

Java Programming 기초

Java의 소개

(James Gosling)
(Green Project)



- 1991
-
- 1995

- -
 -
 -
- -
 -
- -

PC, ,

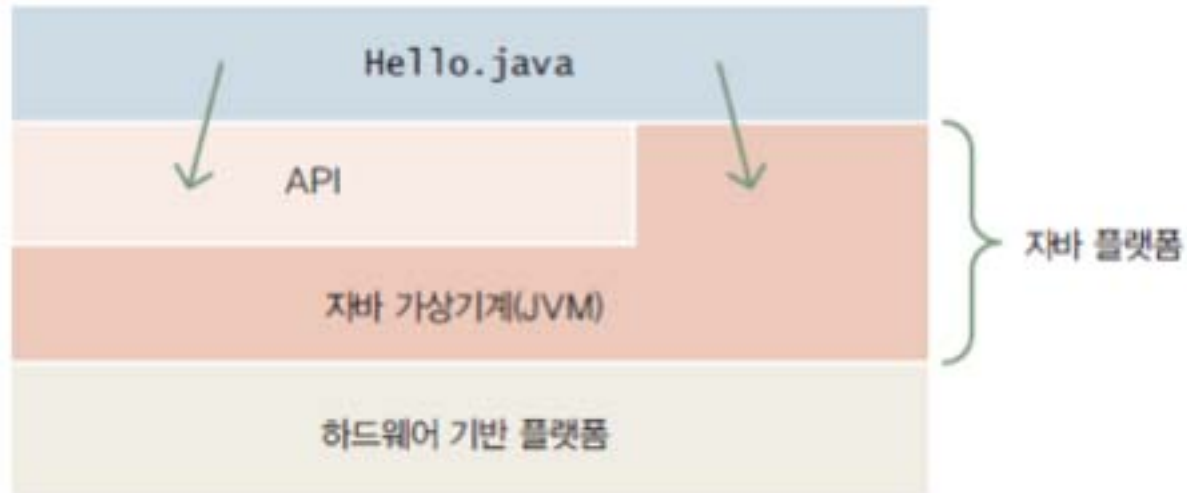
(OAK)

, Netscape

2009

Java 플랫폼

자바 프로그램이 실행되는 하드웨어나 소프트웨어 환경
일반적으로 API란 많은 유용한 기능을 제공하는 라이브러리들의
모임



Java 플랫폼

- **Java SE(Standard Edition)**
- **Java EE(Enterprise Edition)**
- **Java ME(Micro Edition)**



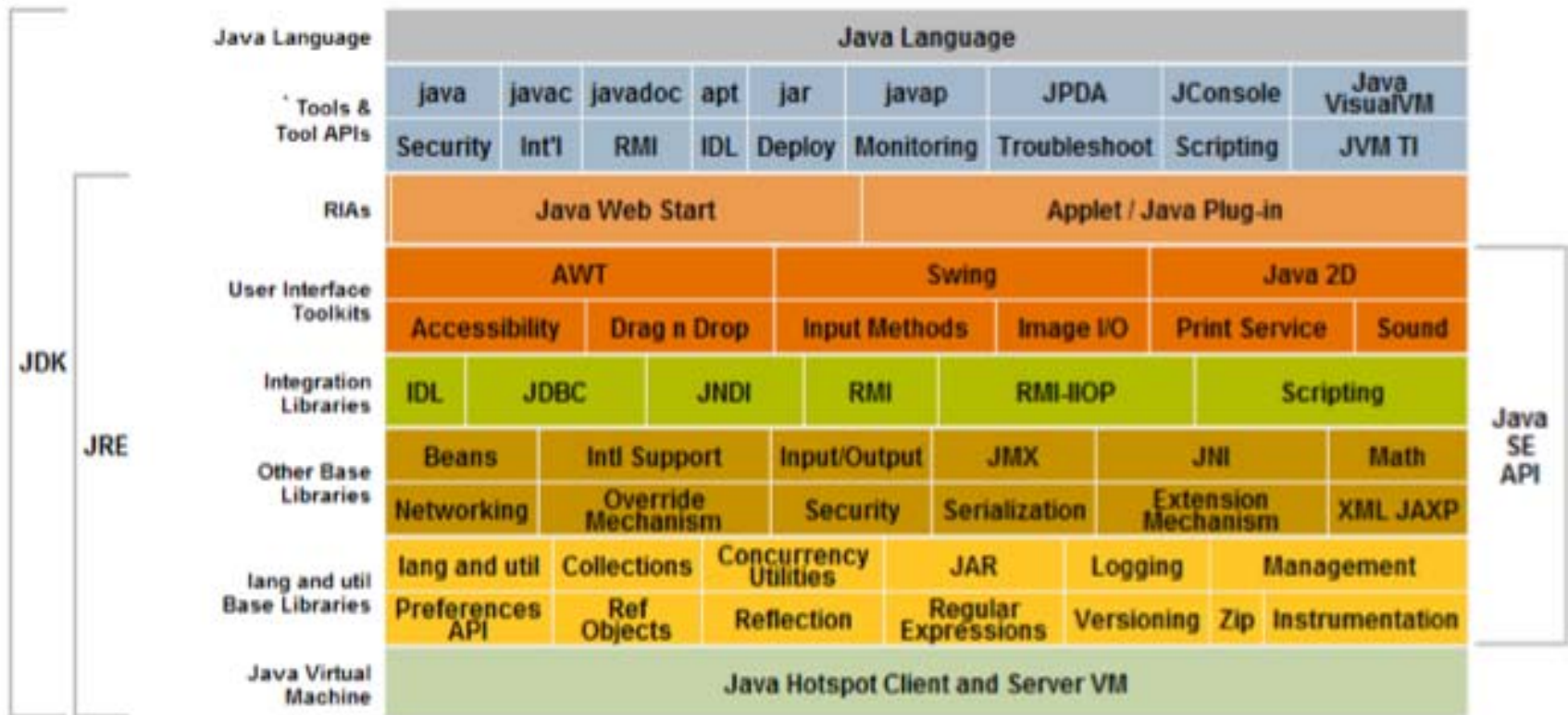
Java 플랫폼(개발도구)

<http://www.oracle.com>

J2SE	Java 2 Standard Edition (응용 프로그램) - 일반적인 범용 컴퓨터 환경을 지원하는 플랫폼
J2EE	Java 2 Enterprise Edition - 자바 기업용 배포판 - 자바를 이용한 다중 사용자, 기업용 응용 개발을 위한 플랫폼 - Java SE + 인터넷 기반의 서버사이드 컴퓨팅 관련 API 추가
J2ME	Java 2 Micro Edition (모바일 프로그램) - 일반적인 컴퓨터환경에 비해 자원 제약이 많은 핸드폰, PDA등의 소형기기 환경을 지원하는 플랫폼

Java 플랫폼

❖자바플랫폼 - Java SE



Java 플랫폼

❖자바플랫폼 - Java SE

JDK

(Java Development Kit
)

- 자바 응용 개발 환경
- 개발에 필요한 도구 포함
- 컴파일러, JRE, 클래스 라이브러리, 샘플

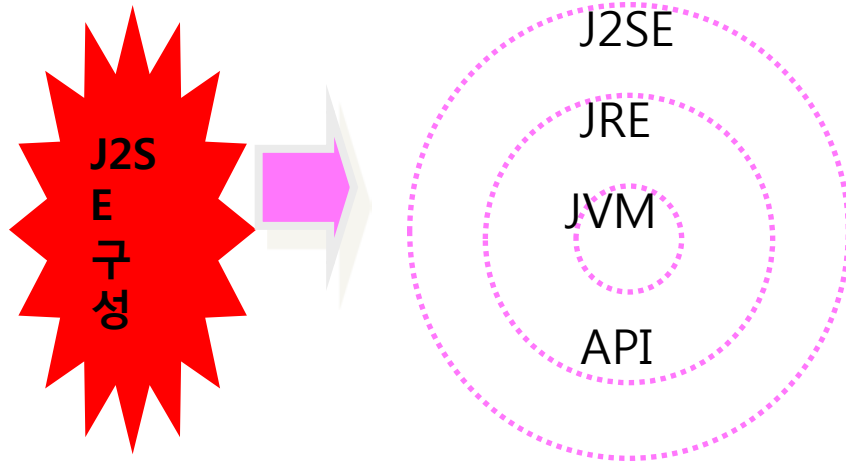
JRE

(Java Runtime Environment
)

- 자바 실행 환경으로
- JVM이 포함되어 있음
- 자바 실행 환경만 필요한 경우
JRE만 따로 다운 가능

Java 플랫폼

❖자바플랫폼 - Java SE



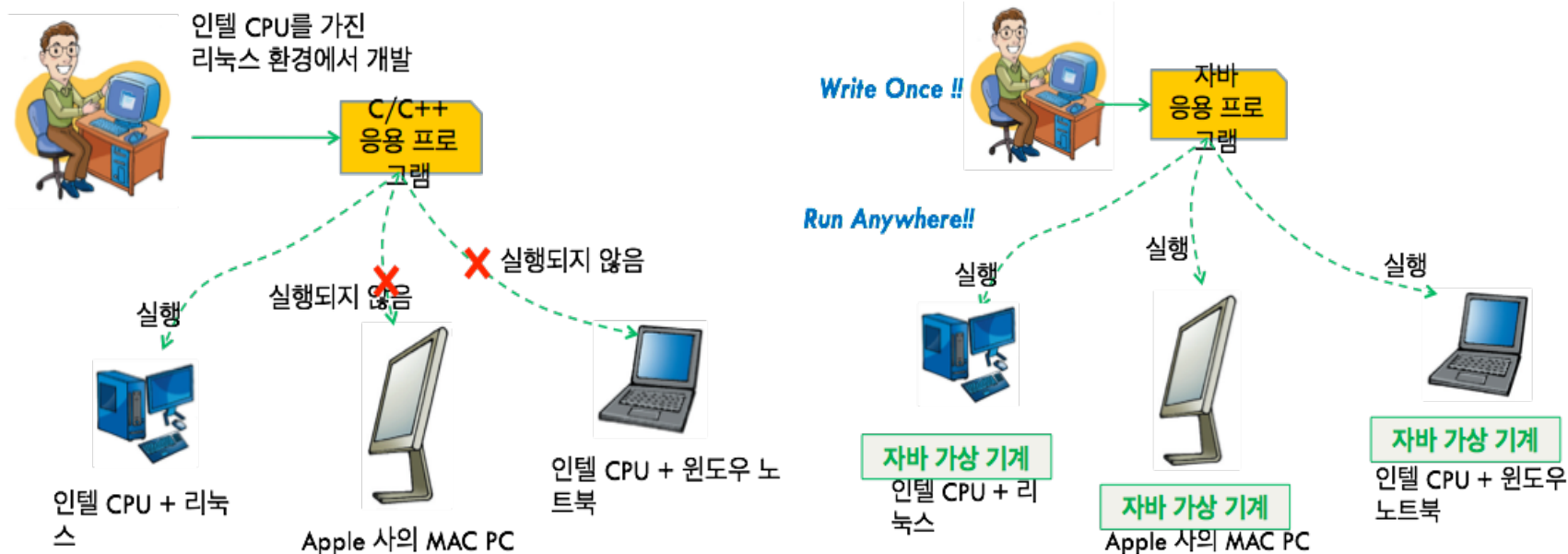
JRE	Java Runtime Edition Java 실행을 위한 최소한 환경
JVM	Java Virtual Machine 메모리관리, 자바운영체제독립
API	Application Programming Interface class들을 묶어 놓은 코드집 (package)

Java Virtual Machine

❖ 자바 가상머신(JVM)

플랫폼 종속적

플랫폼 독립적



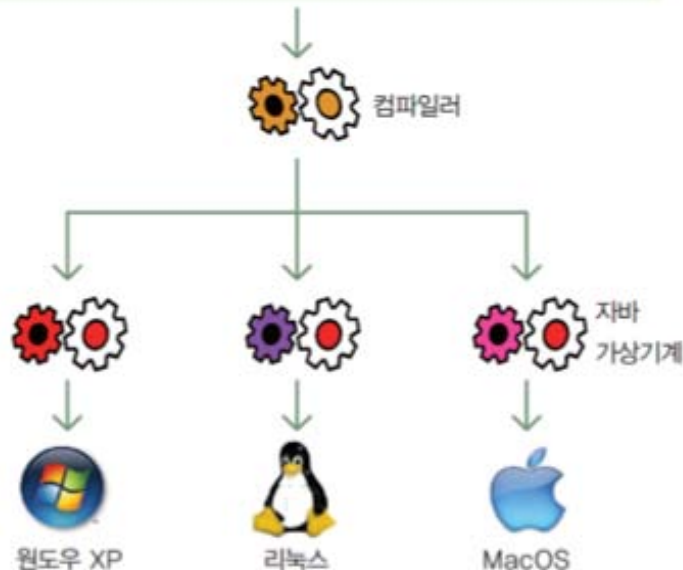
플랫폼 = 하드웨어 플랫폼 + 운영체제 플랫폼

Java Virtual Machine

❖ 자바 가상머신(JVM)

자바 프로그램

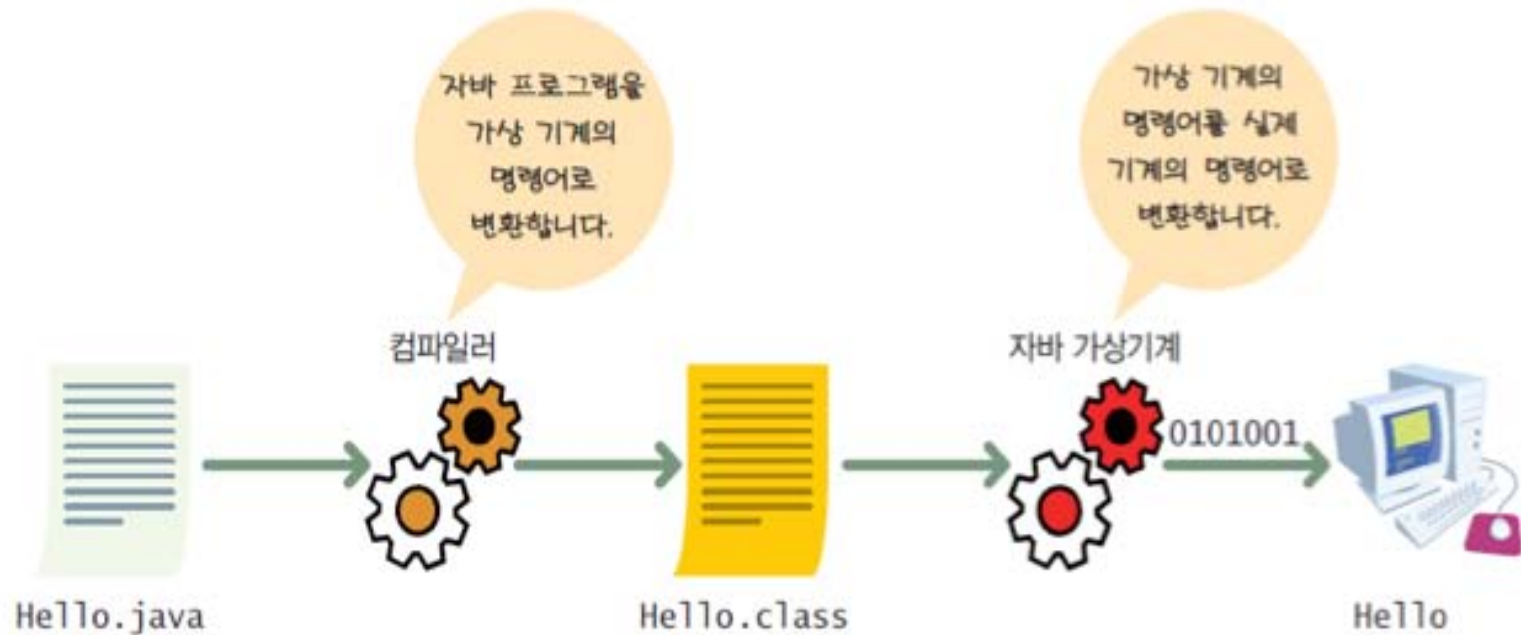
```
class Hello {  
    public static void main(String[] args){  
        System.out.println("Hello World!");  
    }  
}
```



- 가상 기계
- 각 다른 플랫폼에서 동일한 자바실행환경 제공
- 자바 가상 기계 자체는 플랫폼에 종속적
- 오라클 외 IBM, MS 등 다양한 회사에서 제작 공급

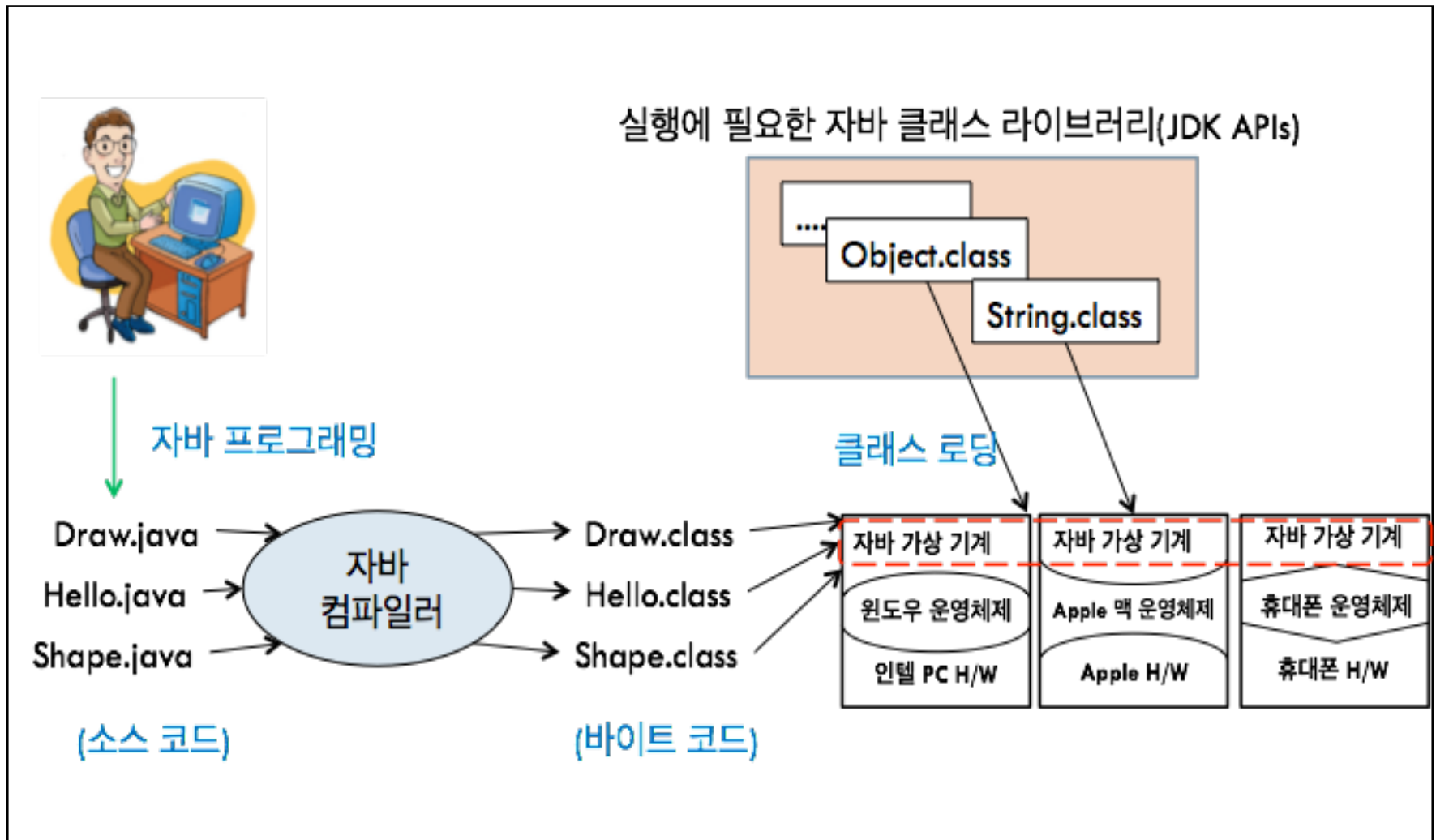
Java Virtual Machine

❖ 자바 가상머신(JVM)



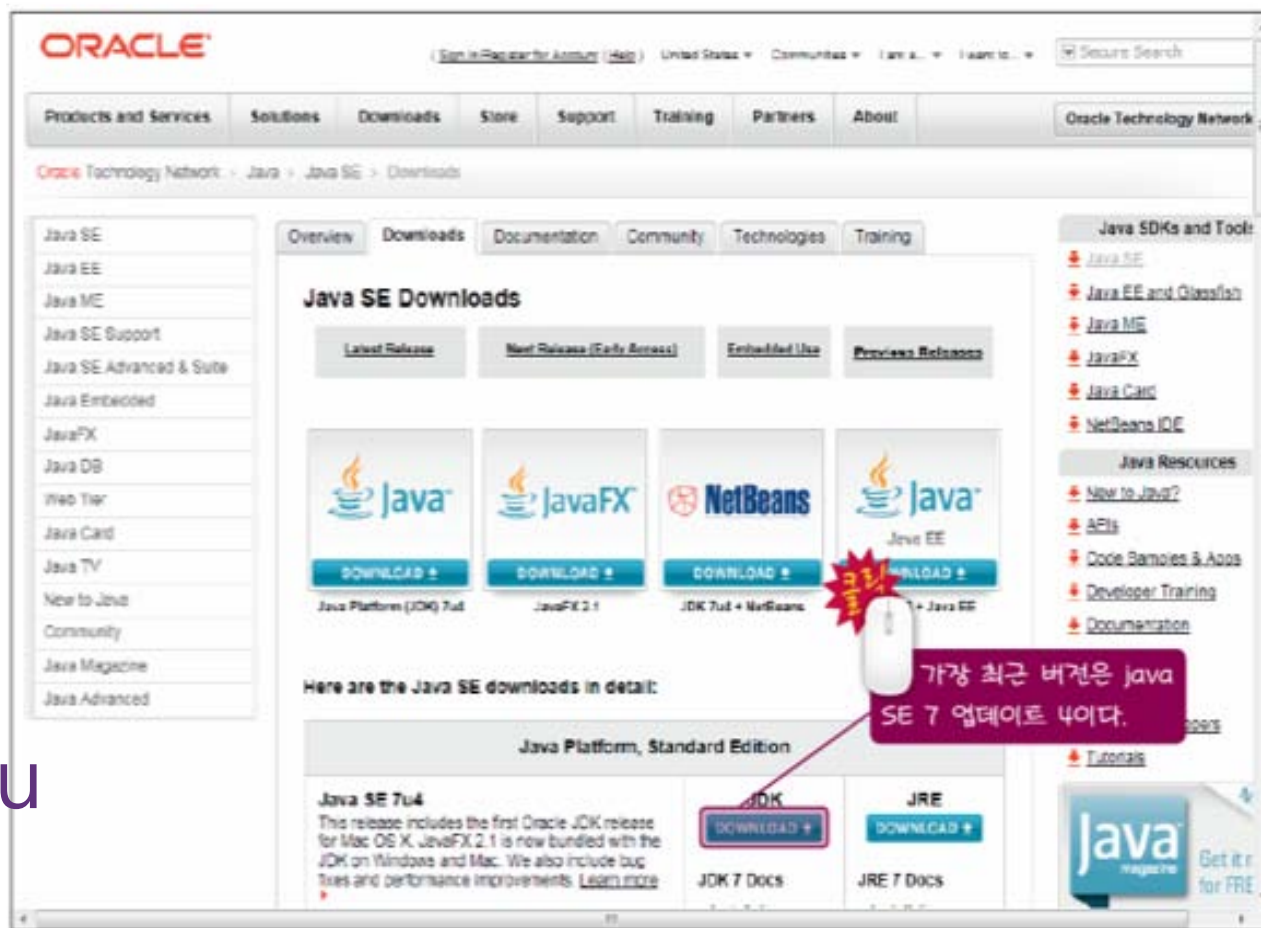
Java Virtual Machine

❖ 자바 가상머신(JVM)

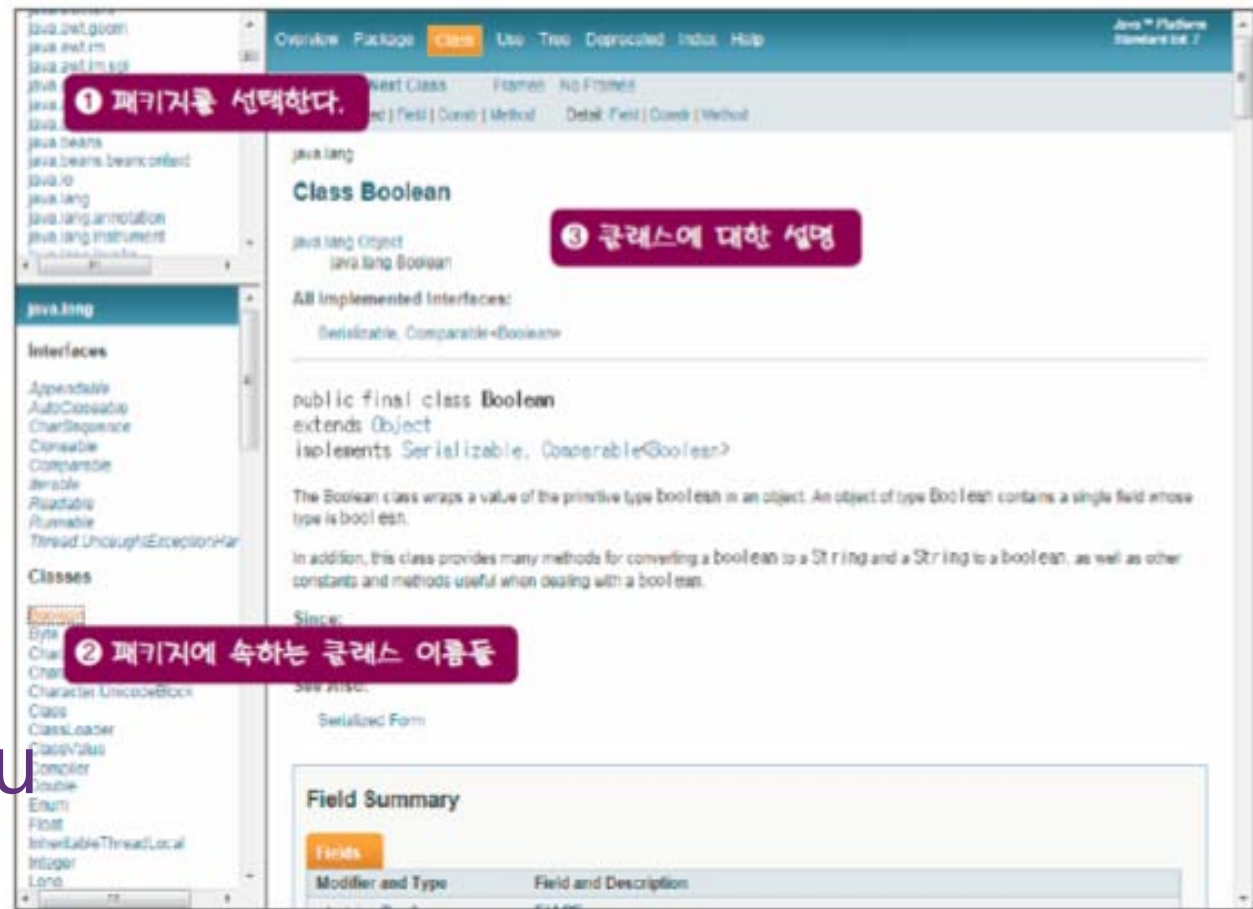


❖자바(JDK) 설치 - 다운로드

Java SE 7u
xx



❖자바(JDK) 설치 - 온라인 API



Java SE 7u
XX

❖자바(JDK) 설치 - JDK 구조

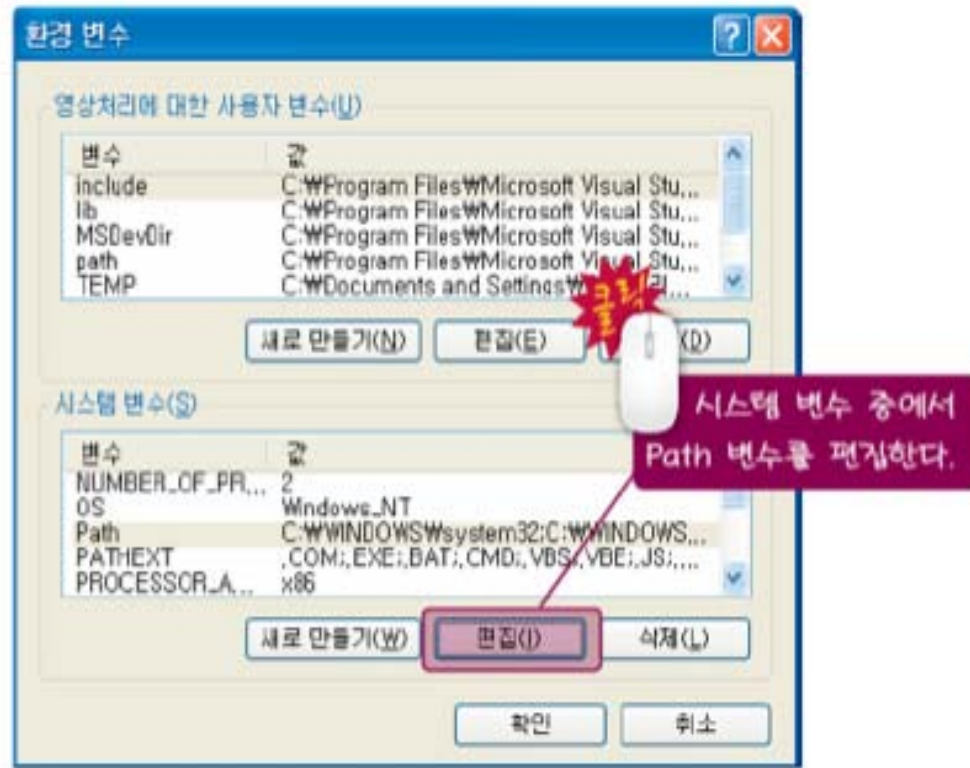
c:\Program Files\Java

폴더	설명
bin	자바 언어를 이용하여 프로그램을 개발하고 실행하며 디버깅, 주석 작업을 도와주는 도구.
db	Java DB, 아파치 Derby 데이터베이스 기술의 선 마이크로 시스템의 배포판 포함.
include	네이티브 코드 프로그래밍을 지원하는 헤더 파일들이다. 이들 파일들은 자바와 C를 동시에 사용하는 프로그램 개발시에 쓰인다.
jre	자바 실행 환경. 자바 가상 기계, 클래스 라이브러리들, 기타 자바 프로그램의 실행을 지원하는 파일들로 이루어져 있다.
lib	개발 도구들이 필요로 하는 추가적인 클래스 라이브러리와 지원 파일들이다.

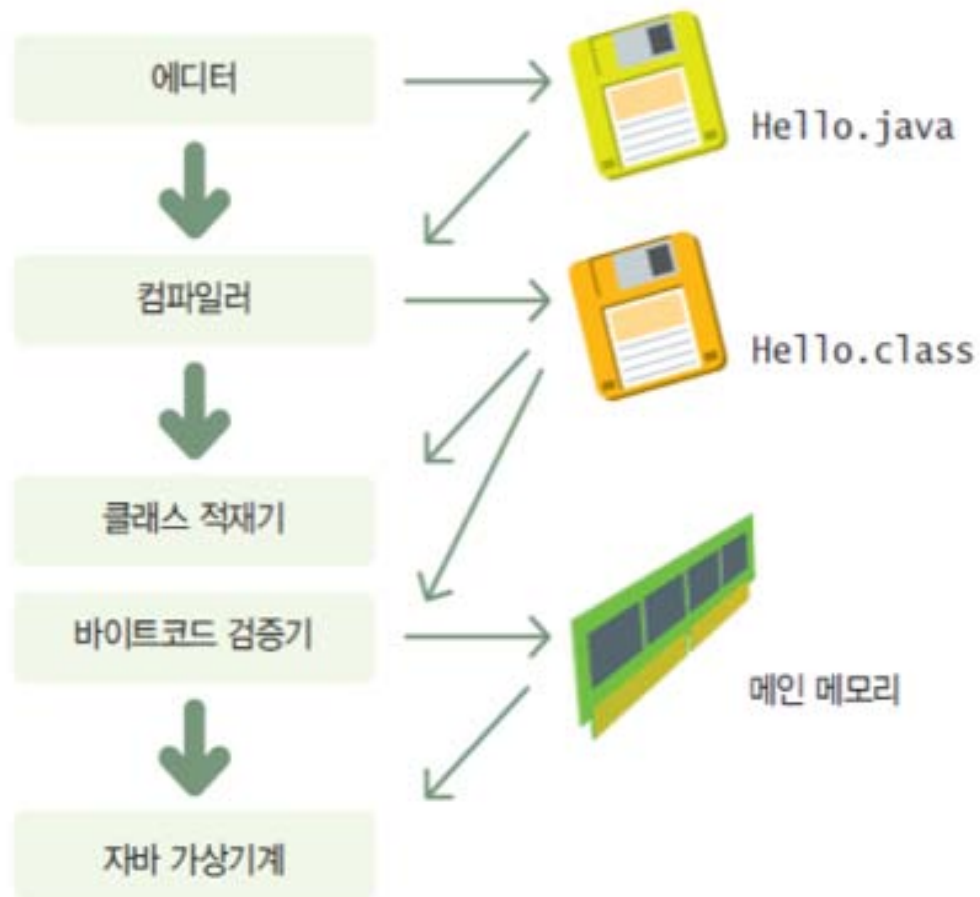
2. 자바 개발도구

❖자바(JDK) 설치 - 환경설정

- 어디에서나 컴파일러를 실행할 수 있도록 경로(path) 설정하기
- [제어판]->[시스템]

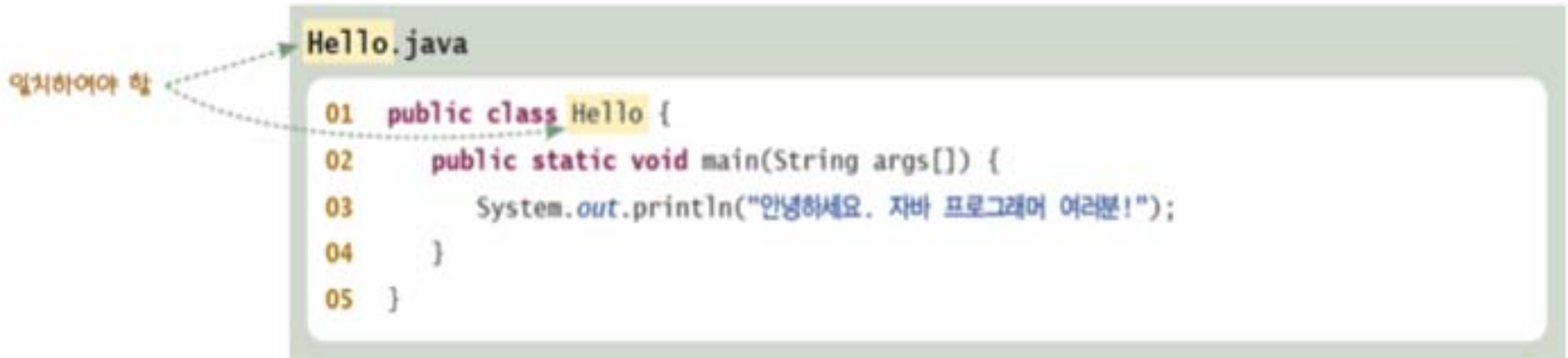


❖자바 개발 단계



❖자바 개발 단계 - 소스 편집

- 파일의 확장자는 .java
- 클래스명과 파일명 일치



일치하여야 함

```
01 public class Hello {  
02     public static void main(String args[]) {  
03         System.out.println("안녕하세요. 자바 프로그래머 여러분!");  
04     }  
05 }
```

❖자바 개발 단계 - 컴파일

```
C:\java\examples>javac Hello.java ←-----소스 파일을 컴파일하여서  
클래스 파일로 변환한다.  
  
C:\java\examples>
```

```
C:\java\examples>dir  
...  
2009-06-05 오후 04:06      454 Hello.class ←-----클래스 파일  
2009-06-05 오후 02:53      144 Hello.java  
                2개 파일      598 바이트
```

❖자바 개발 단계 - 실행

```
C:\java\examples>java Hello ←----- 가상 기계 상에서 클래스 파일을 실행한다.  
안녕하세요. 자바 프로그래머 여러분!  
C:\java\examples>
```

JDK 설치(자바개발도구)

<http://www.oracle.com> 에서 J2SE 7.0 무료다운로드 받아 설치한다.



version

Version 1.1 ~ 1.4 => Version 5.0 ~ 7.0

☞ Version 5.00이 되면서 5가지 문법 변화

- ① Generic Type ② 개선된 루프 ③ 오토박싱/언박싱
- ④ Static import ⑤ Varargs



Path
설정

JDK 설치 후 자주 사용되는 bin폴더 실행파일 들을 path설정한다.

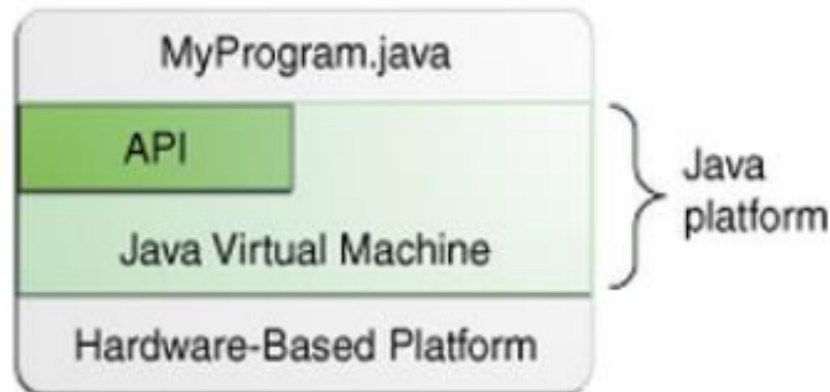
☞ **이유** : bin폴더에 있는 javac.exe(컴파일), java.exe(실행) 파일
을

어떤 경로에서든지 자유롭게 사용하기 위해 설정한다.
(설정하지 않아도 상관없으나 사용자 편의를 위해 설정함)

Java Platform 구성

I. Java Platform 두가지 구성 요소

1. Java Virtual Machine
2. Java Application Programming Interface (API)

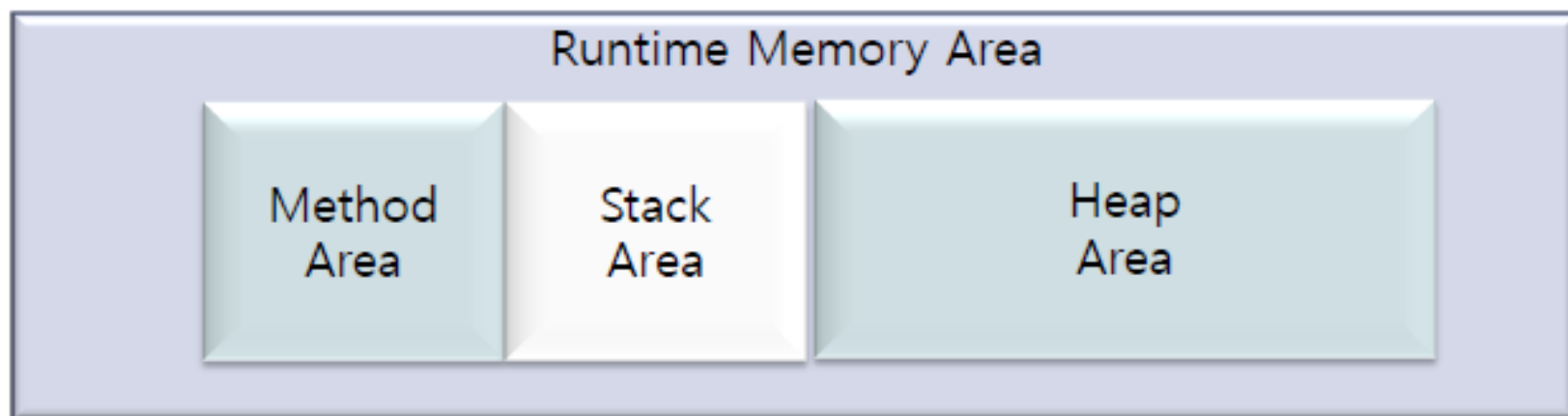


Platform이란?

플랫폼은 프로그램이 실행되는 하드웨어나 소프트웨어 환경
- 운영 체제 및 기본 하드웨어의 조합

JVM의 메모리 구조

1. Method Area(Class Area)
 1. 클래스 내용(Byte code)가 적재되는 영역
 2. static 변수 및 상수가 적재되는 영역
2. Stack Area
 1. 메소드 실행 공간(Method Frame)이 적재되는 영역
3. Heap Area
 1. 생성된 객체가 적재되는 영역



Java문서 구조

Cmd창 open하여 java문서경로 이동한 후 아래문장 실행한다.

클래스이
름.java

```
class 클래스이름{  
    속성=Field=전역변수  
  
    메소드=함수=기능  
}
```



① 컴파일 방법(문서 오류체크)
javac 클래스이름.java

☞ 오류가 있으면 오류메시지가 출력
오류 없으면 클래스이름.class 파일 생김.

② 실행 방법
java 클래스이름

객체란?

1. 유형, 무형의 모든것들 의미
2. 속성과 행위로 구성
 1. Attribute : 객체의 고유한 데이터, 특징
 2. Behavior : 객체의 고유한 동작, 행위, 기능
3. 객체 구분
 1. 물리적 객체 : 현실 세계에서 보고 만질 수 있는 것
 1. 사람, 자동차, 책상, 의자, 꽃 등
 2. 개념적 객체 : 무형의 개념
 1. 비행경로, 날씨 정보등

Java의 특징

- ① OOP (Object Oriented Programming) – 재사용 목적
- ② Platform Independent - 어떤 운영체제에서든 모두 똑같이 실행된다
.
- ③ Security - 보안, 안전성 높음
(JVM이 메모리관리를 자동적으로 실행, 프로그래머가 접근 못함)
- ④ Web에서 돌아가는 Program 작성가능 - Applet 기능
- ⑤ API 제공 - 이미 만들어진 많은 class를 제공
- ⑥ C/C++ 보다 쉽다.
이유 : Pointer개념, 다중상속, 메모리 관리 개념 없음

Java 문법

- ① 대·소문자 구분 - $A \neq a$
- ② 자바의 한 문장은 ; (세미콜론)으로 끝난다.
- ③ 주석 - ㉠ // 한줄주석 ㉡ /* ~ */ 여러줄주석, 부분주석
 ㉢ /** ~ */ Document주석
- ④ { } - class, method 표현
 [] - 배열(Array)
 () - method
- ⑤ ' ' (char)와 " "(String)는 전혀 다르다.
- ⑥ 단어와 단어 사이에 많은 공백 허용 가능.

Java 문법

⑦ Identifier (식별자, 구분자)

class이름, method이름, 변수이름을 의미 => 프로그래머가 만들어 내는 이름들

■ 작성규칙

- ➡ class이름의 첫 글자는 대문자로 시작한다.
- ➡ method이름, 변수이름의 첫 글자 소문자로 시작한다.
- ➡ 예외: 상수는 대문자로 시작
- ➡ 첫 번째 글자는 A~Z, a~z, _ , \$ 한글 가능(숫자 사용X)
- ➡ 두 번째 글자부터 숫자 사용가능
- ➡ 한글 사용가능, _ , \$ 이외의 특수문자 사용 안됨.
- ➡ method는 항상 () 붙는다.
- ➡ 길이제한 없음, 예약어(keyword) 사용안됨.
- ➡ Unicode 지원가능 - 2byte
- ➡ 두단어의 합성으로 이름이 명명될 경우, 두번째 단어의 첫글자를 대문자로 하고 constant는 _을 이용하여 두 단어를 붙이게한다.

권장사항

변수(Variable = Field)

특징 : Data를 저장하는 공간
한가지 Type, 한가지 값만 저장 가능.

변수선언 방법

- ① DataType(자료형) 변수이름 ;
- ② DataType(자료형) 변수이름 = 값 ;

예)

```
int age;  
String name = "장희정" ;  
int year = 1993 ;
```

자료형

① Primitive Type(기본형) – 8가지

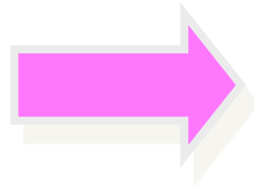
정수형	<ul style="list-style-type: none">byte - 1byte (8bit)short - 2byte (16bit)int - 4byte (32bit) ⇒ 기본형long - 8byte (64bit) ⇒ ex) 5L
실수형	<ul style="list-style-type: none">float - 4byte ⇒ ex) 5.5F or 5.5fdouble - 8byte ⇒ 기본형
문자형	<ul style="list-style-type: none">char - 2byte ⇒ 반드시 <u>한글자만</u> 저장가능. <u>' '</u>만 묶는다. ex) char ch="A"; ⇒ X , char ch='A'; ⇒ unicode값 65저장 char ch='가' ; ⇒ 0 , char ch="가"; ⇒ X
논리형	<ul style="list-style-type: none">논리형 - boolean – 크기없음 ⇒ true, false만 저장가능

자료형

② ObjectType = ReferenceseType(객체형)

PrimitiveType을 제외한 모든 Type

```
class Test{    }  
class Super{  }  
class Hello{  }
```



변수선언 예)

```
Test t;  
Super s;  
Hello h;
```

문자열 저장 => String – ObjectType

ex) String str = "jang" ;

연산자

① 산술연산자

+	더하기
-	빼기
*	곱하기
/	나누기 ex) $5 / 2 \Rightarrow 2$
%	나머지 ex) $5 \% 2 \Rightarrow 1$

② 관계연산자

>	크다
<	작다
>=	크거나 같다
<=	작거나 같다
!=	같지않다
==	같다

③ 대입연산자

=	$a=b$;
+=	$a+=b$; $\Rightarrow a=a+b$;
-=	$a-=b$; $\Rightarrow a=a-b$;
=	$a=b$; $\Rightarrow a=a*b$;
/=	$a/=b$; $\Rightarrow a=a/b$;
%=	$a\%=b$; $\Rightarrow a=a\%b$;

연산자

④ 비트연산자

	or : 양쪽 중 하나만 true이면 true (주기가 긴 연산자)
&	and : 양쪽 모두 true이면 true (주기가 긴 연산자)
^	xor : 양쪽이 같으면 false, 양쪽이 다르면 true

	true	false
true	true	true
false	true	false

&	true	false
true	true	false
false	false	false

^	true	false
true	false	true
false	true	false

주기 긴 연산
자 의미

true | ? => true
false & ? => false



왼쪽의 값만으로도 결과값이 나오
지만
?부분을 무조건 실행한다.

연산자

⑤ 논리연산자

	or : 양쪽 중 하나만 true이면 true (주기가 짧은 연산자)
&&	and : 양쪽 모두 true이면 true (주기가 짧은 연산자)

	true	false
true	true	true
false	true	false

&	true	false
true	true	false
false	false	false

주기 짧은 연
산자 의미

true || ? => true
false && ? => false

왼쪽의 값만으로도 결과값이 나오
므로
?부분은 실행하지 않는다.

연산자

⑥ 증감연산자

++ (1씩 증가)	\hookrightarrow a++ : 선 대입 후 증가 \hookrightarrow ++a : 선 증가 후 대입
-- (1씩 감소)	\hookrightarrow a-- : 선 대입 후 감소 \hookrightarrow --a : 선 감소 후 대입

```
int a=1, b=1;
(⌋) a = b++;
    System.out.println(a);  $\Rightarrow$  1
    System.out.println(b);  $\Rightarrow$  2

(⌋) a = ++b;
    System.out.println(a);  $\Rightarrow$  2
    System.out.println(b);  $\Rightarrow$  2
```

```
int a=1, b=1;
(⌋) a = b--;
    System.out.println(a);  $\Rightarrow$  1
    System.out.println(b);  $\Rightarrow$  0

(⌋) a = --b;
    System.out.println(a);  $\Rightarrow$  0
    System.out.println(b);  $\Rightarrow$  0
```

연산자

⑦ 연결 연산자

(+)

숫자 + 숫자	덧셈 => $5 + 2 = 7$
숫자 + 문자열	붙이기(연결) => $5 + \text{"2"} = 52$
문자열 + 숫자	붙이기(연결) => $\text{"5"} + 2 = 52$ $\text{"A"} + 2 = A2$
문자열 + 문자열	붙이기(연결) => $\text{"5"} + \text{"2"} = 52$ $\text{"Java"} + \text{"짱"} = \text{Java짱}$

연산자

⑧ 조건 삼항 연산자

조건식 ? 조건식결과 참일경우 : 조건식결과 거짓일경우

```
long seed=0;  
Random r = (seed==0) ? new Random() : new Random(seed);
```

형변환

☞ Promotion

- 작은 Type이 큰 Type으로 변신(자동으로 이루어짐)
- 서로 다른 DataType이 연산을 하면 작은 Type이 큰 Type으로 변신

ex) `int a=3; double d=3.2;`
`double c = a + d;`

기본형의 크기순서

`byte < short, char < int < long < float < double`

형변환

☞ Casting

- 큰 Type이 작은 Type으로 변신 (절대 자동아님 - Casting연산자이용)
- 큰 Type을 작은 Type의 변수에 저장할 때 Casting 필요.

Casting방법

(변신하려는DataType)값;

예) `int a = 5.3;` (X) -> Casting \Rightarrow `int a = (int)5.3;` (O)

예) `byte a=2; byte b=3;`
Casting \Rightarrow `byte c = (byte)(a+b);`

예) `int a=3; double b=23;`
`int c = a+b;` -> Casting \Rightarrow `int c = (int)(a+b);`
`int c = a + (int)b;`

제어문

① 조건문(조건에 따라 실행문장을 다르게 한다.)

If문 구조

```
if( 조건문 ){  
    실행문장;  
}else if( 조건문 ){  
    실행문장;  
}else if( 조건문 ){  
    실행문장;  
}  
....  
else{  
    실행문장;  
}
```

switch문 구조

```
switch( expr1 ){  
    case 값 : 실행문장 ; break;  
    case 값 : 실행문장 ; break;  
    case 값 : 실행문장 ; break;  
    .....  
    default : 실행문장  
}
```

- **expr1**는 반드시
int ,byte, short , char 만 가능함.

제어문

② 반복문

for문 구조

```
for ( ①초기화 ; ②조건식 ; ③증감식 ){  
    ④실행문장;  
}  
⑤
```

for문 실행순서

① → ②false경우 → ⑤
②true경우 → ④ → ③ → ②false경우 → ⑤
②true경우 → ④ → ③ → ②

제어문

② 반복문

while문 구조

```
① 초기화;  
while ( ② 조건식 ) {  
    ③ 증감식 ;  
    ④ 실행문장 ;  
}
```

=> ③, ④ 번은 순서 상관없음

do_while문 구조

```
① 초기화;  
do {  
    ③ 증감식 ;  
    ④ 실행문장 ;  
} while ( ② 조건식 );
```

=> ③, ④ 번은 순서 상관없음

특
징

do_while문은 무조건 한번은 실행한다.

제어문

③ 반복문 제어

break	break를 만나면 이하 문장 실행하지 않고 <u>감싸고 있는 반복문을 빠져 나온다.</u>
continue	continue를 만나면 이하 문장 실행하지 않고 <u>다시 반복문을 실행한다.</u>

break 예)

```
for( a=1 ; a<5 ; a++ ){  
    if(a==3) break;  
    System.out.print(a);  
}
```

출력 : 1 2

continue 예)

```
for( a=1 ; a<5 ; a++ ){  
    if(a==3) continue;  
    System.out.print(a);  
}
```

출력 : 1 2 4

제어문

③ 반복문 제어

label

break, continue 사용시 특정위치로 가도록 하기 위해 label 을 지정 할 수 있다. 그러나 label을 많이 사용하게 되면 프로그램의 구조가 복잡하게 되므로 사용을 지양한다.

label 예)

```
outer :  
for(int a=1; a<=5 ; a++){  
    for(int b=1 ; b<=3 ; b++ ){  
        if(b==2) break outer;  
        System.out.print(a);  
    }  
}
```

Method

특징

- ① 객체 안에 선언되어 객체가 가지고 있는 기능이다.
- ② 반드시 class내부에 선언된다.
- ③ 호출해서 사용한다.(호출되기 전에는 실행 안됨.)
- ④ 재귀호출가능(자기 자신 안에서 자신을 호출)
- ⑤ Method 마지막 구현부에서 return 할 수 있다.
(특정한 값을 호출한 주체에게 return 한다.)
- ⑥ 재사용 목적 - 코드의 중복을 피할 수 있다.

Method 작성법

```
modifiers returnType methodName(  
    [DataType 변수이름, DataType 변수이름, ...] ){  
  
    기능구현 ;  
}
```

Method 작성법

modifiers

- 0개 이상 올 수 있다.
- 여러개 올 때 공백으로 구분(일반적으로 접근제한자 먼저 작성)
 - ㉠ **public** : 어디서나 아무나 접근 가능
 - ㉡ **protected** : 상속관계라면 어디서나 접근가능
 - ㉢ (**생략**) : 같은 폴더(package) 내에서 아무나 접근
 - ㉣ **private** : 하나의 class(객체) 내부에서 아무나 접근, 외부에서 접근불가, 자기 자신만 호출 가능

returnType

- ㉠ void : return안함
 - ㉡ PrimitiveType 8가지 중 1개 OR Object Type중 1개
- => ㉡의 경우 메소드 구현부 마지막에 **return 값 ;** 작성

접근제어자

1. 적용 문법

1. **class 선언구 : public or (default)**
2. 변수, 생성자, 메소드 선언구 : 4가지 다 적용 가능

2. 종류 및 용도

	동일 클래스	동일 패키지의 클래스	다른 패키지의 하위클래스	다른 패키지의 클래스
private				
(default)				
protected				
public				

기타 제어자

1. 종류 및 적용 문법

1. class 선언구 : final, abstract
2. 변수 선언구 : final, static
3. 메소드 선언구: final, static, abstract, synchronized
4. static 블록 : static

	클래스	변수	생성자	메소드	블록
final					
static					
abstract					
synchronized					

Method 호출방법

① 가장 일반적인 방법

㉠ 호출하려는 method를 감싸고 있는 class 생성한다.(객체생성: new)

⇒ 객체생성방법

`class이름 변수이름 = new class이름();`

㉡ 생성된 객체변수를 이용하여 method를 호출한다.

⇒ 방법 : **`변수이름.method이름([값, 값, ...]);`**

② static이 붙은 메소드 호출방법

⇒ 객체를 생성하지 않고 호출가능
`class이름.method([값, 값, ...]);`

③ 같은class 내부에서 method 호출하는 방법

⇒ **`this.method이름([값, 값, ...]);`**

객체 생성 및 사용

❖ 객체 생성 및 사용

```
Car    myCar;           // ❶ 참조 변수를 선언  
myCar = new Car();      // ❷ 객체를 생성하고 ❸ 참조값을 myCar에 저장
```

❶ 참조 변수 선언 - Car타입의 객체를 참조할 수 있는 변수 myCar를 선언

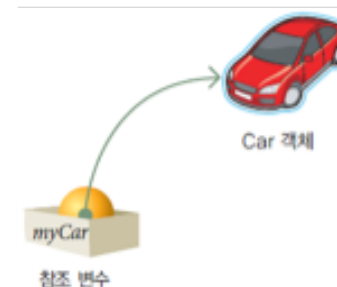


❷ 객체 생성 - new 연산자를 이용하여 객체를 생성, 객체 참조값을 반환



Car 객체

❸ 참조 변수와 객체 연결 - 생성된 새로운 객체의 참조값을 myCar 라는 참조 변수에 대입



객체 생성 및 사용

❖ 객체 생성 및 사용

1. 클래스 정의

```
public class Person {  
    public String name;  
    public int age;  
  
    public Person() {  
    }  
  
    public Person(String s) {  
        name = s;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

2. 객체 생성 및 사용

```
public static void main (String args[]) {  
    Person aPerson;           // 레퍼런스 변수 aPerson 선언  
    aPerson = new Person("김미남"); // Person 객체 생성  
  
    aPerson.age = 30;          // 객체 멤버 접근  
    int i = aPerson.age;       // 30  
    String s = aPerson.getName(); // 객체 메소드 호출  
}
```

객체 생성 및 사용

❖ 객체 생성 및 사용

객체 생성

- ❶ 객체에 대한 참조변수 선언

```
Person aPerson;
```

aPerson



- ❷ 객체 생성

```
aPerson = new Person();
```

aPerson



Person 타입의 객체

name

"김미남"

age

Person() { ... }

getName() { ... }

객체 생성 및 사용

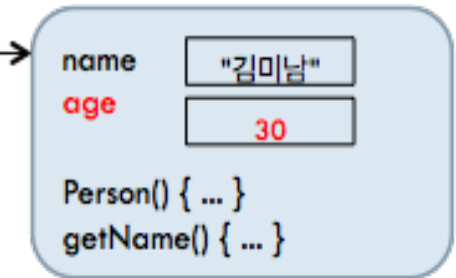
❖ 객체 생성 및 사용

객체 사용

③ 필드 값 설정

```
Person aPerson;
```

aPerson



② 메소드 호출

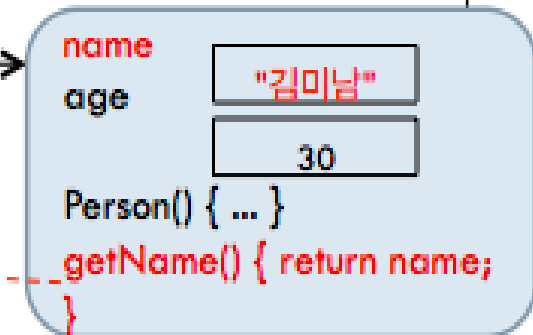
```
String s = aPerson.getName();
```

aPerson



s

"김미남"



객체 레퍼런스 . 멤버

객체 생성 및 사용

❖ 객체 생성 및 사용

참조변수 (Reference Variable)



객체 생성 및 사용

❖ 객체 생성 및 사용

객체 접근방법

myCar가 창조하는 객체로부터

speed라는 필드에 접근

myCar.speed = 100;

도트(.) 연산자 사용!

객체 생성 및 사용

❖ 객체 생성 및 사용

객체 접근방법

myCar가 참조하는 객체로부터

speed라는 필드에 접근

myCar.speed = 100;

객체의 필드에 값 대입

```
public class ClassExample {  
    public static void main (String args[]) {  
        Person aPerson = new Person("홍길동");  
  
        aPerson.age = 30;  
        int i = aPerson.age;  
        String s = aPerson.getName();  
    }  
}
```

객체의 필드에서 값 읽기

객체의 메소드 호출

객체 생성 및 사용

❖ 객체 소멸

가비지(Garbage)

객체에 대한 레퍼런스가 없어지면 객체는 가비지(garbage)
가 됨

자바 가상 기계의 가비지 컬렉터가 가비지 메모리를 반환

가비지 컬렉터(Garbage Collector)

자바에서는 가비지들을 자동 회수, 가용 메모리 공간으로 이동하는
행위

자바 가상 기계 내에 포함된 가비지 컬렉터(garbage collector)에 의
해 자동 수행

```
System.gc(); // 가비지 컬렉션 작동 요청
```

자바문서의 유형

☞ ~ . Java문서 안에

- ① 여러 개의 class 작성 가능하다.
- ② public class는 한 개 만 작성 가능하다.
- ③ public class가 있을 경우 반드시 public class이름으로 파일명을 지정한다.
- ④ main method는 시작을 위해 반드시 필요하기 때문에 시작하는 class안에 작성을 하고 main method가 있는 class이름으로 파일명을 지정해야만 실행 가능하다.
- ⑤ public class안에 main method 작성한다.
- ⑥ class앞에 modifier가 올 수 있는데 접근제한자 중 public or 생략 두가지 중 한가지만 사용가능

변수의 범위

① 지역변수

- method 내부에 선언된 변수
- 선언된 영역(method) 내에서만 접근가능
- 다른 method에서 접근 안됨
- 지역변수는 사용전에 반드시 초기화해야 한다.

ex) int a; ⇨ X int a=값; ⇨ O

② 전역변수

- class 내부에서 선언된 변수
- 모든 method에서 접근가능
- 지역변수와 전역변수 이름이 같은 경우 **this.변수이름;** 하면 전역변수를 의미한다.
- 전역변수 앞에는 접근 제한자(public, protected, 생략, private) 사용가능 하다.
- 전역변수는 객체가 생성되는 시점에 자동초기화 된다.
정수형 : 0 실수형 : 0.0 문자형 : '    ' 논리형 : false
객체형 : String ⇨ null

static

① class

class 앞에는 올 수 없다. (단, Inner class에는 사용가능)

ex) static class Test{ } ⇒ X

② variable

㉠ 전역변수 앞에만 사용가능 ex) static int i;

㉡ 객체생성 없이 외부에서 **class이름.변수이름** 호출가능

㉢ static변수는 같은 class들이 공유하는 공유변수이다.

③ Method

㉠ 객체생성 없이 **class이름.mehtod이름([값,값,...]);** 호출가능

㉡ static method는 일반(static이 없는) method 호출 안됨.

㉢ static method는 static method만 호출 가능.

㉣ static method안에서 this 키워드 사용 안됨.

⇒ 같은 class 내에 static method 호출할 때 => **method이름([값, ...]);**

static

④ static 블록

ex) static{

 기능 구현 ;
}

- ⇒ class 내부에 선언되어 main method보다 먼저 실행된다.
- ⇒ method, 생성자 등의 영역 안에서는 선언 될 수 없다.

OverLoading

- ① 하나의 class 내부에 method이름이 같은 method 여러 개 있는 것
- ② 하나의 method이름을 가지고 기능을 다르게 구현하는 것 ⇨ 이용자 편의
- ③ 작성규칙
 - modifier(접근제한자) 같아도 달라도 상관없음.
 - returnType 같아도 달라도 상관없음.
 - method이름 반드시 같아야 한다.
 - 단, 인수는 무조건 인수의 순서 or Type or 개수가(셋 중하나가) 달라야 한다.

Array(배열)

특징

- ① Data를 저장하는 공간
- ② 하나의 이름으로 여러 개의 값 저장(공간을 나누어서 번지수로 구분)
- ③ 한가지 Type만 저장가능.
- ④ 배열은 객체이다. ⇨ 생성해서 사용한다(new)
- ⑤ 크기변경 안됨
- ⑥ **배열이름.length ⇨ 배열의 길이(크기)**
- ⑦ 배열의 번지수는 0부터 시작

Array(배열)

☞ Array에서 자주 발생하는 Exception(컴파일은 되지만 실행도중 발생)

① `ArrayIndexOutOfBoundsException`

=> 배열의 번지수를 벗어 났을 때 발생

ex) `int a [] = new int [2] ;`

=> `a[0]` , `a[1]` , `a[2]` - 오류

② `NullPointerException`

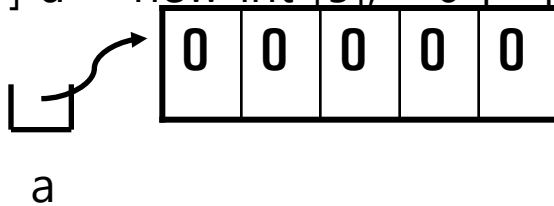
=> 생성되지 않은 배열을 접근(사용)할 때 발생

Array 선언

① 1차원 배열 선언

㉠ `DataType [] 배열이름 = new DataType [개수] ;`

ex) `int [] a = new int [5];` ⇨ 0부터 5까지 다섯 개의 공간을 만듦



중요

▪ 배열의 값 변경

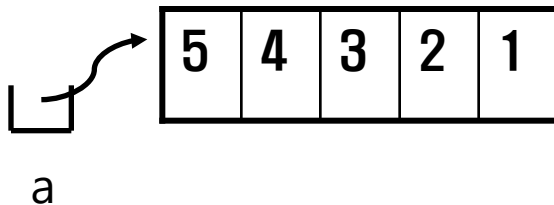
`배열이름[번지수] = 값 ;`

▪ 배열의 값 출력

`System.out.print(배열이름[번지수]);`

㉡ `DataType [] 배열이름 = { 값, 값, 값, ... } ;`

ex) `int a [] = { 5, 4, 3, 2, 1 } ;`

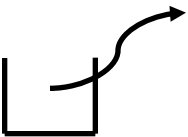


Array 선언

② 2차원 배열 선언

㉠ `DataType [] [] 배열이름 = new DataType [개수] [개수] ;`
ex) `int [] [] a = new int [3] [4];` ⇨ 3행 4열(12개 공간)

중요



	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

a

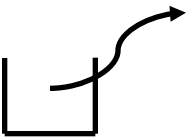
- 배열의 값 변경
`배열이름[행번지수] [열번지수] = 값 ;`
- 배열의 값 출력
`System.out.print(배열이름[행번지수] [열번지수]);`
- `배열이름.length` => 행의 길이
- `배열이름[행번지수].length` => 행번지수의 열의 길이

Array 선언

② 2차원 배열 선언

Ⓛ DataType [] [] 배열이름 = { {값,...} , {값,...} , {값,...} , ... } ;
ex) int a [] [] = { {1,2,3,4} , {2,4,6,8} , {3,6,9,12} }

중요



	0	1	2	3
0	1	2	3	4
1	2	4	6	8
2	3	6	9	12

a

▪ 배열의 값 변경

배열이름[행번지수] [열번지수] = 값 ;

▪ 배열의 값 출력

System.out.print(배열이름[행번지수] [열번지수]);

▪ 배열이름.length => 행의 길이

▪ 배열이름[행번지수].length => 행번지수의 열의 길이

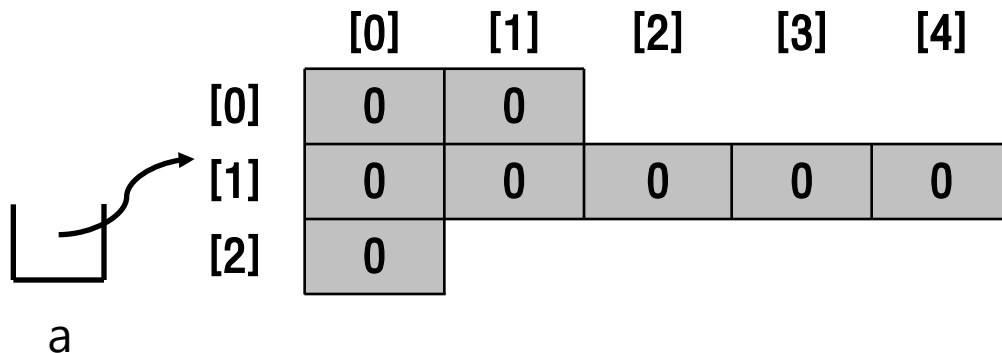
Array 선언

② 2차원 배열 선언

⊖ `DataType [] [] 배열이름 = new DataType [개수] [생략] ;`
ex) `int [] [] a = new int [3] [] ;` => 행만 생성되고 열 생성 안된 상태

☞ **열 생성 필수!**

`a[0] = new int [2];` ⇨ 1행 2열
`a[1] = new int [5];` ⇨ 2행 5열
`a[2] = new int [1];` ⇨ 3행 1열



Constructor(생성자)

- ① 특별한 method이다.
- ② 반드시 method이름이 class이름과 같다. (method이름이 대문자로 시작)
- ③ 일반적인 method선언과 같지만 returnType 자리가 없다.
⇒ `modifier class이름([dataType 변수이름, ...]){ }`
- ④ 객체가 생성(new)되는 시점에 딱 한번 호출한다.
- ⑤ java의 모든 객체는 반드시 한 개 이상의 생성자를 갖는다.
- ⑥ Overloading이 가능하다.
- ⑦ 프로그래머가 생성자를 하나도 작성하지 않으면 default 생성자가 만들어진다.
⇒ `modifier class이름(){ }`

Constructor(생성자)

객체가 생성되는 시점에 하는 일

- ㉠ 전역변수를 0에 준하는 값으로 초기화 ex) `int i;` ⇨ 0
- ㉡ 전역변수를 명시적 초기화 (프로그래머가 직접 값을 입력) ex) `int i=5;`
- ㉢ 생성자 구현부 실행

같은 class내에서 다른 생성자 호출 방법

`this([값, 값, ...]);` ⇨ 반드시 생성자 구현부 첫 번째 줄에서만 가능

UML(class Diagram)

class명

멤버 변수 list

생성자 list
메소드 list

- : **private**
~ : default
: protected
+ : **public**

People

-name : String
-age : int
-address : String

<<create>>

+People()

+People(name:String, age:int, address:String)

+getName() : String

+setName(name:String)

+getAge() : int

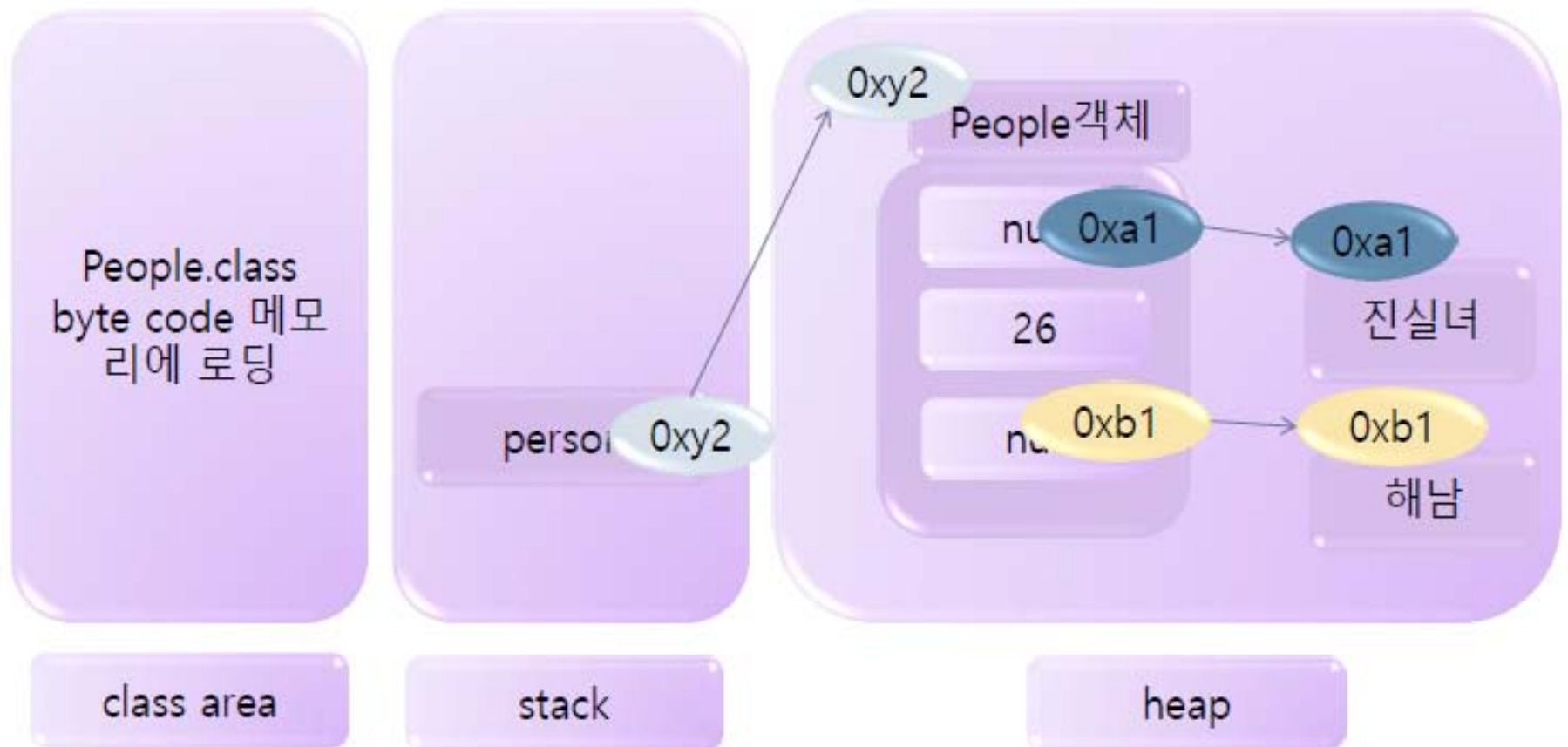
+setAge(int age)

+getAddress() : String

+setAddress(address : String)

객체 생성되어 저장된 JVM구조

```
People person = new People("진실녀", 26, "해남");
```



this

this 기능 3가지

㉠ this.변수이름 ; ⇨ 전역변수

㉡ this.methos이름([값, 값, ...]); ⇨ 하나의 class내에서 다른 method호출

㉢ this([값, 값, ...]);

⇨ 같은 class내에서 다른생성자 호출 (생성자 구현부 첫 줄에서만 사용)

정보 은닉

1. Data(속성)를 외부의 접근으로부터 보호하기 위한 방법
2. 코드의 유지보수를 용이하게 함
3. 구현 방법
 1. 변수 : **private** access modifier 활용
 2. **public** 메소드로 변수 활용
 3. Setter메소드
 1. 값을 받아 저장하는 역할
 2. 잘못된 값에 대한 유효성 체크 로직 이용 가능
 3. `setXxx()` (Xxx : 변수이름)
 4. Getter메소드
 1. 값을 조회하는 역할
 2. `getXxx()` (Xxx : 변수이름)

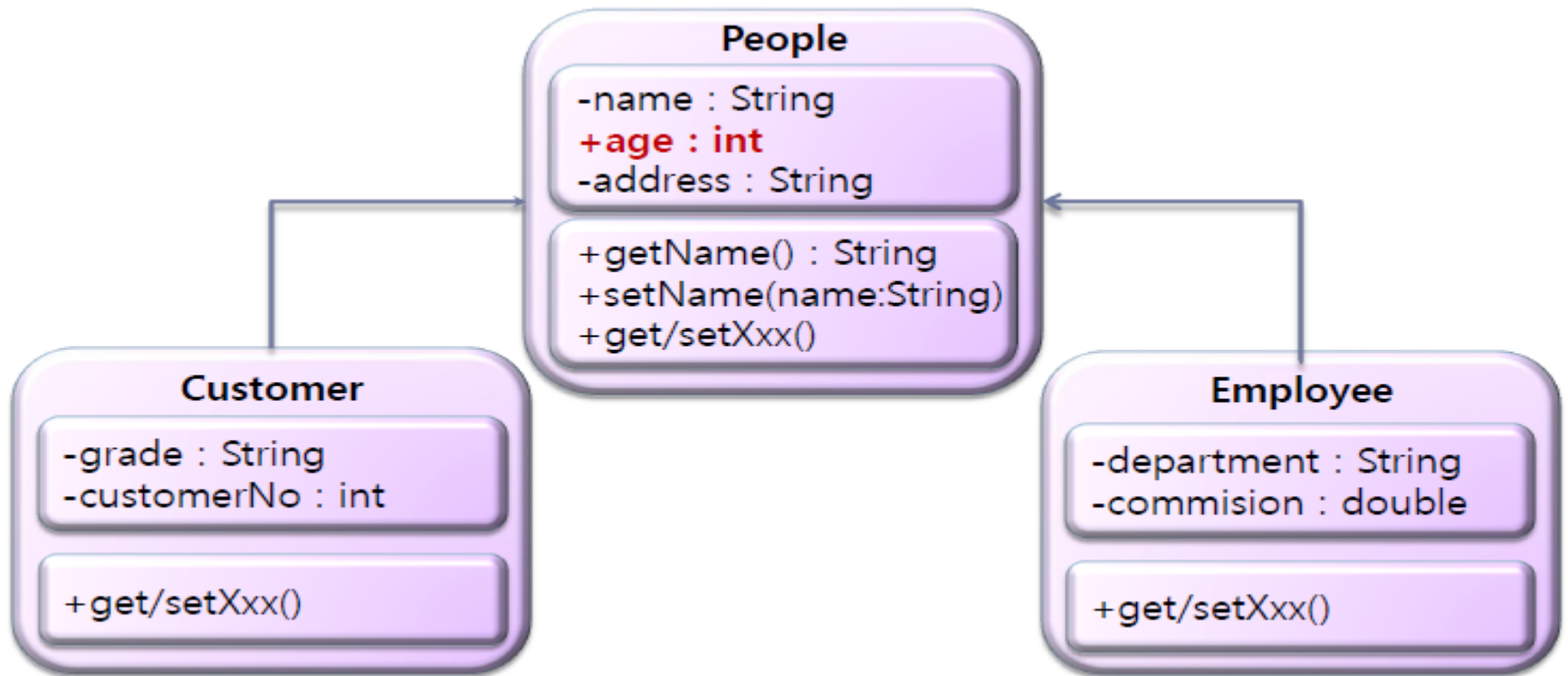
정보 은닉

```
3 public class People {
4     private String name;
5     private int age;
6     private String address;
7
8     public void setAge(int age) {
9         if( age > 0){
10             this.age = age;
11         }else{
12             System.out.println("부적합한 데이터입니다.");
13         }
14     }
15
16     public int getAge() {
17         return age;
18     }
```

Inheritance(상속)

- ① 상속이란 부모(Super) class의 속성(전역변수 = Field)과 method를 상속 받는 것.
- ② 부모 class의 생성자와 private 요소를 제외한 모든 것을 상속받는다.
- ③ 부모 class의 method와 속성을 별도의 선언 없이 내안에 있는 것처럼 접근하여 사용한다.
- ④ 단일상속만 가능
- ⑤ extends 키워드 사용
ex) **class A extends B { } => A가 B를 상속 받는다.**
- ⑥ java의 모든 class는 Object(class)를 상속받는다. (java의 최고 조상)

상속



고객정보[등급:VVIP, 고객번호:1,이름:장희정,나이:40, 주소: 서현동]
직원정보[부서:교육부 ,커미션:80.0,이름:판매왕,나이:30, 주소: 야탑동]

Polymorphism(다형성)

① 반드시 상속관계일 때 성립

② class A extends B{ } 일 때 A를 A라 부를 수 있고 A를 B라 부를 수 있다.

=> class A extends B{ }일 때

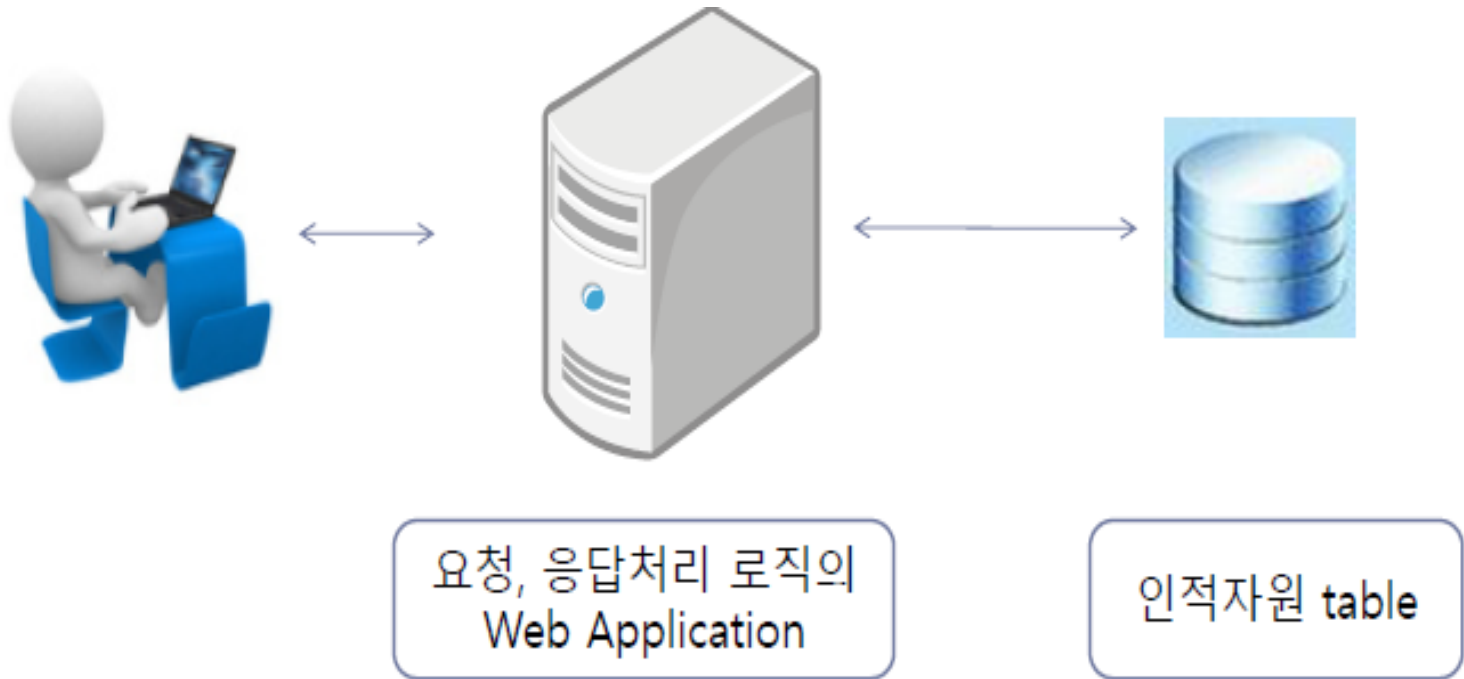
⊐ A a = new A(); ⇨ O

⊐ B b = new A(); ⇨ O

⊐ B c = new B(); ⇨ O

⊐ A d = new B(); ⇨ X

Polymorphism(다형성)

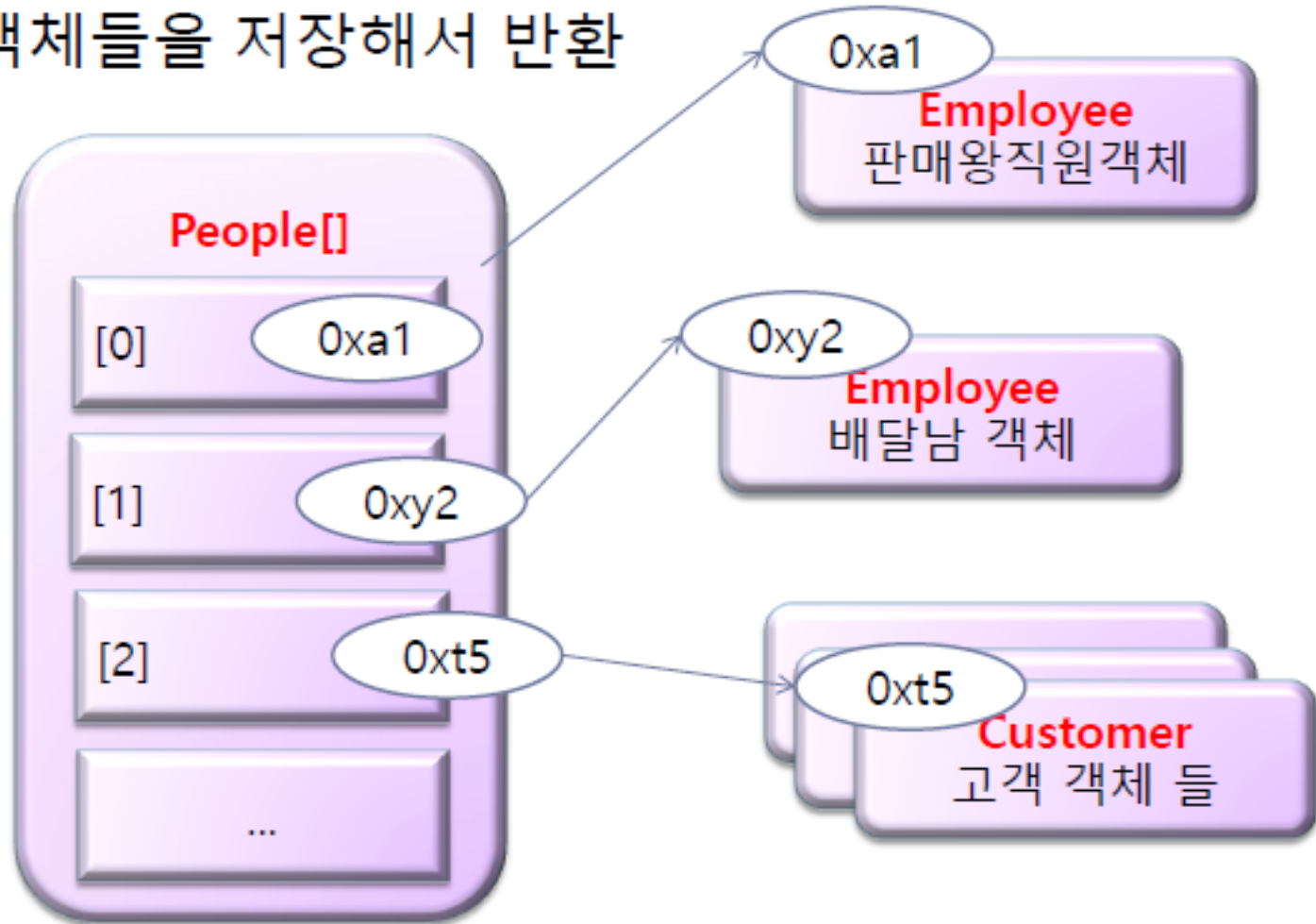


요청 로직 - 쇼핑물 모든 고객 및 직원 정보를 요청하는 상황

고객들과 직원들을 어떻게 단 하나의 로직으로 한번에 제공 즉 반환?
고객, 직원 모두 저장 가능한 타입은?

Polymorphism(다형성)

배열에 객체들을 저장해서 반환



Polymorphism(다형성)

1. 형태가 여러 개인 특성
2. Object Polymorphism
 1. 상위타입의 참조변수로 하위타입의 객체를 참조 할 수 있음
3. 기본 Syntax
 1. 부모타입 변수 = 자식객체;
4. 예
 1. `People [] peopleAll = new People[5];`
 2. `peopleAll[0] = new Customer("고객왕", 50,...);`
 3. `peopleAll[1] = new Employee("영업맨", 30,...);`
 4.

Polymorphism(다형성)

1. People p = new Customer("고객왕", 50,...);
2. p 변수의 제약 조건
 1. p.setGrade("VIP"); 오류
 2. Customer 클래스만이 보유한 자식 멤버 변수, 메소드 호출 불가
 3. 왜? 부모타입 변수엔 은닉
3. 해결책 – 객체타입 형변환 필요
 1. Customer c = (Customer)p;
 2. c.setGrade("VIP");

Polymorphism(다형성)

1. 메소드나 생성자의 매개변수에 다형성을 이용하면 여러 타입의 객체를 매개변수로 받아 처리할 수 있는 메소드로 사용 가능

```
method(new Customer());  
method(new Employee());
```

```
public void method(People e){  
  
    People 및 People하위 타입들에  
    관련된 문장 수행  
  
}
```

다형성이 반영된 API

boolean	<code>contains(Object o)</code> Returns true if this list contains the specified element.
void	<code>ensureCapacity(int minCapacity)</code> Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
E	<code>get(int index)</code> Returns the element at the specified position in this list.
int	<code>indexOf(Object o)</code> Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.

Overriding(재정의)

상속관계에서 Super(부모)class에 정의되어 있는 method를 자식(Sub)class에서 재정의 하는 것.

재정의 방법

- ㉠ modifier의 접근제한자는 반드시 부모 class의 제한자 보다 크거나 같아야 한다.
 - ⇒ 접근제한자 이외의 modifier(abstract)는 같아도 달라도 상관없다.
- ㉡ returnType 무조건 같다.
- ㉢ method이름, 인수 무조건 같다.
- ㉣ 기능을 다르게 만든다.
 - ⇒ 즉, 부모 class의 method 기능이 마음에 들지 않아 다른 기능으로 재정의.

super

super 기능 3가지 (자식class에서 부모를 칭함)

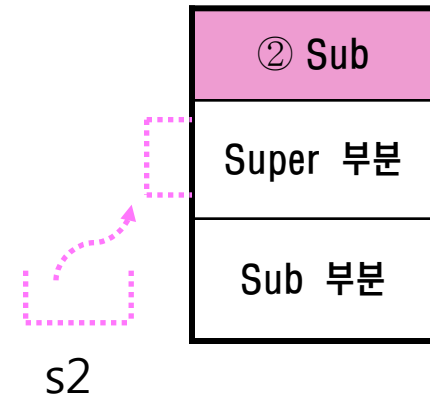
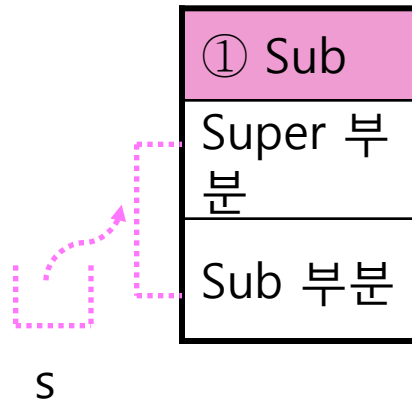
- ㉠ super.변수이름 ⇨ 자식class에서 부모 class의 전역변수 호출
자식class의 변수이름과 부모class의 변수이름 같을 때 구분
- ㉡ super.method이름([값, 값, ...]); ⇨ 자식class에서 부모class의 method호출
- ㉢ super([값, 값, ...]);
⇨ 자식 class의 생성자구현부 첫 번째 줄에서 부모 class의 생성자 호출

ObjectCasing

```
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 접근가능)

Sub s3 = (Sub)s2 ; => ObjectCasing

상속관계 일 때 생성자 개념

- ① 자식 class가 생성 될 때 부모 class의 기본 생성자 호출된다.
- ② 자식 class가 인수가 있는 생성자를 호출하더라도 부모 class의 기본 생성자가 필요하다.
 - ex) new Sub(5); 처럼 호출하더라도 super(); (기본생성자)가 호출된다.
 - ⇒ 자식 생성자의 구현부 첫 번째 줄에서 super(); 생략된 것과 같다.
- ③ 자식 class에서 인위적으로 super(값, 값, ...); 호출하면 부모의 기본생성자 필요 없다.
 - ⇒ 단, 모든 자식생성자에서 super(값, 값, ...); 호출되어 있어야 문법에 오류가 없다.
- ④ 자식 생성자 구현부 첫 번째 줄에서 this(값, 값, ...);은 호출하면 먼저 호출된 자식 생성자에 가서 이동한 자식 생성자 구현부 첫 번째 줄에서 super(); 호출된다.

package

package

- ① ~.java 문서의 첫 번째 줄에 한 개 선언가능
- ② ~.java 문서 컴파일 시 생성되는 class를 선언된 package 폴더에 생성하도록 하는것

Test.java

```
package jang.hee.jung ;  
class Test{  
  
}
```



① 컴파일 방법

```
javac -d 기준디렉토리 Test.java
```

② 실행 방법

```
java jang.hee.jung.Test
```

import

import

- ① ~.java 문서 안에 첫 번째 줄에 작성하여 여러 개 선언가능
(package 함께 선언 될 때에는 package 다음 문장에 선언)
- ② 다른 영역(폴더)에 있는 class를 쓰겠다고 선언 하는 것.

예)

```
import java.awt.Frame;
import java.util.*;
class Test{
    public static void main( String [ ] args ){
        Frame f = new Frame();
        ArrayList list = new ArrayList();
        java.io.File file = new java.io.File("aa.txt");
    }
}
```

Instanceof(연산자)

⇒ **결과값은 boolean형**

- ① 반드시 상속관계일 때 사용한다. (아니면 컴파일 오류)
- ② Object변수 instanceof ObjectType
⇒ 왼쪽의 변수가 오른쪽의 Type이 될 수 있느냐?
- ③ 왼쪽의 변수형이 오른쪽의 Type보다 서브 class일 때 true이다. (왼쪽 < 오른쪽)

final

① variable

변수 앞에 final이 오면 상수가 된다. (변화하지 않는 값)

- ⇒ 값 변경 불가, 상수는 모두 대문자로 표현(권장사항)
- ⇒ 반드시 초기화해야 한다. (전역 변수도 초기화해야 함)

② Method

method 앞에 오면 overriding(재정의) 불가

ex) public final void aa(){ }

③ class

class 앞에 오면 상속 안됨 (객체 생성은 가능)

ex) final class Test{ }

abstract

① variable

변수 앞에 올 수 없다.

② Method

- method 앞에 abstract 붙으면 선언부만 있고 구현부 없다. (기능 없다.)
ex) `public abstract void aa();` ⇨ ;으로 끝
서브 class에서 재정의 하기 위해 존재
- abstract method를 가지고 있는 class는 반드시 abstract class로 선언해야 한다.

③ class

- abstract class는 생성할 수 없다. (new 사용 안됨)
 - 다른 class의 부모(Super)가 되기 위해 존재
 - 상속관계에 있을 때 superclass가 abstract method를 가지고 있으면 서브class에서 abstract method 재정의 해야 한다.
 - ⇨ 만약, 재정의 하지 않으면 서브class는 abstract으로 선언되어야 하며 객체생성 할 수 없다.
- ex) `abstract class A{ }`

interface

- ① class와 유사하지만 class가 아니다. ⇨ 상속X, 생성X
ex) interface A{ }
- ② interface를 구현(implements)하여 **다중상속 같은 효과 얻음.**
- ③ interface의 모든 변수는 **public static final**이다. (상수- 변하지 않는다.)
- ④ interface의 모든 method는 **public abstract**이다. ⇨ 기능X
- ⑤ 위의 ③,④번으로 선언되어 있지 않더라도 무조건 interface 안의 변수는 상수, method는 abstract method이다.
- ⑥ interface는 생성 할 수 없지만 Type(자료형)으로 사용가능
- ⑦ interface는 다른 interface를 상속 받을 수 있으나 class는 상속 받을 수 없다.
- ⑧ class가 interface를 implements하게 되면 **interface에 있는 모든 method 재정의 해야 한다.** 안 하면 abstract class로 선언되어야 한다.
- ⑨ 만약, interface가 interface를 상속 받았고, 서브 interface를 class가 implements 하면
서브, 슈퍼 interface에 있는 모든 method class안에서 재정의 해야 한다.

equals 메소드

- ① equals method는 Object class에서 정의되어 String등의 여러 class에서 Overloading(재정의)되어 있다.
- ② primitiveType은 ==으로 비교하면 값을 비교한다.
- ③ Objectype은 ==으로 비교하면 주소값을 비교한다.
- ④ Object class의 equals는 ==과 같은 역할을 한다. (주소값 비교)
- ⑤ String class의 equals는 ==과 다른 역할을 한다. (값 비교)

중
요

```
boolean b = 주체.equals(Object obj);
```


toString 메소드

- ① Object(객체)를 String으로 변환하는 메소드
- ② Object class의 toString() 메소드는 주소값을 리턴 함.
- ③ WrapperClass , String Class 등이 toString()메소드를 재정의 하여 주소값이 아닌 값을 리턴 함.
- ④ System.out.print(객체) ; 일때 , 인수 객체가 오면 객체.toString() 호출됨.

중요

Object 클래스 변수를 출력하면 주소 값이 출력되지만 String 클래스의 변수를 출력하면 값이 출력됨.

예)

```
Test t = new Test("Jang");    String s = new String("Jang");  
System.out.println( t ); => 주소값출력  
System.out.println( s ); => Jang 출력
```

❖Exception의 개념

잘못된 코드, 부정확한 데이터, 예외적인 상황에 의하여 발생하는 오류

예외(Exception)

자바 프로그램에서 **실행 중**에 발생할 수 있는 경미한 오류를 예외
적절한 처리 모듈을 추가하여 발생한 문제를 복구 가능

IndexOutOfBoundsException과 같이 ○○○ **Exception** 형태

에러(Error)

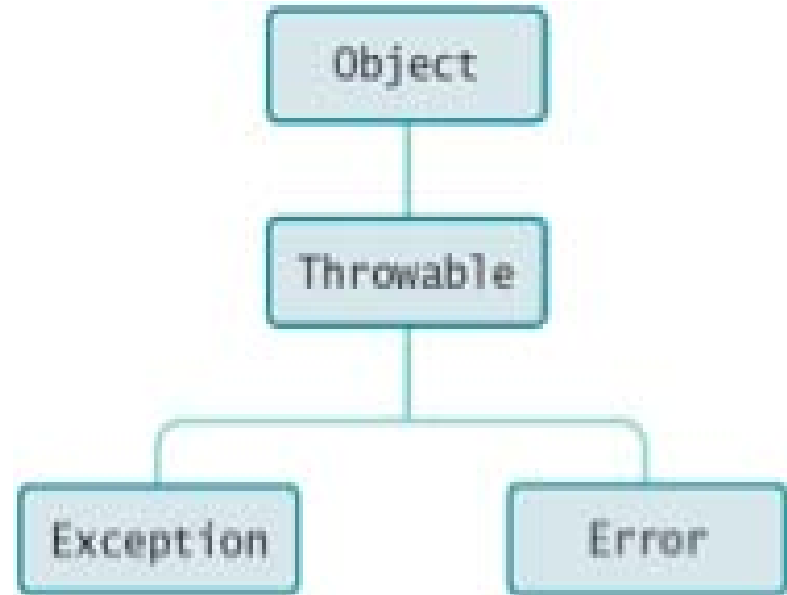
메모리나 내부의 심각한 문제

복구가 불가능한 오류인 **OutOfMemoryError, InternalError** 등

❖Exception의 개념

에러와 예외를 모두 객체로 만들어 처리, 관련 클래스 계층 구조
클래스 **Throwable**

하부로 예외인 **Exception** 클래스와 에러인 **Error** 클래스
Exception 클래스 하부
다양한 예외를 위한 클래스

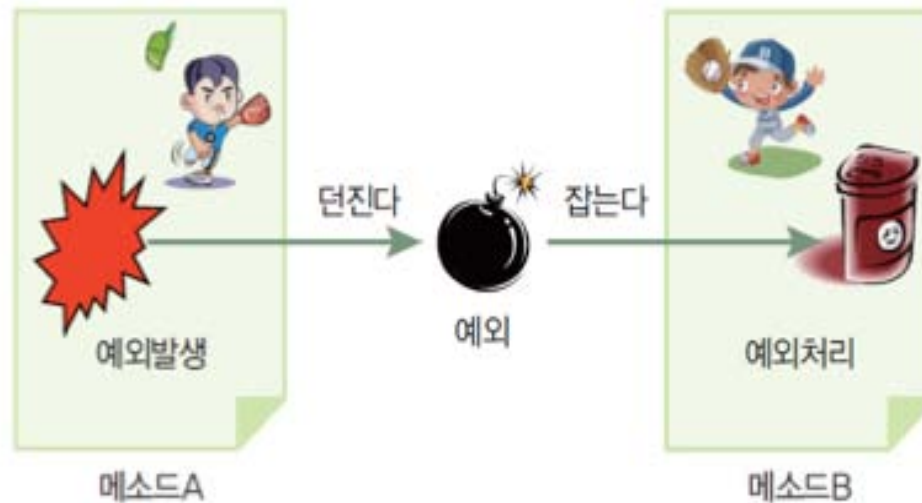


❖Exception의 처리

실행 중에 여러 이유로 예상하지 못했던 문제가 발생한 경우 이를 적절히 처리하는 모듈

try ~ catch ~ finally 문장

- 예외 발생과 상관없이 **finally**의 블록은 실행
- **catch**와 **finally** 둘 중 하나는 옵션



Exception

❖ Exception의 처리

```
try {
```



```
}
```

```
catch {
```



```
}
```

try 블록에서
오류가 발생하면
처리합니다.

```
try {
```

```
    // 예외가 발생할 수 있는 코드
```

```
} catch (예외종류 참조변수) {
```

```
    // 예외를 처리하는 코드
```

```
}
```

```
} finally {
```

```
    // 여기 있는 코드는 try 블록이 끝나면 무조건 실행된다. ← 생략이 가능하다.
```

```
}
```

Exception

❖ Exception의 처리

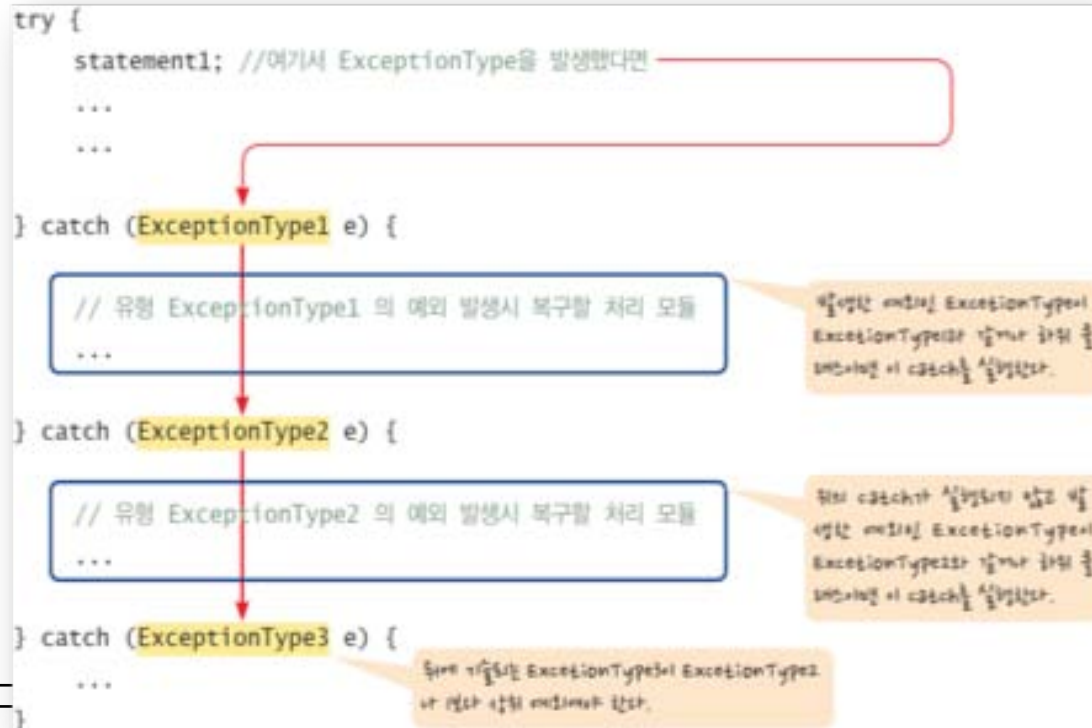


❖Exception의 처리

여러 개의 catch 문을 이용하는 경우

위에서부터 순차적으로 catch 문에서 ExceptionType 인자 유형을 검사
발생된 예외의 유형과 일치하거나 하위 클래스이면 먼저 만나는
catch문만을 실행

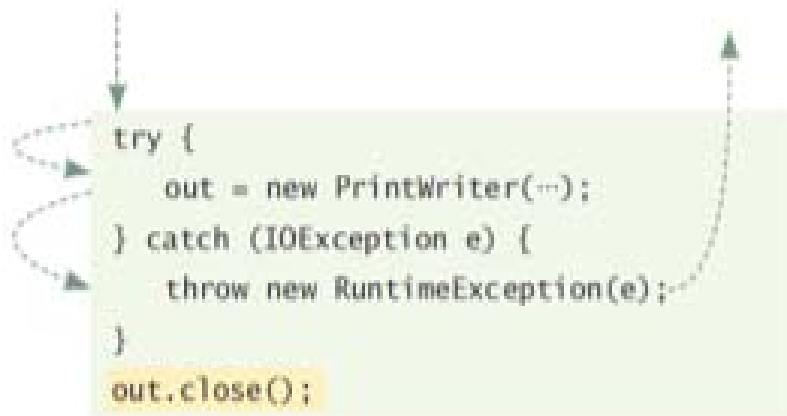
- > 여러 개의 catch 구문에 기술하는 ExceptionType이
하위 클래스부터 먼저 catch문 작성



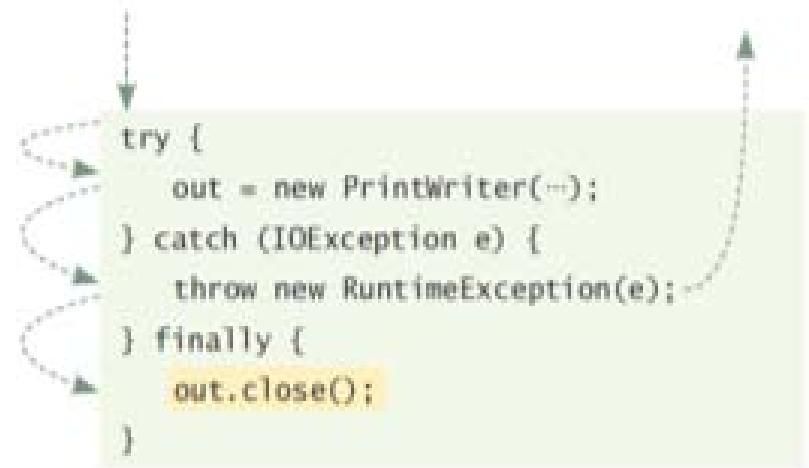
❖Exception의 처리

finally 블록

오류가 발생하였건 발생하지 않았건 항상 실행되어야 하는 코드

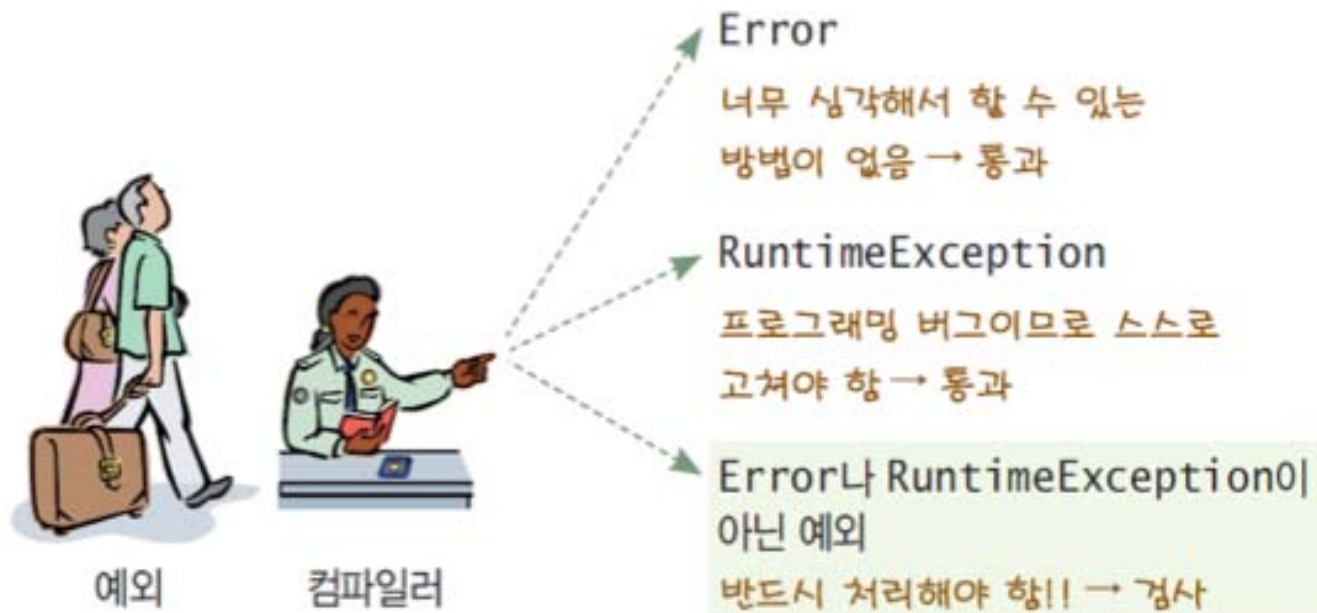


예외가 발생하면 자원이 반납되지 않을 수 있다.



예외가 발생하더라도 확실하게 자원이 반납된다.

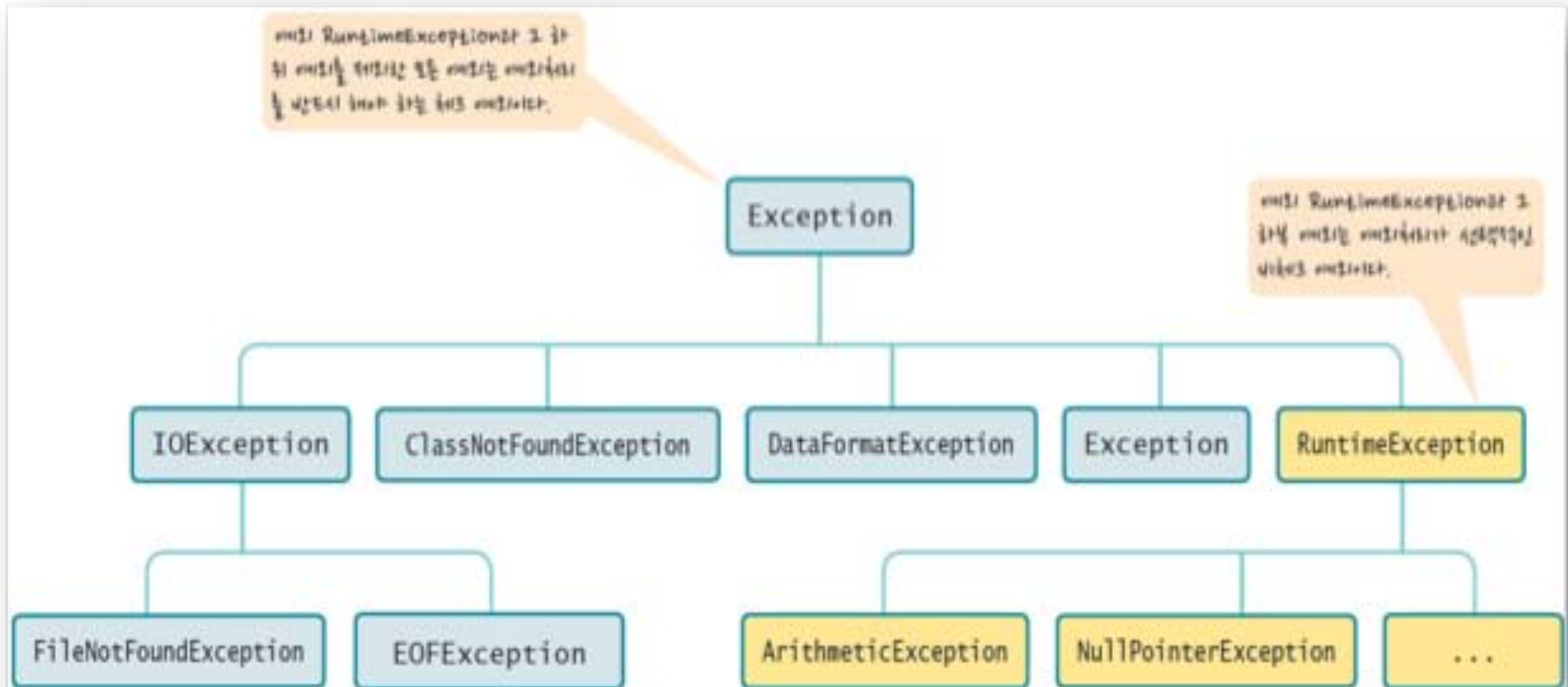
❖Exception의 종류



❖Exception의 종류

RuntimeException

프로그래밍 버그나 논리 오류에서 기인



❖Exception의 종류

RuntimeException

분류	예외	설명
RuntimeException	ArithmeticException	어떤 수를 0으로 나눌 때 발생한다.
	NullPointerException	널 객체를 참조할 때 발생한다.
	ClassCastException	적절치 못하게 클래스를 형변환하는 경우
	NegativeArraySizeException	배열의 크기가 음수값인 경우
	OutOfMemoryException	사용 가능한 메모리가 없는 경우
	NoClassDefFoundException	원하는 클래스를 찾지 못하였을 경우
	ArrayIndexOutOfBoundsException	배열을 참조하는 인덱스가 잘못된 경우

❖Exception의 종류

비체크 예외(unchecked exception)

RuntimeException과 그 하부 예외가 발생할 가능성이 있는 코드들은 try ~ catch 문이 선택적

기타 예외, 체크 예외(checked exception)

Error와 RuntimeException을 제외한 나머지 예외 회복할 수 있는 예외이므로 프로그램은 반드시 처리

❖Exception의 종류

Exception 처리 방법

1) try~catch 구문 사용 방법

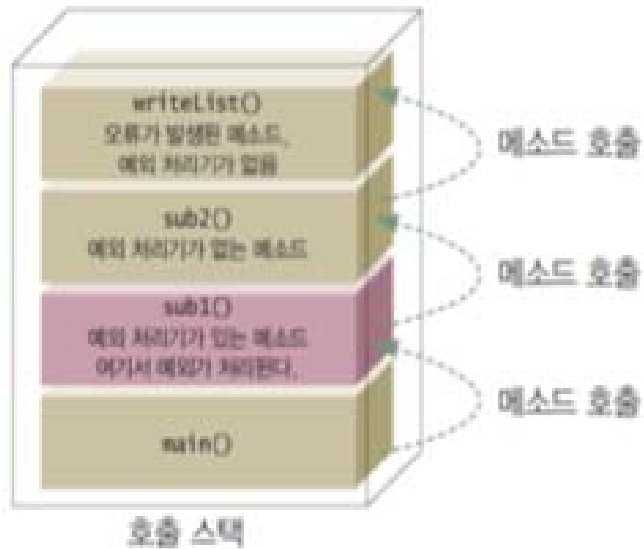
2) throws

- 예외가 발생할 수 있는 구문이 속한 메소드에서 다시 예외를 전달하는
 - 상위 메소드로 예외처리를 미루는 방법

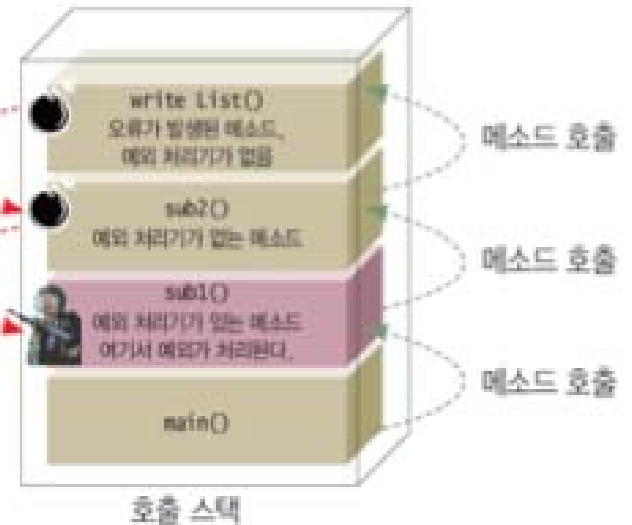
1. Exception

❖Exception의 처리과정

호출 스택을 거슬러가면서 예외 처리기가 있는 메소드를 찾음

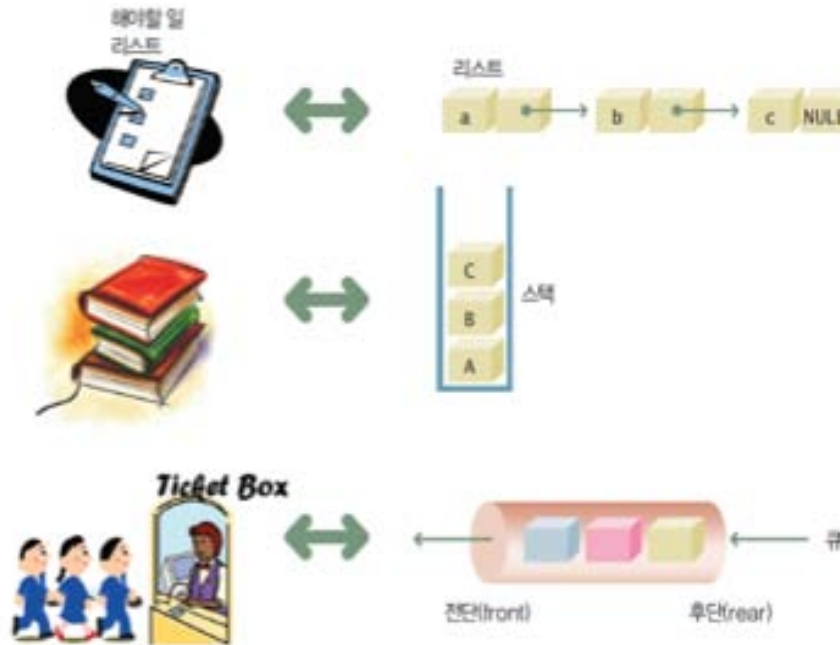


예외가 발생한다.
예외가 전달된다.
예외를 잡는다.



❖Collection의 개념

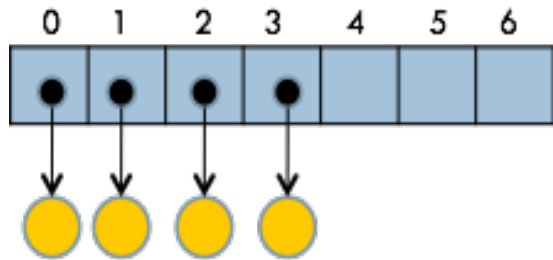
자바에서 자료 구조를 구현한 클래스
다양한 객체들을 삽입, 삭제, 검색 등 관리하는데 사용됨
요소(element)라고 불리는 가변 개수의 객체들의 모음
리스트(list), 스택(stack), 큐(queue), 집합(set), 해시 테이블(hash table) 등



❖ Collection의 개념

배열 vs 컬렉션

배열(Array)



컬렉션(Collection)



고정 크기 이상의 객체를 관리할 수 없다.

배열의 중간에 삽입, 삭제가 어렵다.

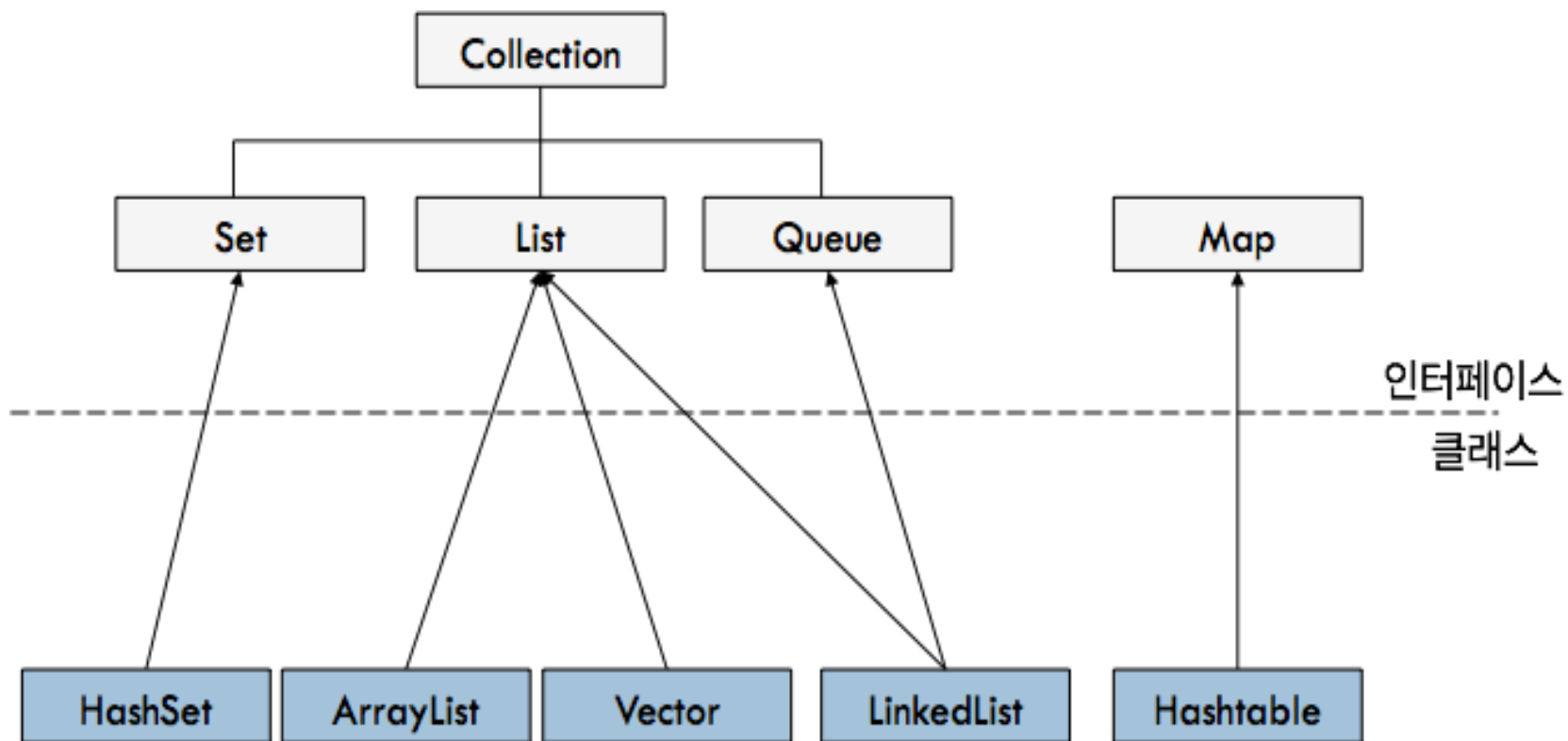
가변 크기로서 객체의 개수를 염려할 필요 없다.

배열의 중간에 객체가 삽입, 삭제가 쉽다.

Collection Framework

❖ Collection의 개념

Collection 종류



❖Collection의 개념

Collection 종류

인터페이스	설명
Collection	모든 자료 구조의 부모 인터페이스로서 객체의 모임을 나타낸다.
Set	집합(중복된 원소를 가지지 않는)을 나타내는 자료 구조
List	순서가 있는 자료 구조로 중복된 원소를 가질 수 있다.
Map	키와 값들이 연관되어 있는 사전과 같은 자료 구조
Queue	극장에서의 대기줄과 같이 들어온 순서대로 나가는 자료구조

Collection Framework

❖Collection의 개념

분류	메소드	설명
기본 연산	<code>int size()</code>	원소의 개수 반환
	<code>boolean isEmpty()</code>	공백 상태이면 true 반환
	<code>boolean contains(Object obj)</code>	obj를 포함하고 있으면 true 반환
	<code>boolean add(E element);</code>	원소 추가
	<code>boolean remove(Object obj)</code>	원소 삭제
	<code>Iterator<E> iterator();</code>	원소 방문
벌크 연산	<code>boolean addAll(Collection<? extends E> from)</code>	c에 있는 모든 원소 추가
	<code>boolean containsAll(Collection<?> c)</code>	c에 있는 모든 원소가 포함되어 있으면 true
	<code>boolean removeAll(Collection<?> c)</code>	c에 있는 모든 원소 삭제
	<code>void clear()</code>	모든 원소 삭제
배열 연산	<code>Object[] toArray()</code>	컬렉션을 배열로 변환
	<code><T> T[] toArray(T[] a);</code>	컬렉션을 배열로 변환

❖Collection의 종류

Collection

컬렉션에 대해 연산을 수행하고 결과로 컬렉션을 반환

주요 메소드

컬렉션에 포함된 요소들을 소팅하는 `sort()` 메소드

요소의 순서를 반대로 하는 `reverse()` 메소드

요소들의 최대, 최소값을 찾아내는 `max()`, `min()` 메소드

특정 값을 검색하는 `binarySearch()` 메소드

모두 static 메소드

❖Collection의 종류

Vector

여러 객체들을 삽입, 삭제, 검색하는 컨테이너 클래스

배열의 길이 제한 단점 극복

원소의 개수가 넘쳐나면 자동으로 늘어나는 가변 길이

Vector에 삽입 가능한 것

객체, null도 삽입 가능

기본 데이터 타입은 불가능(Wrapper 객체로 만들어 삽입 가능)

Vector에 객체 삽입

벡터의 맨 뒤에 객체 추가 : 공간이 모자라면 자동 늘림

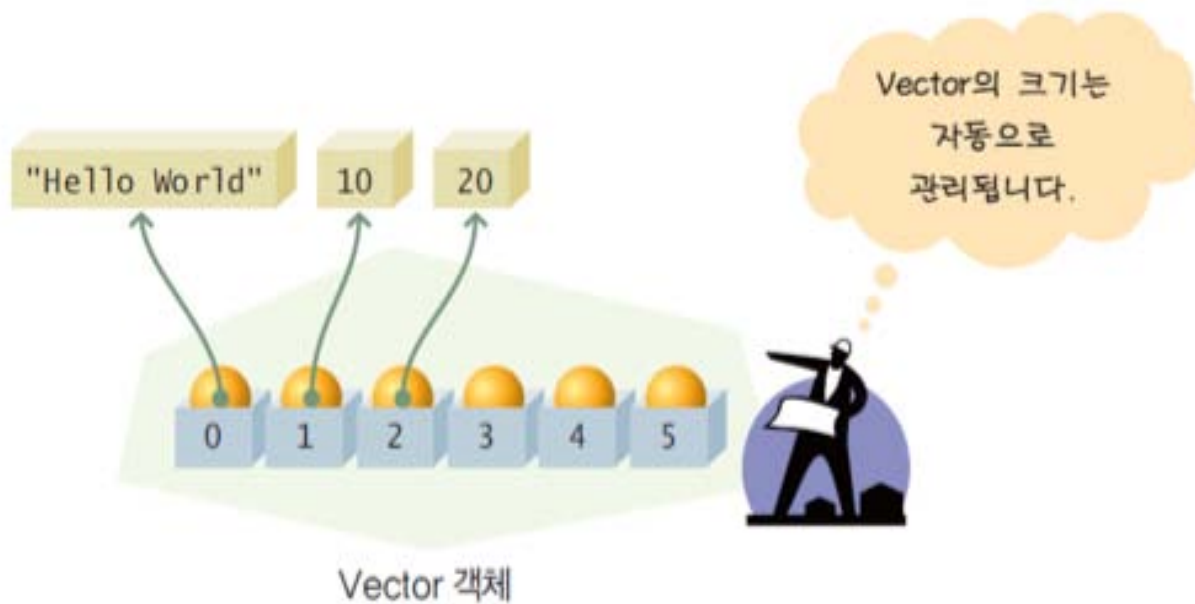
벡터 중간에 객체 삽입 : 삽입된 뒤의 객체는 뒤로 하나씩 이동

Vector에서 객체 삭제

임의의 위치에 있는 객체 삭제 가능 : 객체 삭제 후 하나씩 앞으로 자

❖Collection의 종류

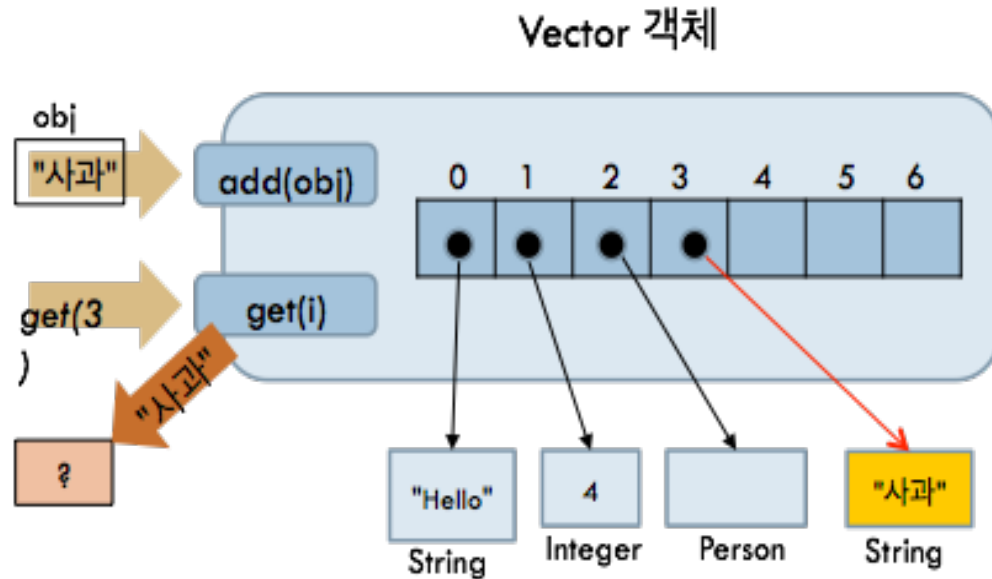
Vector



❖Collection의 종류

Vector

add()를 이용하여 요소를 삽입하고 get()을 이용하여 요소를 검색합니다



❖Collection의 종류

ArrayList

가변 크기 배열을 구현한 클래스

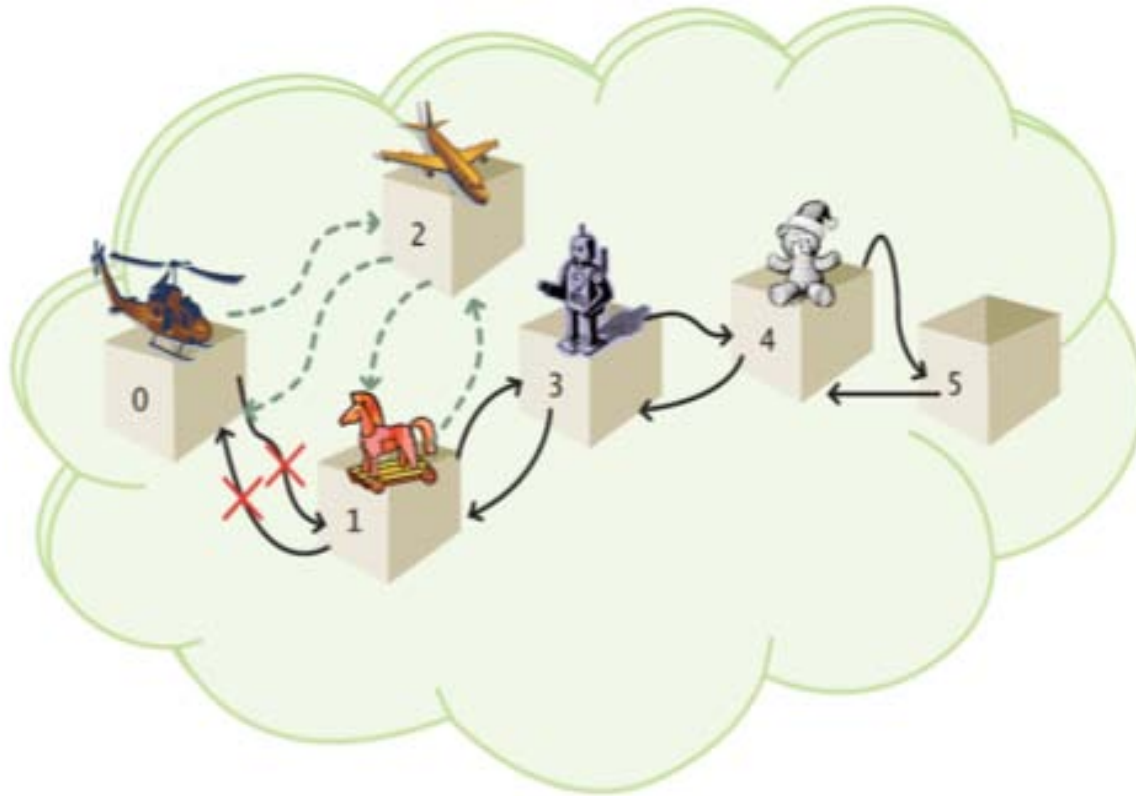
벡터와 거의 유사하며 자동으로 스레드 동기화를 지원하지 않는 점이 가장
다수의 스레드가 동시에 ArrayList에 접근할 때 ArrayList는 동기화
ArrayList에 접근하는 곳에서 스레드 동기화를 수행하여야 함



Collection Framework

❖Collection의 종류

LinkedList



❖Collection의 종류

Set

원소의 중복을 허용하지 않음

HashSet

해쉬 테이블에 원소를 저장하기 때문에 성능면에서 가장 우수
하지만 원소들의 순서가 일정하지 않은 단점이 있음

TreeSet

레드-블랙 트리(red-black tree)에 원소를 저장
값에 따라서 순서가 결정되며 하지만 HashSet보다는 느림

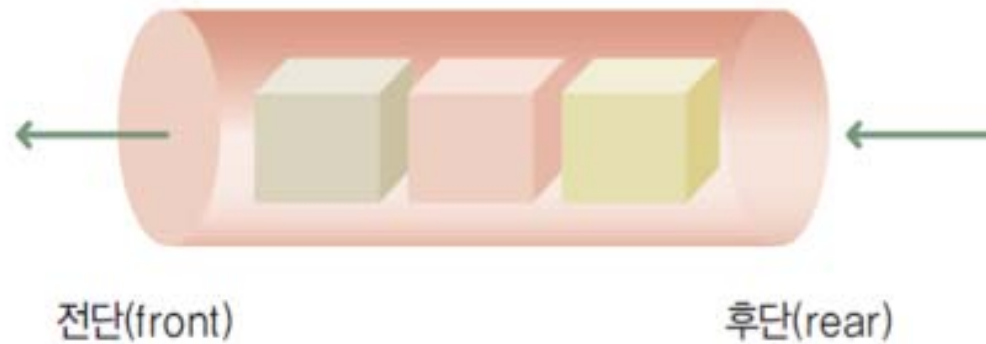
LinkedHashSet

해쉬 테이블과 연결 리스트를 결합한 것, 원소들의 순서는 삽입되었던

❖Collection의 종류

Queue

큐는 먼저 들어온 데이터가 먼저 나가는 자료 구조
FIFO(First-In First-Out)



❖Collection의 종류

Map

사전과 같은 자료 구조
키(key)에 값(value)이 매핑



Map

❖Generic의 개념

Generic

다양한 타입의 객체를 동일한 코드로 처리하는 기법
제네릭은 컬렉션 라이브러리에 많이 사용



❖Generic의 개념

Generic

제네릭 도입 이전에는 Object 클래스를 이용하여 모든 객체들을 참조 다운 캐스팅이 반드시 필요하며 이 부분이 문제를 발생

컴파일러는 elementAt이 Object 타입을 반환할 것이라는 밖에는 알지 못
프로그래머가 명시적으로 타입을 지정하여야 함

규모가 큰 프로그램의 작성 시 프로그래머가 타입을 실수로 잘못 지정하는
런타임에 ClassCastException이 발생

제네릭을 이용하면 컴파일러가 요소의 타입을 알 수 있어 타입 불일치를 방지

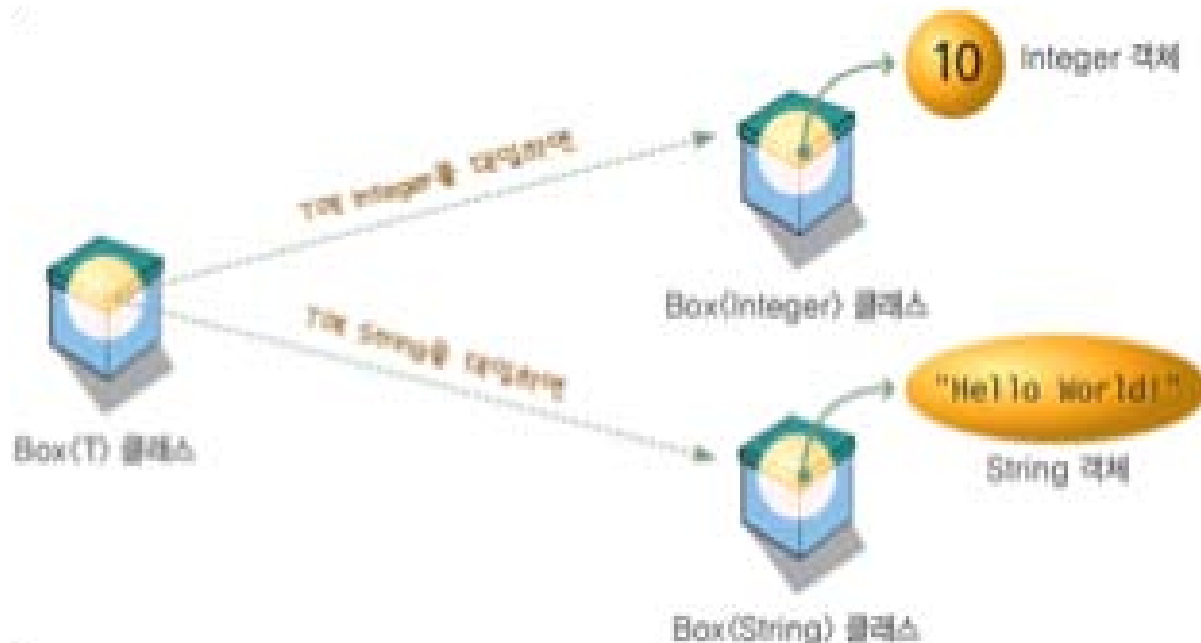
❖Generic의 개념

Generic

타입 매개변수

제네릭 클래스에서는 타입을 변수로 표시

타입 매개변수는 객체 생성 시에 프로그래머에 의하여 결정



❖Generic의 개념

Generic

타입 매개변수

'<'과 '>'사이의 문자로 표현

하나의 대문자를 타입 매개 변수로 사용

많이 사용하는 타입 매개 변수 문자

E : Element를 의미하며 컬렉션에서 요소를 표시할 때 많이 사용

T : Type을 의미한다.

V : Value를 의미한다.

K : Key를 의미

타입 매개변수가 나타내는 타입의 객체 생성 불가

ex) ~~T a = new T();~~

타입 매개 변수는 나중에 실제 타입으로 대체됨

어떤 문자도 매개 변수로 사용될 수 있음

❖Generic의 개념

Generic Method

타입 매개 변수를 이용하여 제네릭 메소드 정의 가능
타입 매개 변수의 범위가 메소드 내부로 제한

```
public class Array
{
    public static <T> T getLast(T[] a)
    {
        return a[a.length-1];
    }
}
```

←-----제네릭 메소드 정의

```
String[] language = { "C++", "C#", "JAVA" };
String last = Array.<String>getLast(language); // last는 "JAVA"
```

❖Iterator의 개념

**Vector, ArrayList, LinkedList와 같은 리스트 자료구조에서
요소를 순차적으로 검색할 때 Iterator 인터페이스를 사용
iterator() 메소드를 호출하면 Iterator 객체를 반환
Iterator 객체는 검색한 위치를 기억하고 있어 인덱스 없이 순차적 검색이**

메소드	설명
<code>boolean hasNext()</code>	다음 반복에서 사용될 요소가 있으면 <code>true</code> 반환
<code>E next()</code>	다음 요소 반환
<code>void remove()</code>	마지막으로 반환된 요소를 제거

3. Thread

❖ Thread의 개념

멀티 태스킹(multi-tasking)

여러 개의 애플리케이션을 동시에 실행하여서 컴퓨터 시스템의 성능을 높인다.

음악을 들으면서
운동을 할 수 있다.



인쇄를 하면서
문서 편집을 할 수 있다.

3. Thread

❖ Thread의 개념

다중 스레딩(multi-threading)

하나의 프로그램이 동시에 여러 가지 작업을 할 수 있도록 하는 것
각각의 작업은 스레드(thread)라고 함



3. Thread

❖ Thread의 개념

프로세스 vs 스레드

1) 프로세스

실행되고 있는 프로그램

프로세스마다 고유한 저장공간을 사용하며 독립적으로 실행

단점

여러 프로세스를 실행하려면 프로세스 간의 정보 교환에 많은 시간이

2) 스레드(Thread)

가벼운 프로세스(light-process)로,

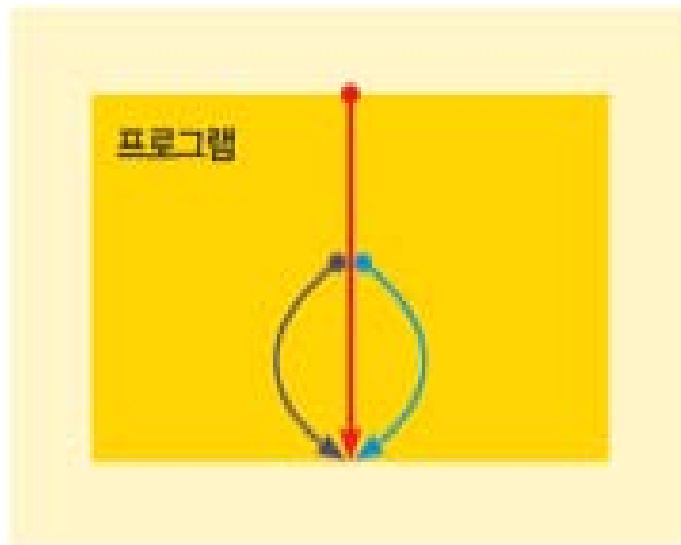
하나의 프로그램 내부에서 실행되는 스레드는 프로세스보다 오버헤드가

처리할 작업을 동시에 실행

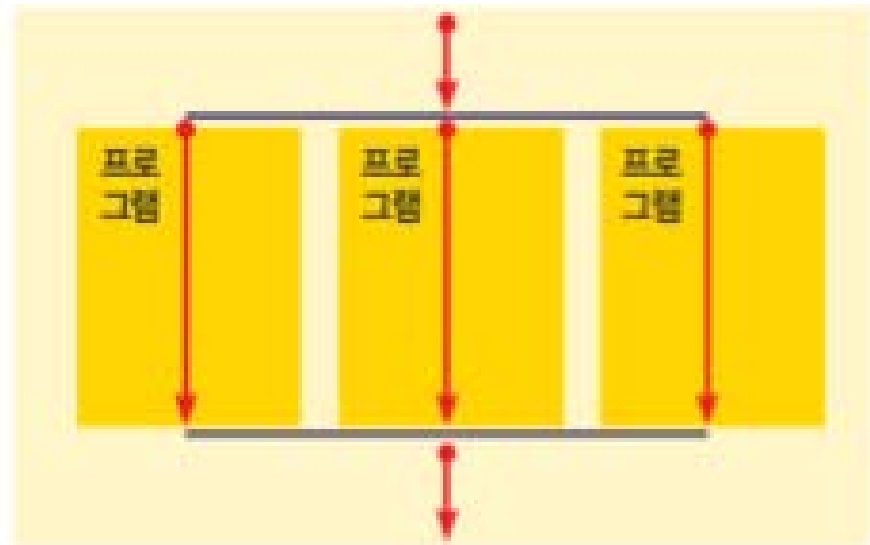
3. Thread

❖ Thread의 개념

프로세스 vs 스레드



(1) 다중 스레드



(2) 다중 작업

3. Thread

❖ Thread의 생성과 실행

스레드 생성 방법

스레드 생성 방법

Thread 클래스를 상속하는 방법

Thread 클래스를 상속받은 후에 run() 메소드를 재정의한다.

Runnable 인터페이스를 구현하는 방법

run() 메소드를 가지고 있는 클래스를 작성하고, 이 클래스의 객체를 Thread 클래스의 생성자를 호출할 때 전달한다.

3. Thread

❖ Thread의 생성과 실행

스레드 생성 및 실행 절차

1) Thread 클래스를 상속하는 방법

- Thread를 상속받아서 클래스를 작성
- run() 메소드를 재정의
- Thread 객체를 생성한다.
- start()를 호출하여서 스레드를 시작

2) Runnable 인터페이스를 구현하는 방법

- Runnable 인터페이스를 구현한 클래스를 작성
- run() 메소드를 재정의
- Thread 객체를 생성하고 이때 MyRunnable 객체를 인수로 전달
- start()를 호출하여서 스레드를 시작

3. Thread

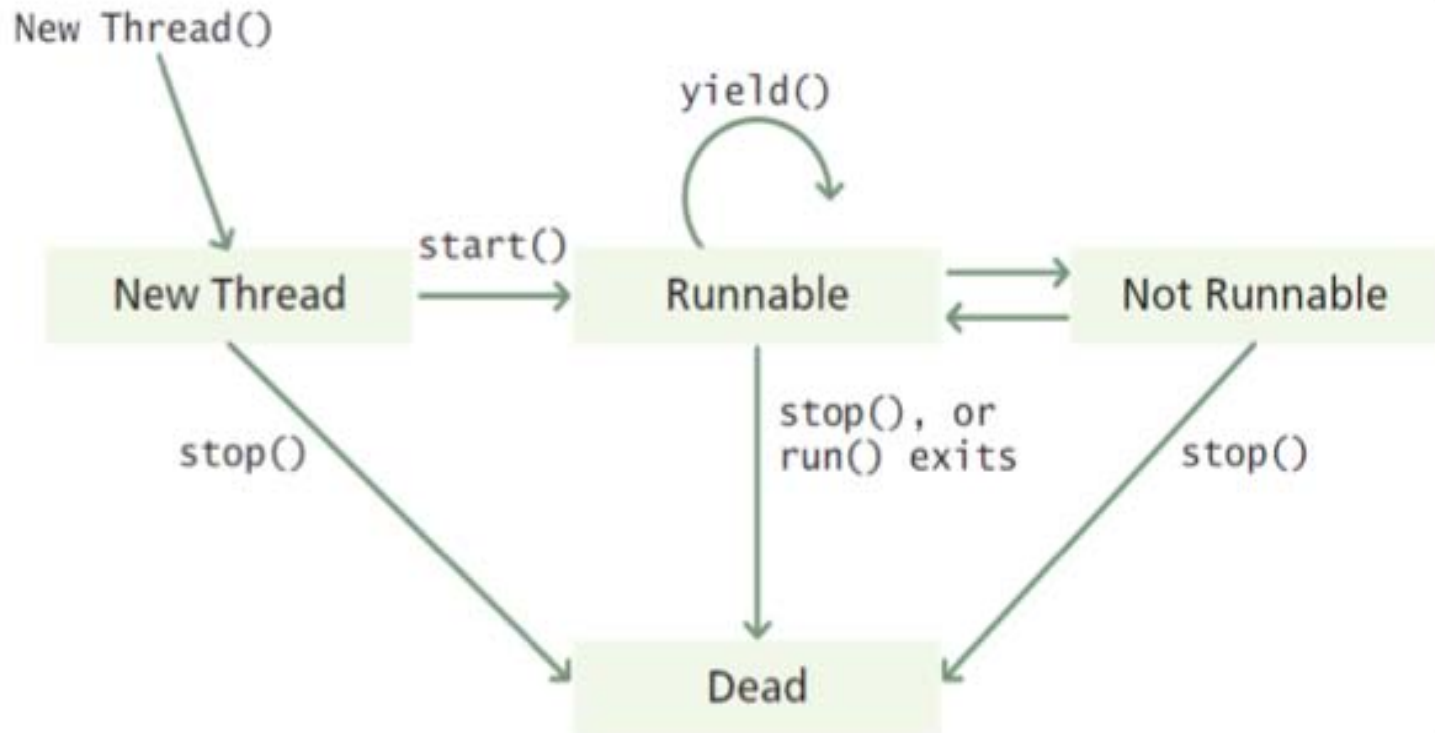
❖ Thread의 생성과 실행

스레드 클래스

메소드	설명
Thread()	매개 변수가 없는 기본 생성자
Thread(String name)	이름이 name인 Thread 객체를 생성한다.
Thread(Runnable target, String name)	Runnable을 구현하는 객체로부터 스레드를 생성한다.
static int activeCount()	현재 활동 중인 스레드의 개수를 반환한다.
String getName()	스레드의 이름을 반환
int getPriority()	스레드의 우선순위를 반환
void interrupt()	현재의 스레드를 중단한다.
boolean isInterrupted()	현재의 스레드가 중단될 수 있는지를 검사
void setPriority(int priority)	스레드의 우선순위를 지정한다.
void setName(String name)	스레드의 이름을 지정한다.
static void sleep(int milliseconds)	현재의 스레드를 지정된 시간만큼 재운다.
void run()	스레드가 시작될 때 이 메소드가 호출된다. 스레드가 하여야하는 작업을 이 메소드 안에 위치시킨다.
void start()	스레드를 시작한다.
static void yield()	현재 스레드를 다른 스레드에 양보하게 만든다.

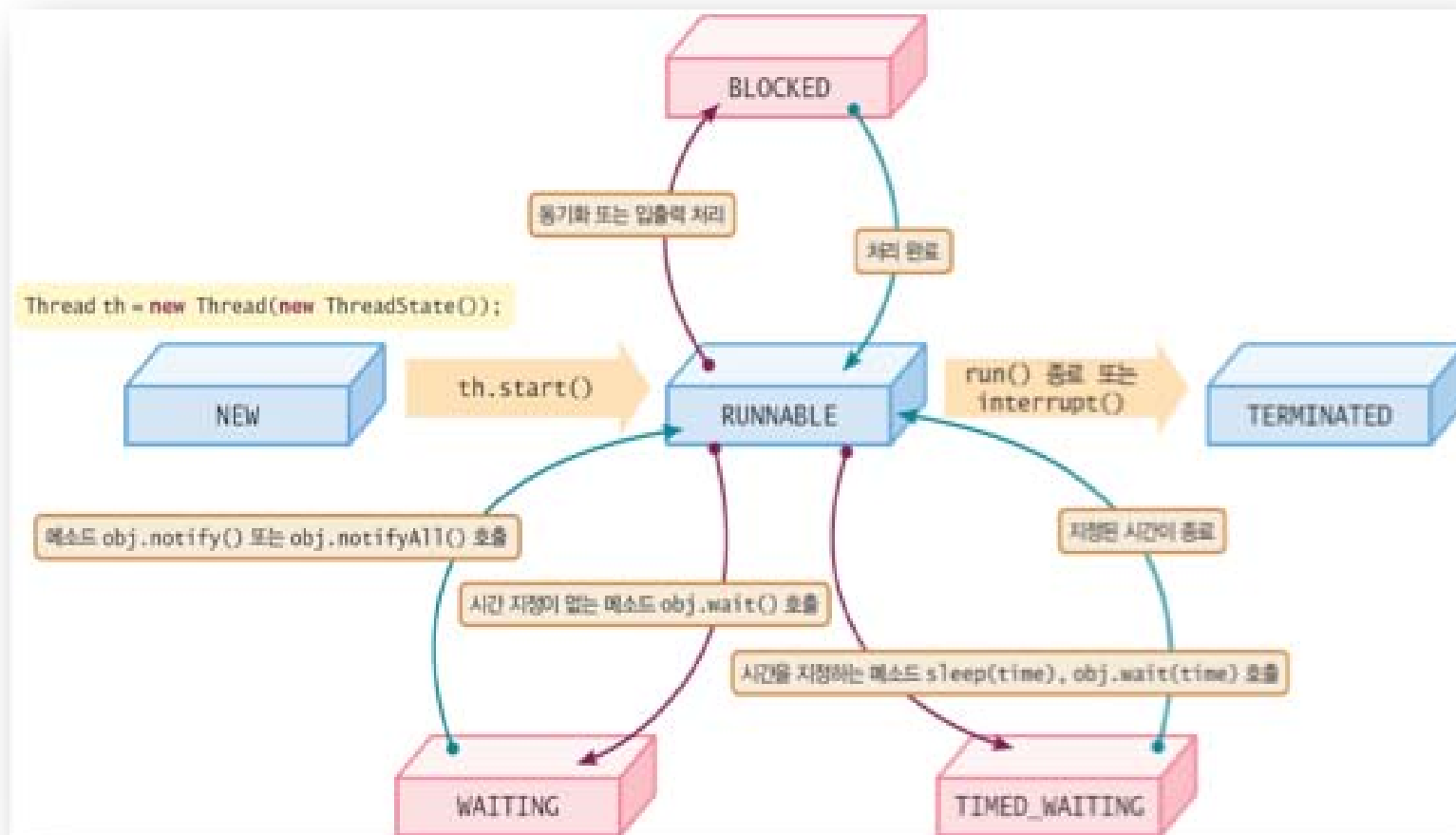
3. Thread

❖ Thread의 상태



3. Thread

❖ Thread의 상태



3. Thread

❖ Thread의 상태

NEW : 스레드는 객체가 생성되면 이동되는 상태

RUNNABLE : 메소드 `start()`가 호출되면 이동

TERMINATED : 스레드가 완전히 종료된 상태

더 이상 NEW 또는 RUNNABLE 등의 다른 상태로 전이가 불가능한 상태

BLOCKED, WAITING, TIMED_WAITING

3. Thread

❖ Thread의 상태

인터럽트(interrupt)

- 하나의 스레드가 실행하고 있는 작업을 중지하도록 하는 메카니즘

```
for (int i = 0; i < messages.length; i++) {  
    try {  
        Thread.sleep(1000);  
    } catch (InterruptedException e) {  
        // 인터럽트를 받은 것이다. 단순히 리턴한다.  
        return;   
    }  
    System.out.println(messages[i]);  
}
```

←----- 인터럽트 처리는 여기에서 해준다.

만약, 스레드가 실행 중에 sleep()을 호출하지 않는다면 InterruptedException

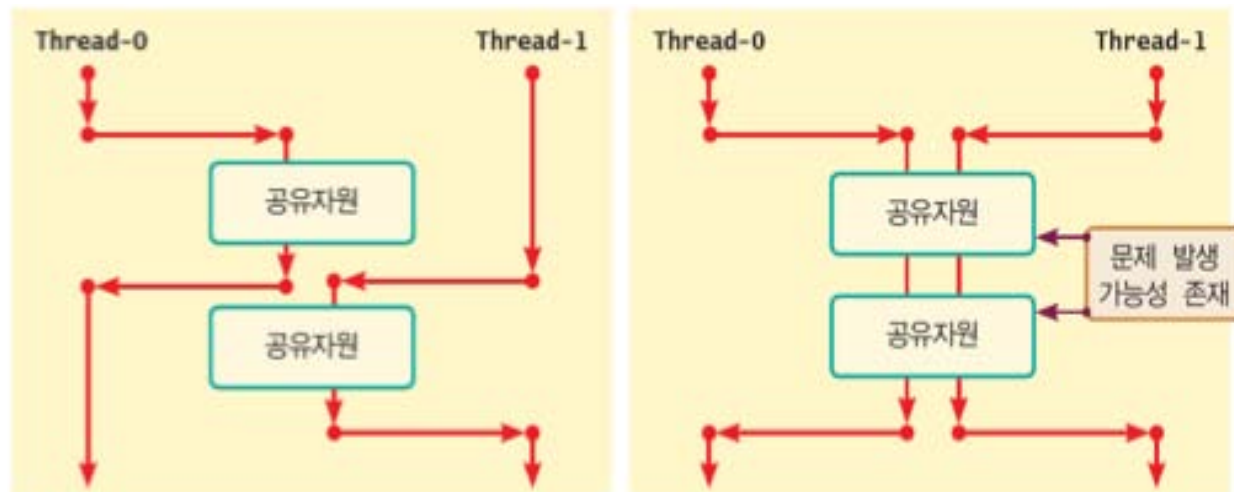
```
if (Thread.interrupted()) {  
    // 인터럽트를 받은 것이다. 단순히 리턴한다.  
    return;  
}
```

3. Thread

❖ Thread의 동기화

다중 스레드 상에서 공유 자원의 처리 문제
예상하지 못한 문제 발생 가능성 존재
스레드 A가 공유 자료를 처리하는 도중에
다른 스레드 B가 그 자료를 처리한다면

다중 스레드에서 수행해야 할 작업이 하나의 단위로 처리되지 않아



(1) 다중 스레드 동기화 처리

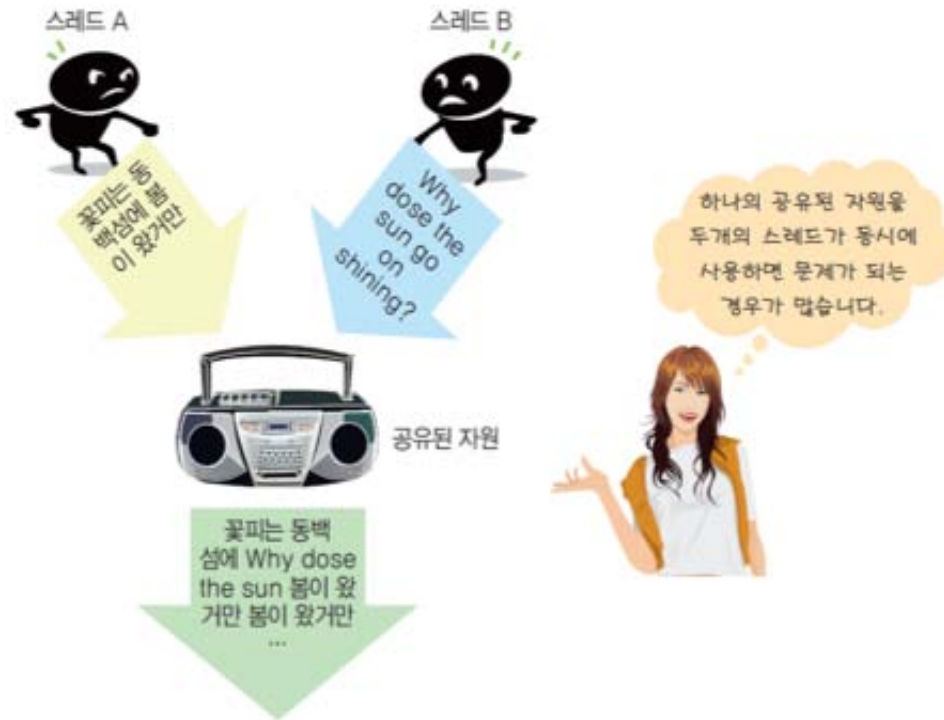
(2) 다중 스레드 동기화 문제

3. Thread

❖ Thread의 동기화

동기화(synchronization)

- 한 번에 하나의 스레드 만이 공유 데이터를 접근할 수 있도록 제어하는



3. Thread

❖ Thread의 동기화

임계영역(critical section)

다중 스레드에서 하나의 스레드가 배타적으로 공유 자원을 독점하도록

임계영역의 실행

- 하나의 스레드가 은행계좌의 인출 메소드 `withdraw()` 기능을 수행
중간에 다른 스레드가 들어와 일을 할 수 없도록 잠금(lock) 장치 필요
인출 기능을 모두 수행했다면 다른 스레드가 일을 하도록
잠금 장치를 해지(unlock) 해야 할 것

은행 계좌 모의 구현

신청한 금액을 인출하는 메소드 `withdraw()`

신청한 금액을 입금하는 메소드 `deposit()`

3. Thread

❖ Thread의 동기화

은행 계좌 모의 구현

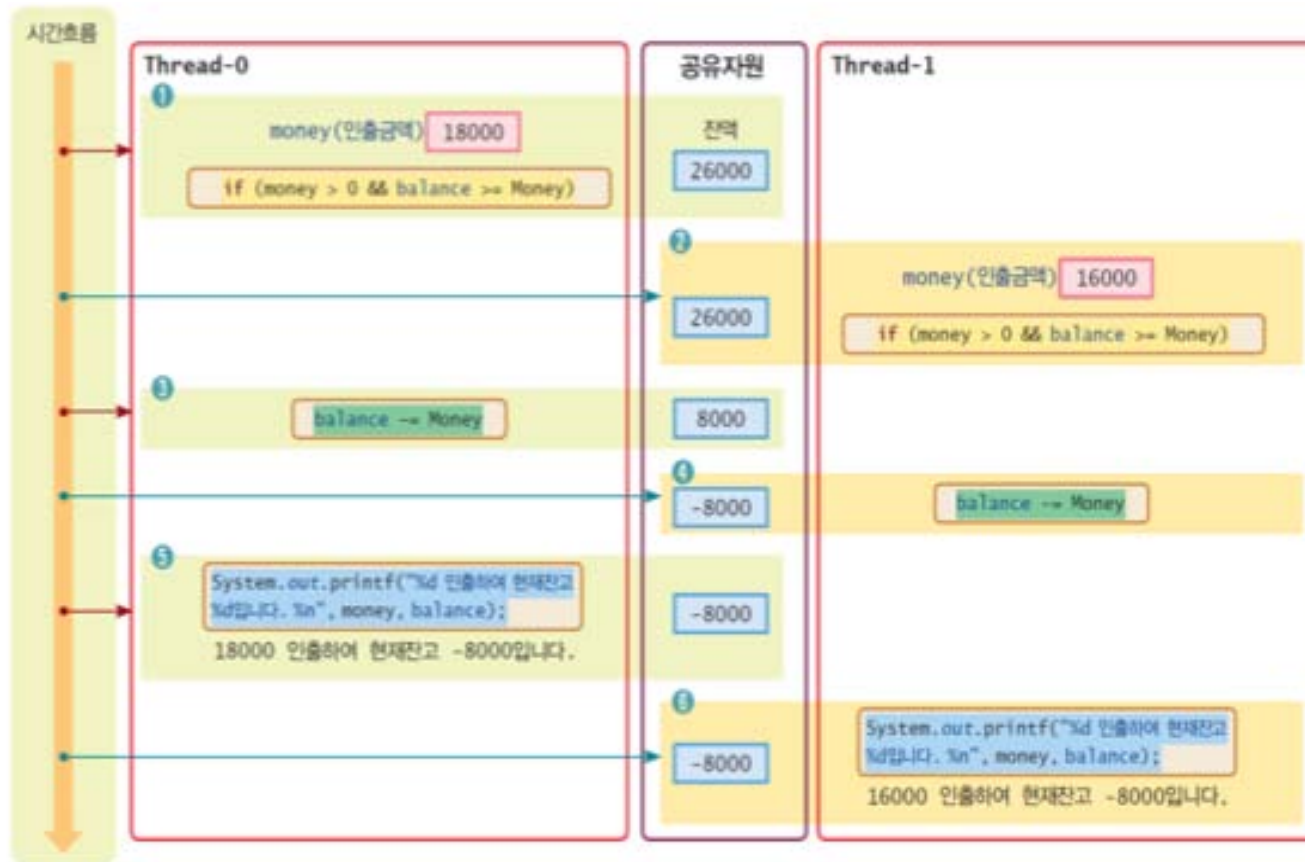
신청한 금액을 인출하는 메소드 `withdraw()`

신청한 금액을 입금하는 메소드 `deposit()`

3. Thread

❖ Thread의 동기화

은행 계좌 모의 구현



3. Thread

❖ Thread의 동기화

동기화(synchronization) 기법

다중 스레드에서 임계영역의 독점적 처리 해결 방법

동기화 방법

1) 메소드의 동기화

메소드의 동기화는 메소드의 지정자인 키워드 `synchronized`를 기술

2) 블록의 동기화

블록의 동기화는 `synchronized (Object) {...}`

`Object`는 잠금 장치를 수행하는 객체

3. Thread

❖ Thread의 동기화

동기화 방법

```
public synchronized void withdraw(int money) {  
    if (money > 0 && balance >= money) {  
        balance -= money;  
    }  
    ...  
}
```

```
public void deposit(int money) {  
    synchronized (this) {  
        if (money > 0) {  
            balance += money;  
        }  
    }  
}
```

3. Thread

❖ Thread의 동기화

동기화 효율을 높이는 방법

Object의 메소드 `wait()`와 `notify()`, `notifyAll()`을 사용

1) 메소드 `wait()`

동기화 블록에서 객체의 특정한 작업을 위해 처리 중인 스레드를
`WAITING` 또는 `TIMED_WAITING` 상태로 이동

2) `notify()` 또는 `notifyAll()`

다시 스레드 상태를 `RUNNABLE`로 이동시켜 스레드 작업을 다시

특징

Object 클래스의 메소드이므로 모든 객체에서 사용 가능
키워드 `synchronized`를 사용하는 동기화 내부에서만 사용 가능

3. Thread

❖ Thread의 동기화

동기화 효율을 높이는 방법

1) 메소드 wait()

```
synchronized (obj) {  
    while (<작업 수행 조건이 만족하지 못하면>)  
        obj.wait(timeout);  
        // 또는 obj.wait()의 사용  
    ...    // 조건을 만족하여 작업을 계속 수행  
}
```



```
obj.notifyAll();  
// obj.notify();
```

WAITING 상태에서 notify() 또는 notifyAll()에 의해
RUNNABLE 상태로 이동한다.

3. Thread

❖ Thread의 동기화

동기화 효율을 높이는 방법

2) notify() 또는 notifyAll()

```
public synchronized void withdraw(int money) {  
    ...  
    int count = 0;  
    while (balance < money) {  
        ...  
        //지속적으로 잔금이 부족하여 메소드 종료  
        if (++count > 3) {  
            System.out.println("잔액이 부족하여 출금처리 못하고 종료합니다.");  
            return;  
        }  
        ...  
        try {  
            //wait();  
            wait(1000);  
        } catch (InterruptedException e) {  
            System.err.println(e);  
        }  
    }  
    balance -= money;  
    ...  
}
```

세 번까지 스레드를 깨워 다시 출금 조건을 검사하도록 하여 그 이후는 출금을 하지 않고 출금 처리를 종료하도록

스레드를 깨워 다시 출금 조건을 검사하도록

```
public synchronized void deposit(int money) {  
    ...  
    balance += money;  
    //notify();  
    notifyAll();  
}
```

Collection 프레임워크

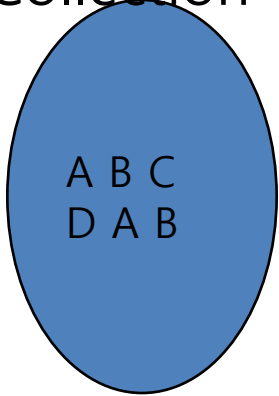
다수의 데이터를 쉽게 처리 할 수 있는 표준화된 방법을 제공하는 클래스들

● 컬렉션 프레임워크의 핵심 인터페이스

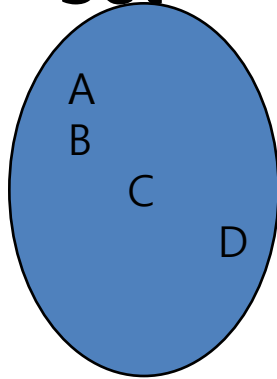
인터페이스	특징
List	데이터 순서 있고 중복 허용 (ArrayList , LinkedList, Stack, Vector)
Set	데이터 순서 없고 중복 안됨 (HashSet, TreeSet)
Map	키와 값의 쌍으로 이루어진 데이터 집합 순서 없고 키는 중복 안되며 값은 중복 허용함. (HashMap, TreeMap, Hashtable, Properties)

Collection 프레임워크

Collection



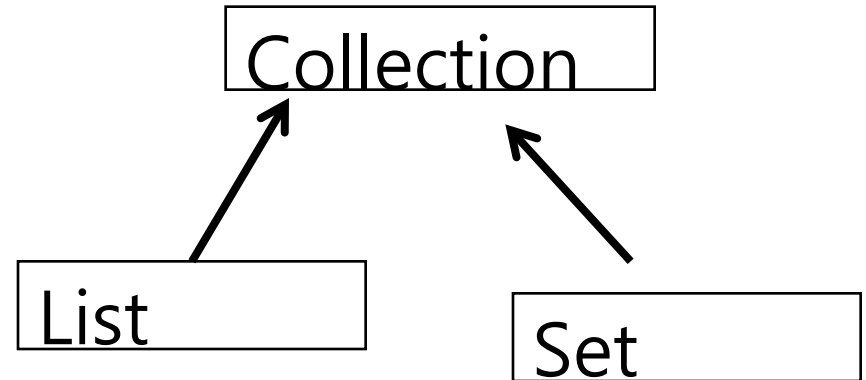
Set



List



Map



Map

[컬렉션 상속계층도]

WrapperClass

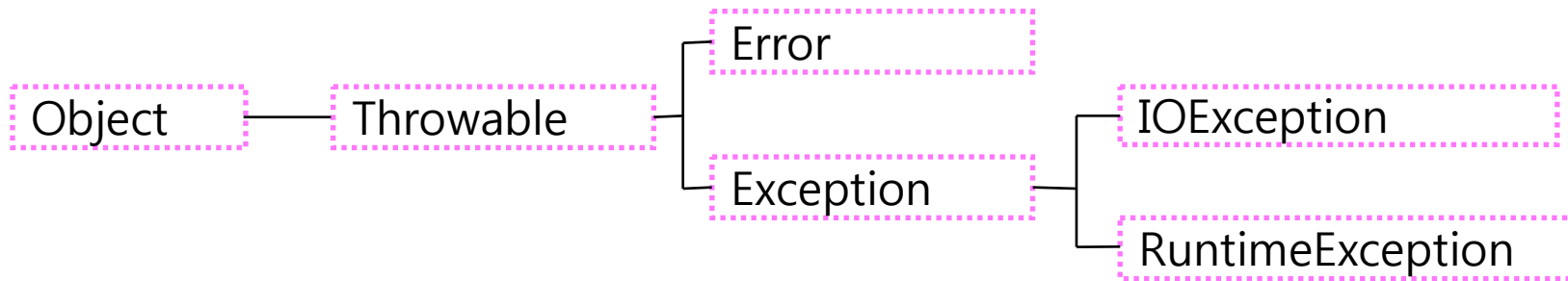
primitiveType를 Object로 변환 하는 클래스

Primitive Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Exception

프로그램 실행도중 예외가 발생하여 프로그램이 중단되는 것을 미리 예방하는 것.

예외가 발생한 부분은 어쩔 수 없어도 이외의 영역은 끝까지 수행될 수 있도록 함.



Exception

직접처리방법

```
try{
    예외발생 가능성 있는 코드;
}catch(예외가능성Exception 변수이름){
    예외발생시 해야 할 일;
}catch(예외가능성Exception 변수이름){
    예외발생시 해야 할 일;
}
...
finally{
    예외발생여부와 상관없이 무조건 실행해야 할 일;
}
```

중요

- catch문 여러 번 작성 할 때 반드시 서브클래스부터 작성한다.
- try는 catch 또는 finally 와 함께 사용함.

Exception

던지는 방법

```
public void test() throws ~Exception , ...{  
  
}
```

중요

- 예외발생 가능성이 있는 메소드 선언부에서 예외발생가능성이 있는 Exception의 종류를 throws를 이용하여 던짐.
- 메소드를 호출한 곳에서 예외처리를 직접 하도록 함.

예외 강제로 발생 시키기

```
throw new Exception();
```

Exception

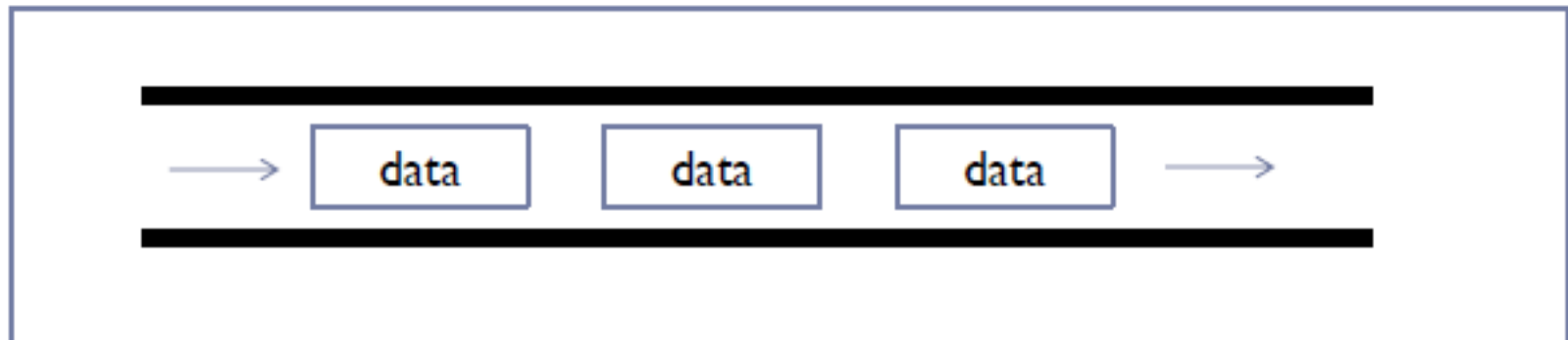
예외만들기

```
class A extends Exception{  
    public A(){  
        super("애들은 가라!");  
    }  
}
```

A a = new A(); => 생성자를 통해 예외메시지를 만듦
throw a; => 예외를 발생시키고자 할 때 throw를 이용하여 예외발생 시킴.

IO Stream

1. 데이터를 운반(입출력)하는데 사용되는 연결통로
2. 데이터의 흐름
3. 연속적인 데이터의 흐름을 물(stream)에 비유해서 붙여진 이름
4. 자바에서의 스트림은 단방향
 1. 데이터를 읽어 들이는 스트림
 2. 데이터를 내보내는 스트림
 3. 따라서, 입, 출력 작업을 동시에 수행할 시 각각의 스트림이 필요함



IO Stream

1. 전송되는 data 타입에 따라

	1 byte(binary)	2 byte(character)
입력	InputStream	Reader
출력	OutputStream	Writer

1. 입/출력 장치에 따라

1. NodeStream
2. 입/출력 장치에 직접 연결되는 스트림
3. 반드시 필요

2. 기능(처리 능력)에 따라

1. FilterStream, ProcessingStream
2. 옵션(필요에 따라 사용가능)
3. 0개 이상 사용가능