent)	키가 눌려질 때
		void keyReleased(KeyEvent)	눌러진 키가 떼어질 때
		void keyTyped(KeyEvent)	유니코드 키가 입력될 때
Mouse	MouseListener	void MousePressed(MouseEvent)	마우스버튼이 눌릴 때
		void MouseReleased(MouseEvent)	눌린 버튼이 떼어질 때
		voidMouseClicked(MouseEvent)	클릭될 때
		void MouseEntered(MouseEvent)	커서가 컴포넌트 위에 올라올 때
		void MouseExited(MouseEvent)	커서가 컴포넌트를 벗어날 때

Swing GUI(이벤트 리스너)

이벤트 리스너 클래스를 선언할 때 `ActionListener` 인터페이스를 구현하였고 내부의 추상메서드 `actionPerformed(ActionEvent e)`를 작성하였습니다.

만약 이벤트가 발생하면 `ActionEvent e`를 통해 발생한 이벤트 객체를 받아오고

```
JButton button = (JButton) e.getSource();
```

위에서 `getSource` 메서드를 통해 `ActionEvent`의 정보를 읽어와서 저장합니다.

```
button.setText("버튼을 클릭하셨습니다.");
```

그리고 `setText()` 메서드를 통해서 해당 버튼의 글자를 바꾸는 동작을 하게됩니다.

작성한 이벤트 리스너를 사용하기 위해서는

```
button.addActionListener(new Listener1());
```

컴포넌트가 선언된 클래스 내부에서 컴포넌트에 리스너를 붙여주면 됩니다.

Swing GUI(이벤트 리스너)

이벤트 종류	리스너 인터페이스	추상 메서드	발생 상황
Mouse	MouseMotionListener	void mouseDragged(MouseEvent e)	마우스가 드래그 될 때
		void mouseMoved(MouseEvent e)	마우스가 움직일 때

메서드	메서드의 활용
int getX()	현재 마우스 포인터의 x좌표
int getY()	현재 마우스 포인터의 y좌표
short getButton()	현재 클릭한 버튼(왼쪽, 오른쪽)
int getClickCount()	마우스를 클릭한 횟수

Swing GUI(이벤트 리스너)

- 생성자

- `JFrame()` : 타이틀이 없는 프레임을 생성
- `JFrame(String title)` : 타이틀이 있는 프레임 생성

- 구성

- **JRootPane** : 실직적인 윈도우 기능을 수행하는 경량의 컨테이너
 - glassPane과 layerPane으로 구성
 - layerPanem은 JMenuBar와 contenPane을 포함
- **layerPane** : 루트 페인에 대해 레이어를 할 수 있도록 여러 층의 패널을 포함 할 수 있는 패널로 위 부분은 menuBar와 아래 부분은 contentPane으로 구성
- **glassPane** : 기본적으로 숨겨진 상태로 되어 있으며 다른 패널 위에 존재하는 패널
 - 마우스 이벤트를 처리하기 위해 가장 먼저 루트 페인에 추가된다.
- **contentPane** : 일반적인 컴포넌트들을 가질 수 있는 패널
 - 프레임 객체의 `getContentPane()`메소드를 이용해서 얻을 수 있다.
- **menuBar** : 윈도우의 메뉴를 제공하는 역할로 생략이 가능한 선택항목
 - `JMenu`, `JmenuItem` 등을 이용해서 메뉴를 구성하여 `setJMenuBar()`메소드를 이용해서 등록 할 수 있다.

Swing GUI

- 메소드

MenuBar getMenuBar()	프레임의 메뉴바를 리턴
int getState()	프레임의 상태를 리턴
String getTitle()	타이틀 바의 문자열을 리턴
void remove(MenuComponent m)	프레임에서 지정한 메뉴를 제거
void setResizable(Boolean resizable)	프레임의 크기를 사용자가 변할 수 있게 지정
void setSize(int width, int height)	사이즈를 'width' 및 'height'로 변경
pack()	컨테이너의 크기를 구성 요소들의 크기로 설정
void setVisible(boolean b)	'b'에 의해 컴퍼넌트 표시 or 비표시 설정
void setLocation(int x, int y)	이 컴퍼넌트를 새로운 위치로 이동
setBounds(int x, int y, int width, int height)	위치, 너비, 높이 지정
getContentPane()	프레임의 contentPane 오브젝트를 돌려줍니다.
setIconImage(Image image)	프레임의 최소화된 아이콘에 표시되는 이미지를 설정
add(컴포넌트)	컴포넌트를 부착

Swing GUI (Component)

- JPanel

- 컴포넌트들을 그룹 별로 묶어서 처리할 때 사용하는 컨테이너
- 일반적으로 Frame에 컴포넌트들을 직접 붙이지 않고 Panel 이용
- 생성자

JPanel()	디폴트의 레이아웃 사용해 새로운 Panel 생성
JPanel(LayoutManager layout)	지정된 레이아웃 매니저를 이용하여 새로운 Panel 생성
JPanel(boolean isDoubleBuffered)	지정된 버퍼를 이용해 새로운 Panel 생성

- JScrollPane

- 스크롤을 이용해서 컴포넌트들을 보여주는 컴포넌트
- 스크롤을 이용해서 보여주는 화면을 상하좌우로 이동하여 포함된 컴포넌트의 원래크기 유지
- JList, JTextArea, JTextPane 등이용

Swing GUI (Component)

- 원도우를 구성하는 작은 부품
- 컨테이너의 **add**를 통해 부착
- **Border** : 경계선
- **AbstractBorder** : 최상위 추상 클래스
- **Border** 객체를 생성한 후 컴포넌트의 **setBorder()**를 이용해서 설정
- **setToolTipText(일반 텍스트 또는 html코드)**를 이용해서 툴 팁 작성 가능

Swing GUI (Component)

- **JLabel**

- 컴포넌트에 텍스트와 이미지를 모두 넣을 수 있습니다.
- getText와 setText(String str)을 이용해서 텍스트 설정

- **AbstractButton**

- JButton, JToggleButton, JRadioButton 등의 모든 버튼의 종류를 추상화한 클래스
- 버튼에 텍스트와 이미지를 모두 넣을 수 있고 다양한 샘플에서 보여줄 버튼의 이미지를 각각 지정하여 Roll-Over 버튼을 만들 수도 있습니다.
- JButton(Icon icon), JButton(String text, Icon icon)

- **JToggleButton**

- 버튼의 상태를 기본과 선택된 상태 두 가지를 가지는 버튼으로 선택된 상태를 계속 유지
- 기본상태와 선택된 상태를 구분하기 위해 서로다른 아이콘 지정 사용 가능

- **JRadioButton**

- AWT의 Checkbox 클래스를 이용한 라디오 형 체크박스와 유사한 형태의 컴포넌트
- 여러 항목 중에서 하나의 항목만 선택할 수 있도록 만든 컴포넌트

Swing GUI (Component)

- JTextField

- 생성자
 - JTextField()
 - JTextField(int cols) : 지정된 열의 수를 갖는 텍스트 필드 생성
 - JTextField(String text) : 지정된 텍스트로 초기화 한 텍스트 필드
 - JTextField(String text int cols)
- 메소드
 - int getCarePosition() : 텍스트 내에서 현재 마우스 포인터의 위치를 리턴
 - String getText() : 텍스트 내의 표시되는 텍스트를 리턴
 - void select(int selection, int selectionEnd) : 시작위치부터 마지막 위치까지 선택
 - void selectAll() : 텍스트 컴퍼넌트 내의 모든 텍스트를 선택
 - void setBackground(Color c) : 백그라운드 칼라를 설정
 - void setCaretPosition(int pos) : caret의 위치를 설정
 - void setText(String t) : 표시되는 텍스트를 지정된 텍스트로 설정

Swing GUI (Component)

- **JTextArea**

- 여러 줄을 입력할 수 있는 컴포넌트로 자체적으로 스크롤이 처리되지 않으므로 JScrollPane에 포함시켜 스크롤 지원합니다.
- 생성자
 - JTextArea()
 - JTextArea(int rows, int cols)
 - JTextArea(String text)
 - JTextArea(String text, int rows, int cols)
- 줄 바꿈 자동으로 해주지 않음 (옵션 사용)
 - setWrapStyleWord()

Swing GUI (Component)

- JComboBox

<생성자>

JComboBox(ComboBoxModel m)	지정한 콤보박스 모델을 사용하는 콤보박스 객체 생성
JComboBox(Object[] items)	지정한 배열의 자료를 보여주는 콤보박스 객체 생성
JComboBox(Vector items)	지정한 벡터의 자료를 보여주는 콤보박스 객체 생성

<메소드>

void addActionListener(Action a)	콤보박스에서 아이템을 선택했을 때 발생하는 이벤트를 받기위해 지정된 리스너 추가
void addItemListener(ItemListener a)	콤보박스에서 아이템의 선택이 바꿨을 때 발생하는 이벤트를 받기 위해 지정된 리스터 추가
void showPopup()	팝업 윈도우를 보이게 한다.
Object getItemAt(int index)	지정한 인덱스의 아이템을 얻어온다.
void addItem(Object o)	지정한 아이템을 목록에 추가한다.
void insertItemAt(Object o, int index)	지정한 아이템을 지정한 인덱스 위치에 추가
void removeAllItems()	모든 아이템을 제거한다.
void removeItem(Object o)	지정된 아이템을 목록에서 제거한다.
void removeItemAt(int index)	지정된 인덱스의 아이템을 목록에서 제거한다.
void setEditable(Boolean flag)	콤보박스의 편집 가능 상태를 지정한다.

Swing GUI (Component)

- **JList**

<생성자>

JList(ListModel m)	지정한 리스트 모델을 사용하는 리스트 객체 생성
JList(Object[] listData)	지정한 배열의 자료를 보여주는 리스트 객체 생성
JList(Vector listData)	지정한 벡터의 자료를 보여주는 리스트 객체 생성

<메소드>

void addListSelectionListener (ListSelectionListener listener)	리스트에서 선택하는 항목이 바뀌었을 때 발생하는 이벤트를 받기 리스너 추가
void ensureIndexisVisible(int index)	지정한 인덱스의 아이템이 보여지도록 리스트 스크롤 시킨다.
void setSelectionBackground (Color selectionBackground)	선택된 아이템의 배경색을 지정한다.
void setSelectionForeground (Color selectionForeground)	선택된 아이템의 전경색을 지정한다.

Swing GUI (Component)

void setSelectionModel (ListSelectionModel selectionModel)	지정된 ListenerModel로 지정한다.
boolean isSelectionEmpty()	선택된 아이템이 있는지 검사하여 있으면true, 없으면false
int getSelectedIndex()	선택된 첫 번째 아이템의 인덱스를 얻어온다.
int[] getSelectedIndices()	선택된 모든 아이템의 인덱스를 배열로 얻어온다.
Object getSelectedValue()	선택된 아이템을 얻어온다.
Object[] getSelectedValues()	선택된 모든 아이템을 배열로 얻어온다.
void setSelectedIndex(int index)	지정된 인덱스의 아이템을 선택된 상태로 만든다.
void setSelectedValue(Object o, boolean shouldScroll)	지정된 객체를 가진 아이템을 선택된 상태로 만들고 shouldScroll이 true면 그 아이템이 보이도록 스크롤 한다.
void setModel(ListModel m)	지정된 모델로 리스트의 모델을 지정한다.
void setListData(Object[] listData)	지정된 배열 자료를 사용하는 모델을 만들고 지정한다.
void setListData(Vector listData)	지정된 백터 자료를 사용하는 모델을 만들고 지정한다.

Swing GUI (Component)

- BoxLayout

- Swing에 추가된 Layout
- Box : 컴포넌트들을 왼쪽에서 오른쪽으로 수평배치되거나 위에서 아래로 수직배치 해 주는 객체
- 생성
 - Box.createVerticalBox() : 수평으로 컴포넌트 추가
 - Box.createHorizontalBox() : 수직으로 컴포넌트 추가
 - Box.createVerticalStrut(int width)
 - Box.createHorizontalStrut(int width)

Swing GUI (Component)

- **JTable**

- 데이터를 행과 열로 구성되어 있는 테이블 형식으로 보여주는 컴포넌트
- JTable 역시 Scrollable 인터페이스가 구현되었음
- JTable을 사용하기 위해서는 먼저 데이터를 저장할 모델을 만들고 View인 JTable에 연결

Swing GUI (Component)

<생성자>

JTable()	기본 자료 모델, 기본 컬럼 모델, 기본 선택 모델로 초기화된 테이블 객체 생성
JTable(int rows, int cols)	DefaultTableModel을 이용해 빈 셀을 행과 열의 수만큼 테이블 객체 생성
JTable(Object[] rowData, Object colsName)	2차원 배열에 값들을 보여주는 테이블 객체를 생성, 주어진 테이블 데이터와 컬럼의 이름을 가진다.
JTable(TableModel dm)	주어진 dm을 가지는 자료 모델, 기본 컬럼 모델, 기본 선택 모델로 초기화된 테이블 객체를 생성
JTable(TableModel dm, TableColumnModel cm)	주어진 dm을 가지는 자료 모델, cm을 가지는 컬럼 모델, 기본 선택 모델로 초기화된 테이블 객체를 생성
JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm)	주어진 dm을 가지는 자료모델, cm 을 가지는 컬럼 모델, sm을 가지는 선택 모델로 초기화된 테이블 객체 생성
JTable(Vector rowData, Vector colsName)	주어진 벡터의 값들을 보여주는 테이블 객체 생성, 주어진 테이블 자료와 컬럼 이름을 가짐

Swing GUI (Component)

<메소드>

void setModel(TableModel dataModel)	주어진 자료 모델을 테이블 자료 모델로 지정
void setColumnModel (TableColumnModel columnModel)	주어진 컬럼 모델을 테이블의 컬럼 모델로 지정
setCellEditor(TableCellEditor anEditor)	주어진 셀 자료를 테이블의 셀 에디터로 지정
void setValueAt(Object aValue, int row, int cols)	테이블 모델의 셀에 주어진 행과 열의 위치에 자료를 설정한다.
Object getValueAt(int row, int cols)	주어진 행과 열 위치에 있는 셀값을 얻어옴
void setAutoResizeMode(int mode)	컬럼의 자동 크기 조절 모드를 설정
void setIntercellSpacing (Dimension intercellSpacing)	셀 사이의 간격을 설정한다.
void setSelectionBackground (Color selectionBackground)	선택된 셀의 배경색을 설정한다.
void setSelectionForeground (Color selectionForeground)	선택된 셀의 전경색을 설정한다.
void setShowHorizontalLines (boolean showHorizontalLines)	셀 사이의 수평 구분선을 그릴지 설정

Swing GUI (Component)

void selectAll()	전체 테이블의 셀, 행과 열을 선택
void setCellselectionEnabled (boolean cellSelectionEnabled)	행과 열을 동시에 선택할 수 있도록 하는 선택 모드를 사용할지를 설정한다.
void setColumnSelectionAllowed (boolean columnSelectionAllowed)	테이블의 열을 동시에 선택할 수 있도록 하는 선택모드를 사용할지 설정
void setRowSelectionAllowed (boolean rowSelectionAllowed)	테이블의 행을 동시에 선택할 수 있도록 하는선택모드를 사용할지 설정
int getSelectedColumn()	선택된 열의 첫번째 위치를 얻어온다. 선택된 열이 없으면 -1을 얻음
int[] getSelectedColumns()	선택된 열들을 배열 형태로 얻음
int getSelectedRow()	선택된행의 첫번째 위치를 얻어온다. 선택된 행이 없으면 -1
int[] getSelectedRows()	선택된 행들을 배열 형태로 얻어온다.
boolean isCellSelected(int row, int column)	주어진 위치의 셀이 선택 상태인지를 얻음
int getColumnCount()	열 수를 얻음
int getRowCount()	행 수를 얻음
Object getValueAt(int rowIndex, int colsIndex)	주어진 위치의 셀값을 얻음
void setValueAt(Object aValue, int rowIndex, int colsIndex)	주어진 위치의 셀값을 설정

Swing GUI(Event)

- GUI 컴포넌트에서 발생되는 모든 행위
- 예를 들어 버튼을 클릭하거나 윈도우 종료단추를 클릭하는 행동
- **ActionEvent(JButton, JTextField, JCheckBox, JFileChooser, JMenuItem, JRadioButton..)**
 - 버튼이 클릭되거나 리스트, 메뉴 등이 선택되었을 때 발생하는 이벤트
 - ActionListener 인터페이스의 actionPerformed(ActionEvent) 메서드를 이용해서 처리

필드명	해당 키
ALT_MASK	ALT 키
CTRL_MASK	Ctrl 키
SHIFT_MASK	Shift 키

메소드	해당 키
getActionCommand()	이벤트를 발생시킨 객체의 문자열을 가져온다
getSource()	이벤트를 발생시킨 객체의 위치값을 가져온다.
getModifiers()	이벤트가 발생되었을 때 같이 사용된 modifier키들을 가져온다.

Swing GUI (Event)

- KeyEvent(JFrame, JButton, JTextField, JTextArea..)

- 키보드를 통해서 키 입력이 오면 발생
- KeyListener 인터페이스나 KetAdapter 추상클래스를 이용
- 이벤트 처리 메소드
 - keyPressed(KeyEvent e) : 컴포넌트에서 키가 눌러지면 호출
 - keyReleased(KetEvent e) : 키에서 뗄 때 호출
 - ketTyped(KeyEvent e) : 키보드를 통해 문자가 입력되었을 때 호출

메소드	내용
getKeyChar()	이벤트에 의해 입력된 문자값을 가져온다.
getKeyCode()	이벤트에 의해 입력된 문자에 해당하는 코드값을 가져온다.

필드명	해당 키
VK_0 ~ VK_9	숫자 0~9
VK_A ~ VK_Z	영문자 A~Z
VK_F1 ~ VK_F24	기능키 F1 ~ F24
VK_ENTER	엔터 키
VK_SPACE	스페이스바 키
VK_BACKSPACE	BackSpace 키
KEY_PRESSED	키가 눌려진 이벤트 의미
KEY_RELEASED	키가 눌렸다 놓아진 이벤트 의미
KEY_TYPED	키 입력 이벤트 의미

Swing GUI(Event)

- **MouseEvent(JList를 제외한 거의 모든 컴포넌트)**

- 마우스 버튼을 누르거나 특정 컴포넌트 안에 진입하거나 벗어날 때 호출되는 이벤트
- MouseListener 인터페이스나 MouseAdapter 추상 클래스를 이용해서 처리
- 메소드
 - **mouseClicked(MouseEvent e)** : 마우스를 클릭했을 때 호출
 - **mouseEntered(MouseEvent e)** : 마우스 커서가 컴포넌트 영역에 들어오면 호출
 - **mouseExited(MouseEvent e)** : 마우스 커서가 컴포넌트 영역에서 벗어나면 호출
 - **mousePressed(MouseEvent e)** : 마우스 버튼이 눌러지면 호출
 - **mouseReleased(MouseEvent e)** : 마우스 버튼이 눌러졌다 띄어지면 호출

Swing GUI (Event)

필드명	설명
MOUSE_CLICKED	마우스 버튼이 클릭된 경우 발생되는 이벤트
MOUSE_ENTERED	마우스 커서가 컴포넌트 영역으로 들어왔을 때 발생되는 이벤트
MOUSE_EXITED	마우스 커서가 컴포넌트 영역 밖으로 나가면 발생되는 이벤트
MOUSE_PRESSED	마우스 버튼이 눌러졌을 때 발생하는 이벤트
MOUSE_RELEASED	마우스 버튼이 눌렀다 띄었을 때 발생하는 이벤트
MOUSE_DRAGGED	마우스 버튼이 클릭된 상태에서 움직일 때 발생되는 이벤트
MOUSE_MOVED	마우스 커서가 움직일 때 발생되는 이벤트

메소드	내용
getClickCount()	마우스 눌려진 횟수를 얻어온다.
getPoint()	마우스 이벤트가 발생한 좌표를 얻어온다.
getX()	마우스 이벤트가 발생한 X좌표를 얻어온다.
getY()	마우스 이벤트가 발생한 Y좌표를 얻어온다.

Swing GUI(Event)

- **MouseEvent(JList를 제외한 거의 모든 컴포넌트)**

- 마우스를 움직이면 호출되는 이벤트
- MouseMotionListener 인터페이스 or MouseMotionAdapter 추상 클래스 이용
- 메소드
 - **mouseDragged(MouseEvent e)** : 마우스 버튼이 눌러진 상태로 이동하면
 - **mouseMoved(MouseEvent e)** : 마우스를 이동하면 호출

- **WindowEvent(JFrame)**

- 윈도우를 활성화, 아이콘화, 비활성화 작업 시 발생하는 이벤트
- WindowListener 인터페이스 or WindowAdapter 추상 클래스 이용
- 메소드
 - **windowOpened(WindowEvent e)** : 윈도우가 열릴 때
 - **windowClosing(WindowEvent e)** : 윈도우가 닫힐 때
 - **windowActivated(WindowEvent e)** : 윈도우가 활성화 되었을 때
 - **windowDeactivated(WindowEvent e)** : 윈도우가 활성화 되었을 때
 - **windowIconified(WindowEvent e)** : 윈도우가 아이콘화 되었을 때
 - **windowDeiconified(WindowEvent e)** : 윈도우가 최소화 상태에서 원래대로 되돌아올 때

Swing GUI(Event)

필드명	설명
WINDOW_ACTIVATED	윈도우가 활성화될 때 발생
WINDOW_DEACTIVATED	윈도우가 비활성화 될 때 발생
WINDOW_CLOSED	윈도우가 닫힐 때 발생
WINDOW_CLOSING	윈도우가 사용자의 요청으로 닫힐 때 발생
WINDOW_ICONIFIED	윈도우가 아이콘화 될 때 발생
WINDOW_OPENED	윈도우가 생성될 때 발생

메소드	내용
getWindow()	이벤트가 발생된 윈도우를 가져온다.

Swing GUI (Event)

- 이벤트 관련 객체

- Listener 인터페이스 - 하나 또는 2개 이상의 이벤트 처리를 위한 메소드를 소유한 인터페이스
- Adapter 클래스 - 2개 이상의 메소드를 가진 Listener를 변환한 추상 클래스

- 이벤트 처리

- 원하는 Listener를 클래스에 implements하거나 Adapter 클래스를 상속
 - 이벤트에 따라 호출되는 메소드 재정의
 - 넘어오는 매개변수의 getSource()가 이벤트를 발생시킨 컴포넌트를 리턴
 - 컴포넌트에 이벤트 리스너를 추가
-
- 이벤트 처리는 **Anonymous Class**를 이용해서 한번에 처리 가능

Swing GUI (Pane)

- 특정 목적에 맞도록 작성된 컴포넌트

- **JEditorPane**

- 여러 가지 형태의 문서를 보여줄 수 있는 컴포넌트
- 일반 텍스트나 html, rft 문서를 보여줄 수 있습니다.

- **JTabbedPane**

- 여러 패널을 담을 때 사용하는 컴포넌트
- 기능별로 분류된 옵션들을 동시에 보여 줄 필요가 없고, 필요 시 하나의 패널만 보여주기 위해서 사용하는 컴포넌트
- 사용 방법은 타이틀이나 아이콘을 가지는 탭을 클립함으로써 여러개의 패널 중에 선택된 탭을 보여줌
- 탭의 위치는 기본적으로 왼쪽 위에 위치

Swing GUI (Pane)

- JOptionPane

- 실행하는 도중에 사용자로부터 데이터를 입력 받거나 특정한 메시지를 출력시켜 확인시키는 작업
- 자체적으로 실행시키는게 아니므로 showDialog()메소드를 통해 확인해야한다.

종류	기능	호출 함수
MessageDialog	사용자에게 메시지를 보여준다.	showMessageDialog()
ConfirmDialog	Yes, No, Cancel과 같은 버튼으로 확인	showConfirmDialog()
InputDialog	사용자로부터 자료를 입력받기 위한 다이어로그	showInputDialog()
OptionDialog	위 세 가지를 포함한 다이어로그	showOptionDialog()

Swing GUI (JFileChooser)

- 프로그램을 실행하는 도중에 데이터를 파일로부터 불러오거나 파일에 저장할 수 있도록 선택 다이어로그 생성
- 파일 선택창은 **FileSystemView**, **FileView**, **FileFilter** 등과 같은 컨트롤러와 함께
- **FileSystemView**는 파일 시스템과 디렉토리 정보를 제공
- **FileView**는 디렉토리 내부에 있는 파일들에 대한 정보를 제공
- **FileFilter**는 파일을 원하는 종류만 보여줄 수 있도록 걸러주는 역할

Swing GUI (JFileChooser)

- 멤버필드

- static int CANCEL_OPTION : 취소를 선택했을 때 리턴되는 값
- static int APPROVE_OPTION : 예나 확인을 선택했을 때 리턴되는 값

- 생성자

- JFileChooser() : 디폴트 경로
- JFileChooser(File currentDirectory) : 지정한 경로를 보여줌
- JFileChooser(String currentDirectoryPath) : 지정한 경로를 보여줌

Swing GUI (JMenu)

- **JMenuBar, JMenu, JMenuItem**으로 구성
- 사용 방법
 - 1단계: JMenuBar 객체를 생성
 - 2단계: setJMenuBar(JMenuBar객체)를 이용해서 컨테이너에 메뉴바 부착
 - 3단계: JMenu객체를 생성해서 JMenuBar 객체의 add(JMenu객체)로 부착
 - 4단계: JMenuItem 객체를 생성해서 JMenu객체의 add (JMenuItem 객체)로 부착
- 스윙에서는 **JMenuItem, JCheckBoxMenuItem, JRadioButtonMenuItem**의 메뉴 아이템이 제공
- 메뉴 객체에서 **setMnemonic(int mnemonic)**을 이용해서 단축키도 지정이 가능
- ALT와 조합되서 단축키가 됩니다.

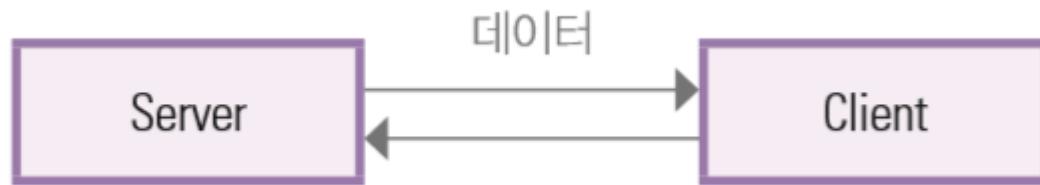
Network Programming

네트워크(Network)는 분산되어 있는 두 개 이상의 컴퓨터를 통신망으로 연결한 것을 말합니다. 즉 네트워크 프로그래밍이란 다른 컴퓨터와 송수신할 수 있는 프로그램을 만드는 것을 말합니다. 자바는 네트워크 프로그래밍에 필요한 대부분을 객체화하였기 때문에 사용자는 네트워크 어플리케이션의 통신 부분을 쉽게 프로그래밍할 수 있으며 고수준의 통신 프로그램을 만들 수 있습니다.



Client/Server

클라이언트(Client)란 서비스를 사용하는 컴퓨터를 말합니다. 반대로 서버(Server)는 서비스를 제공하는 컴퓨터입니다. 즉 일반적으로 우리들이 접속하는 사이트를 웹 서버라고 하고 우리들이 사이트에 접속하기 위해 사용하는 개인 컴퓨터를 클라이언트라고 합니다. 서버는 다수의 클라이언트를 위해 존재하기 때문에 일반적으로 매우 큰 용량과 성능을 가집니다.



IP주소

IP 주소는 컴퓨터와 네트워크에서 장치들이 서로를 인식하고 통신을 하기 위해 컴퓨터마다 부여되는 고유한 주소입니다.

무선 LAN 어댑터 Wi-Fi:

연결별 DNS 접미사.....:

링크-로컬 IPv6 주소.....: fe80::f435:805f:36d:4eec%2

IPv4 주소: 192.168.10.143

서브넷 마스크: 255.255.255.0

기본 게이트웨이: 192.169.10.1

IP 주소 = 네트워크주소 + 호스트 주소

IP주소

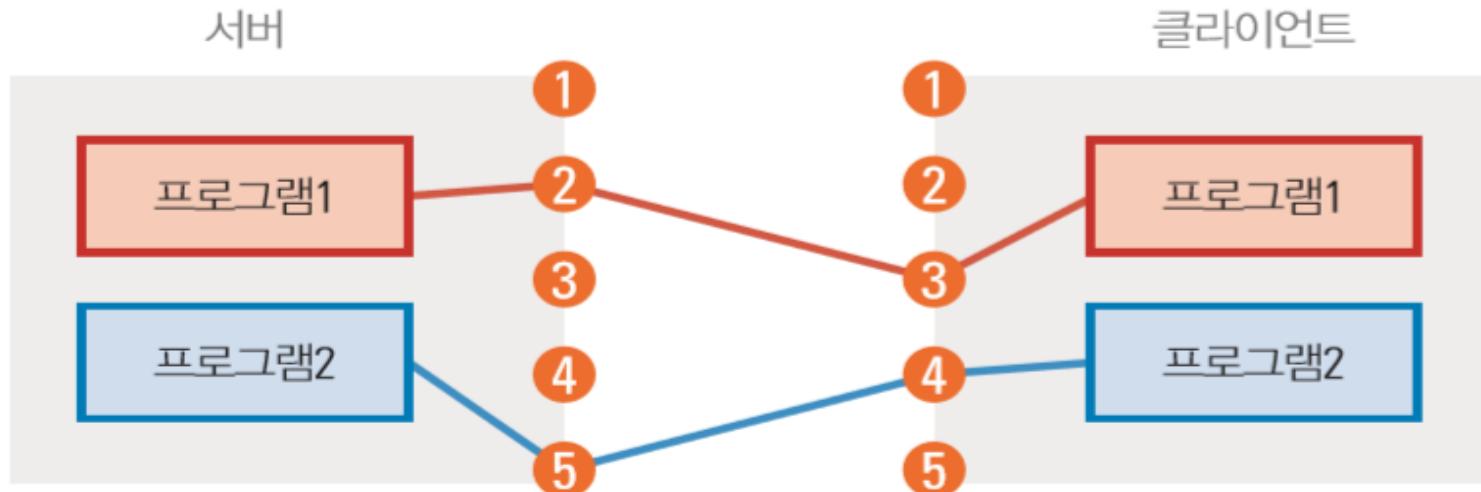
메서드	설명
byte[] getAddress()	IP 주소를 byte 배열로 반환합니다.
static InetAddress[] getAllByName(String host)	도메인명에 지정된 모든 호스트의 IP 주소를 배열에 담아 반환합니다.
static InetAddress getByName(String host)	도메인명을 통해 IP 주소를 얻습니다.
String getHostAddress()	호스트의 IP 주소를 반환합니다.
String getHostName()	호스트의 이름을 반환합니다.
boolean isMulticastAddress()	IP 주소가 멀티캐스트 주소인지 확인합니다.
boolean isLoopbackAddress()	IP 주소가 loopback 주소인지 알려줍니다.

소켓프로그래밍

소켓 프로그래밍이란 소켓을 이용한 통신을 하는 프로그램을 작성하는 것을 말합니다. 소켓(Socket)이란 프로세스간의 통신에 사용되는 양쪽 끝단(endpoint)을 의미하는데 이는 네트워크상에서 서버, 클라이언트 두 개의 프로그램이 특정 포트를 통해 양방향 통신이 가능하도록 만들어주는 장치입니다.

포트(port)는 가상의 통신 선로입니다. 하나의 컴퓨터 안에는 여러 개의 프로세스가 수행되고 있으므로 특정한 응용프로그램과 통신하기 위해서는 그 응용프로그램으로 통하는 선로가 필요한데 이를 포트라고 합니다. 포트 번호는 0부터 65535까지의 번호를 사용하지만 1023 아래의 포트는 기존 다른 통신 프로그램이 사용하는 경우가 많기 때문에 1023번 이상의 숫자에서 사용하는 것이 좋습니다.

소켓프로그래밍



항목	TCP	UDP
연결방식	<ul style="list-style-type: none">• 연결 후 통신• 1:1통신방식	<ul style="list-style-type: none">• 연결하지 않고 통신• 1:1부터 n:n 통신방식
특징	<ul style="list-style-type: none">• 데이터의 전송순서가 보장• 데이터의 수신여부 확인• UDP보다 느림	<ul style="list-style-type: none">• 데이터의 전송순서 보장되지 않음• 순서가 바뀔 수 있음• 데이터의 수신여부 확인안함• TCP보다 속도가 빠름

소켓프로그래밍(TCP)

- ① 서버 쪽에서는 서버소켓을 특정 포트와 결합하여 클라이언트의 연결요청을 기다립니다.
- ② 클라이언트 쪽에서는 소켓을 생성하여 서버의 IP 주소와 연결할 포트를 가지고 서버에 연결을 요청합니다.
- ③ 서버소켓은 클라이언트로부터 연결 요청을 받으면 서버 쪽에 새로운 소켓을 생성해서 클라이언트의 소켓과 연결합니다.
- ④ 클라이언트와 서버의 소켓이 연결되었으므로 서로간의 통신이 가능해 집니다.

- 소켓(Socket)

프로세스간의 통신을 하며 `InputStream`과 `OutputStream`을 가지고 있다. 이 스트림을 통해서 통신이 이루어진다.

- 서버소켓(ServerSocket)

포트와 연결(bind)되어 클라이언트의 연결을 기다리다가 연결이 되면 `Socket`을 생성해서 소켓끼리 연결되게 한다.

소켓프로그래밍(TCP)

- ① ServerSocket을 생성하고 8000번 포트에 연결(bind)합니다.

```
serverSocket = new ServerSocket(8000);
```

- ② 클라이언트로부터의 연결이 올 때까지 계속해서 이 상태로 기다립니다. 만약 이 상태에서 클라이언트로부터 연결이 확인(accept)되면 서버소켓은 다른 소켓을 생성해서 클라이언트와 연결합니다.

```
Socket socket = serverSocket.accept();
```

- ③ OutputStream을 통해서 “TEST 확인”이라는 메시지를 클라이언트에게 전송합니다.

```
OutputStream out = socket.getOutputStream();
DataOutputStream dos = new DataOutputStream(out);
dos.writeUTF("TEST 확인");
```

- ④ 사용이 완료된 소켓과 스트림은 종료시켜줍니다. 그리고 다시 서버소켓은 클라이언트의 연결을 기다립니다.

```
dos.close();
socket.close();
```

소켓프로그래밍(UDP)

UDP 소켓 프로그래밍은 DatagramSocket과 DatagramPacket을 사용합니다. TCP처럼 연결이 확인된 후에 데이터를 전송하는 것이 아니기 때문에 따로 서버소켓이 필요하지는 않으며 DatagramSocket을 통해 접속을 요청하면 데이터를 DatagramPacket에 담아서 전송합니다.

DatagramPacket은 헤더와 데이터로 구성되어 있고 헤더에는 수신할 호스트의 정보가 저장되어 있습니다.

소켓프로그래밍(URL)

URL 클래스

URL : 인터넷에서 접근 가능한 자원의 주소를 표현 형식

`http://blog.naver.com:80/javaking75/index.html?copen=1#9380390`

프로토콜 : 자원에 접근하기 위해 서버와 통신하는데 사용되는 통신규약

호스트명 : 자원을 제공하는 서버의 이름(`www.naver.com`)

포트번호 : 통신에 사용되는 서버의 포트번호(80)

경로명 : 접근하려는 자원이 저장된 서버상의 위치 (`/javaking75/`)

파일명 : 접근하려는 자원의 이름(`index.html`)

쿼리(query) : URL에서 '?' 이후의 부분 (`copen=1`)

참조(anchor) : URL에서 '#' 이후의 부분 (`#9380390`)