

Java Programming Language

Yu Hyun Seok
mechi00@hanmail.net

자바 프로그래밍의 기초

(The Java Programming Language Basics)

1. Java의 주요특징
2. 객체지향(Object-Oriented)
3. 기본적인 컴포넌트
(Identifiers, Keywords and Types)
4. 표현식과 흐름제어
(Expressions and Flow Control)
5. 배열(Array)

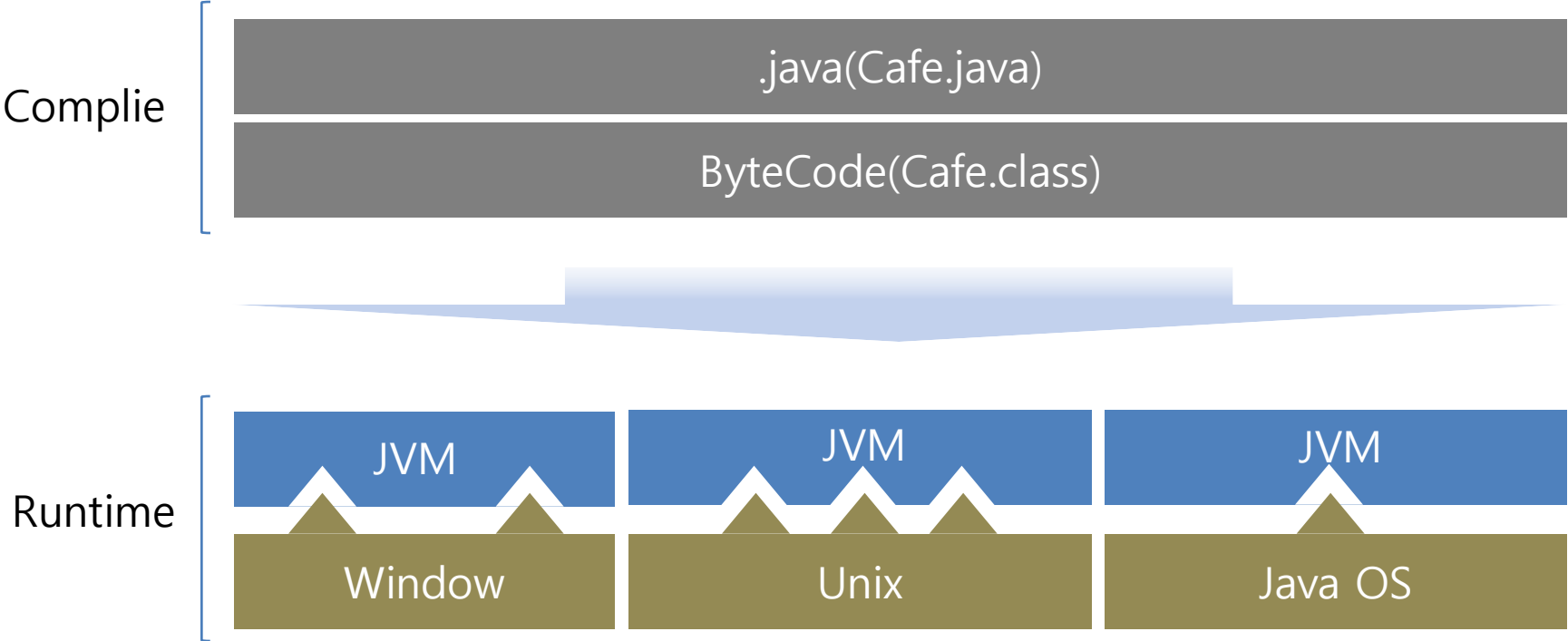
1-1) Java의 기본 목적

- OOP(Object-Oriented Programming)
객체 지향 프로그래밍 언어이다.
- Portable(platform independent)
휴대성, 이식성이 용이하며, platform에 독립적이다.
- Multi Threading
process를 수행시키기 위해서는 많은 자원과 시간이 필요
thread 기술로 문제점을 많이 최소화

1-2) 기본 목적을 위해 제공되는 3가지

- 자바 가상 머신(JVM)
- 가비지 컬렉터(Garbage Collector)
- 코드의 보안(Code Security)

1-3) 자바 가상 머신(JVM : Java Virtual Machine)



1-4) Garbage Collector

- 오랫동안 사용하지 않거나 메모리 부족현상이 발생하게 되면 JVM의 Garbage Collector에 의해 제거된다.
- C 나 C++ 의 경우 일일이 프로그래머가 control

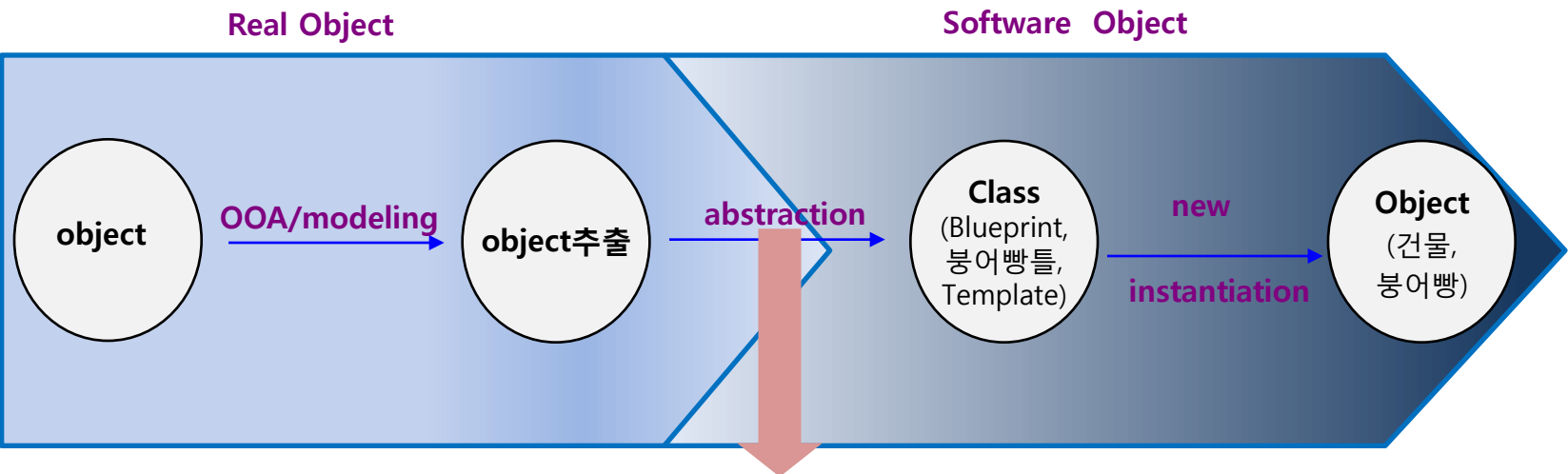
1. Java의 주요특징
2. 객체지향(Object-Oriented)
3. 기본적인 컴포넌트
(Identifiers, Keywords and Types)
4. 표현식과 흐름제어
(Expressions and Flow Control)
5. 배열(Array)

- Object, Class, attribute, method
- OOP의 3대개념
 - 은닉화(encapsulation)
 - 상속성(inheritance)
 - 다형성(polymorphism)
- OOP의 다른특징
 - Overloading
 - Interface
 - 추상화(abstract)
 - 모듈성(modularity)
 - 계층성(hierarchy)

2-1) 객체(object)

현실세계 : 현실세계에 존재하는 구분 가능한 속성과 행위를 가진 구체적인 사물

Software : 행위와 속성을 가진 모든 것



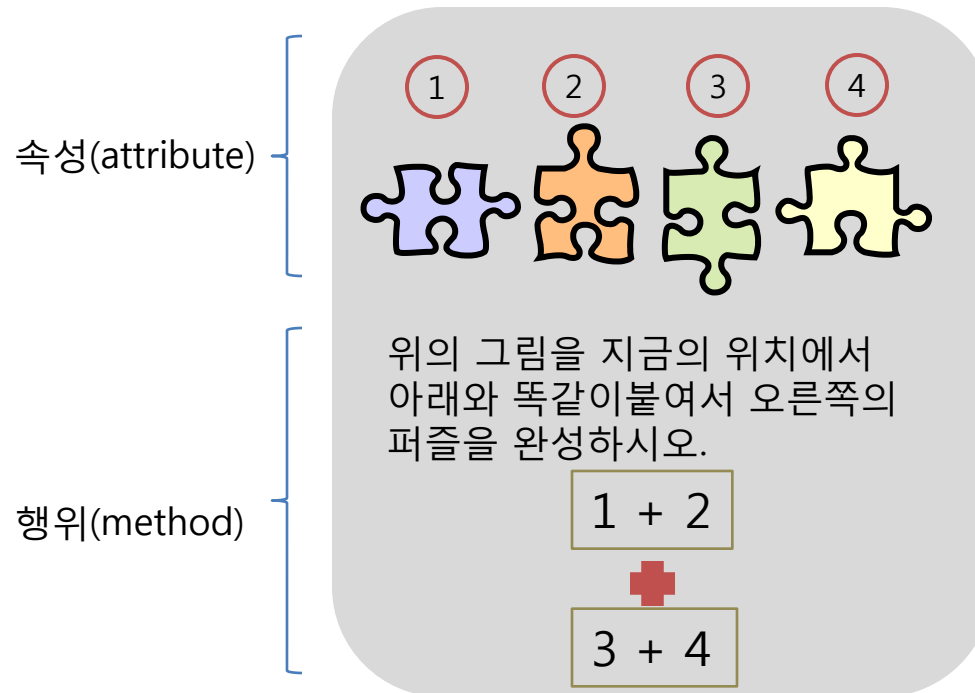
■ 추상화(abstraction)

: 나타내려는 객체의 특징과 목적을 간략하게 표현하는 것

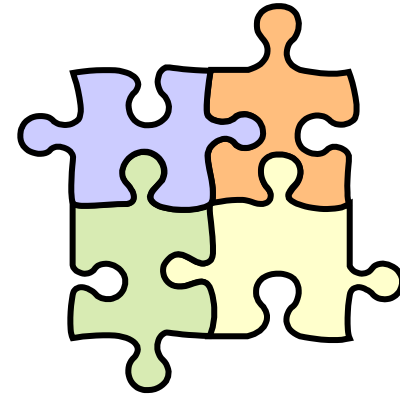
- 간략화(simplificatioin) : 의미를 잃어 버리지 않을 정도로 가장 간단하게 정리한 상태
- 클래스화(classficatioin) : 추상화 단계에서 얻어진 클래스 이름, 메소드, 속성, 생성자 등을 코드로 바꾸는 것

2-2) 클래스(class)

- Blueprint for Object(설계도)
- Object의 상세설명(specification)이다.



예상 artifact



2-3) 속성(attributes)

Basic syntax of an attribute

<modifiers> <type> <name>

Examples:

```
public class Cafe {  
    private int cup;  
    private float money = 6000.0F;  
    private String coffeeType = "Café Moca";  
}
```

2-4) 행위(methods) - 1

Basic syntax of an attribute

```
<modifiers> <return_type> <name> ([argument_list]) {  
    <statement>  
}
```

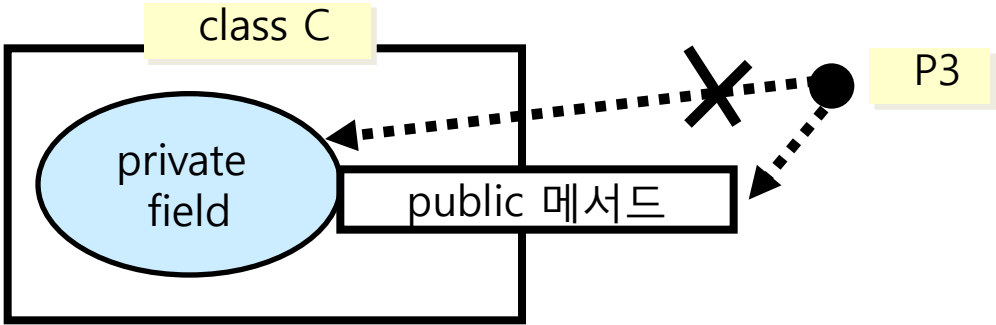
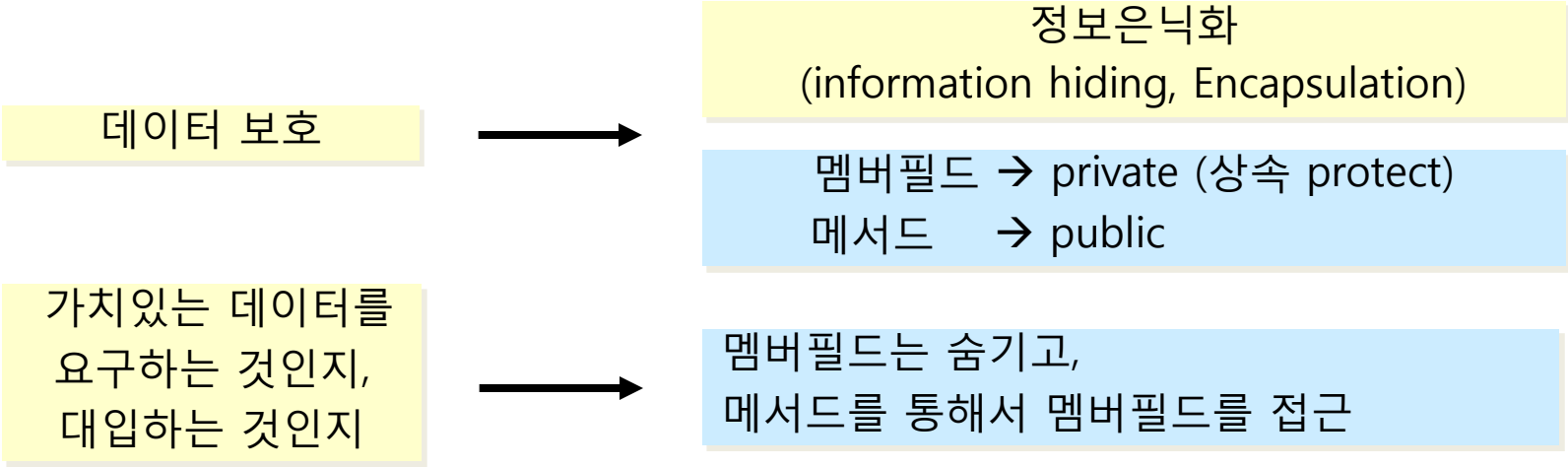
Examples:

```
public class CoffeeMachine{  
    public int money; // 자판기에 넣을 돈 준비  
  
    public int getMoney(){  
        return money;  
    } // 자판기에서 거스름 돈을 받는다.  
  
    public void setMoney(int money){  
        this.money= money;  
    } // 자판기에 돈을 넣는다.  
}
```

2-4) 행위(methods) -2

1. 객체 **타입**에 따른 구분
 - Static
 - non-static
2. **반환값**의 유무에 따른 구분
 - return 값이 존재하는 메서드
 - void 메서드
3. **작성**에 따른 구분
 - API
 - 사용자 정의 메서드

2-5) 은닉화(encapsulation)

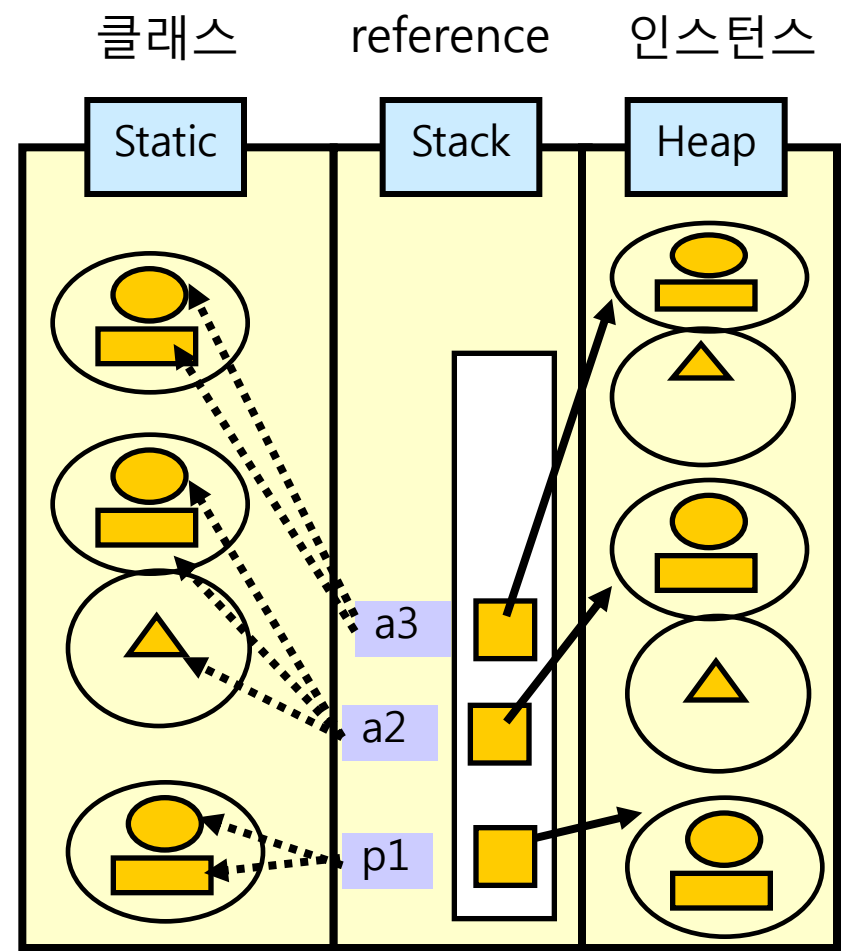
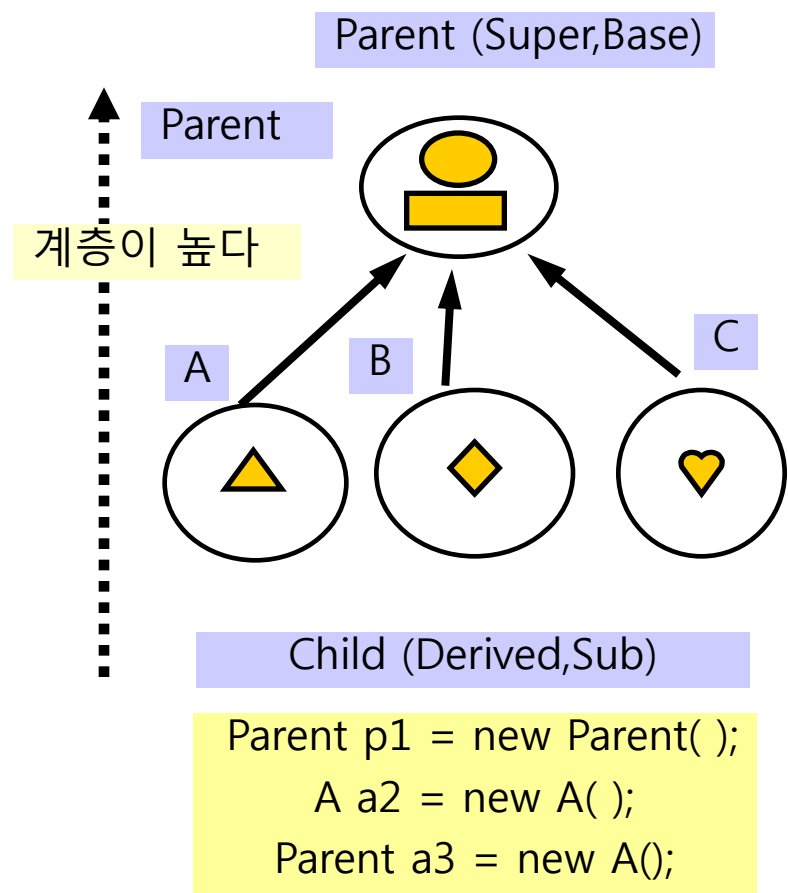


2-6) 상속성(inheritance)

→ 메모리의 4대특징

1. 자식이 생성되면 부모가 생성된다.
2. 자식의 설계도가 올라가면 부모의 설계도도 올라간다.
3. 생성된 주소는 부모의 주소를 가리킨다.
4. 설계도에 공개된 메소드만 사용할 수 있다.

2-7) 상속성(inheritance)



2-8) 다형성(polymorphism) : code

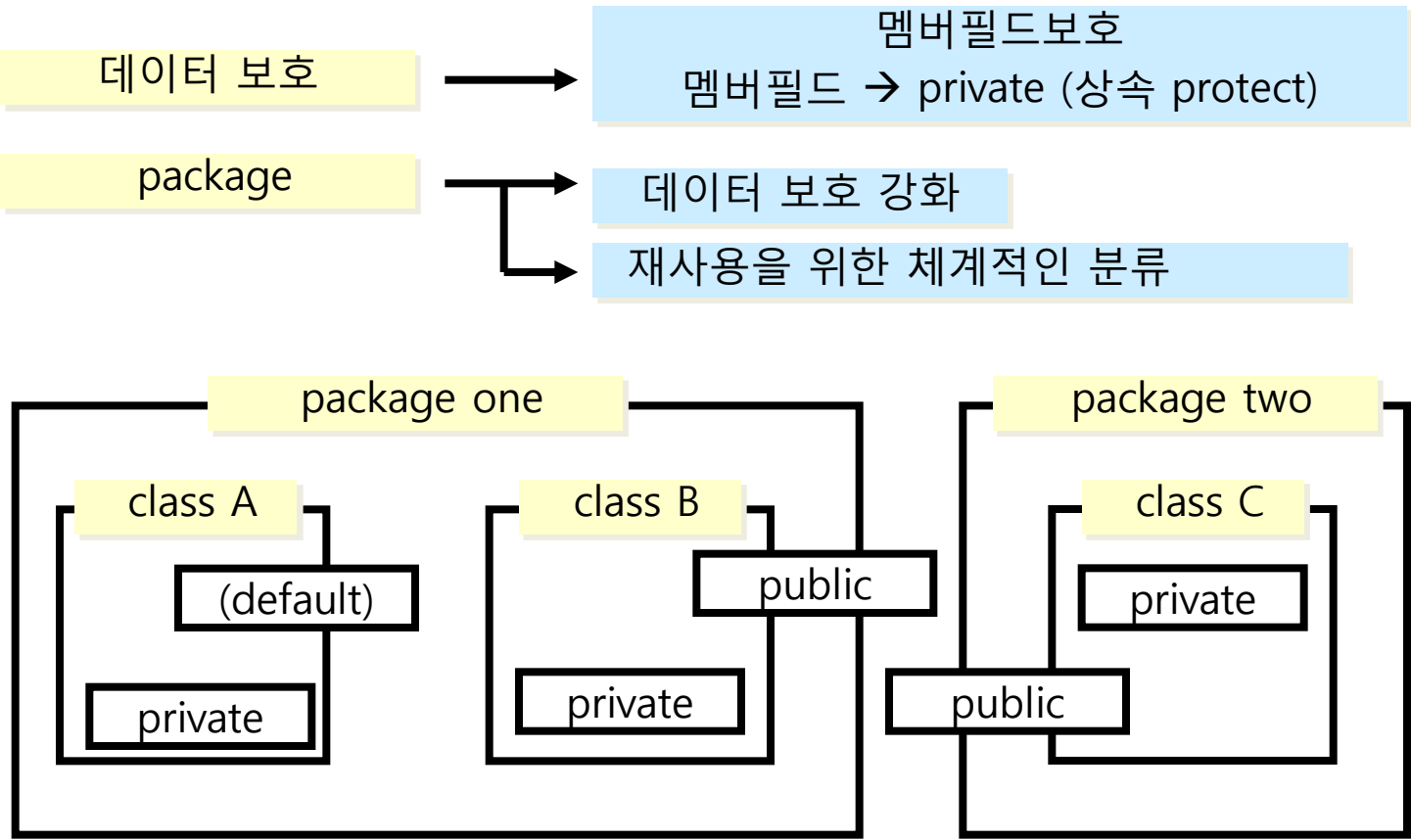
```
public interface Vehicle{  
    void open();  
}
```

```
public class Airplane implements Vehicle{  
    public void open(){  
        System.out.println("Airplane의 문이 열렸습니다.");  
    }  
}
```

```
public class Car implements Vehicle{  
    public void open(){  
        System.out.println("Car의 문이 열렸습니다.");  
    }  
}
```

```
public class Truck implements Vehicle{  
    public void open(){  
        System.out.println("Truck의 문이 열렸습니다.");  
    }  
}
```

2-9) 접근 제한자(access modifier)와 패키지(package)



1. Java의 주요특징
2. 객체지향(Object-Oriented)
- 3. 기본적인 컴포넌트**
(Identifiers, Keywords and Types)
4. 표현식과 흐름제어
(Expressions and Flow Control)
5. 배열(Array)

3-0) 명명법

1. Pascal

시작은 대문자, 의미가 바뀔 경우 대문자 사용
클래스, 인터페이스, 생성자의 명명시 사용

Ex) `class HelloJavaMain {}`

2. Camel

시작은 소문자, 의미가 바뀔 경우 대문자 사용
메서드, 멤버필드 등의 명명시 쓰인다.

Ex) `public void printValue (int number) {}`

3. Hungarian

변수 이름 앞에 변수의 자료형의 약자를 넣는 방법

Ex) `txtNumber1`

4. 전체 대문자 사용

상수 Ex) `Static final NUM = 1;`

5. 전체 소문자 사용

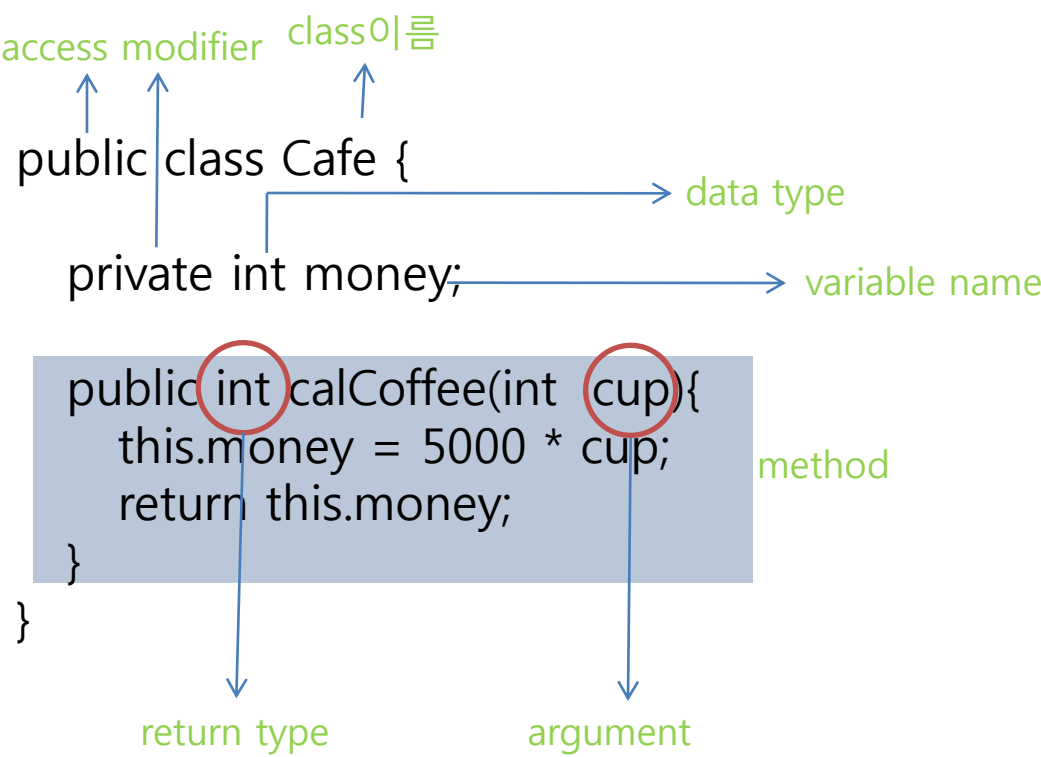
Package 이름 명명시

6. 예약어는 변수로 사용할 수 없다.

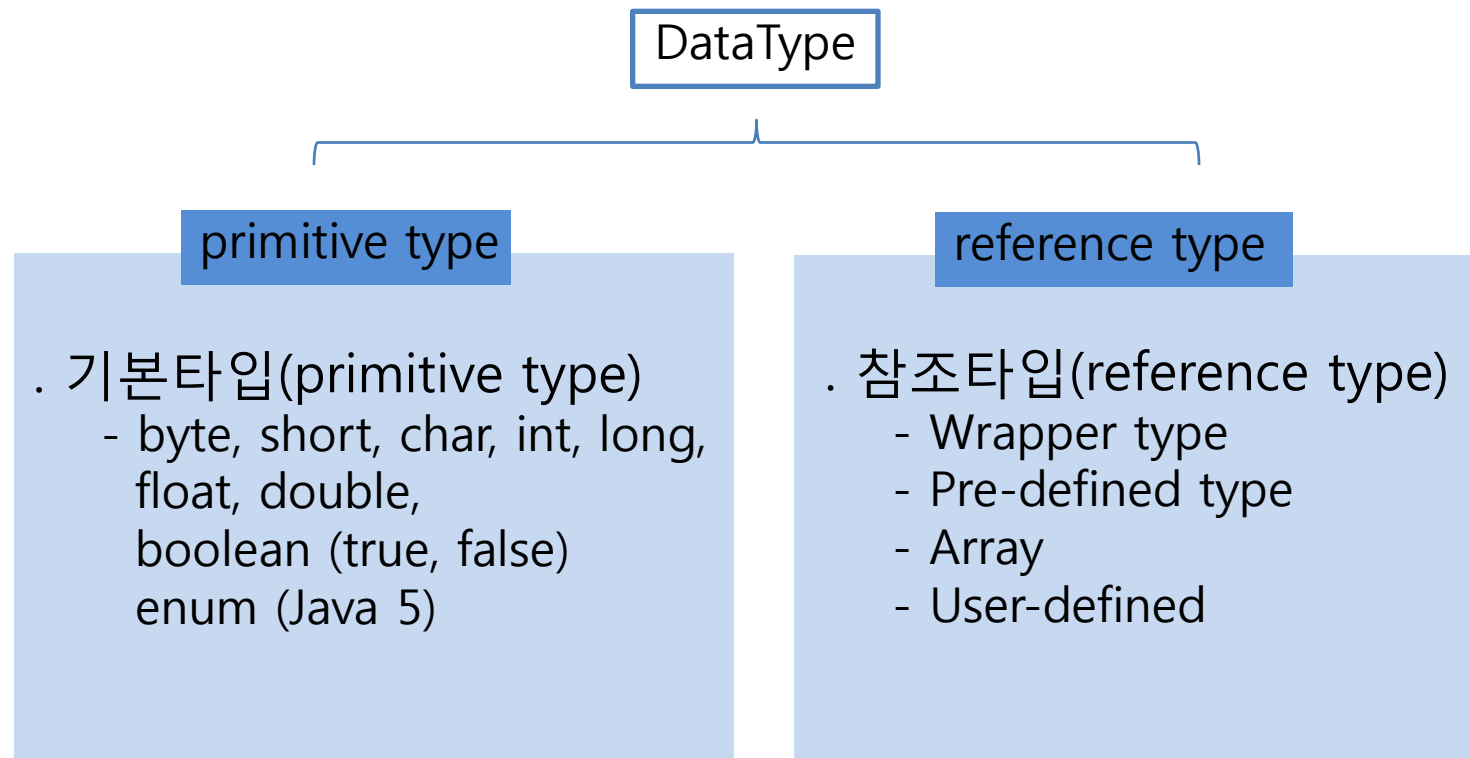
3-1) 주석(Comment)

주석표현	명칭	설명
//	한 줄 주석	// 이 한줄이 주석입니다.
/* */	여러 줄 주석	/* 이 부분은 주석입니다. */
/** */	문서 주석	/** 주석을 위한 클래스 입니다.*/

3-2) 클래스의 구조(Statement)



3-3) 데이터 타입(DataType)



3-4) 데이터 타입(DataType)의 형변환 [java2 or java5]

int -> Object 형 으로 변환하려면??
[primitive] -> [reference]

int a = 5;

Java5 Object o = a; ①

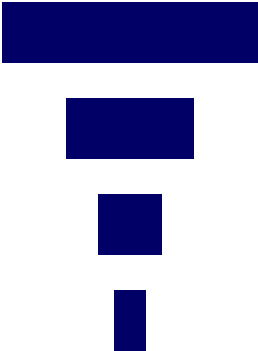
Java2 Object o = new Integer(a); ②



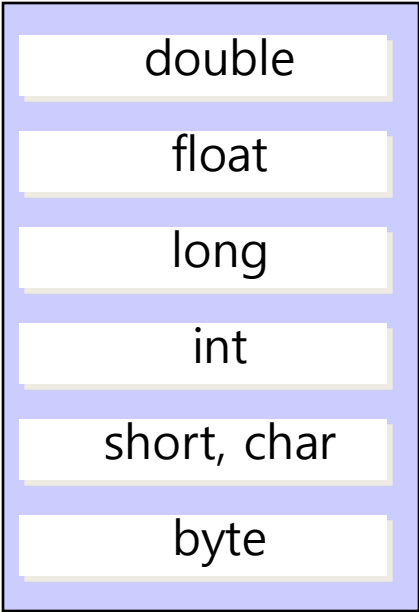
Java5 : Wrapper 클래스를 **암시적(자동)**으로 처리해준다.

3-5) 기본타입(primitive type)

그릇의 크기가 다양한 이유?



(down) cast,
explicit



(auto) promotion,
implicit

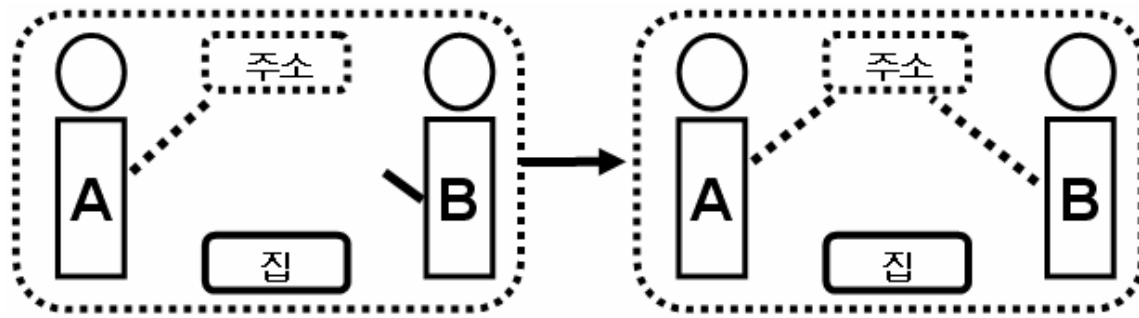
3-6) 기본타입(primitive type)의 크기

double	8 byte	$\pm 4.9\text{E-}324 \sim 1.8\text{E}308$
float	4 byte	$\pm 1.4\text{E-}05 \sim 3.4\text{E}038$
long	8 byte	$-2^{63} \sim 2^{63} - 1$
int	4 byte	$-2^{31} \sim 2^{31} - 1$
short, char	2 byte	$-2^{15} \sim 2^{15} - 1, 0 \sim 2^{16} - 1$ (65535)
byte	1 byte	$-2^7 \sim 2^7 - 1$

3-7) 참조타입(reference variable type)

- . Reference type → mutable
 - assign by reference
 - pass by reference

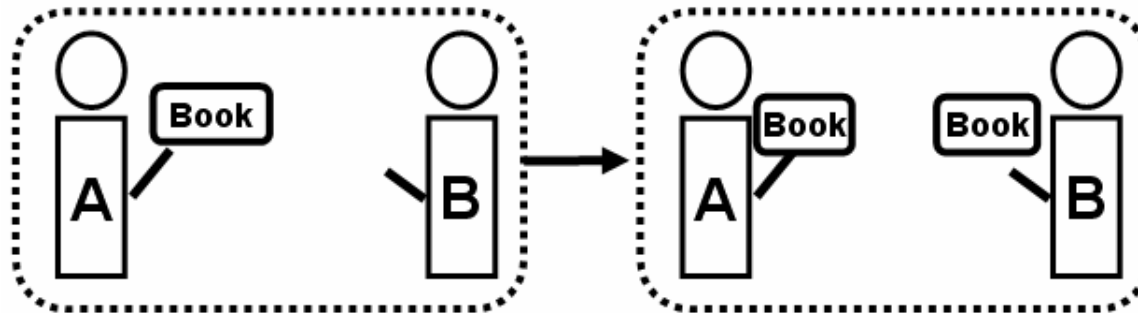
- . Assign by reference



3-8) 기본타입(primitive variable type)

- . Primitive type → immutable
→ assign by value
→ pass by value

- . Assign by value



3-9) 키워드(keywords)

abstract	do	implements	private	this
boolean	double	import	protected	throw
break	else	instanceof	public	throws
byte	extends	int	return	transient
case	false	interface	short	true
catch	final	long	static	try
char	finally	native	strictfp	void
class	float	new	super	volatile
continue	for	null	switch	while
default	if	package	synchronized	

3-10) 생성자(constructor)

- 모양은 메소드와 비슷하나 클래스 이름만 같고 리턴타입이 없다.
- 객체 초기화시 사용
- 객체생성시 한번만 호출된다.
- VendingMachine() 와 같이 argument가 없는 생성자는 default 생성자이다.

```
public class VendingMachine{  
    public VendingMachine() { }  
}
```

3-11) this keyword

- 클래스 내부에서 this는 멤버(멤버필드나 멤버 메서드)를 의미한다.

this() → 자기 멤버필드 초기화

```
public class Mabang{  
    public Mabang(){  
        this(3);  
    }  
  
    public Mabang(int n){  
        this.init(n);  
        // 실질적으로 만들어질 마방진  
        this.tung = n-1;  
        // 벽을 쳤을때 '통~~'하고 맨위나 아래로 넘김.  
    }  
}
```

3-12) 관습 : 약속 (Conventional)

키워드	사용법
package	package com.hankyung.uhs
classes	class Coffee
interfaces	interface Car
methods	balanceAccount()
variable	coffeeType
constants	PUB_COUNT MIN_SIZE

1. Java의 주요특징
2. 객체지향(Object-Oriented)
3. 기본적인 컴포넌트
(Identifiers, Keywords and Types)
- 4. 표현식과 흐름제어**
(Expressions and Flow Control)
5. 배열(Array)

4-1) 스트링(String)

1) 컨케트네이션(concatenation)

```
int a = 1;  
int b = 2;  
String str = "can";  
System.out.println(str + a + b);  
System.out.println(a + b + str);
```



"can12"
"3can"

2) 이뮤터블 하다(immutable)

```
String str = "홍길동";  
change(str);
```

```
public void chage(String str){  
    str+"길용이어라~~";  
}
```



참조타입이므로
str = "aa"
값을 받지 않으면
값이 변하지 않는다.
따라서 str의 결과는??

결과 : "홍길동"

4-2) 논리 연산자(Logical Operators)

&& - AND, || - OR, != - NOT

```
public class LogicalOpt{
    public int makeOptRtn(int num1, int num2, int opt){
        int isOptRtn = 0;
        if(opt == 1){
            isOptRtn = num1 + num2;
        }

        if (opt == 2) {
            isOptRtn = num1 * num2;
        }

        if ((num1<num2) && opt == 3) {
            isOptRtn = num1 - num2;
        }

        return isOptRtn;
    }
}
```

4-3) 비트 연산자(Bit Logical Operators)

! – NOT, & - AND, | - OR, ^ - XOR

~ 0 1 0 0 1 1 1 1

1 0 1 1 0 0 0 0

0 0 1 0 1 1 0 1
& 0 1 0 0 1 1 1 1

0 0 0 0 1 1 0 1

0 0 1 0 1 1 0 1
^ 0 1 0 0 1 1 1 1

0 1 1 0 0 0 1 0

0 0 1 0 1 1 0 1
0 1 0 0 1 1 1 1
0 1 1 0 1 1 1 1

4-4) 쉬프트 연산자(Shift Operators)

1357 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 1 0 1

1) Left-Shift

1357 << 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 1 0 1 0 0 0 0 0

2) Right-Shift

1357 >> 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0

3) >>> 무조건 오른쪽으로 해당 값만큼 이동

1357 >>> 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0

4-5) 캐스팅(Casting)

1) 다운캐스팅(down-casting)

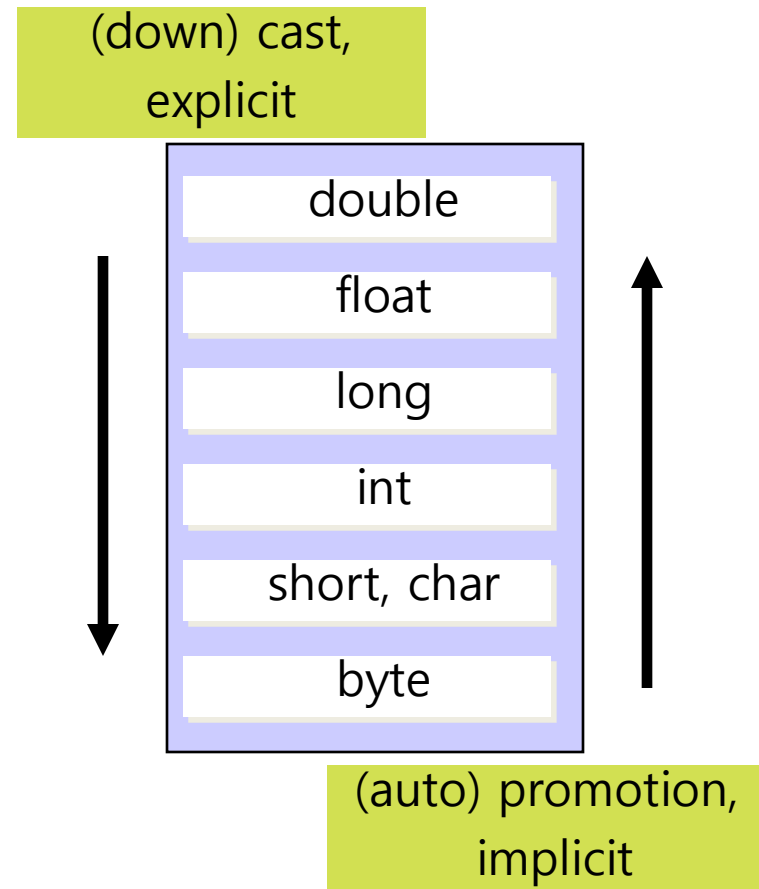
```
long bigValue = 99L;  
int smValue = (int)bigValue;
```

결과 : 99

2) 업캐스팅(up-casting)

```
int sb = 30;  
long tt1 = sb;
```

결과 : 30



4-6) 흐름 제어(Flow Control)

1) 반복문(Loop Statement)

```
while( i<10 ){  
    System.out.println(" [while] 자바 프로그래밍의 중심에 서다.");  
}
```

```
for(int i=0; i < 10 ; i++){  
    System.out.println(" [ for ] 자바 프로그래밍의 중심에 서다.");  
}
```

```
do{  
    System.out.println(" [ do-while ] 자바 프로그래밍의 중심에 서다.");  
}while(i<10)
```

4-6) 흐름 제어(Flow Control)

2) 분기문 (branch statement)

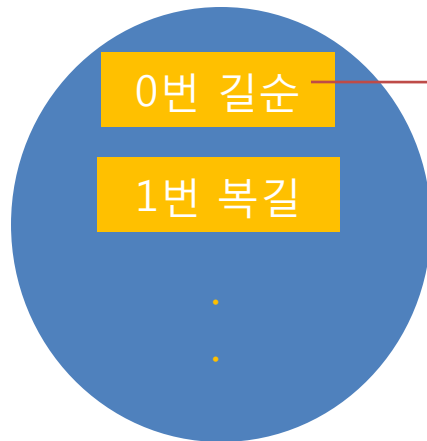
```
switch( isBestVehicle) {  
  
    case 1:  
        str = “우주선”;  
        break;  
    case 2:  
        str = “비행기”;  
        break;  
    case 3:  
        str = “열기구”;  
        break;  
    case 4:  
        str = “풍선”;  
        break;  
  
}
```


1. Java의 주요특징
2. 객체지향(Object-Oriented)
3. 기본적인 컴포넌트
(Identifiers, Keywords and Types)
4. 표현식과 흐름제어
(Expressions and Flow Control)
5. 배열
(Array)

5-1) 배열(Array)

정의 : 배열은 같은 타입을 여러 개 나열한 것
특징 : 같은 타입의 변수를 여러 개 만들 경우에 사용
하나의 이름으로 위치에 따라 값을 사용

Ex) 장미반



장미반 0번인 길순
장미반 1번인 복길



장미반[0] = "길순"
장미반[1] = "복길"

위와같이 사용하기 위하여

```
String str[0] = "길순"  
String str[1] = "복길"
```

5-2) 배열선언(array declare)

기본타입 : `int[] ar <single>; int[][] ar ; <multi>`

참조타입 : `Top[] art<single> ; Top[][] art ; <multi>`

```
int[] a = null;           // 사용방법 #1
a = new int[5];
a[0]=2;
a[1]=5;
a[2]=3;
a[3]=9;
a[4]=8;
```

```
int[] b = new int[]{ 2 , 5 , 3 , 9 , 8 };
           //사용방법 #2
```

```
int[] c = { 2 , 5 , 3 , 9 , 8 };
           //사용방법 #3
```

5-3) 배열의 복사(array copy)

얕은카피(shallow copy) : 배열의 주소값을 넘긴다.(reference assignment)

깊은카피(deep copy) : 배열의 값 자체를 넘긴다.(value assignment)

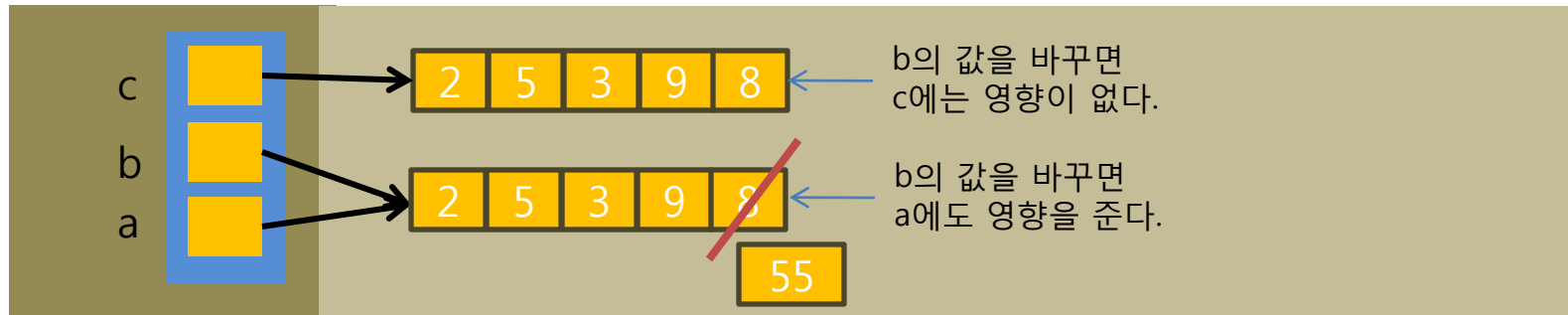
```
int[] a = {2,5,3,9,8};
```

```
int[] b = a; //reference assignment
```

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

```
System.arraycopy(a,0,c,0,a.length); //value assignment
```

```
b[4] = 55;
```



진정한 객체 지향 프로그래밍

(More Object-Oriented Programming)

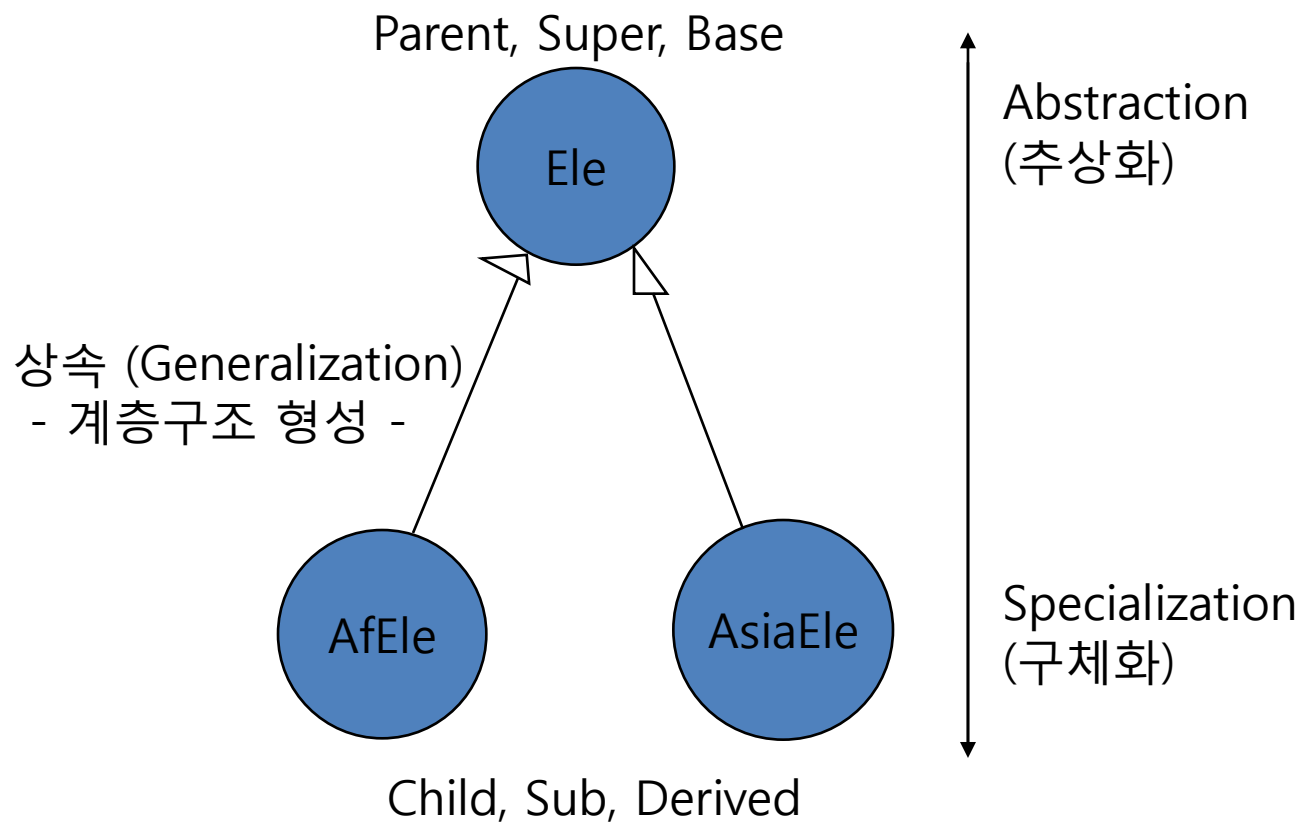
6. 클래스 디자인

(Class Design)







7. 진화된 클래스의 특징

(Advanced Class Features)

6-1) OOP(Object Oriented Programming) 용어



6-2) 관계(Relationship)

	extends	Generalization
	implements	Realization
	(has,link)	Association
	(단순 구성)	Aggregation
	(떨어질 수 없는 관계)	Composition
	(다른클래스 의존)	Dependency

6-3) OOP(Object Oriented Programming) 3대 개념

은닉화(encapsulation)

Data보호(field) → member field private
→ member method public
메서드를 통하여 멤버필드에 접근

상속성(inheritance)

부모의 member를 물려받는 것

다형성 (polymorphism)

부모의 메소드가 자식의 형태에 따라 다양한 형태로 호출

6-4) 접근제한자(access modifier)

Private

- 한 클래스 내에서만 사용가능
상속 받아도 사용이 불가능

Default (아무것도 표시하지 않을 경우)

- 같은 패키지 내에서 접근가능

Protected

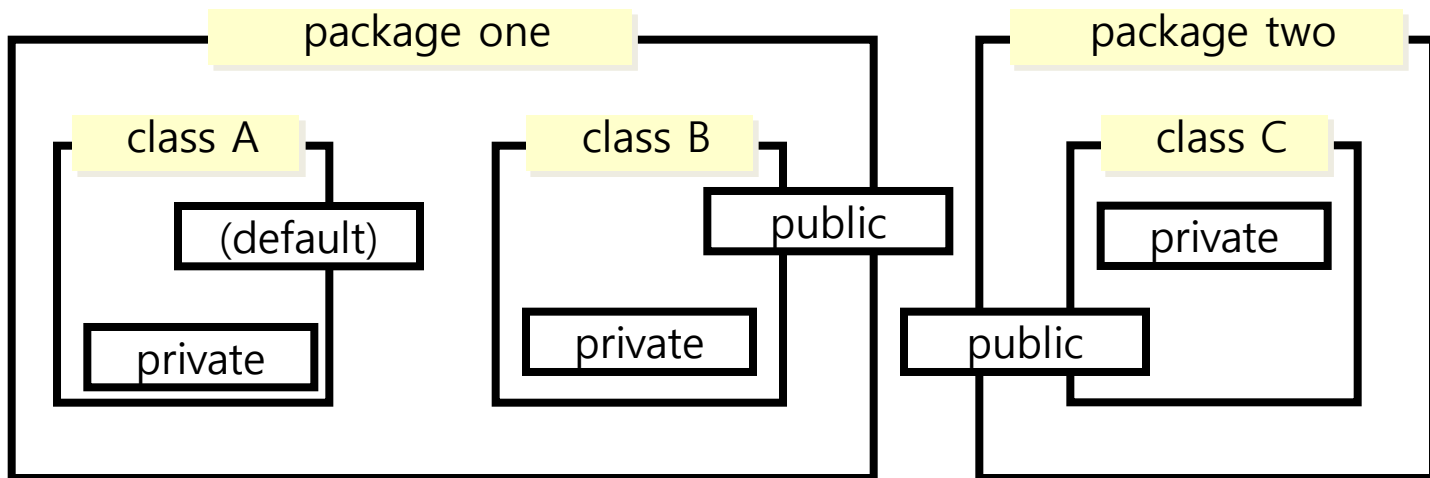
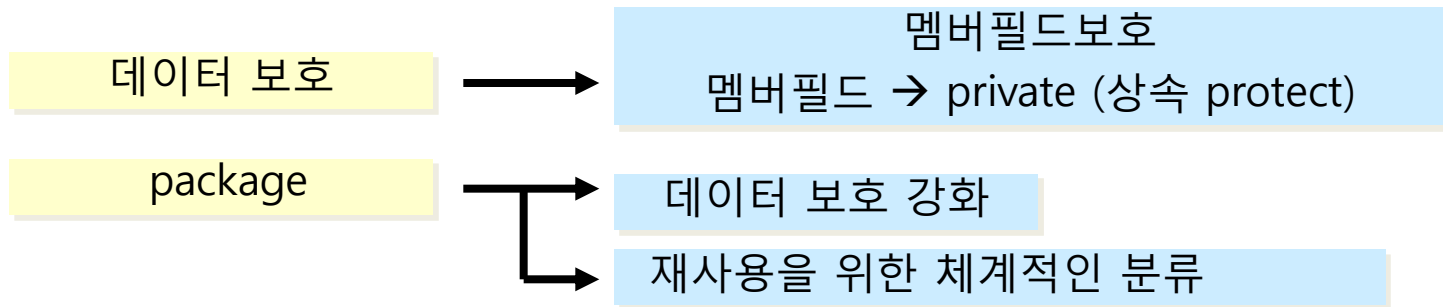
- 상속 시는 public, 상속을 받지 않으면 Default

Public

- 어디서든 접근가능

2-8 앞에내용 참조

2-8) 접근 제한자(access modifier)와 패키지(package)



2-5 앞에내용 참조

2-5) 은닉화(encapsulation)

데이터 보호



정보은닉화

(information hiding, Encapsulation)

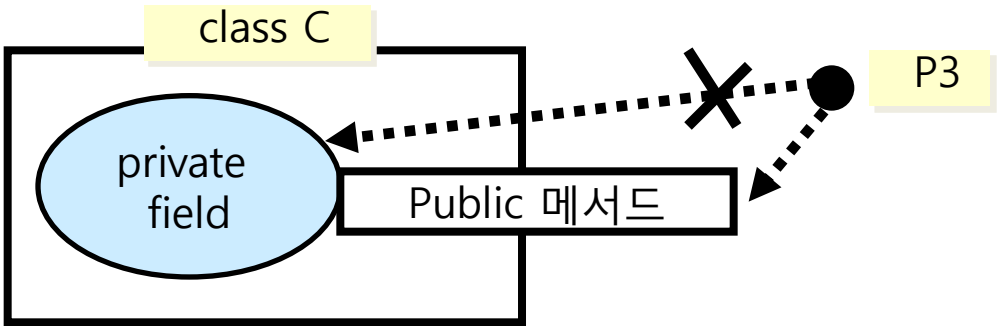
멤버필드 → private (상속 protect)

메서드 → public

가치있는 데이터를
요구하는 것인지,
대입하는 것인지



멤버필드는 숨기고, 메서드를 통해서
멤버필드를 접근

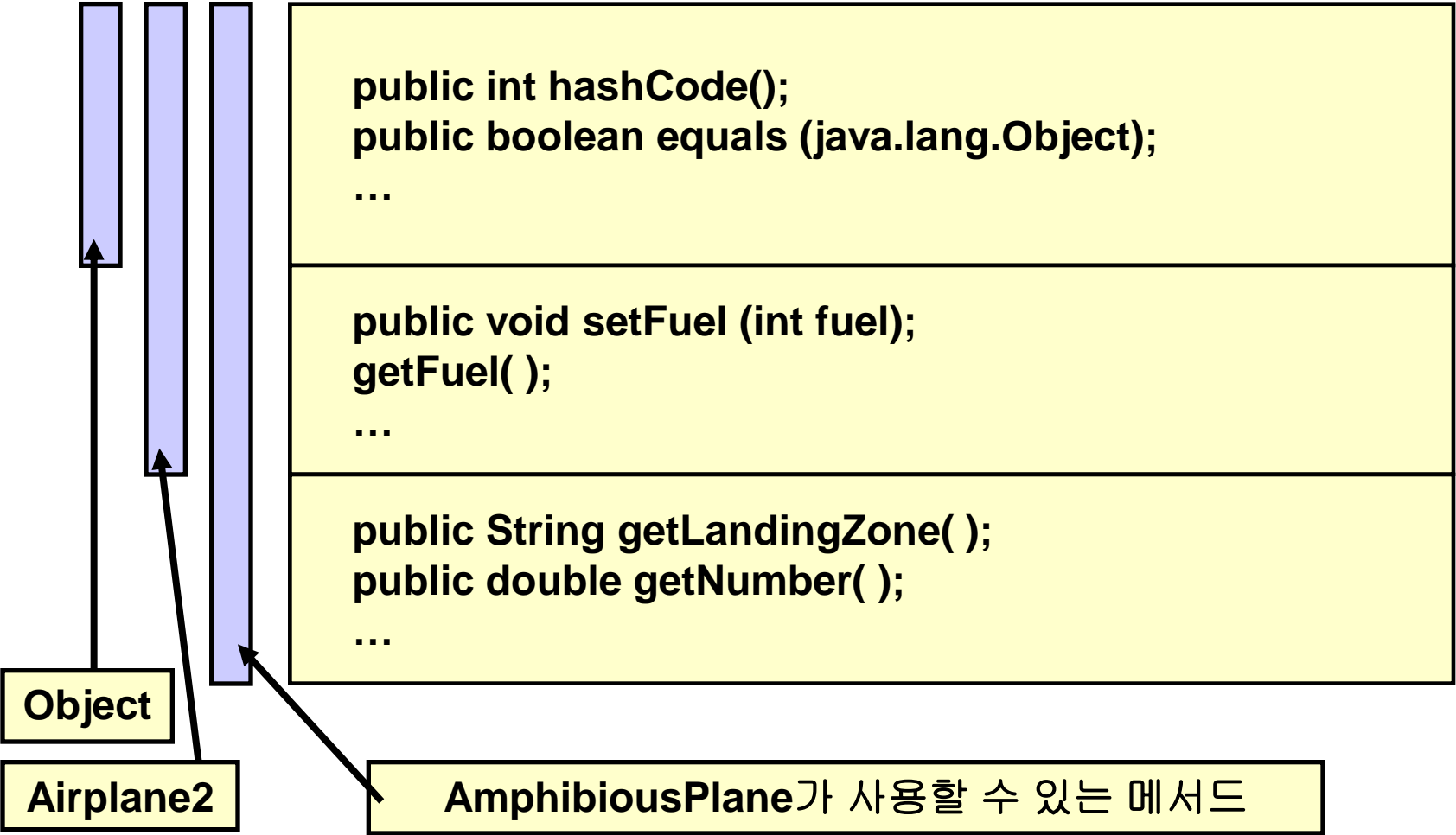


6-5) 상속성(inheritance)

- 부모의 멤버(변수&메서드)를 물려받아 사용
(단, 멤버는 static이 아니어야 한다)
- 생성자는 물려받지 못함
- extends 키워드를 사용
- 부모의 초기화 시 super,
자신의 초기화 시 this를 사용

ex) `public Truck extends Car{ }` //사용법

6-5) 상속성(inheritance)- 그림설명



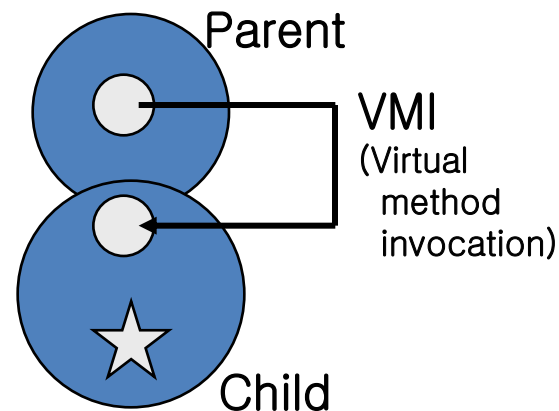
6-6) 오버라이딩(overriding)

Overriding이란?

자식클래스에서 메서드의 역할을 변경하거나 확장 시에
상속 받은 method 를 새로 정의하는 것

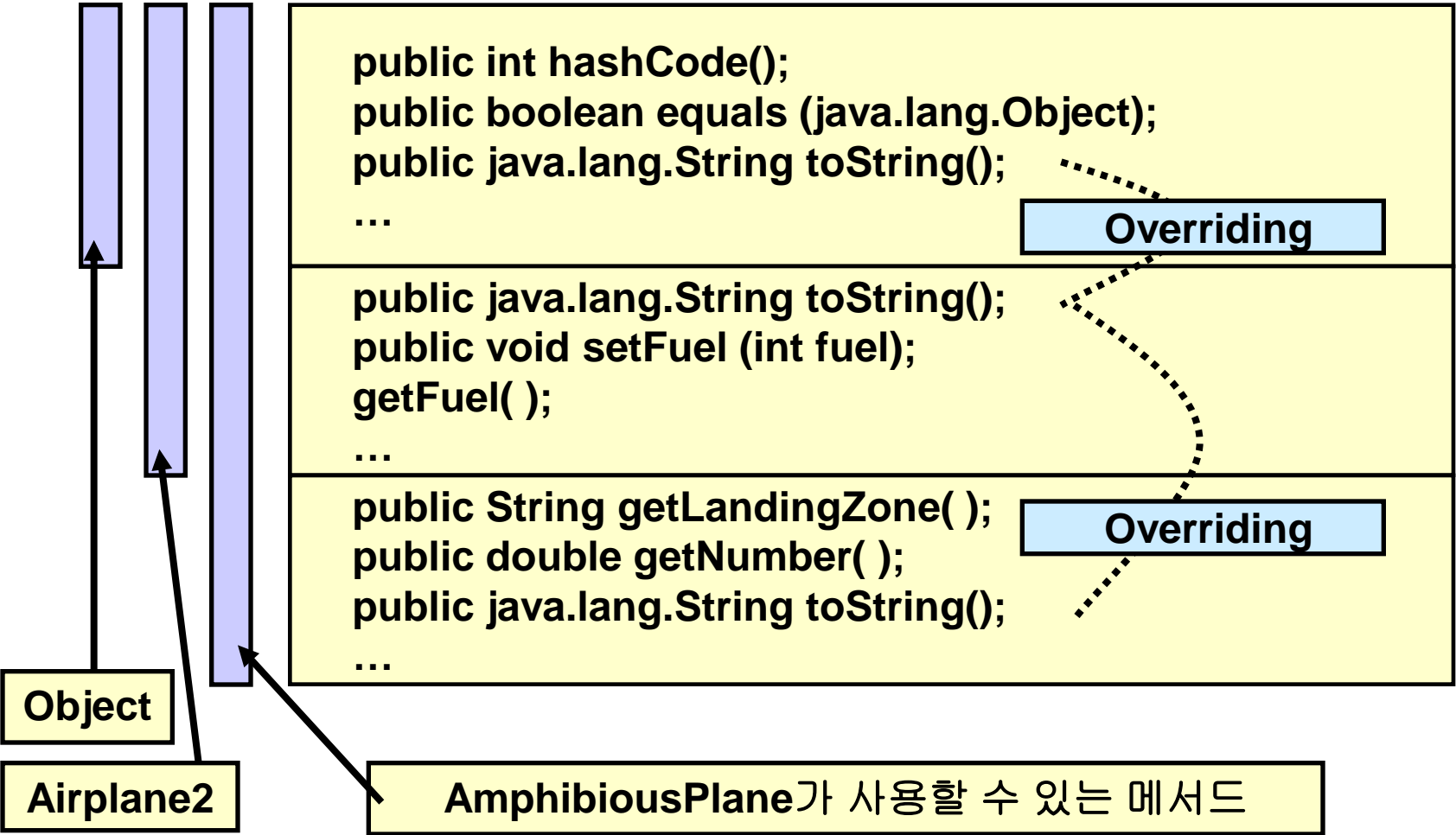
Overriding의 조건

1. 부모클래스의 메서드와 **이름**이 동일 할 것
2. 매개변수의 타입과 **개수,순서**가 동일 할 것
3. **리턴 타입**이 같을 것
(접근 제어자는 부모의 메서드 보다 같거나 크게)



상속 시 부모의 method 를 호출하면 VMI에 의해 자식 method 가 호출됨

6-6) 오버라이딩(overriding)- 그림설명



6-6) 오버라이딩 (overriding) - 예제

```
public class Parent{  
    public void overrideEx(){  
        System.out.println("한경닷컴");  
    }  
}
```

```
public class Child extends Parent{  
    public void overrideEx(){  
        System.out.println("한경닷컴 교육센터");  
    }  
}
```

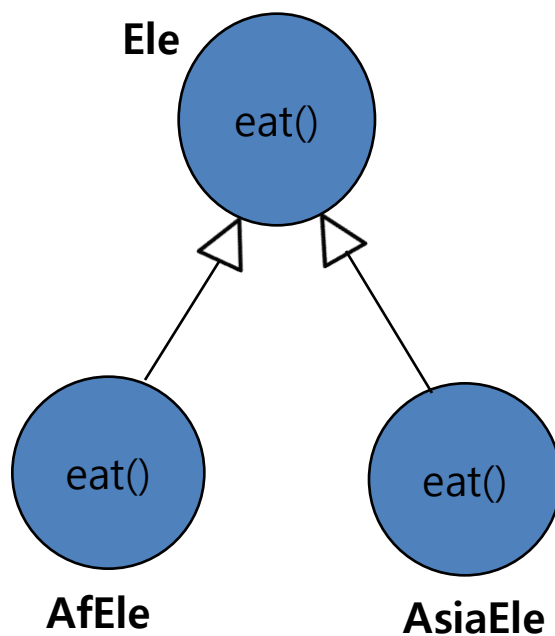
```
Parent p = new Child();  
p.overrideEx();
```

=====

결과 : 한경닷컴 교육센터

6-7) 다형성(polymorphism)

정의 : 다양한 형태를 나타낼 수 있는 능력
자식의 종류에 따라 다양하게 호출



상속받는 자식에 따라서
다양한 방법으로 호출 할 수 있다.
(overriding 등)

6-7) 다형성(polymorphism)의 발생원리

부모의 이름으로 자식 생성 할 수 있다.

```
Ele e1 = new AfEle();
```

```
Ele e2 = new AsEle();
```

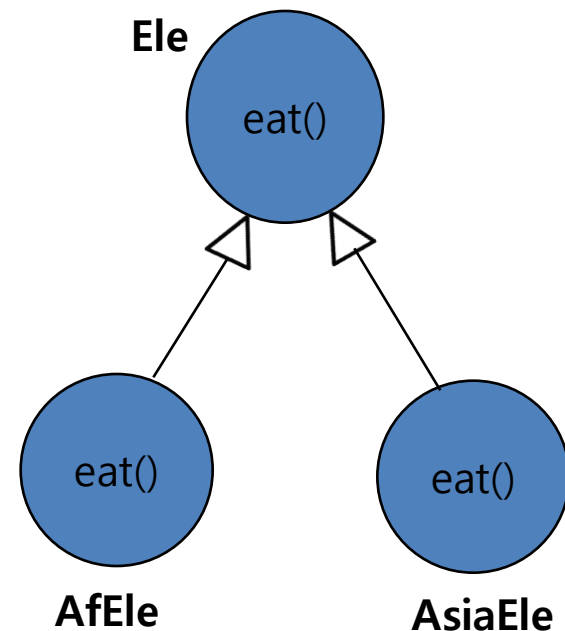
부모의 이름으로 자식을 받을 수 있다.

```
AfEle af1 = new AfEle();
```

```
Ele e3 = af1;
```

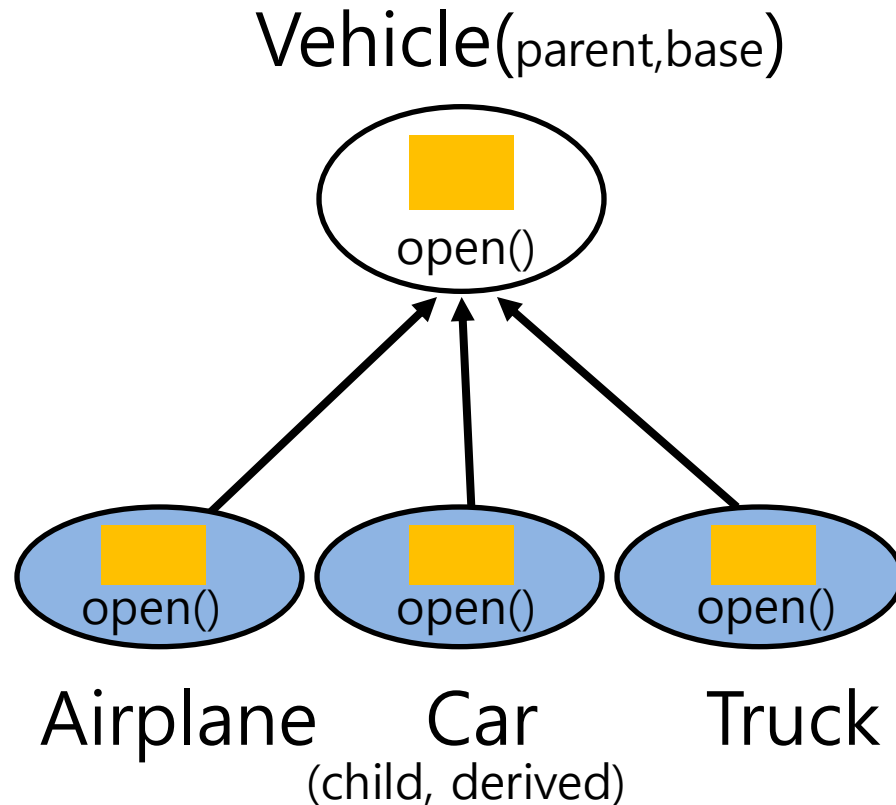
부모의 method로 자식의 method를 호출할 수 있다.

단, 해당 method는overriding 되어있어야 한다.



6-7) 다형성(polymorphism)

자식의 형태에 따라 부모의 메소드가 다양한 형태로 구현된다.
(단, 상속관계 일경우에)



```
Vehicle vh = new Airplane( );  
vh.open();
```

```
Vehicle vh = new Car( );  
vh.open();
```

```
Vehicle vh = new Truck( );  
vh.open();
```

6-8) 오버로딩(overloading)

같은 이름의 메서드지만 변수의 **타입, 개수** 등이 다름

(EX)

```
public void overLoadingTest();  
public void overLoadingTest(double d);  
public void overLoadingTest(double d, double e);  
public void overLoadingTest(double d, double e, double f);  
public void overLoadingTest(double d, int i, double f);
```

```
//public void overLaodingTest(String s, long i, int j);  
//public int overLoadingTest(String s, long i, int j);  
//리턴 타입만 다른 것은 overloading안됨!
```

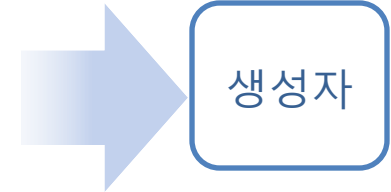
6-9) 생성자(constructor)

- 메서드 중 가장 먼저 호출된다.
- 초기화를 위해 생성
- 클래스 이름과 같다.
- 메서드 형태이지만 리턴 타입이 없다.
- New 키워드 사용시 자동 호출됨
- 생성자를 만들지 않으면 내용이 비어있는 상태의 기본 생성자가 자동생성
- 생성자는 상속될 수 없다.

```
public Car{  
    public Car(){} : 생성자  
}
```

6-10) 생성자 오버로딩(overloading constructors)

- 모양은 메소드와 비슷하나 클래스 이름만 같고 리턴타입이 없다.
- 객체 초기화시 사용
- 객체생성시 한번만 호출된다.

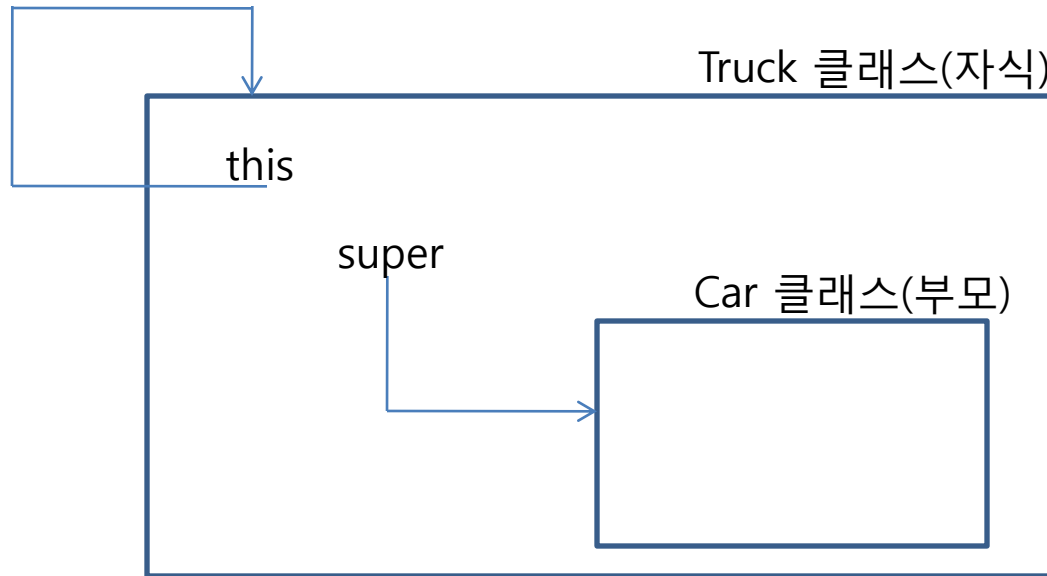


- Car() 와 같이 argument가 없는 생성자는 default 생성자이다.

```
public Class Car{  
    ...  
    public Car( );  
    public Car(String name, String id );  
    public Car(String name, String id, String price );  
    ...  
}
```



6-11) this, super 키워드



6-11) super 키워드

- 오버라이딩이 제공되면서, 자식 클래스에서 부모 클래스의 오버라이딩된 메서드나 멤버변수를 접근할 수 있어야 하는 필요성이 생겼다.
- super : 상속관계에 있을경우에 부모의 클래스를 참조할 경우에 사용된다.
부모의 멤버(field, method)를 초기화 한다.

```
public Class Car extends Vehicle{  
    ...  
    super( );  
    super(String name, String id );  
    super(String name, String id, String price );  
    ...  
}
```


6-11) this 키워드

- 자기 멤버(field, method)를 초기화

```
public class VendingMachine{  
  
    String coffeeType = null;  
  
    public VendingMachine(){ // this();  
    }  
  
    public VendingMachine(int coffee, int water){  
        this(coffee, water);  
    }  
  
    public int calCoffee(String coffeeType){  
        this.coffeeType = coffeeType;  
    }  
}
```

6-12) instanceof 연산자(operator)

- 객체가 해당 클래스 or interface에 의해 생성되었는지 체크하는 연산자
- return 값은 true, false
- *<reference field> instanceof <type>*

- ex)

```
public class Employee extends Object
public class Manager extends Employee
```

```
=====
public void doSomething(Employee e){
    if(e instanceof Manager){
        System.out.println("true");
    }else{
        System.out.println("false");
    }
}
```

6-13) Object 클래스(object class)

- 모든 클래스의 부모클래스이므로 자동상속
- 상속받지 않아도 사용가능

equals()	해쉬코드를 이용하여 참조타입을 비교
hashCode()	객체를 위한 해쉬코드(객체를 유일하게 식별할수 있는 정수값)를 반환
#clone()	객체를 복제
getClass()	객체의 실제 자료형의 클래스 반환
toString()	객체에 대해 표현하고 있는 문자열 반환 일반적으로 하위클래스는 toString 메서드를 재정의

6-14) toString 메소드(The toString Method)

- Object를 String타입으로 convert시켜주는 메소드이다.
- ```
Date now = new Date();
System.out.println(now);
```

## 6-15) == 연산자와 equals 메소드 비교

( The ==Operator Compared With the equals Method )

- == : 두개의 reference가 같은 Object일때 사용한다.
- equals : 동일하지 않은 reference Object일때 사용한다.
- return값은 true or false

ex)

```
Car car1 = new Car();
```

```
Car car2 = new Car();
```

```
car1.equals(car2)
```

\* Equals는 기본적으로 Hashcode를 비교

- 모든 객체는 다른 객체로 취급
- 객체의 값을 비교하려면 overriding해야 함
- Equals overriding시 hashCode()도 함께 Overriding 해야 한다.

## 6-16) 래퍼 클래스(Wrapper Class)

- Wrapper Class를 통해서 기본타입과 참조타입간 변환하려고 할때 사용

| Primitive Data Type | Wrapper Class |
|---------------------|---------------|
| boolean             | Boolean       |
| byte                | Byte          |
| char                | Char          |
| short               | Short         |
| int                 | Int           |
| long                | Long          |
| float               | Float         |
| double              | Double        |

ex)

```
Integer it = new Integer(5)
int a = it.intValue();
```

```
String s = "12345";
int b = Integer.parseInt(s);
```

# 진정한 객체 지향 프로그래밍

(More Object-Oriented Programming)

## 6. 클래스 디자인

(Class Design)

## 7. 진화된 클래스의 특징

(Advanced Class Features)

## 7-1) static 키워드

- 클래스에서는 사용 못하고, **메소드**와 **멤버변수**에만 사용될 수 있다.
- 클래스 객체 생성 없이 사용할 수 있다  
클래스 이름.method() or 클래스 이름.멤버변수로 접근가능  
ex) Car.speed() or Car.wheelNum
- **Non-static 메소드**는 static 메소드를 사용할수 있지만  
**static 메소드**는 해당 멤버가 아니면 Non-static 메소드에 접근할수 없다.



## 7-2) final 키워드

final class

- 상속이 불가능

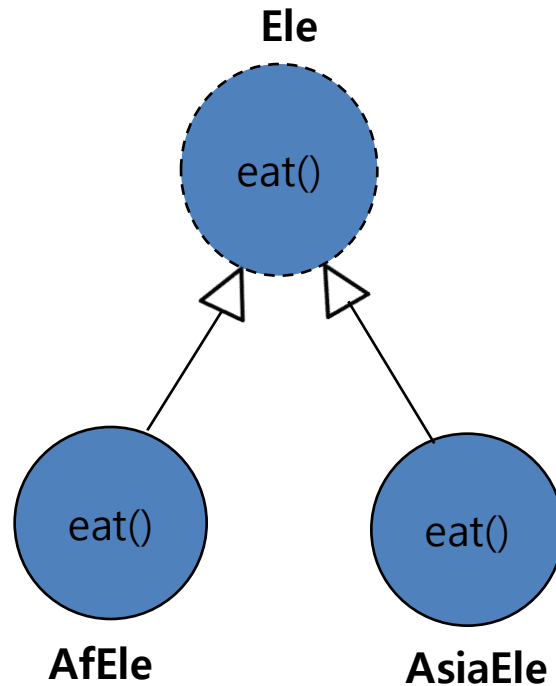
final method()

- 오버라이딩 불가능

final 변수(variable)

- 변수에 저장된 값을 변경할수 없음

## 7-4) 추상 클래스(Abstract Class)- 그림설명

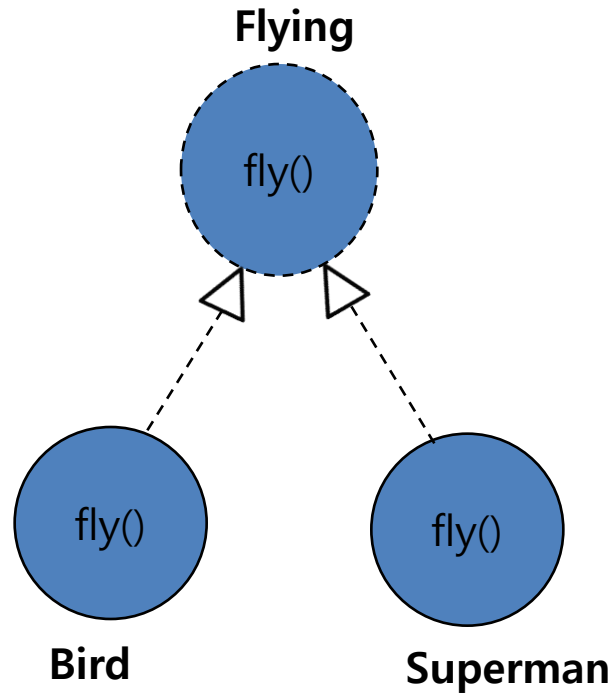


- 상속받는 클래스가 Elephant라는 연관성이 있다.

### 7-3) 추상 클래스(Abstract Class)

- 메소드 중 하나라도 abstract이면 abstract class
- abstract메소드가 하나라도 있을 경우 abstract class 이며 상속을 강요한다.
- 연관성이 있는 클래스간의 상속강요
- 추상클래스에서 추상메소드가 있다면 자식에서 반드시 그 메소드를 구현해야 한다. 만약 자식에서 구현을 하지 않으면 에러발생.

## 7-5) 인터페이스(Interface) - 그림설명

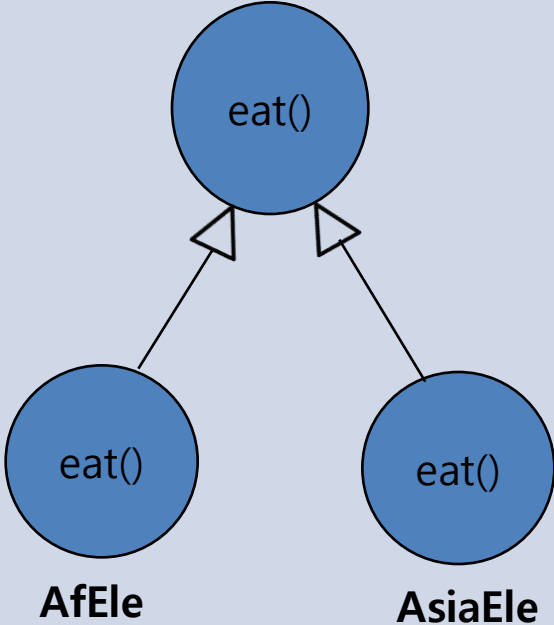
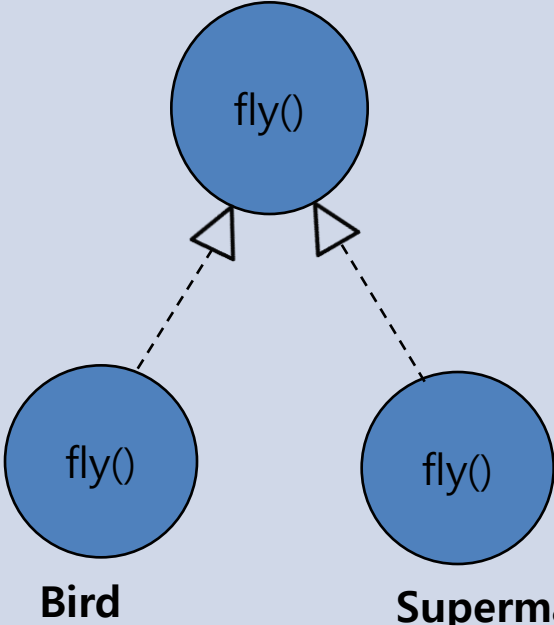


- 모두 날다(fly())의 기능외에는 연관성이 없다.

## 7-5) 인터페이스(Interface)

- 모든 메서드가 추상메서드로 구성
- 변수와 추상메서드의 나열로만 구성
- 자기 이름으로 자신의 객체 생성불가
- 밀접한 관계를 가지는 추상클래스와는 달리 단순한 기능을 공통점으로 가진다
- 추상메서드를 반드시 구현해야 하는 강제성
- 인터페이스를 이용하여 다중상속을 흉내냄
- 구현된 메서드가 하나라도 있으면 에러  
(메서드가 선언만 되어있음)

7-5) 추상(abstract) 클래스와 인터페이스(interface) 비교

|     | Abstract Class                                                                                                  | Interface                                                                                                           |
|-----|-----------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| 차이점 | 연관된 것 중 상속강요                                                                                                    | 연관이 적은 것들의 상속강요                                                                                                     |
| 키워드 | extends                                                                                                         | implements                                                                                                          |
|     | <div><p><b>Ele</b></p></div> | <div><p><b>Flying</b></p></div> |
|     |                                                                                                                 | interface간에는 extends로 상속                                                                                            |

## 7-5) 네스티드 클래스(Nested Class)

- 특정 외부클래스에게만 public, 다른 외부 클래스에게는 private 권한을 주고싶을때 사용
- 특징
  1. 반드시 클래스 생성후 사용
  2. 내부 클래스는 외부 클래스의 멤버를 마음대로 사용가능
  3. 외부에서 단독으로 사용 불가
  4. 외부 클래스 이름\$내부 클래스 이름.class로 생성
- 예제

```
class 외부_클래스{
 class 내부_클래스{
 ...
 }
}
```

# 어플리케이션 완성

## (Building Applications)

### 8. 예외처리

(Exceptions)

### 9. 텍스트 기반 어플리케이션

(Text-Based Applications)



## 8) 예외와 예외처리

### 가. 예외와 예외처리방법

- 자바에서 제공되는 예외처리에는 두 가지 방법이 있습니다.

#### 1. try/catch/finally 구문을 사용

- try블록에서 에러가 발생하면 catch 블록의 내용이 처리
- Finally은 에러여부와 관계없이 무조건 실행
- 상위에러는 나중에 처리

#### 2. throw, throws

- 원하는 위치에서 예외를 발생시킨다
- 에러만 발생시키므로 누군가는 catch해야 함

## 8) 예외와 예외처리

나. 예외처리 기본 구문 - try/catch 문

```
try{
 int a = Integer.parseInt("123");
 System.out.println(a);

}catch(NumberFormatException ee){

 System.out.println("int타입이 아닙니다");

}
```

## 8) 예외와 예외처리

### 다. 예외처리 추가 구분 - finally 문

- finally 블록은 try/catch 구문의 제어가 끝난 후에 반드시 처리해야 할 마무리 작업들을 처리하는 구문

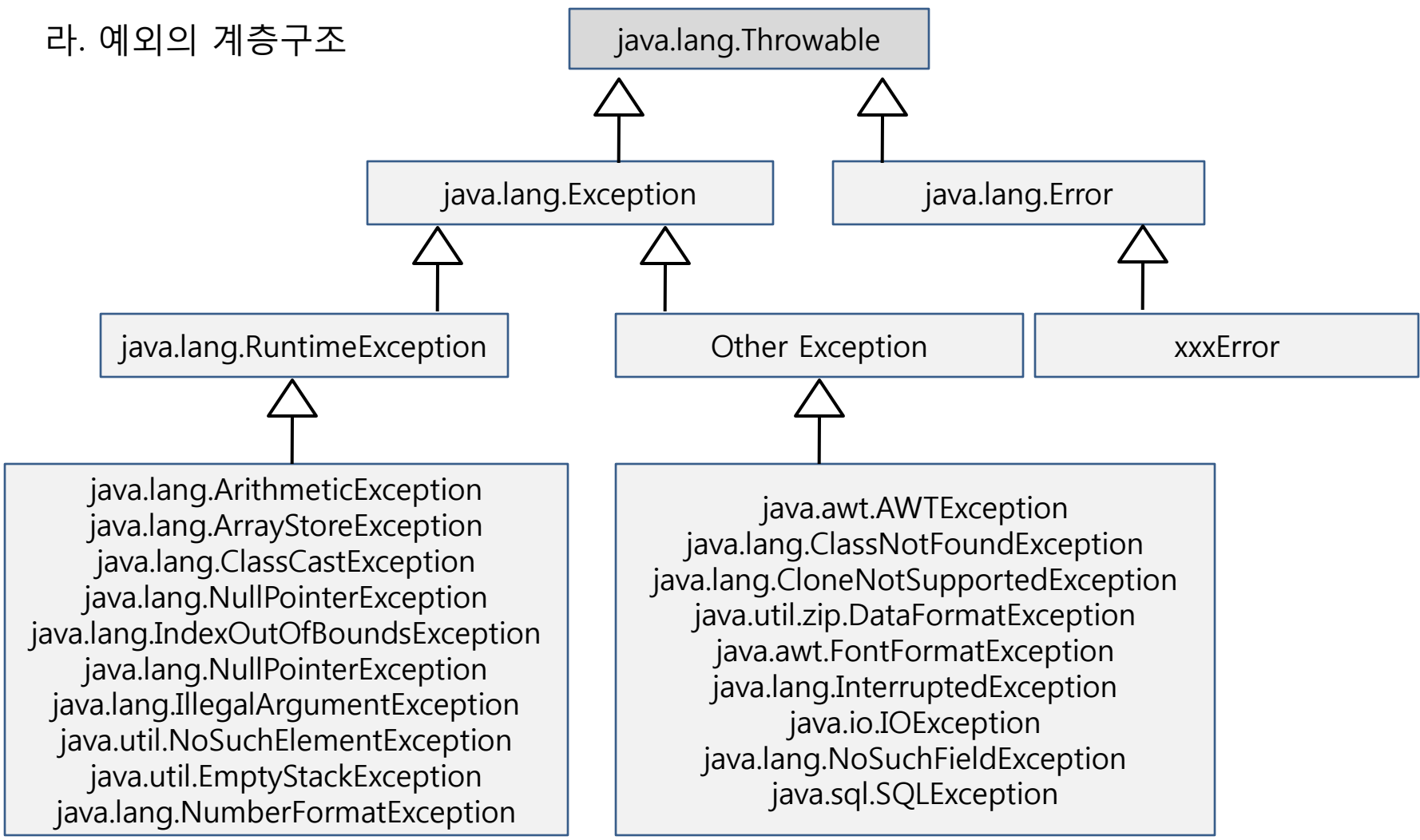
```
try{
 int a = Integer.parseInt("123");
 System.out.println(a);

}catch(NumberFormatException ee){
 System.out.println("int타입이 아닙니다");
}

finally{
 System.out.println("여기는 무조건 보인다");
}
```

8) 예외와 예외처리

라. 예외의 계층구조



## 8) 예외와 예외처리

마. 예외 발생시키기 및 처리하기 -throw, throws

- 자바에서는 프로그래머가 원하면 예외를 강제로 발생시킬 수도 있습니다.
- 자바에서 제공해 주지 않는 예외에 대해서는 자바 프로그램의 개발자가 새롭게 정의하여 사용할 수 있고, 필요에 따라 이 예외를 발생시킬 수 있습니다.

※ 주의

main 메서드에서도 throws 할 수 있으나 가급적 사용하지 않는 것이 좋다

## 8) 예외와 예외처리

마. 예외 발생시키기 및 처리하기 -throws

사용예)

```
public int sub(int a, int b) throws MyZeroException{
 if(b==0){
 throw new MyZeroException("0으로 나누지 말란 말야~~ ");
 }
 return (a/b);
}
```

## 8) 예외와 예외처리

마. 예외 발생시키기 및 처리하기 -throw

```
public static int divide(int a, int b) throws ArithmeticException {
 int c;

 try {
 System.out.println("divide: before a / b");
 return(a / b);
 }
 catch(ArithmeticException e) {
 System.out.println("divide: before throw e");
 throw e;
 }
}
```

# 어플리케이션 완성

## (Building Applications)

8. 예외처리  
(Exceptions)

9. 텍스트 기반 어플리케이션  
(Text-Based Applications)

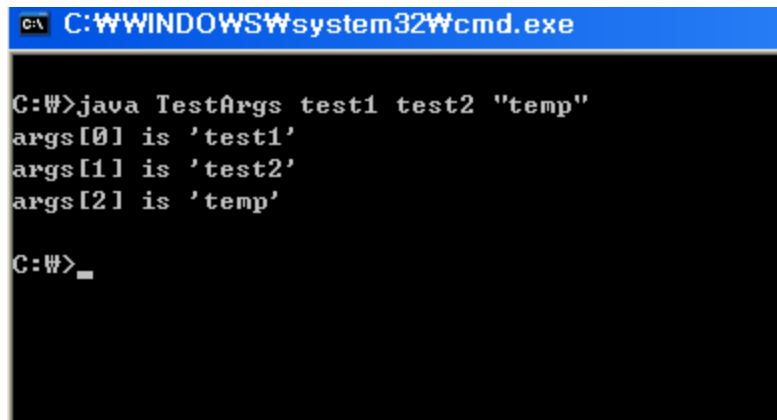


## 9) Text-Based Applications

- 명령프롬프트 상의 인자를 사용한 프로그램을 작성

```
public class TestArgs {
 public static void main(String[] args) {
 for(int i=0;i<args.length;i++){
 System.out.println("args["+i+"] is '"+ args[i]+"");
 }
 }
}
```

- 실행결과



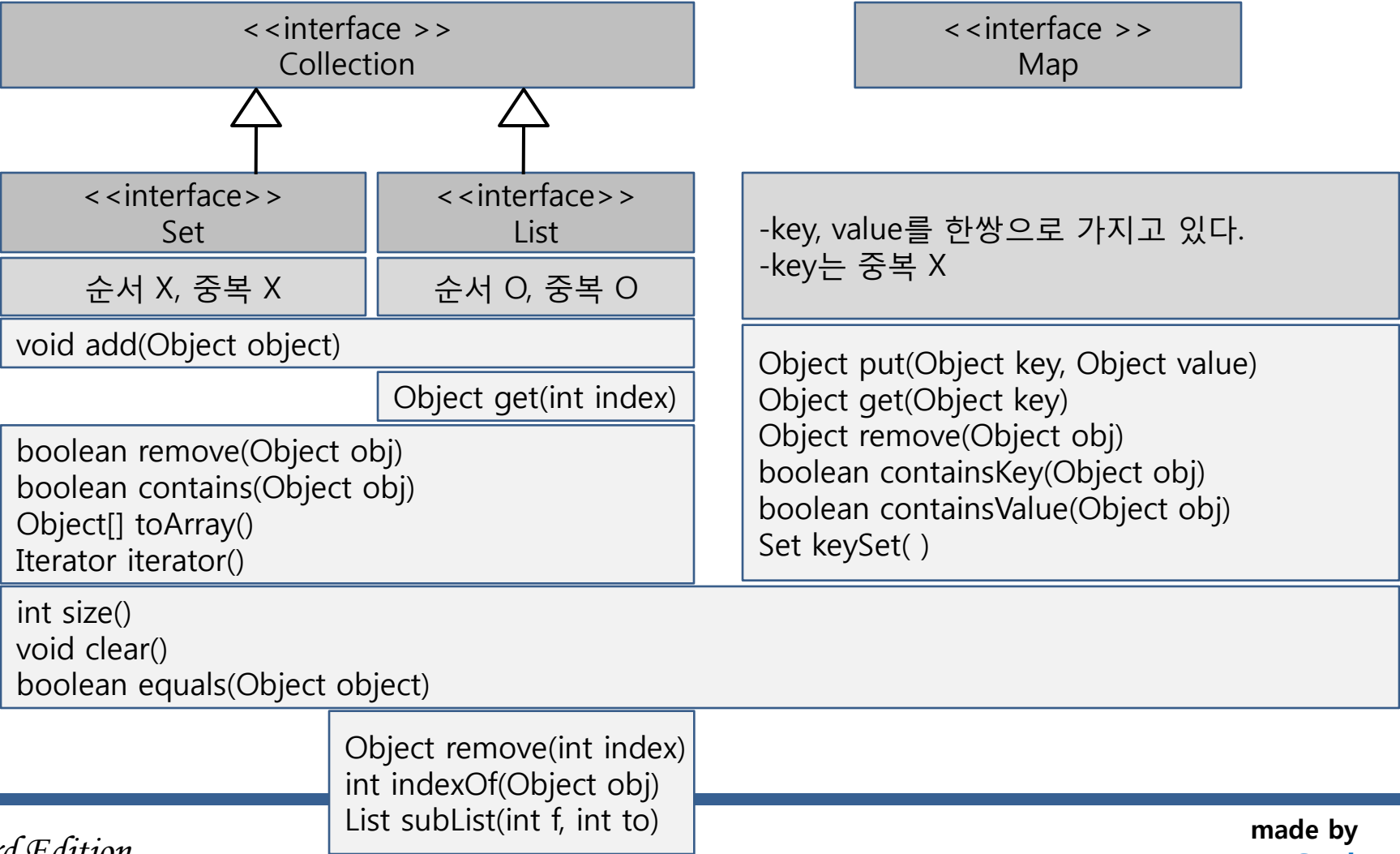
```
C:\WINDOWS\system32\cmd.exe

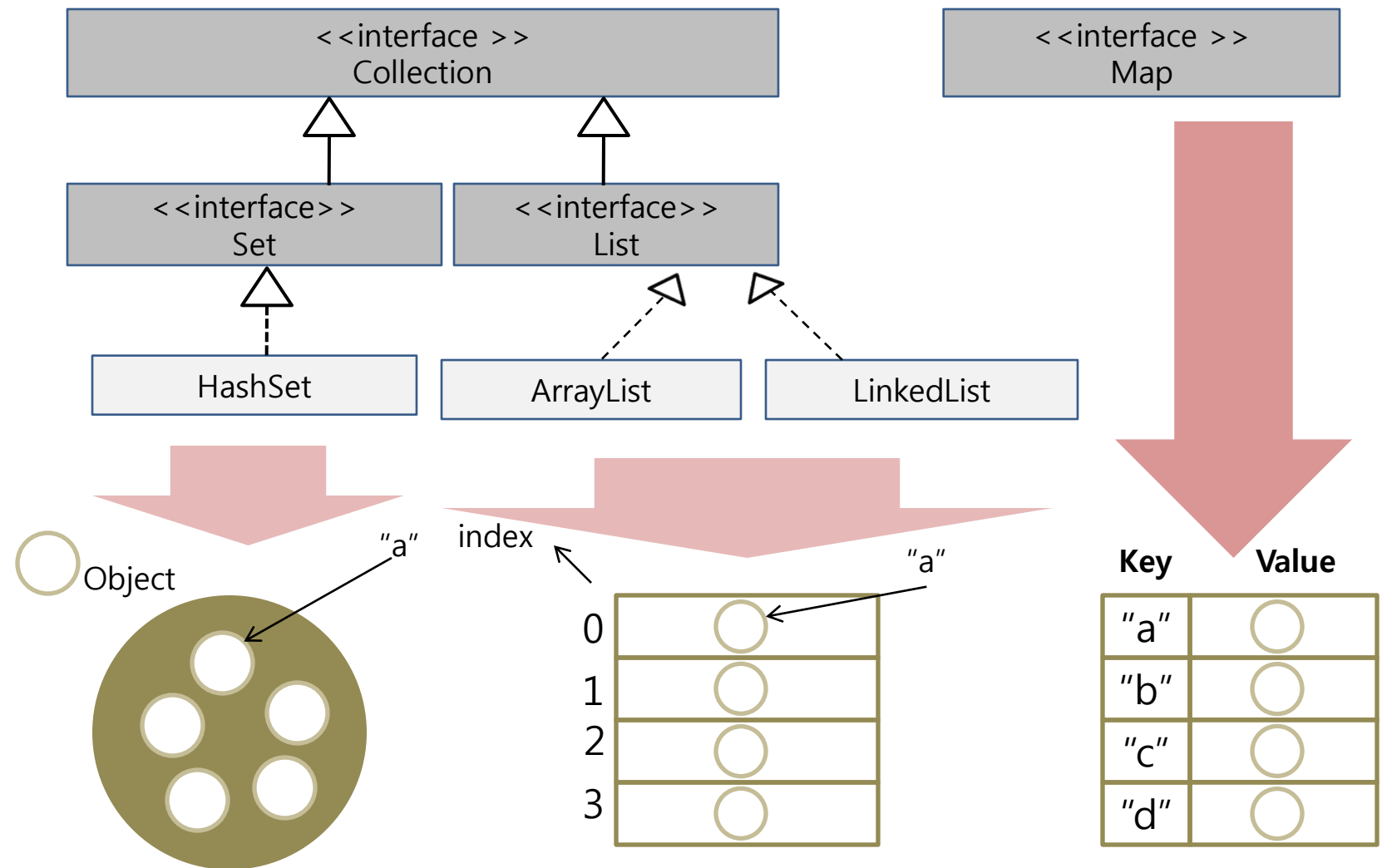
C:\W>java TestArgs test1 test2 "temp"
args[0] is 'test1'
args[1] is 'test2'
args[2] is 'temp'

C:\W>_
```

# 9) Text-Based Applications

- Java2 SDK의 Collection계열과 Map계열 비교





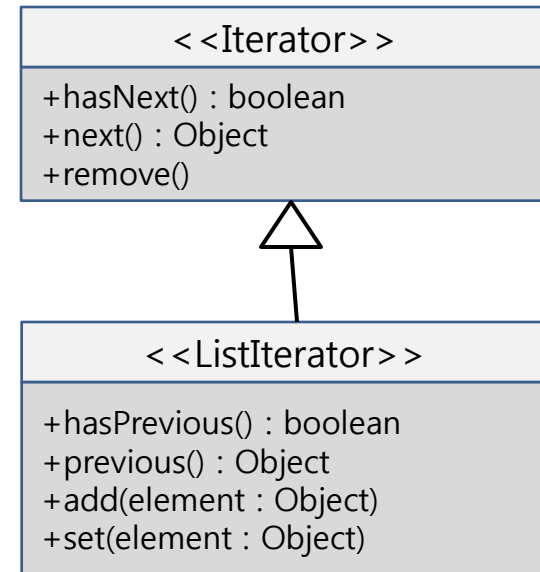
## 9) Text-Based Applications

### - Iterator

Collection객체 담겨있는 것을 꺼낼때 사용하는 process

```
List list = new ArrayList();

// add some elements
Iterator elements = list.iterator();
while(elements.hasNext()){
 System.out.println(elements.next());
}
```



## 9) Text-Based Applications

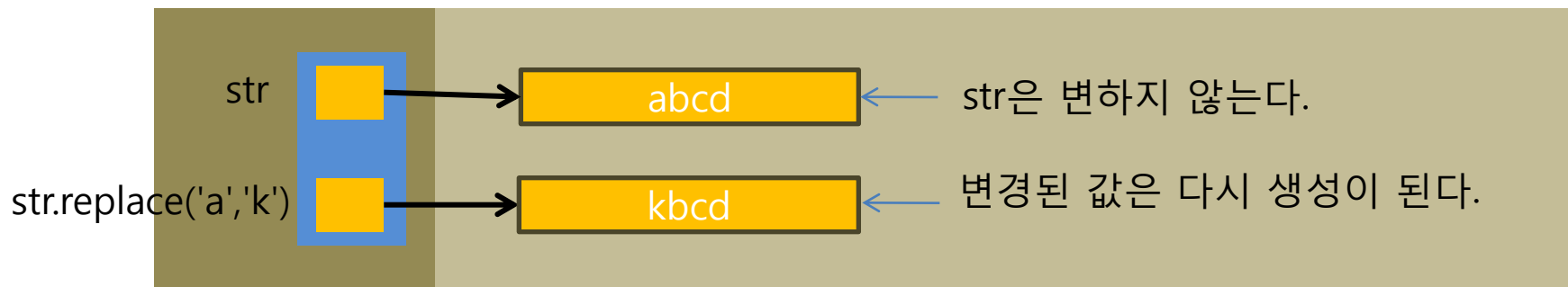
- String 클래스 - 연산자
  - immutable(값이 변하지 않는다.)
  - 새로운 문자열 만드는 연산자
    - : concat, replace, substring, toLowerCase, toUpperCase, and trim
  - 검색연산자
    - : endsWith, startsWith, indexOf, and lastIndexOf
  - 비교
    - : equals, equalsIgnoreCase, and compareTo
  - 그밖에
    - : charAt and length

## 9) Text-Based Applications

- String 클래스 - 그림설명

```
- String str = "abcd";
 str.replace('a','k');
 System.out.println(str);
```

결과 : abcd



## 9) Text-Based Applications

### - StringBuffer 클래스

- mutable(값이 변한다)

- StringBuffer()

- 빈 StringBuffer 생성

- StringBuffer(int capacity)

- StringBuffer를 초기화시 buffer를 capacity의 크기만큼 할당한다.

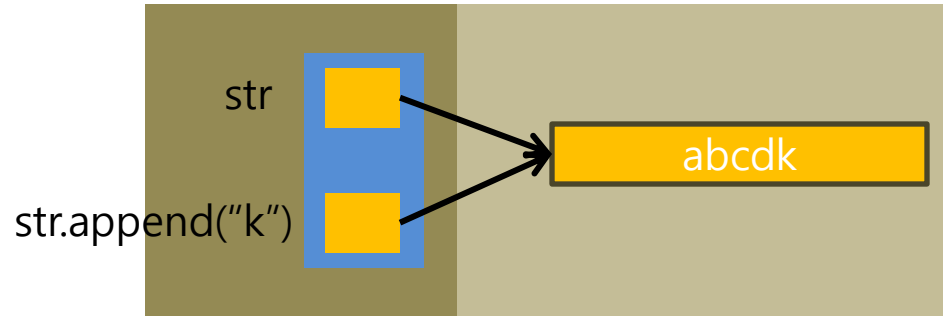
- StringBuffer(String initialString)

- StringBuffer 초기화시 initialString로 생성된다.

- ex)

```
StringBuffer str = new StringBuffer("abcd");
str.append("k");
System.out.println(str);
```

결과 : abcdk



## 9) Text-Based Applications

### - I/O관련 클래스 - 개념

Input / Output (입출력) : 입력과 출력에 관한 내용을 처리

Stream : 연속적인 data의 흐름

1. 한방향
2. 전용단위 byte
3. 다른 스트림 끼리는 섞이지 않는다

~stream : Byte

~er : Character

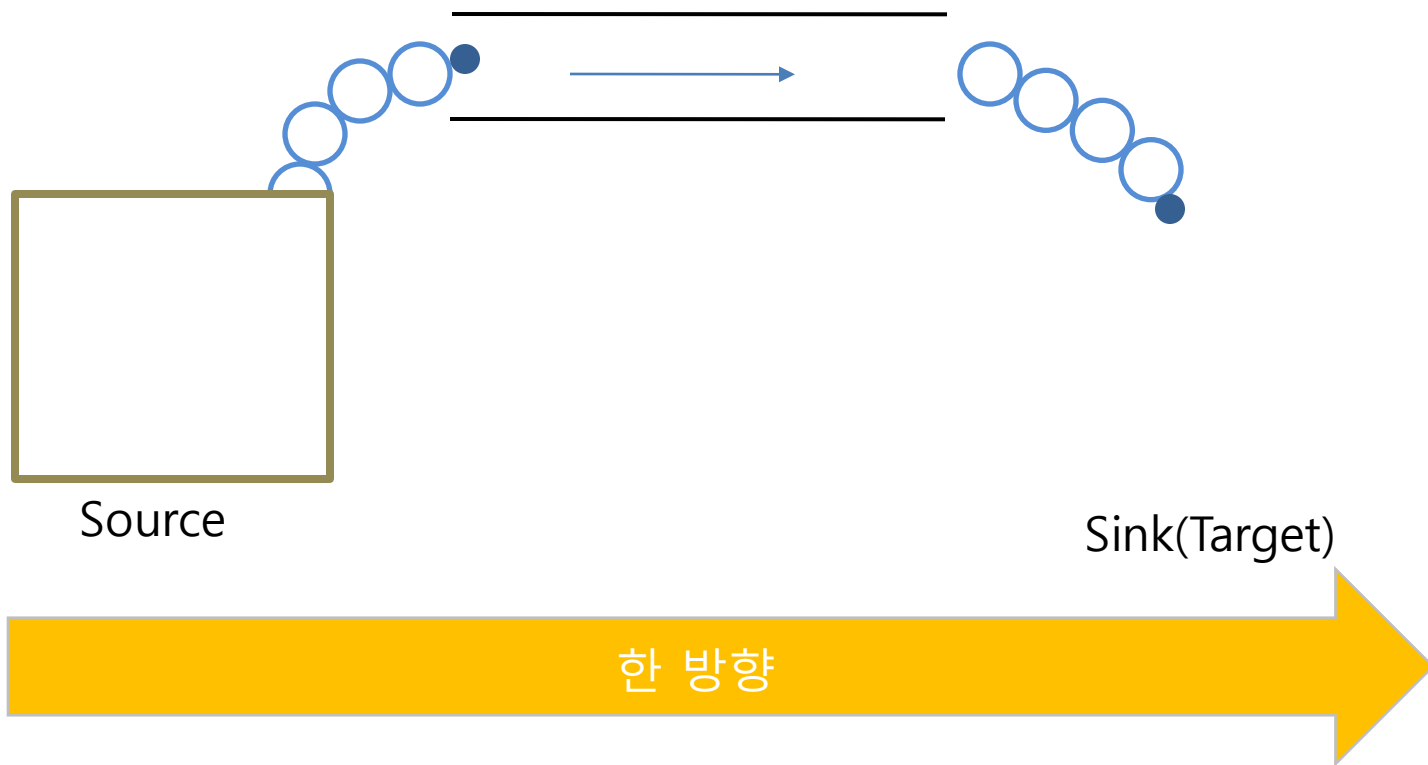
~node : 수도꼭지

~filter : 호스( 성능향상 )

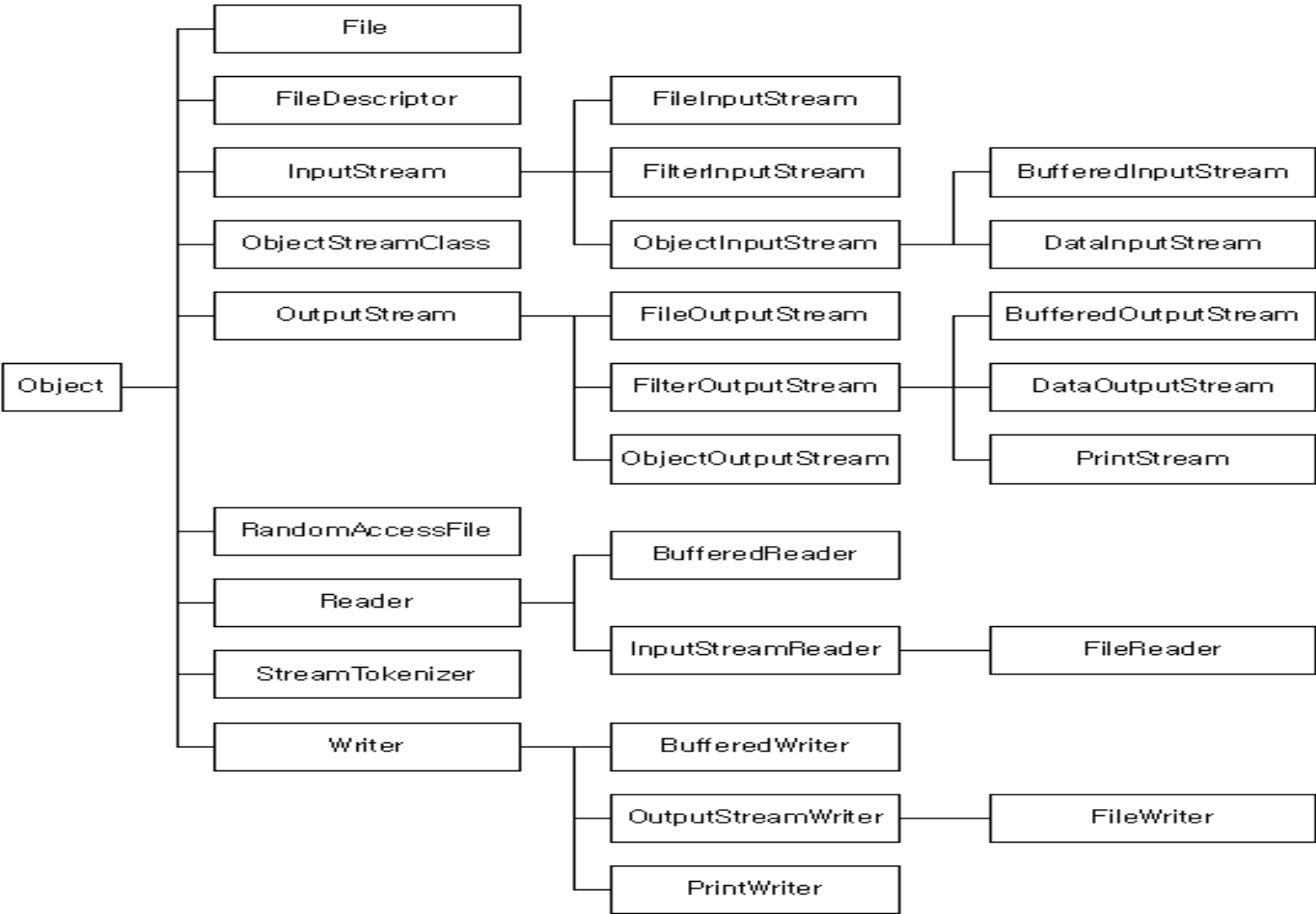


## 9) Text-Based Applications

- I/O 관련 클래스 - 개념[그림]



9) Text-Based Applications – I/O 계층구조



# 사용자 그래픽 인터페이스

## (Developing Graphical User Interfaces)

### 10. Java GUI 만들기

(Building Java GUIs)

### 11. GUI 이벤트 핸들링

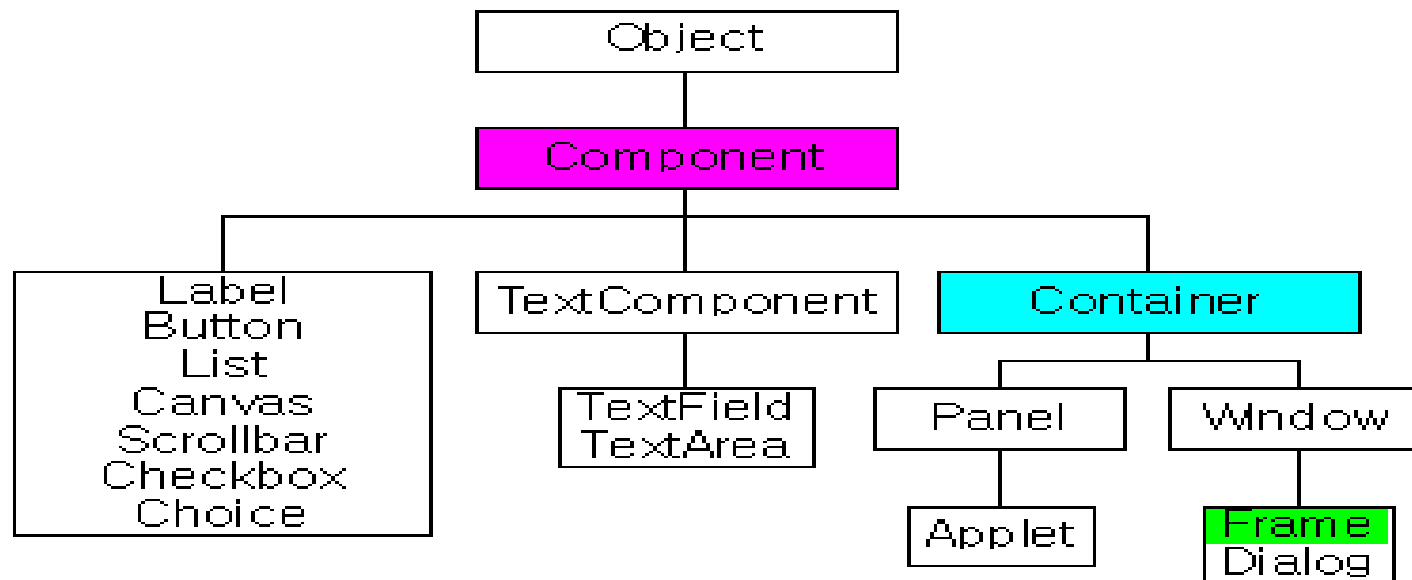
(GUI Event Handling)

### 12. GUI 기반 어플리케이션

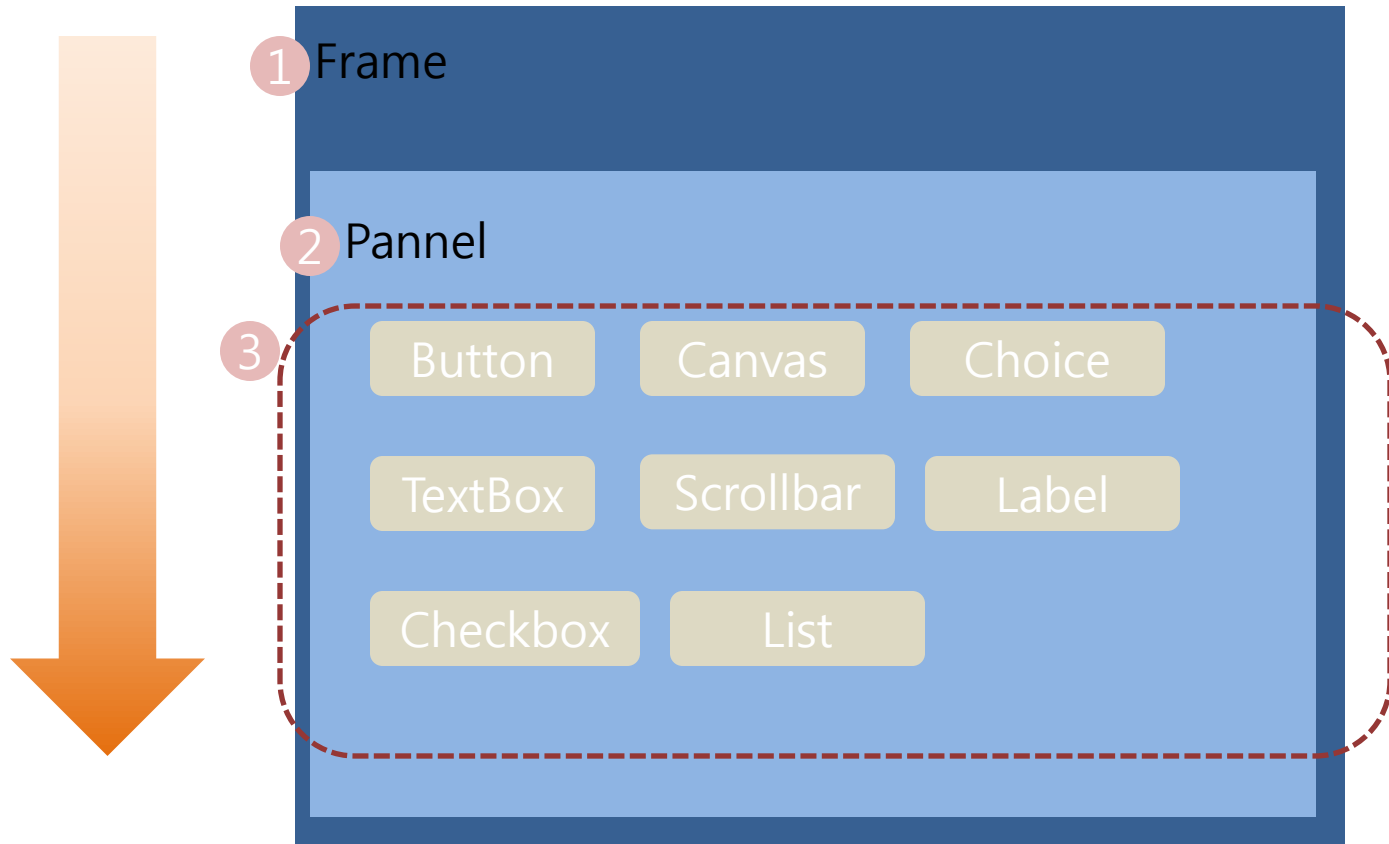
(GUI-Based Applications)

## 10-1) AWT

- 자바에서는 그래픽 프로그램을 하기 위한 클래스 라이브러리를 제공해 주는데, 이것을 우리는 AWT(Abstract Window Toolkit)라고 부르고 있습니다.
- 그래픽 프로그램을 작성하기 위한 컴포넌트 라이브러리를 AWT라고 생각하면 됩니다.
- AWT의 가장 큰 장점은 **한 번 구축되면 어떠한 환경에서도 동작한다**는 것입니다.



## 10-2) AWT - 화면 구성 순서



### 10-3) AWT - 창(Frame)

- Frame 만들기(창)

Awt1.java

```
import java.awt.*; // x1

public class Awt1{
 public static void main(String[] args){
 Frame f=new Frame("AWT의 Frame"); // 프레임 객체(f)를 만든다.
 f.setSize(200,100); // f의 크기를 결정한다.
 f.setVisible(true); // f를 보이게 한다.
 }
}
```

## 10-4) AWT - Component 클래스의 유용한 메소드

```
public int getX(), public int getY()
```

- 컴포넌트의 x 또는 y 좌표를 반환한다.

```
public int getWidth(), public int getHeight()
```

- 컴포넌트의 너비 또는 높이를 반환한다.

```
public void setLocation(int x, int y)
```

- 컴포넌트의 위치를 (x, y)로 이동시킨다.

```
public void setSize(int width, int height)
```

- 컴포넌트의 너비를 width로, 높이를 height로 바꾼다.

```
public void setBounds(int x, int y, int width, int height)
```

- 위치와 크기를 동시에 바꾼다.

```
public void setLocation(xid setVisible(boolean b)
```

-b가 true이면 컴포넌트를 화면상에서 보이게 한다.

-b가 false이면 컴포넌트를 감춘다.

## 10-5) AWT - component 추가하기

- Label a = new Label("레이블 테스트");
- Button b = new Button("버튼 테스트");
- TextField tf = new TextField(20);
- 
- 
-



## 10-6) AWT - Layout Manager

Container 계열의 컴포넌트들은 다른 컴포넌트들을 포함할 수 있습니다.

컴포넌트를 포함할 때 어떻게 배치할 것인지에 대한 방법을 레이아웃(Layout)이라고 합니다. 보통 레이아웃을 지정하는 클래스를 레이아웃 관리자(배치관리자)라고 하며 다음과 같이 5가지 종류가 있습니다.

- ◆ FlowLayout : 삽입되는 순서대로 보이는 것
- ◆ BorderLayout : 동.서.남.북.중앙의 다섯 곳의 위치에 삽입
- ◆ GridLayout : Grid 형식으로 분할한 뒤 분할된 곳에 순서대로 컴포넌트를 삽입하게 됩니다.
- ◆ CardLayout : 카드처럼 컴포넌트를 겹치서 배치하며, 제일 앞에 있는 컴포넌트만 보이게 하는 방식입니다.
- ◆ GridBagLayout : 여러 칸에 하나의 컴포넌트를 삽입하는 방식

## 10-7) AWT - FlowLayout을 이용한 컴포넌트 배치

```
import java.awt.*;
public class FlowFrame extends Frame {
 private Button b1 = new Button("1번버튼");
 private Button b2 = new Button("2번버튼");
 private Button b3 = new Button("3번버튼");
 private Button b4 = new Button("4번버튼");
 private Button b5 = new Button("5번버튼");
 private Choice ch = new Choice();

 //멤버변수 초기화 영역
 {
 ch.add("Green");
 ch.add("Red");
 ch.add("Blue");
 }
 public FlowFrame() {
 this.setLayout(new FlowLayout());
 this.add(b1);
 this.add(b2);
 this.add(b3);
 this.add(b4);
 this.add(b5);
 this.add(ch);
 }
} //end of FlowFrame class
```

## 10-8) AWT - BorderLayout을 이용한 컴포넌트 배치

```
import java.awt.*;
class BorderFrame extends Frame {
 private Button b1 = new Button("1번버튼");
 private Button b2 = new Button("2번버튼");
 private Button b3 = new Button("3번버튼");
 private Button b4 = new Button("4번버튼");
 private Button b5 = new Button("5번버튼");
 public BorderFrame() {
 //this.setLayout(new BorderLayout(10,5));
 this.add("North", b1);
 this.add("West", b2);
 this.add("Center", b3);
 this.add("East", b4);
 this.add("South", b5);
 }
} //end of BorderFrame class
```

## 10-9) AWT - GridLayout을 이용한 컴포넌트 배치

```
public class GridFrame extends Frame {
 //Button형 b1, b2, b3, b4, b5가 존재한다고 가정
 public GridFrame() {
 this.setLayout(new GridLayout(3,2));
 this.add(b1);
 this.add(b2);
 this.add(b3);
 this.add(b4);
 this.add(b5);
 }
}
```

## 10-10) AWT - CardLayout 을 이용한 컴포넌트 배치

```
public class CardFrame extends Frame{
 private CardLayout card = new CardLayout();
 //레이블 b1, b2, b3, b4, b5가 존재한다고 가정
 public CardFrame() {
 this.setLayout(card);
 this.add(b1,"card-b1");
 this.add(b2,"card-b2");
 this.add(b3,"card-b3");
 this.add(b4,"card-b4");
 this.add(b5,"card-b5");
 }
}
```

## 10-11) AWT - GridBagLayout을 이용한 컴포넌트 배치

```
import java.awt.*;

public class GridBagFrame extends Frame {
 //삽입할 컴포넌트 생성
 private Label lblReceive = new Label("받는 사람: ", Label.RIGHT);
 //GridBagLayout 생성
 private GridBagLayout gBag = new GridBagLayout();
 public GridBagFrame() {
 this.setLayout(gBag);
 //GridBagLayout에 삽입하는 과정
 this.insert(lblReceive, 0, 0, 1, 1);
 this.pack();
 }

 private void insert(Component cmpt, int x, int y, int w, int h) {
 GridBagConstraints gbc = new GridBagConstraints();
 gbc.fill = GridBagConstraints.BOTH;
 gbc.gridx = x;
 gbc.gridy = y;
 gbc.gridwidth = w;
 gbc.gridheight = h;
 this.gBag.setConstraints(cmpt, gbc);
 this.add(cmpt);
 }
} //end of GridBagFrame class
```

# 사용자 그래픽 인터페이스

## (Developing Graphical User Interfaces)

10. Java GUI 만들기  
(Building Java GUIs)

11. GUI 이벤트 핸들링  
(GUI Event Handling)

12. GUI 기반 어플리케이션  
(GUI-Based Applications)

# 사용자 그래픽 인터페이스

## (Developing Graphical User Interfaces)

10. Java GUI 만들기  
(Building Java GUIs)

11. GUI 이벤트 핸들링  
(GUI Event Handling)

12. GUI 기반 어플리케이션  
(GUI-Based Applications)



# 진화된 자바 프로그래밍

## (Advanced Java Programming)

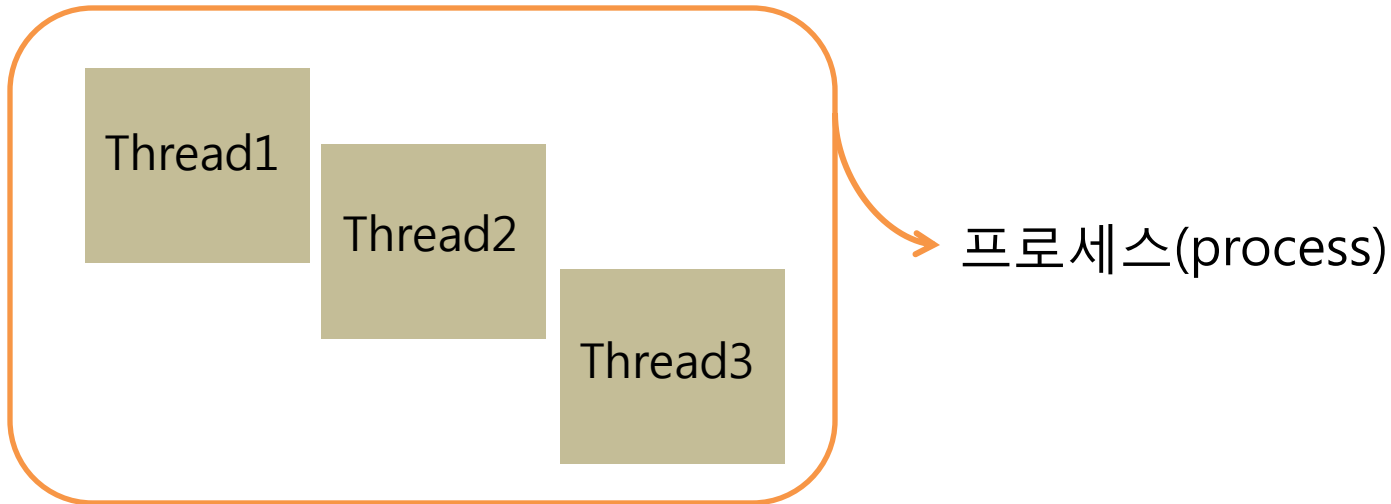
13. 스레드  
(Threads)

14. 진화된 I/O 스트림  
(Advanced I/O Streams)

15. 네트워킹  
(Networking)

### 13-1) 스레드(Threads)란?

- \* 프로세스(process) : 실행되고 있는 프로그램
- \* 스레드(Thread) : 독립된 작은 실행단위



## 13-1) 쓰레드(Threads) 정리-1

1. 여러 작업을 나눠서 할 때 사용
2. 작은 실행 단위 (프로세스위에서 동작한다)
3. 문맥교환 (Switching Context) : 동시동작을 흉내냄
4. 같은 객체를 여러 thread 에서 사용할 때는 동기화를 위해 lock 을 걸어줘야 한다
5. multi browsing
6. 경쟁방식
  - Thread Manager / JVM -> Deamon

## 13-1) 쓰레드(Threads) 정리-2

### 7. << Runnable >> run() 구현

#### A. 구현 방법

```
MyRun { run(); }
Thread Manager => run()
Thread t1=new Thread (my);
t1.start();
```

#### B. Thread 를 Extends 해서 run() 을 오버라이딩 t1.start();

#### C. Thread 를 start 해줘야 ready 상태가 되고, t1, t2, t3 ... running 상태가 된다 !! manager 가 스케줄링을 하면서 작업 완료가 되면 dead() 상태가 된다

## 13-1) 쓰레드(Threads) 정리-3

D. yield() / sleep()

E. 하나의 자원을 여러 개의 thread 에서 작업할 때 동기화 문제가 생긴다  
Synchronized (deadlock 이 걸릴수 있다 => wait(), notifyall() 을 이용 )

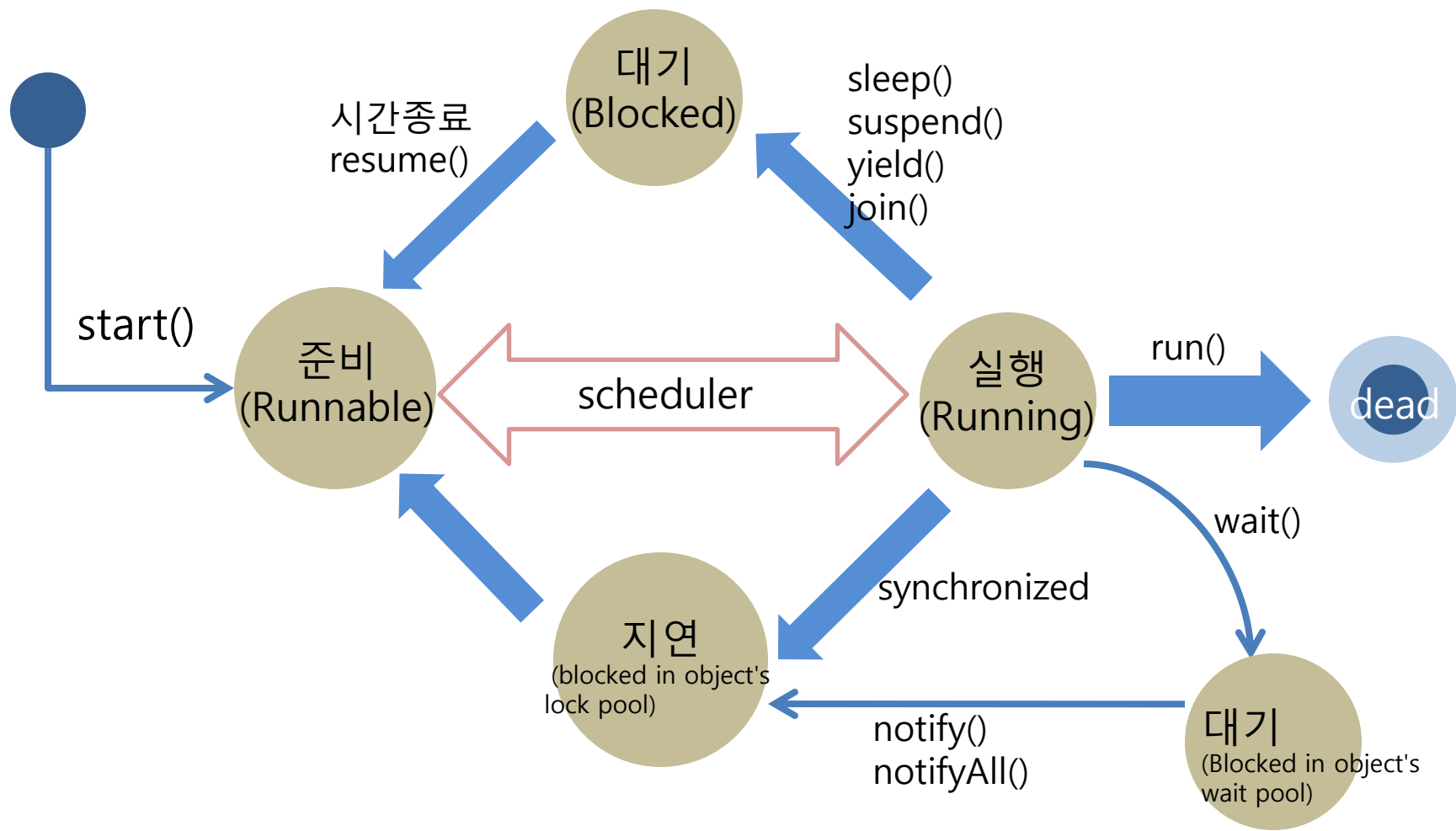
F. Deadlock 걸리는 상태 감시

1. 세마포어
2. key
3. Critical Zone

G. Start()Dead 상태

H. Wait() => notify() => notifyall()

13-2) 쓰레드(Threads)의 life cycle



# 진화된 자바 프로그래밍

## (Advanced Java Programming)

13. 스레드  
(Threads)

14. 진화된 I/O 스트림  
(Advanced I/O Streams)

15. 네트워킹  
(Networking)

## 14-1) I/O관련 클래스 - 개념

Input / Output (입출력) : 입력과 출력에 관한 내용을 처리

Stream : 연속적인 data의 흐름

1. 한방향
2. 전용단위 byte
3. 다른 스트림 끼리는 섞이지 않는다

~stream : Byte

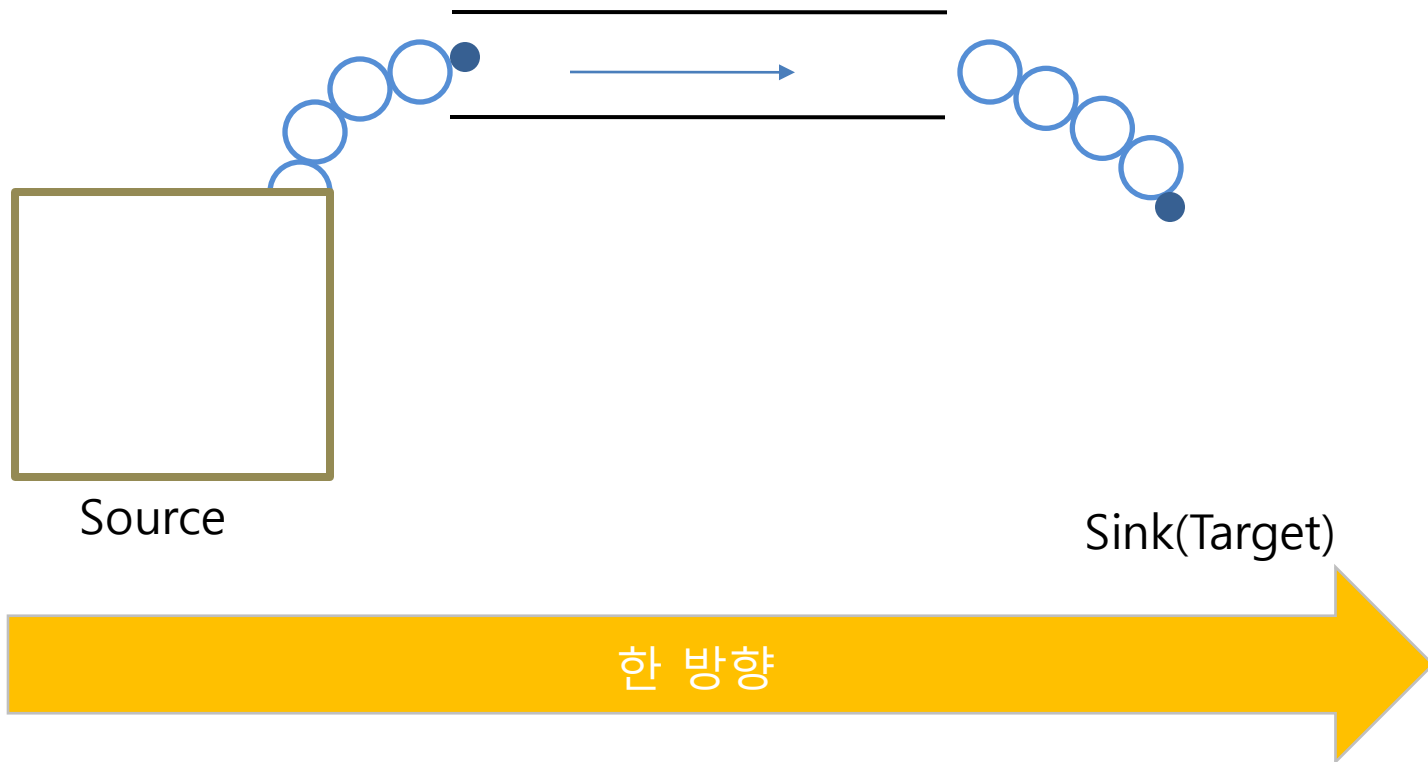
~er : Character

~node : 수도꼭지

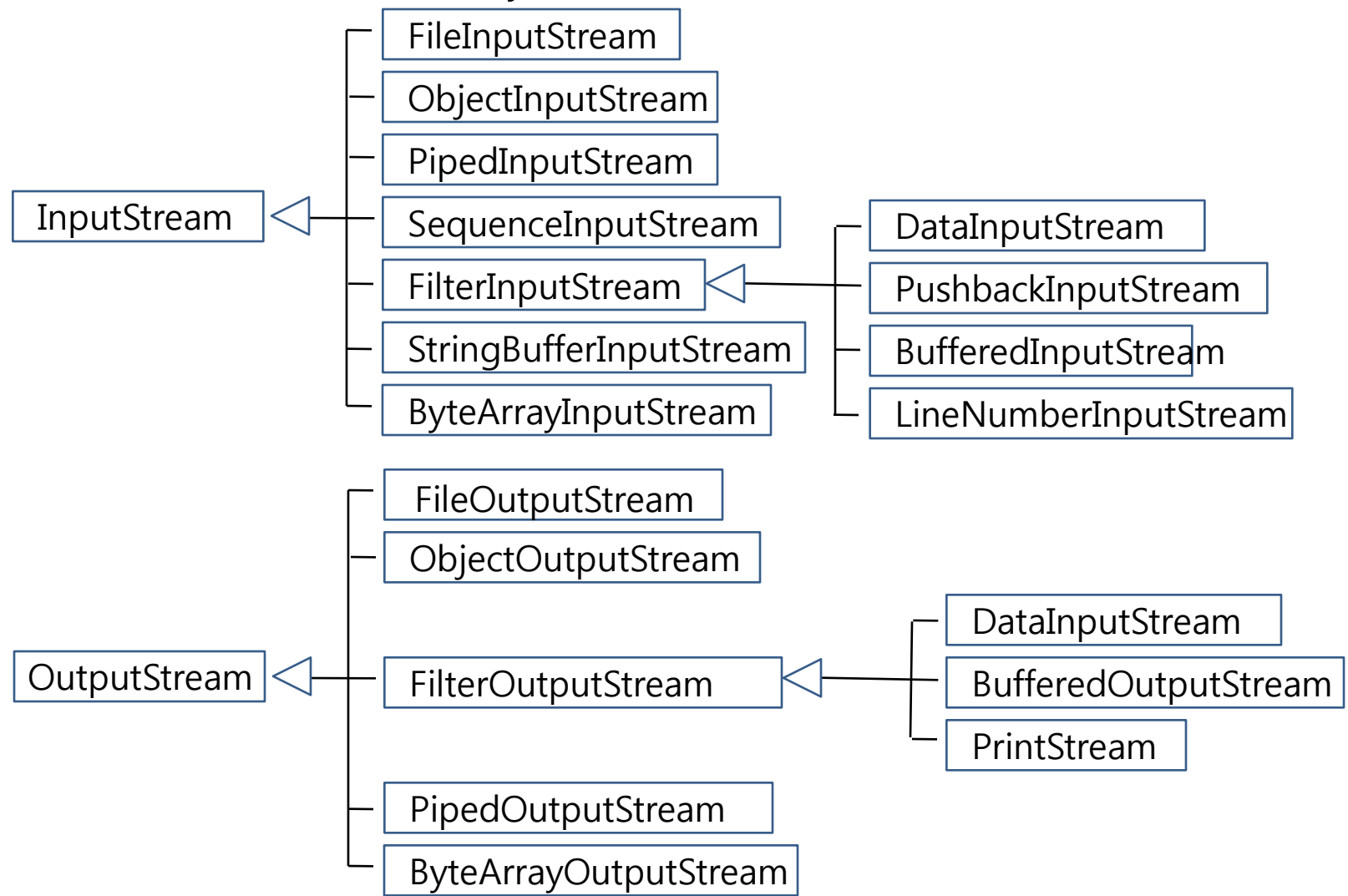
~filter : 호스( 성능향상 )



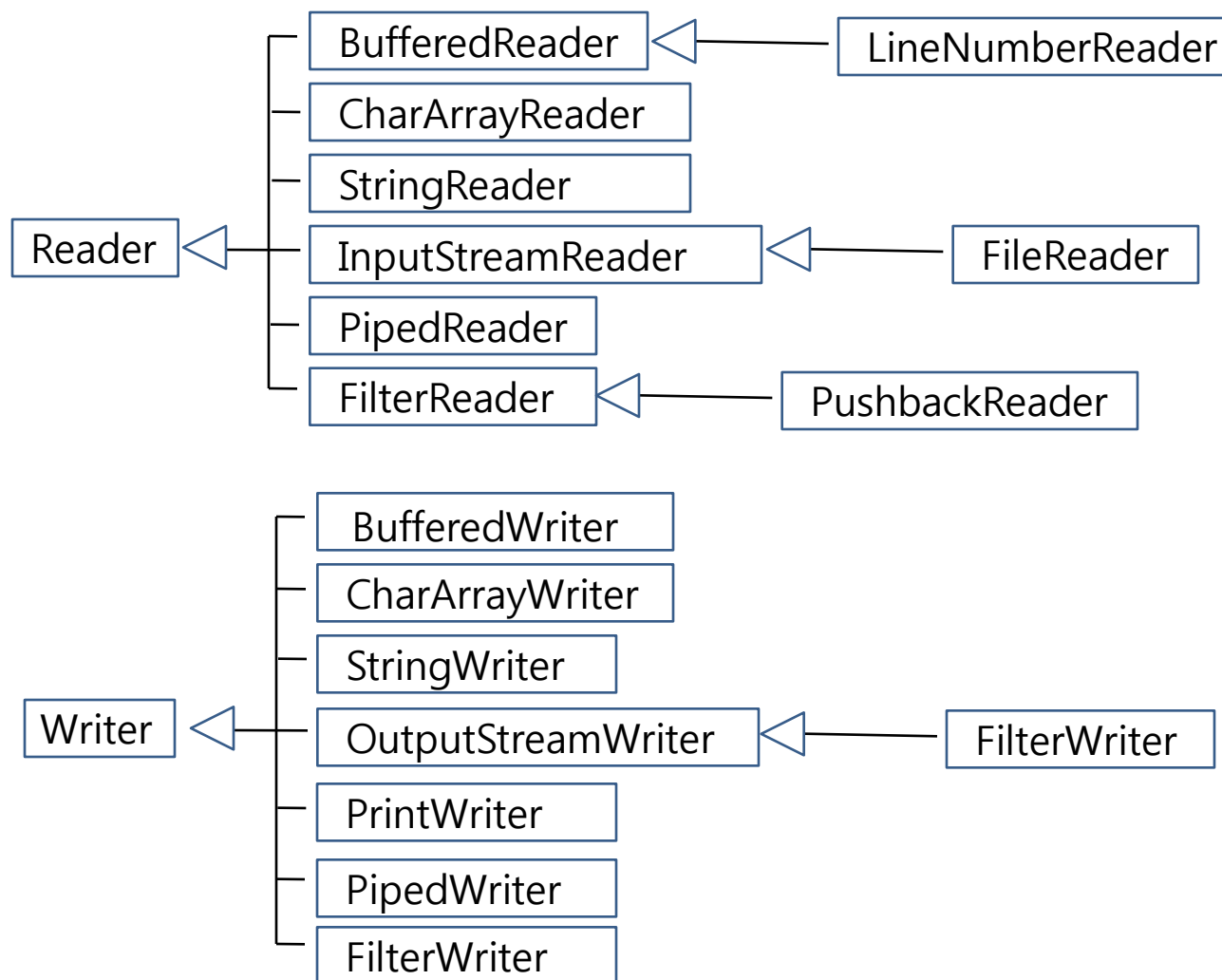
- I/O 관련 클래스 - 개념[그림]



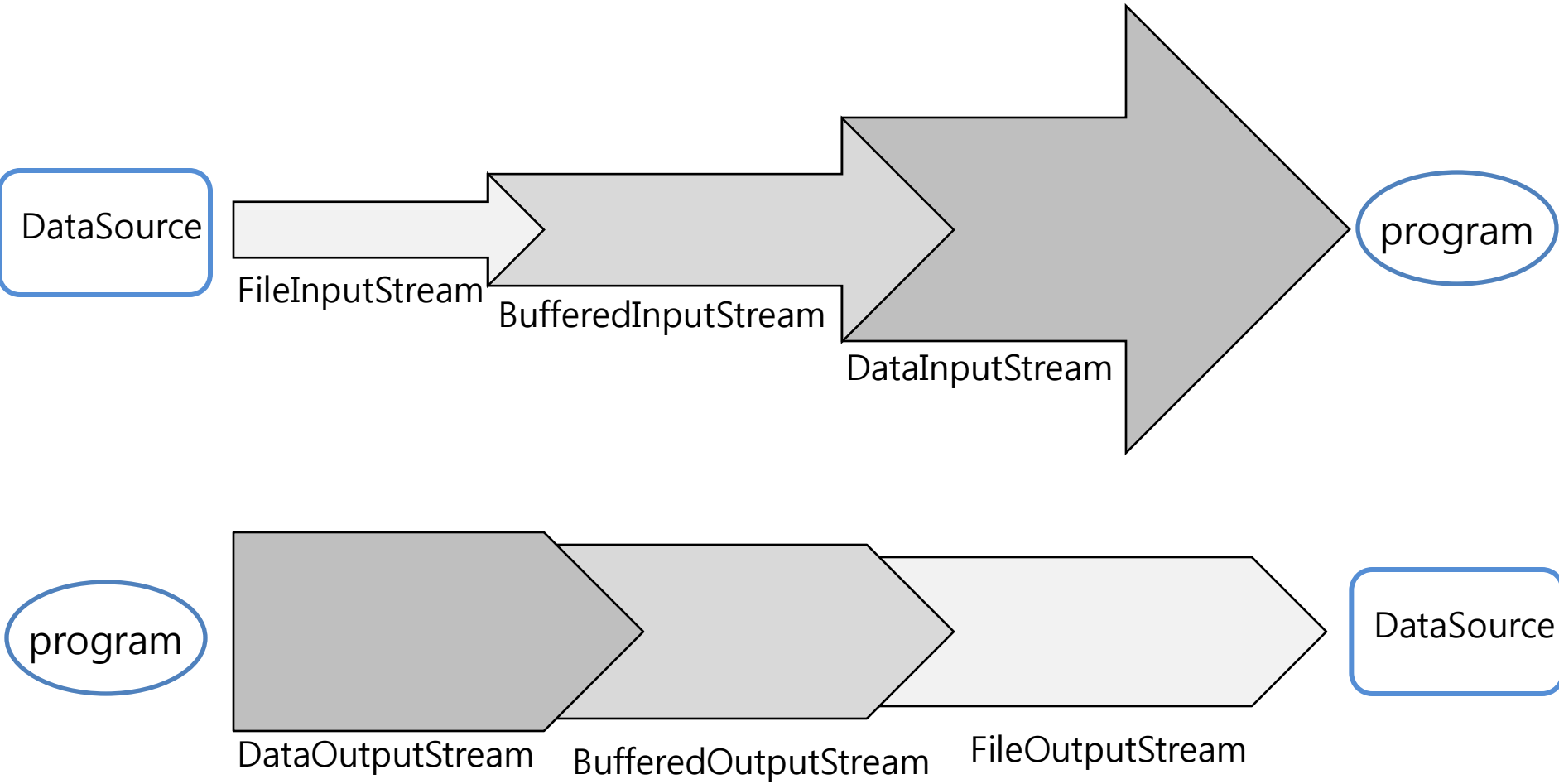
14-2) I/O 계층구조 - Basic Byte



## 14-2) I/O 계층구조 - Character Stream Classes



14-2) I/O Stream Chaining



# 진화된 자바 프로그래밍

## (Advanced Java Programming)

13. 스레드  
(Threads)

14. 진화된 I/O 스트림  
(Advanced I/O Streams)

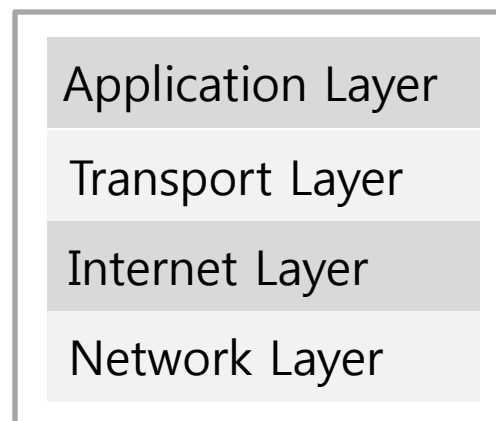
15. 네트워킹  
(Networking)

## 15-1) TCP/IP

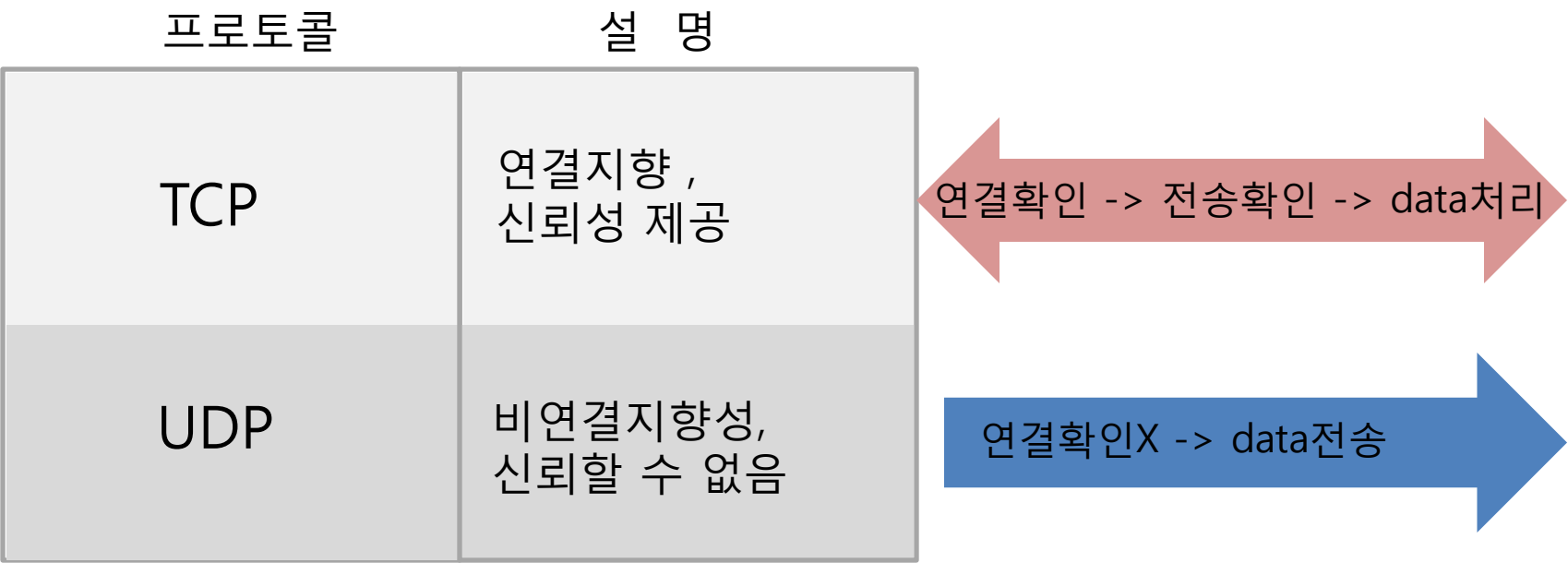
OSI 7계층



TCP/IP model



15-2) TCP 와 UDP



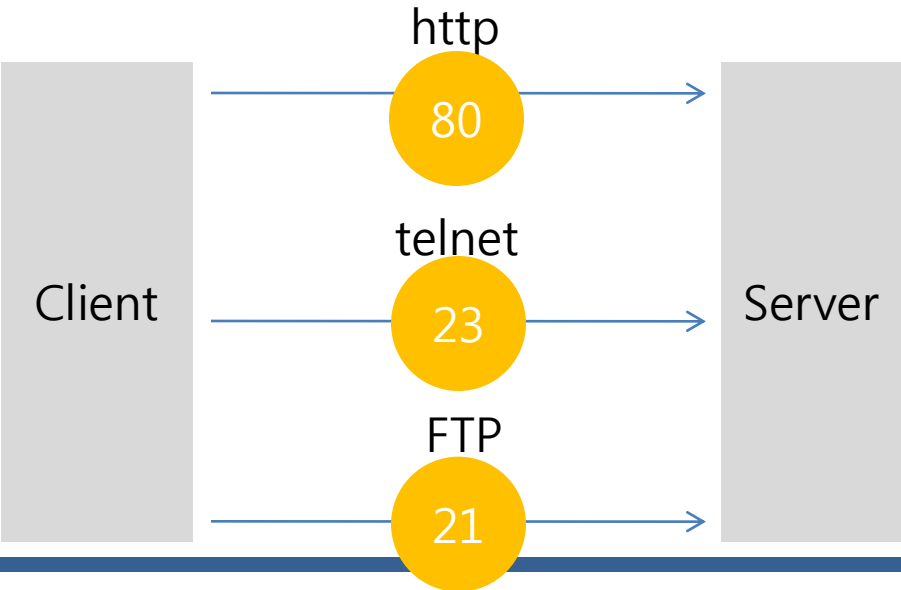
15-3) IP주소, port번호

IP 주소 :

- 컴퓨터가 인식하는 IP번호에 해당하는 사용자가 인식할 수 있는 고유한 주소
- ex) [www.hankyung.com](http://www.hankyung.com) : domain name
- 203.249.33.7 : ip address

Port 번호 :

- 해당서비스를 사용하기 위해 사용하는 길



| 프로토콜     | 포트  | TCP/UDP |
|----------|-----|---------|
| echo     | 7   | TCP/UDP |
| discard  | 9   | TCP/UDP |
| daytime  | 13  | TCP/UDP |
| ftp-data | 20  | TCP     |
| FTP      | 21  | TCP     |
| Telnet   | 23  | TCP     |
| SMTP     | 25  | TCP     |
| time     | 37  | TCP/UDP |
| whois    | 43  | TCP     |
| finger   | 79  | TCP     |
| HTTP     | 80  | TCP     |
| POP3     | 110 | TCP     |
| NNTP     | 119 | TCP     |



# [0] JAVA 환경설정

# [0] JAVA 환경설정

## 1. JAVA = Execute Files + Library Files

1) Execute File : \*.exe, C:\j2sdk1.4.2\bin에 위치

2) Library File : \*.class, \*.jar

(j2sdk에서는 rt.jar 파일이  
자주 쓰이며 위치는 C:\j2sdk\jre\lib)

# [0] JAVA 환경설정

\* C:\wj2sdk1.4.2\lib\ext 폴더

⇒ \*.jar 파일을 별도의 환경설정 없이  
사용하고자 할 때 ...\ext 폴더에 저장.

<장점> classpath를 별도로 설정할 필요 없음.

<단점> Memory 소요.

cf) [IT용어] [\*\*JAR\*\* \[ java archiver \]](#)

클래스 파일의 효율적인 배포를 위해

여러 클래스 파일들을 하나로 묶어 단일의 파일로 만드는 포맷.

로컬상에서 편리한 관리는 물론 자바 Java 프로그램 실행 중에  
원격지에서 하이퍼텍스트 전송 규약 HTTP 등을 통해 download  
되어 바로 사용이 가능하다.

# [0] JAVA 환경설정

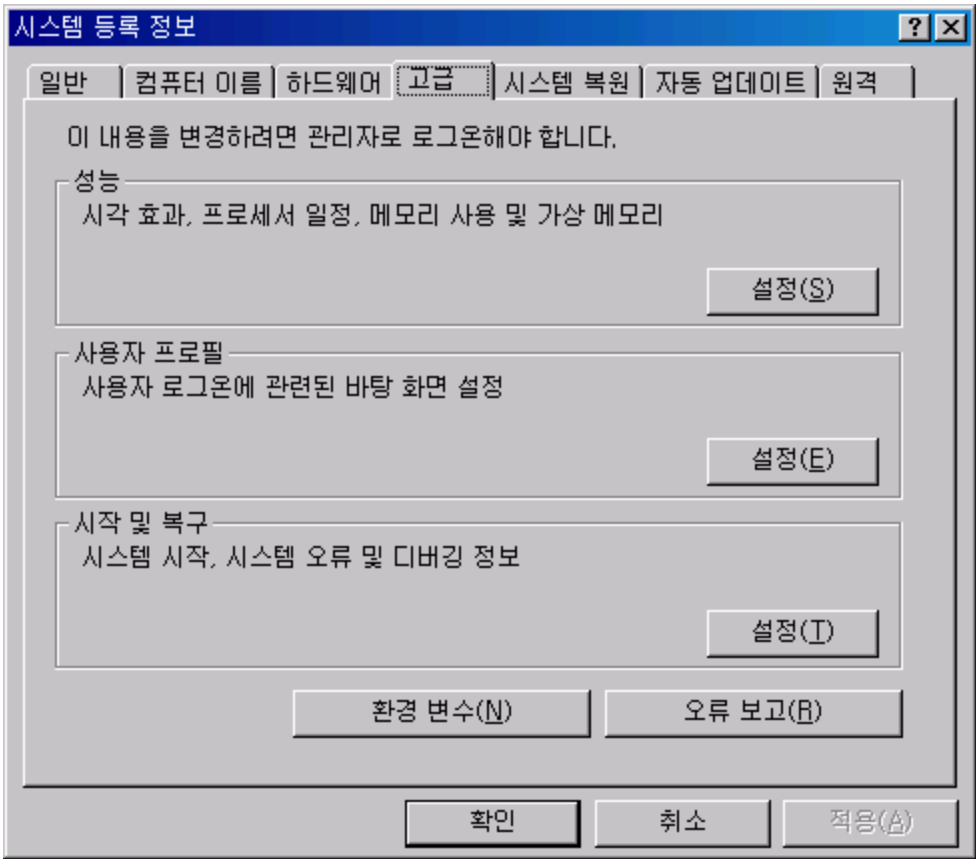
[환경설정 순서 (그림)]

1. 바탕화면에 있는 "내 컴퓨터"에서 마우스 오른쪽 클릭 후 "속성" 선택.



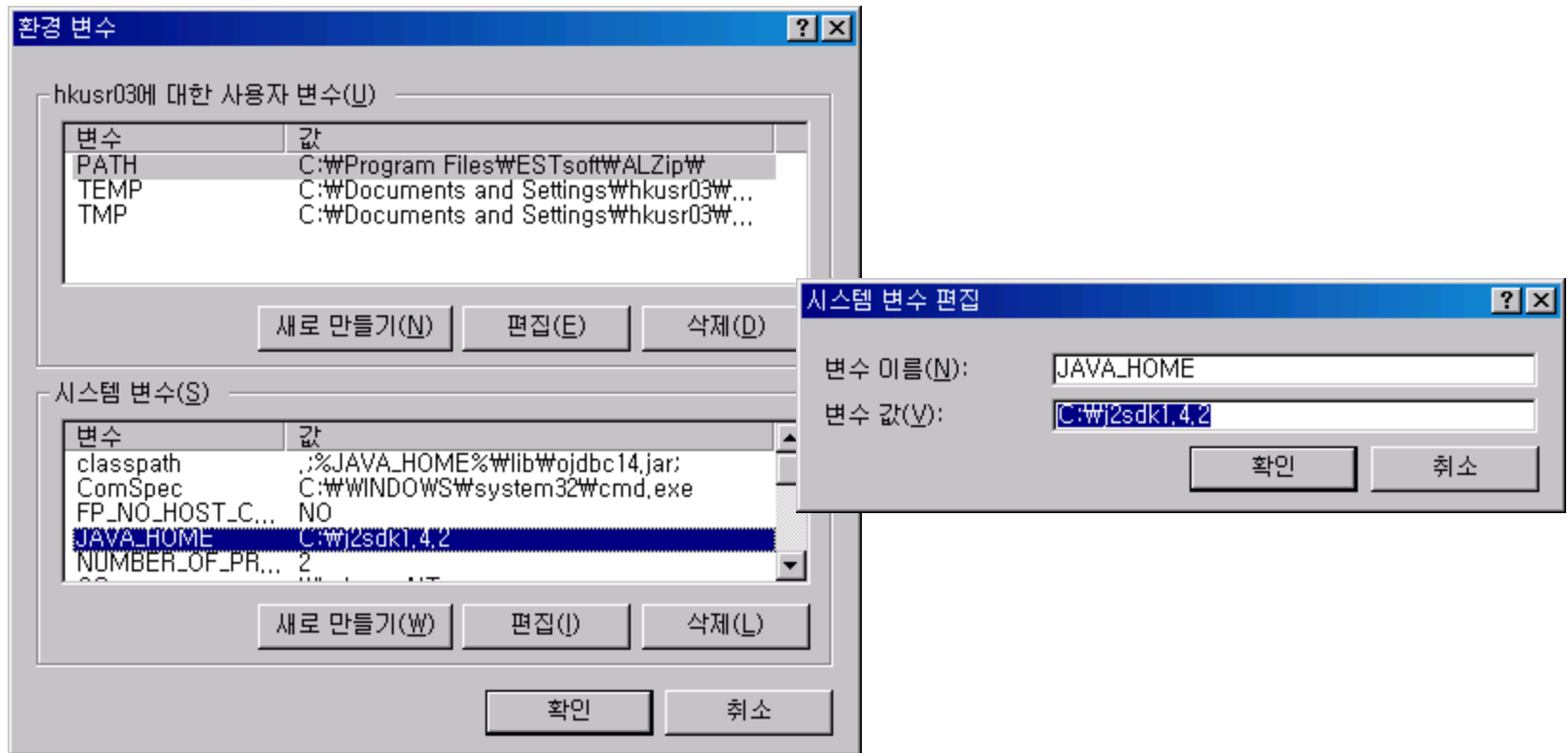
# [0] JAVA 환경설정

2. 시스템 등록 정보에서 “고급” => “환경 변수”를 선택.



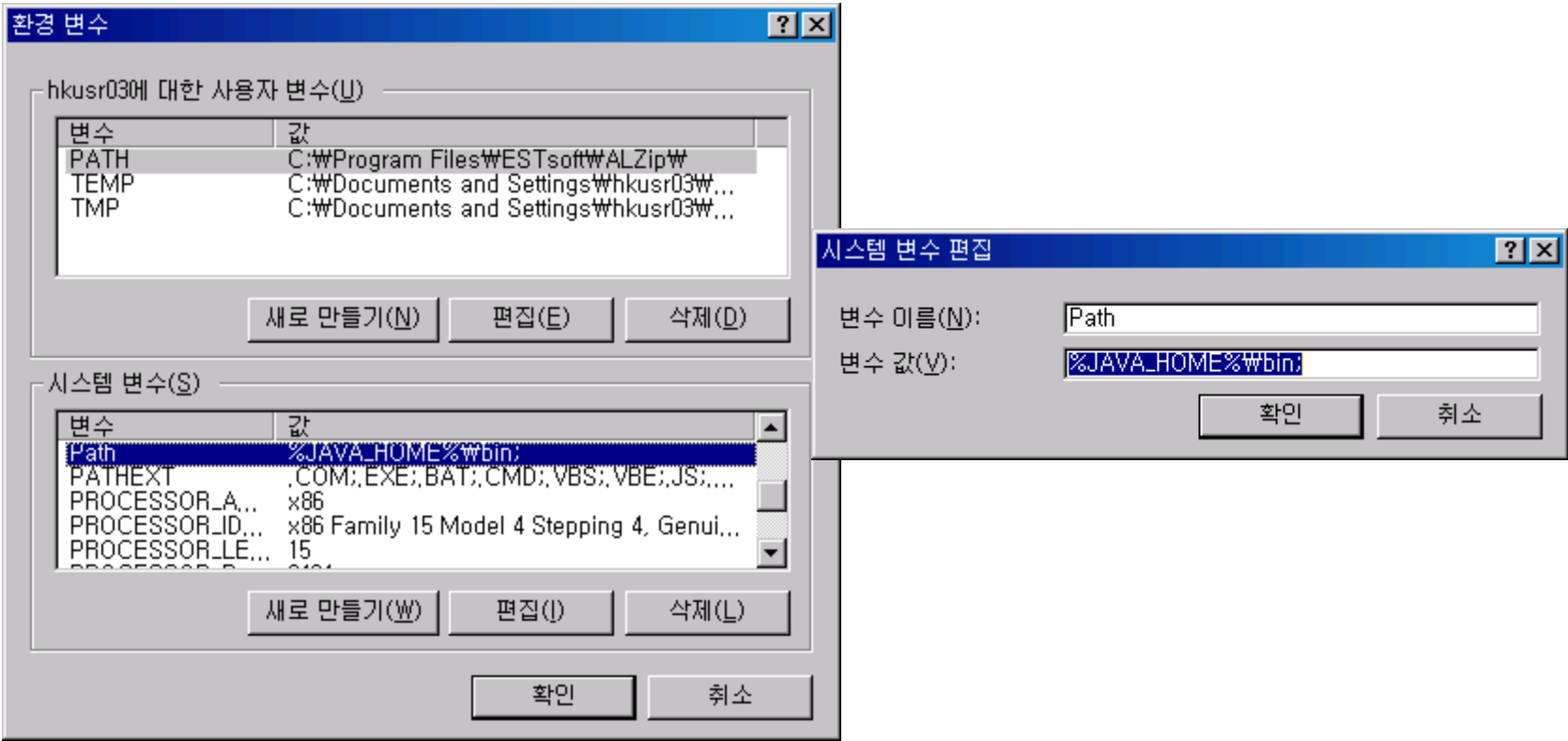
# [0] JAVA 환경설정

3. “시스템 변수” 항목에서 “새로 만들기” 버튼을 클릭하여  
JAVA\_HOME, path, classpath 항목을 추가
- 1) JAVA\_HOME (“C:\wj2sdk1.4.2” 대신 JAVA\_HOME으로 설정)



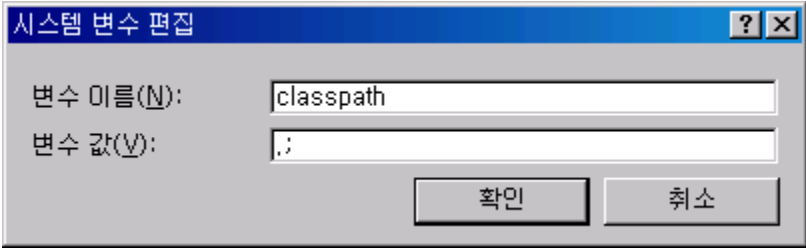
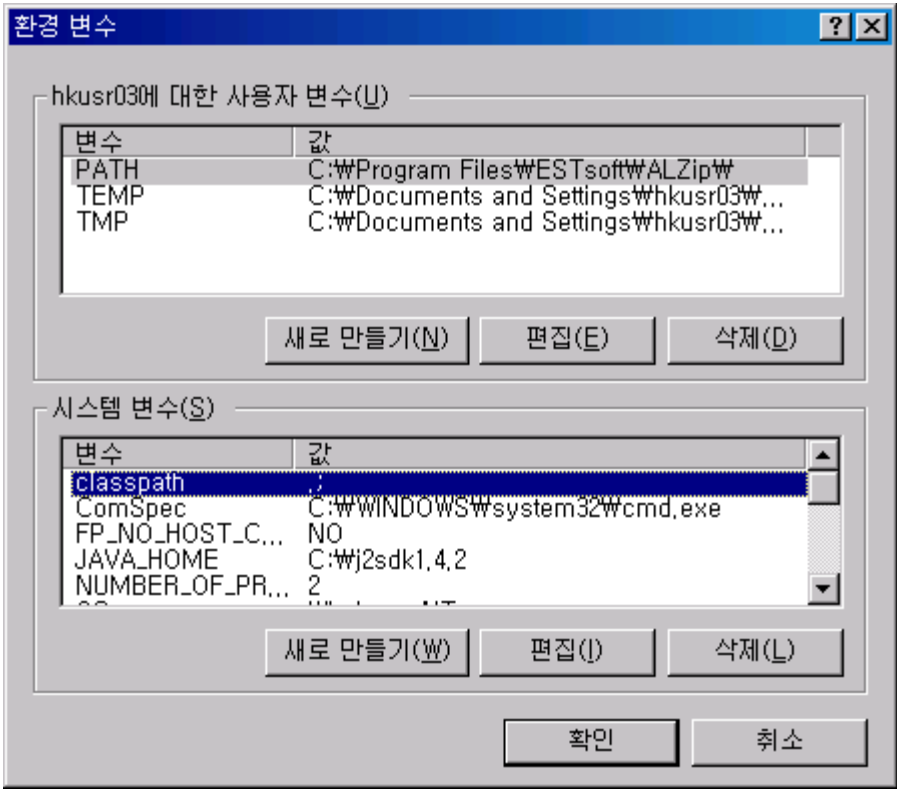
# [0] JAVA 환경설정

## 2) Path 설정 (%JAVA\_HOME%\bin;으로 설정)



# [0] JAVA 환경설정

## 3) classpath 설정

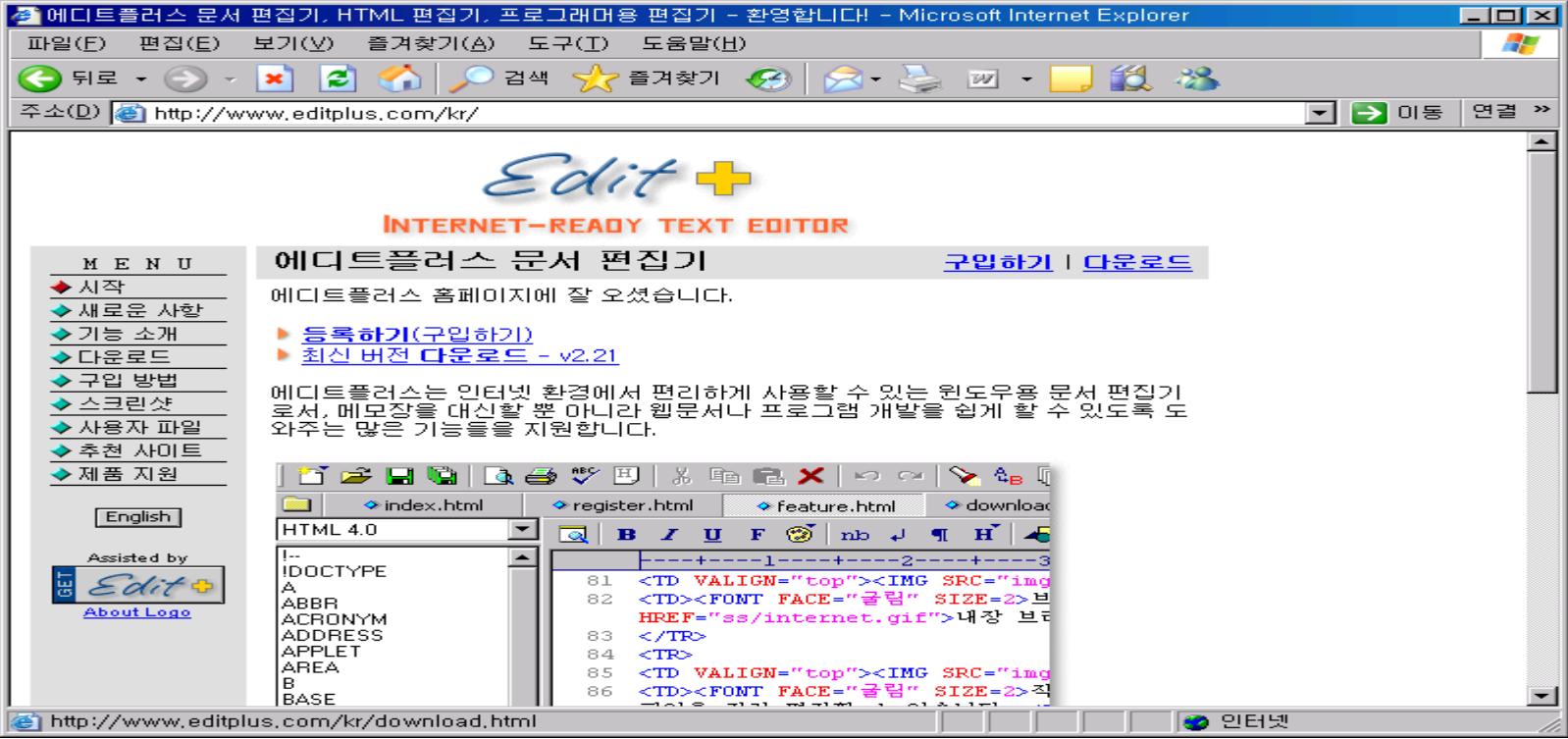




# [0] JAVA 환경설정

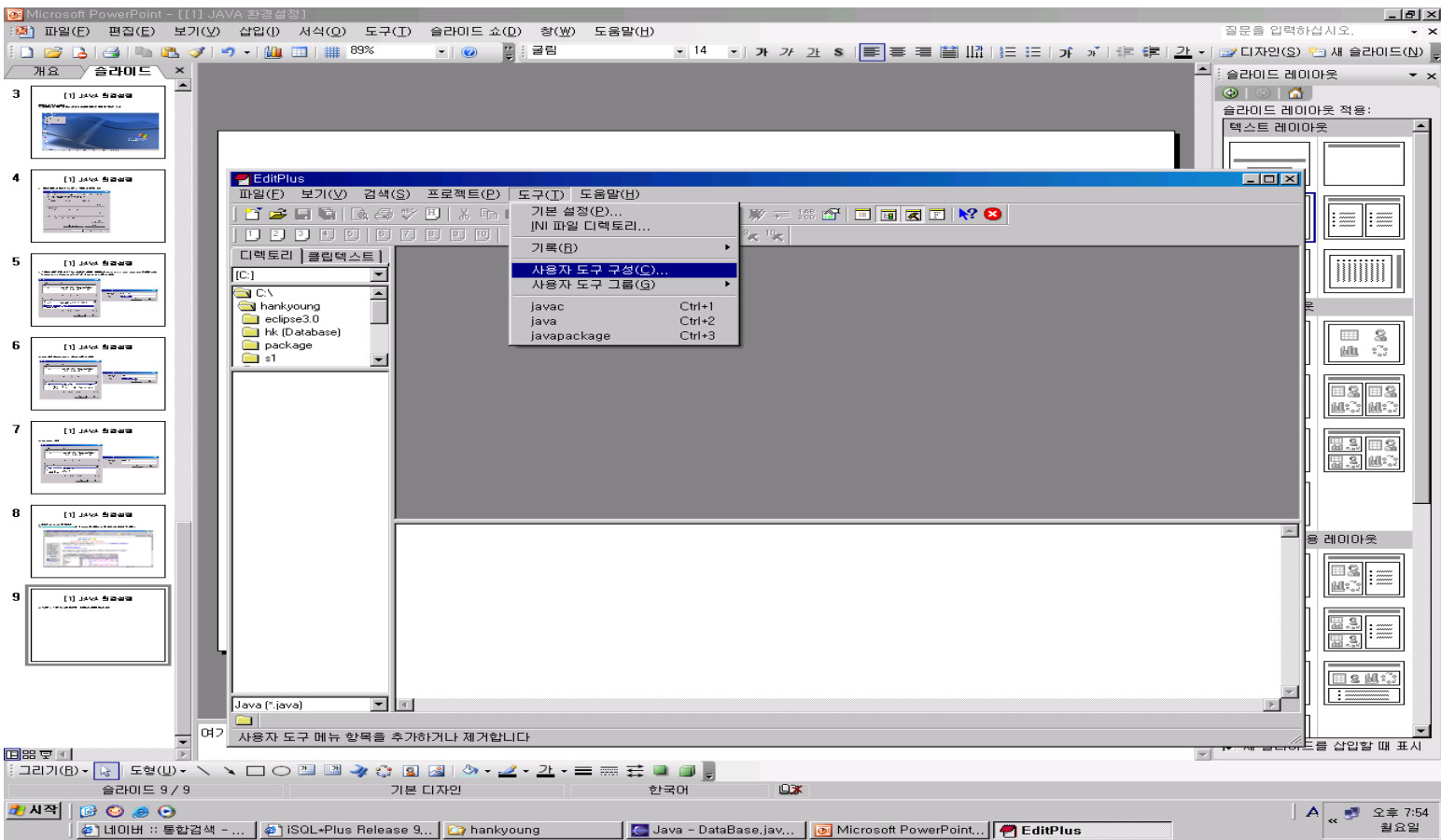
\* 편집기 (EditPlus) 환경설정

1) <http://www.editplus.com/kr>에서 프로그램 다운로드 후 설치  
(설치과정은 생략.)



# [0] JAVA 환경설정

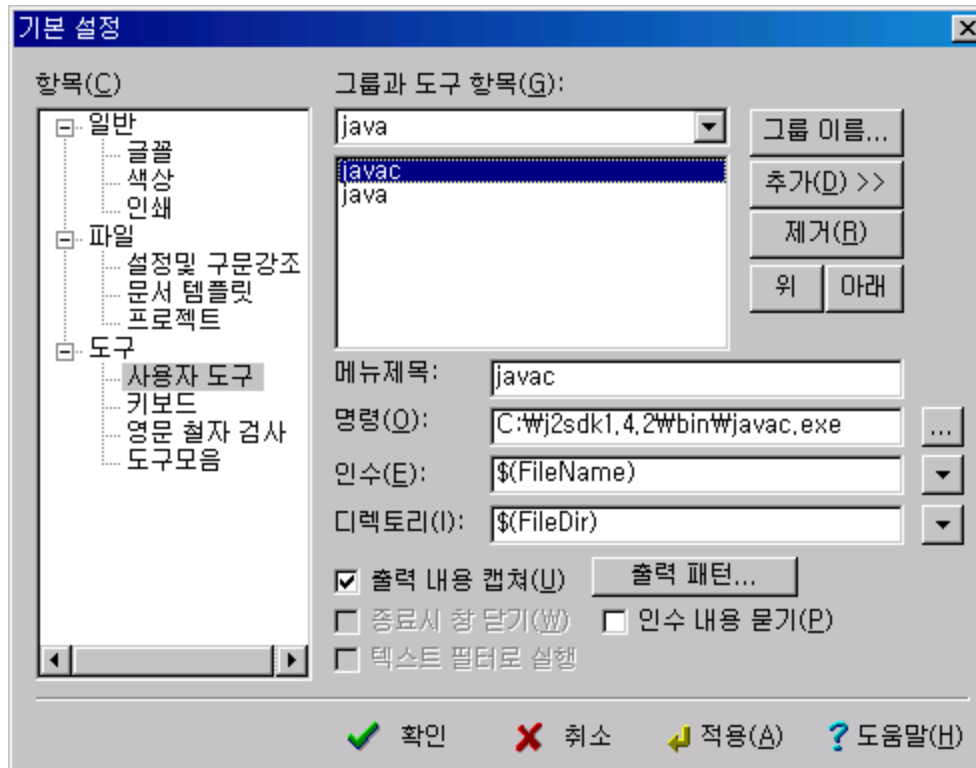
2) “도구” => “사용자 도구 구성”을 선택



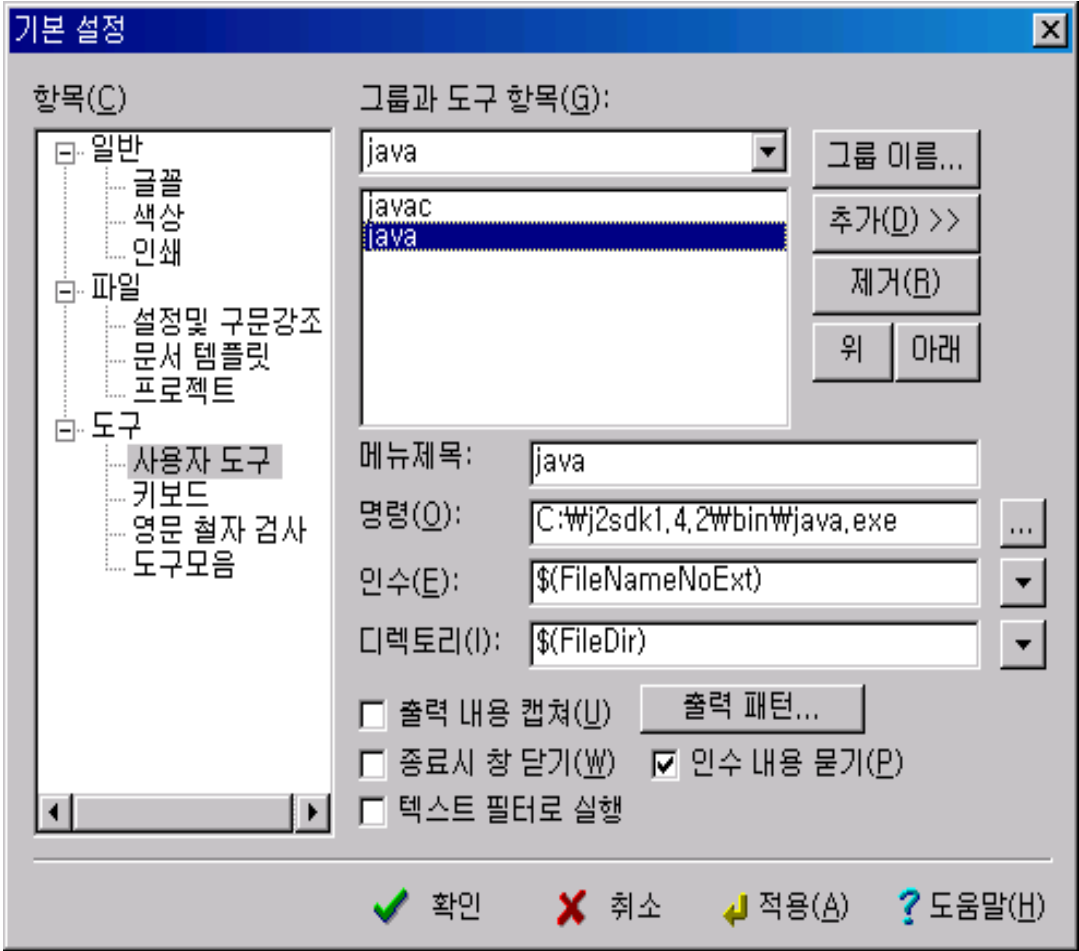
# [0] JAVA 환경설정

## 3) 아래와 같이 환경설정

(그룹 이름을 클릭하여 java라고 입력한 후 '추가'를 선택하여 javac와 java를 입력.)

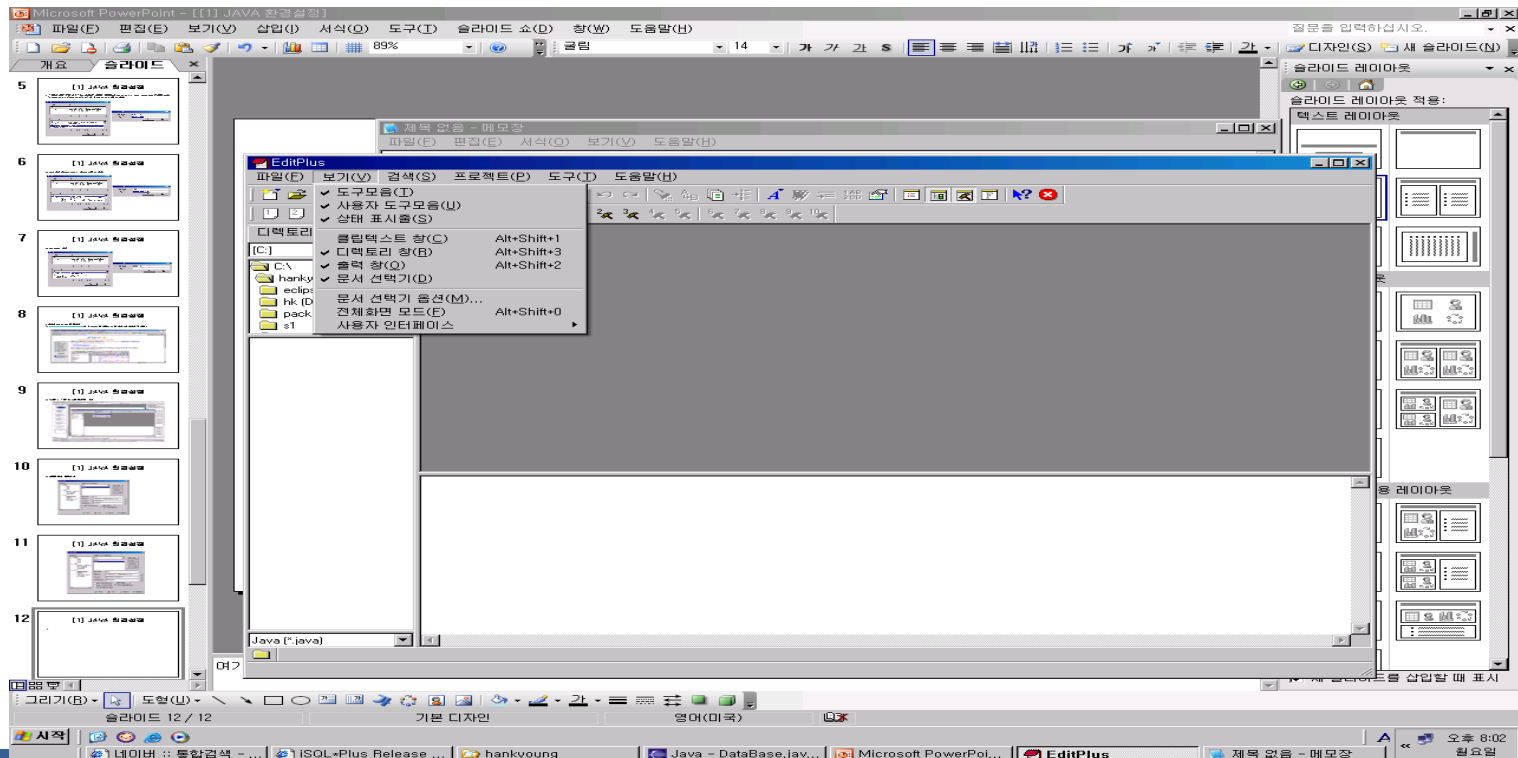


# [0] JAVA 환경설정



# [0] JAVA 환경설정

- 4) 사용자 도구 구성이 끝나면 보기 => 사용자 도구모음을 클릭한다.  
(종이모양 그림 중 3번이 자바 프로그램의 기본 골격이며  
1번 망치그림은 자바 컴파일(javac),  
2번 망치그림은 자바 실행 프로그램(java)임.)



# [1] JAVA 구성

- \* 자바를 사용하면서 가장 많이 쓰는 5가지 실행파일들  
=> java (실행), javac (컴파일), javadoc(설명서 만들기),  
javap(역컴파일), jre(실행환경)
- \* JRE+API = SDK (관련용어는 3 Page 참조.)
- \* 자바는 크게 OS / JVM / Class로 나뉘며 JVM은 자바언어로 만들어진 프로그램을 실행가능하게 해주는 프로그램이며 class 부분에는 SDK, 실행파일 등이 포함되어 있다.

[자바 플랫폼]

<1> 가상머신(Java Virtual Machine) : 자바 프로그램을 실행하기 위해 소프트웨어적으로 구현한 하드웨어

<2> API(Application Programming Interface) :

JVM에서 프로그램을 실행할 때 사용될 수 있는 라이브러리[자바언어]

JVM에서 실행하기 위한 프로그램을 작성할 수 있도록 해주는 프로그래밍 언어.

# [1] JAVA 구성

[IT용어] 자바 실행 환경 [ -實行環境, Java Runtime Environment ]

플랫폼 자바를 이용해 개발된 소프트웨어는 어떤 플랫폼으로부터도 독립적인 형식인 자바 바이트 코드로 배포되지만 이것을 실행하려면 그 플랫폼을 해석할 수 있는 형식인 native code로 변환해야 한다.

즉, 자바 실행 환경 **JRE**는 이 변환과 실행을 행하는 자바 가상머신과 그 주변의 소프트웨어이다.

[IT용어] 응용 프로그램 인터페이스 [ 應用-, application program interface ]

응용 프로그램이 컴퓨터 운영 체제(OS)나 데이터베이스 관리 시스템(DBMS) 등 다른 프로그램의 기능을 이용할 수 있도록 한 인터페이스. 응용 프로그래밍 인터페이스라고도 하며, 일반적으로 API라는 약어로 부른다.

실제로는 OS 등의 기능과 그 기능을 사용하는 방법을 정의한 함수의 집합을 말한다.

[IT용어] 소프트웨어 개발 키트 [ -開發-, software development kit ]

응용 프로그램의 개발을 간편하고 용이하게 하기 위해 프로그래머에게 유상이나 무상으로 제공되는 개발 도구.

프로그래머들이 솔루션 개발 시 자사의 응용 프로그램 인터페이스(API)를 채택하게 하기 위한 목적으로 대부분 무료로 제공된다.

개발 도구 및 라이브러리, 관련 도큐먼트, 개발 툴 등으로 구성된다.

# Conversion

## 1. 기본

1) Casting: 큰 것 -> 작은 것

예) int->char: (char)int->char

2) Promotion: 작은 것 -> 큰 것

예) char->int

## 2. 참조

1) Casting: 부모 -> 자식

예) Vector v=new Vector; //Vector 부모

String s1=new Vector();

String s1=new String("Hello world");

V.add(s1);

String s2=(String)v.elementAt(0); //Object 타입을 String 타입으로 캐스팅

2) Promotion: 자식 -> 부모

예) SuperClass supC;

subClass subC;

subC=subC; // 자식타입을 부모타입으로 캐스팅



# Operation

1. 5칙연산: +, -, \*, /, %
2. 단축대입: +=, -=, \*=, /=, %=
3. 단항: +5, -5, ++, --
4. 이항: 3+5
5. 삼항: C? A:B
6. Bit 연산자: &, |, ^, ~, <<, >>, >>>
7. 논리연산자(True, False):&, |, &&, --, !