



Java 프로그래밍 과정

백명숙(vega2k8s@gmail.com)

Chap 01. Getting Started

학습 목표

1. 자바 기술의 특징
2. JVM 기능, GC
3. Sample Program 작성
4. Write, Compile, Run

Java 의 역사

- 1991년 Sun사의 James Gosling에 의해 가전제품에 이용하기 위해 개발이 시작(Green Project).
- 초기에 개발된 언어를 Oak라 하였으며 전자기기의 내장된 프로그램을 위해 사용. Oak는 별로 관심을 끌지 못하였다.
- 1994년 Gosling은 Oak를 JAVA(커피의 속어)로 다시 명명하고 당시 인터넷에서 급격히 성장한 WWW에 자바를 적용 결정
- Hot Java 검색기 개발. 넷스케이프, 익스플로러 자바 지원

Java 의 역사

- 1991년
 - Green 프로젝트가 생기면서 Java의 모태가 탄생하기 시작
 - James Gosling, Mike Sheridan, Patrick Naughton이 TV와 시청자가 서로 상호작용을 하는 것을 만들기 위해 Oak 라는 언어 탄생
- 1994년
 - World Wide Web 등장
 - Oak에서 Java로 명칭변경
 - Java, Hot Java project 시작
- 1995년
 - Hot Java, Java, Java context, source code가 Web에 공개
 - 플랫폼 : Sun SPARC Solaris, Windows NT, Windows95, Linux
 - Java beta1 발표(Sun Microsystems)
 - Netscape 지원결정
 - Java beta2 발표
 - JavaScript 발표(Sun & Netscape)

Java 의 역사

- 1996년
 - Java1.0 발표
 - Netscape2.0 Java 지원
- 2000년 J2SE 1.3 출시
- 2002년 J2SE 1.4 출시
- 2004년 J2SE 5.0 출시
 - Generic, foreach 루프, static import, Type Safe Enum
 - AutoBoxing/unBoxing, Concurrent API
- 2006년 J2SE 6 출시
 - JavaSE 6까지는 Sun Microsystems에서 Java에 대한 주요 스펙을 만듦
 - JAX-WS(Web Service Client), 모니터링 및 관리기능 강화, 스크립트언어지원
- 2011년 J2SE 7 출시
 - JavaSE 7부터는 Oracle이 Java에 대한 주요 스펙을 만듦
 - String을 이용한 switch 구문, NIO 2.0, Fork-Join에 의한 병렬처리
 - try-with-resources구문,
- 2014년 J2SE 8 출시
 - Lamda, 함수형 프로그래밍, Functional Interface, Stream, default method
- 2017년 9월 J2SE 9 출시

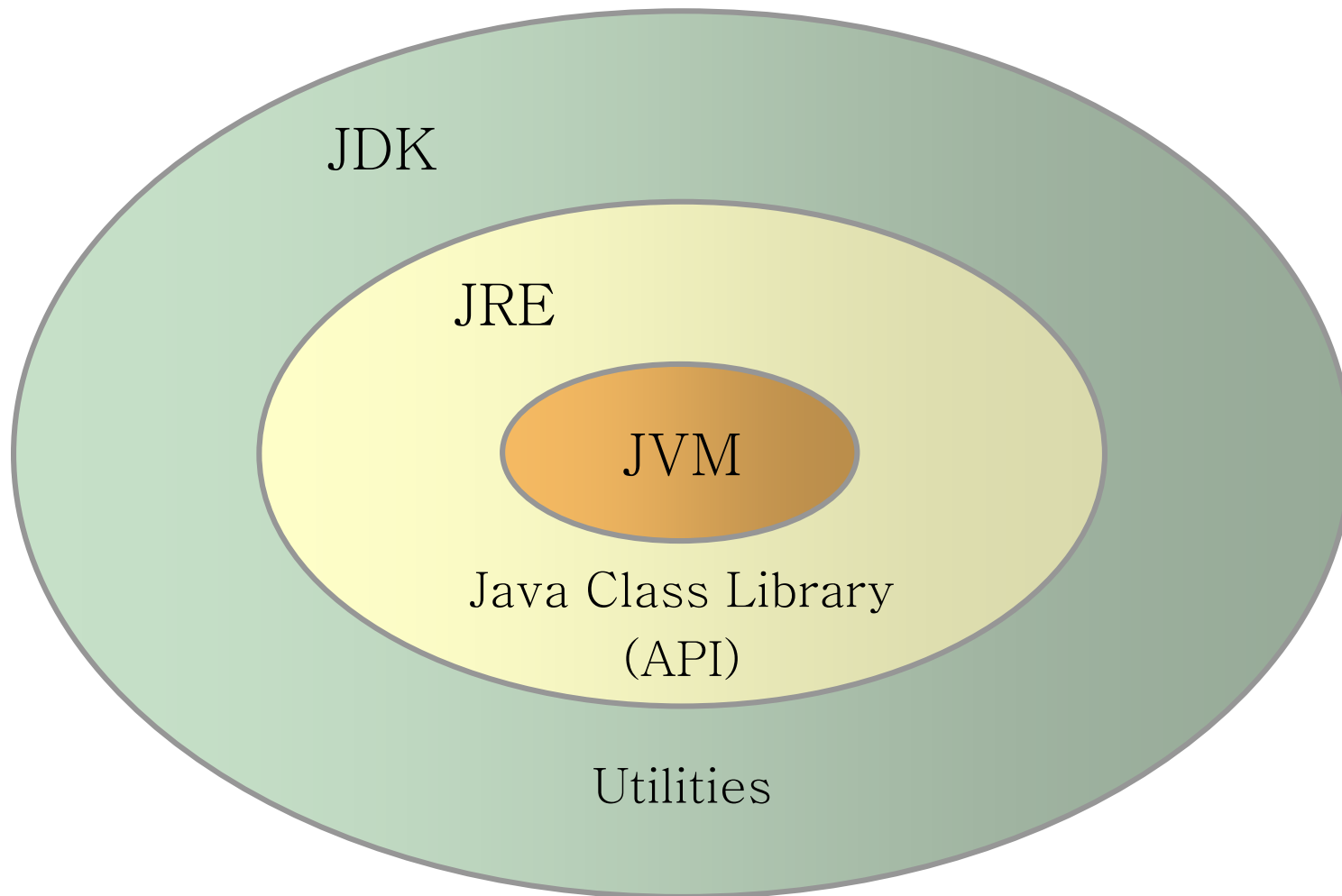
Programming Language

Development Environment

Application Environment

Deployment Environment

사용자/개발자 입장에 따라 설치하는 범위가 달라진다.



J2SE : Java 2 Standard Edition

J2EE : Java 2 Enterprise Edition

J2ME : Java 2 Micro Edition

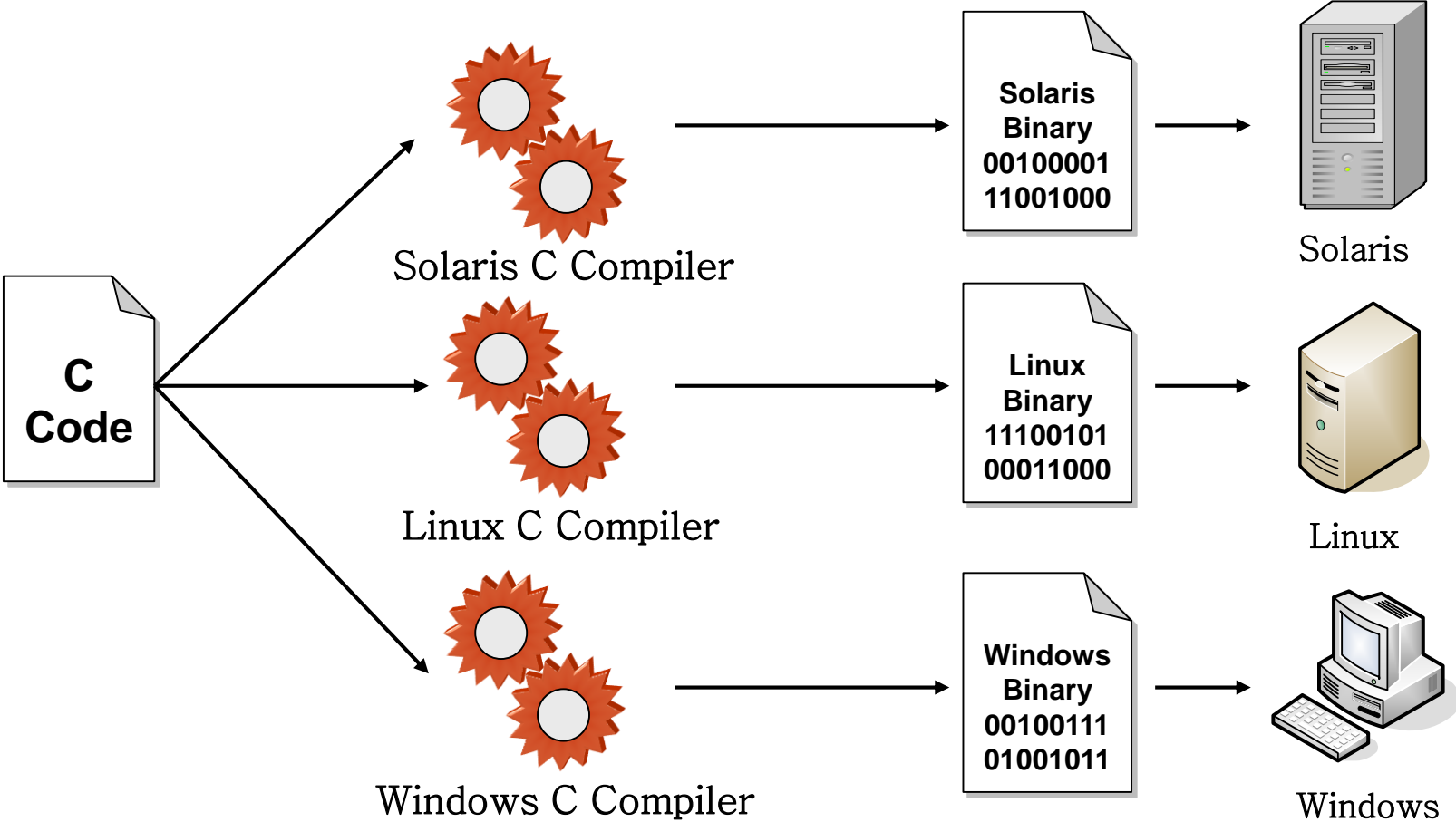
- ❑ 사용하기 쉬운 언어 제공
 - 다른 언어의 단점 보완 (포인터/메모리)
 - 객체 지향 언어
 - 능률적이고 명확한 코드를 작성하게 해 준다.
- ❑ Interpreted Environment
 - 개발 속도 향상
 - 코드 이식성 (bytecode)
- ❑ Thread 제공
- ❑ 클래스의 동적 메모리 Load
- ❑ 변경 된 class들을 원격지로부터 다운로드하여, Runtime시 반영 (Applet)
- ❑ 보다 나은 코드 안정성

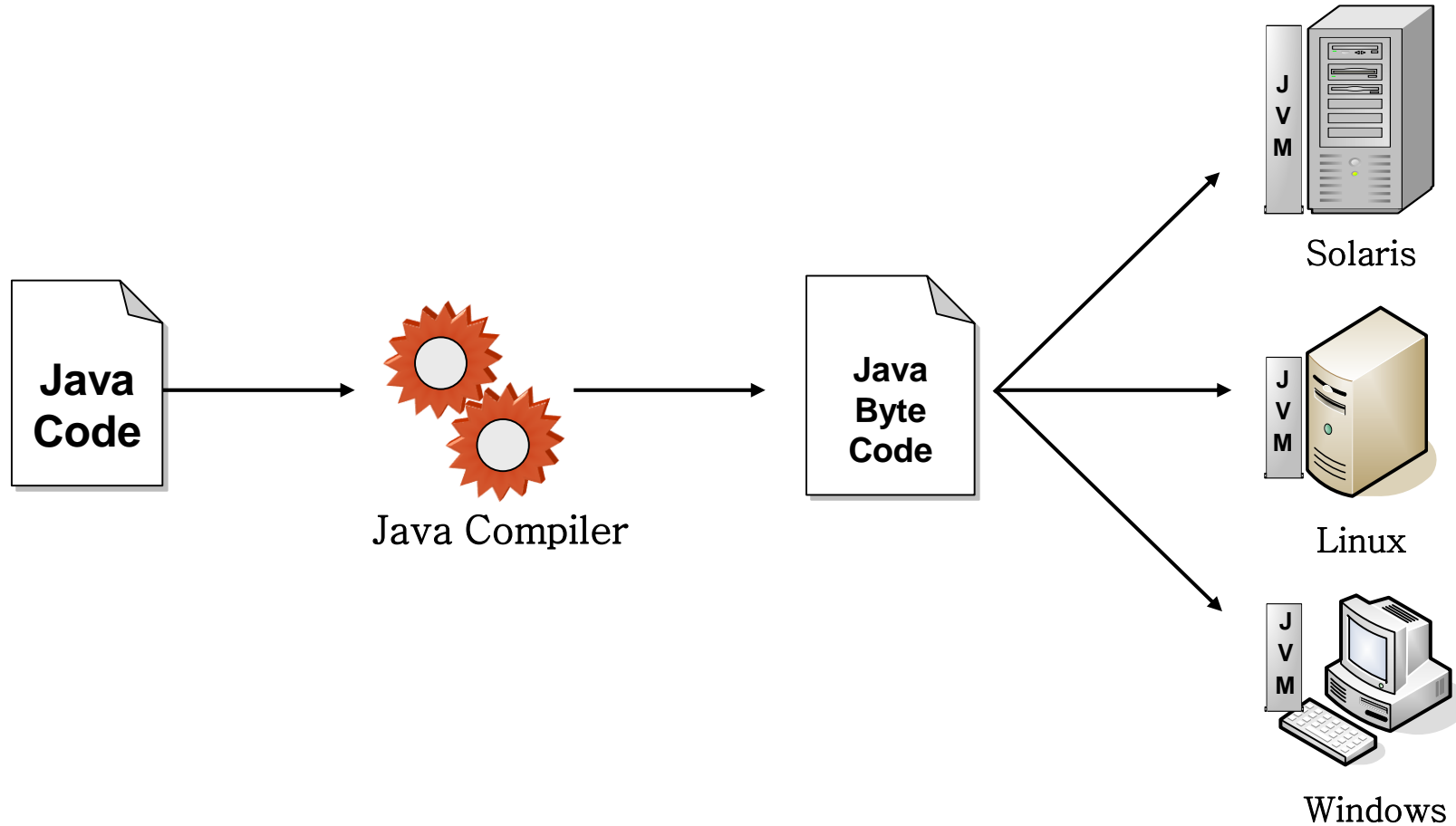
Java Virtual Machine

Garbage Collection

Code Security

- ❑ H/W Platform 규약을 제공한다.
- ❑ Platform 독립적인 bytecode를 해석하고, 실행한다.
- ❑ S/W 또는 H/W로 구현 된다.
- ❑ 실행환경은 일반 PC이거나, Web Brower일 수 있다
- ❑ 설치 된 Platform에 종속적이다.





- 더 이상 사용되지 않는 메모리는 재사용 가능하게 해제시켜야 한다.
- 다른 언어에서는 메모리 해제는 프로그래머가 직접 코드로서 구현해야 한다.
- 자바에서는 메모리를 지속적으로 감시하면서 더 이상 사용되지 않는 메모리를 해제 시켜준다.

□ Garbage Collector

- 더 이상 사용 되지 않는 메모리를 검사한다.
- 자동적으로 일어난다. (개발자는 신경 쓸 필요 없음)
- 일어나는 시점은 구현 된 JVM마다 다를 수 있다. (Vendor Dependent)
- 개발자가 `System.gc();` 로 호출할 수 있다.

TestGreeting.java

```
public class TestGreeting {  
    public static void main( String[] args ) {  
        Greeting hello = new Greeting();  
        hello.greet();  
    }  
}  
  
class Greeting {  
    public void greet() {  
        System.out.println( "hi" );  
    }  
}
```


Greeting.java

```
public class Greeting {  
    public void greet() {  
        System.out.println( "hi" );  
    }  
}
```

TestGreeting.java

```
public class TestGreeting {  
    public static void main( String[] args ) {  
        Greeting hello = new Greeting();  
        hello.greet();  
    }  
}
```

Console 에서의 컴파일과 실행

□ Compile

```
javac -d c:\Wdata\Wclass Greeting.java
```

```
javac -d c:\Wdata\Wclass -classpath c:\Wdata\Wclass  
TestGreeting.java
```

①

명령어

②

컴파일된 클래스파일이
저장될 위치 지정

③

컴파일을 위해 필요한
클래스 파일들을 찾아올
위치 지정

④

확장자를 포함한
소스파일명

□ Run

```
java -classpath c:\Wdata\Wclass TestGreeting
```

①

명령어

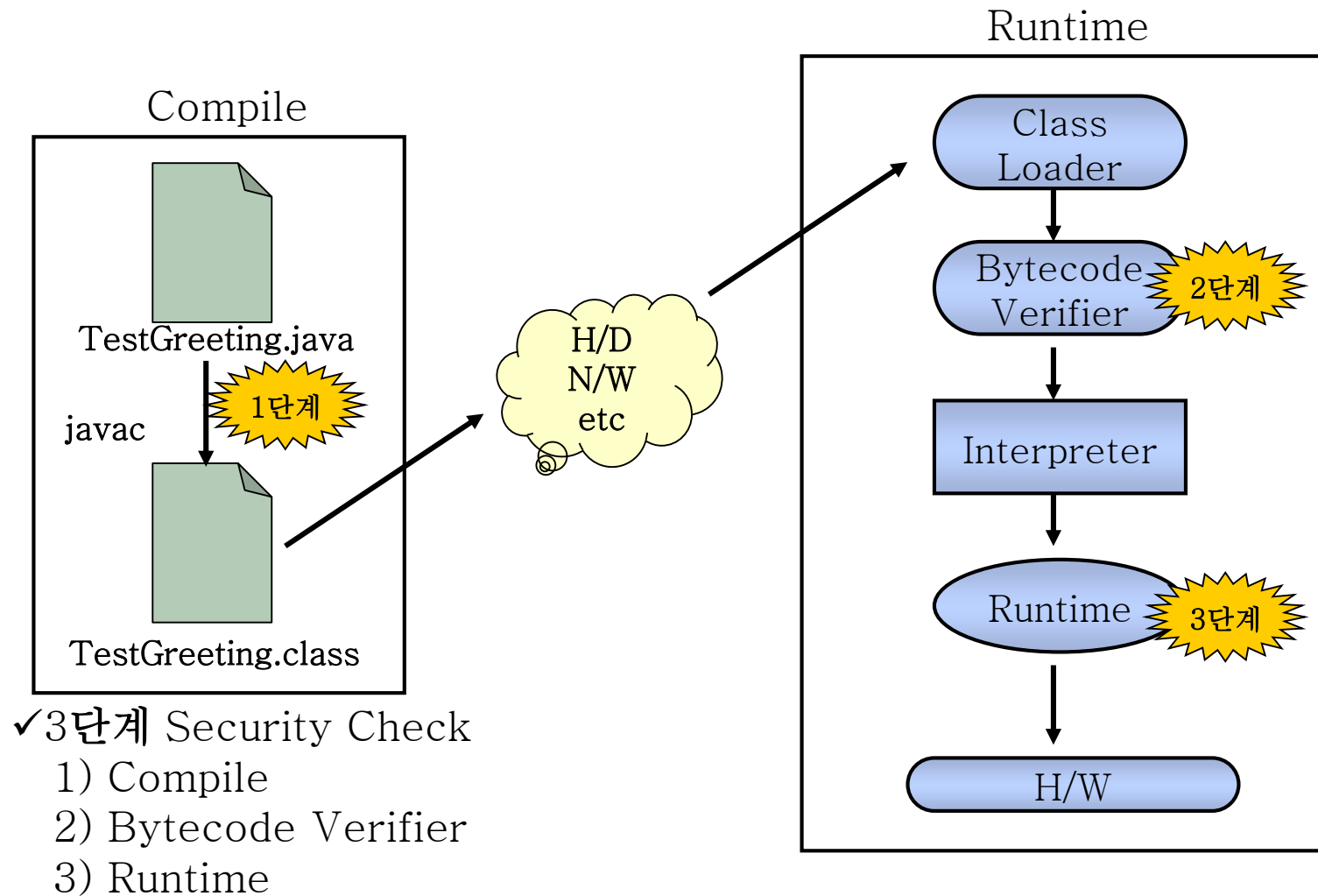
②

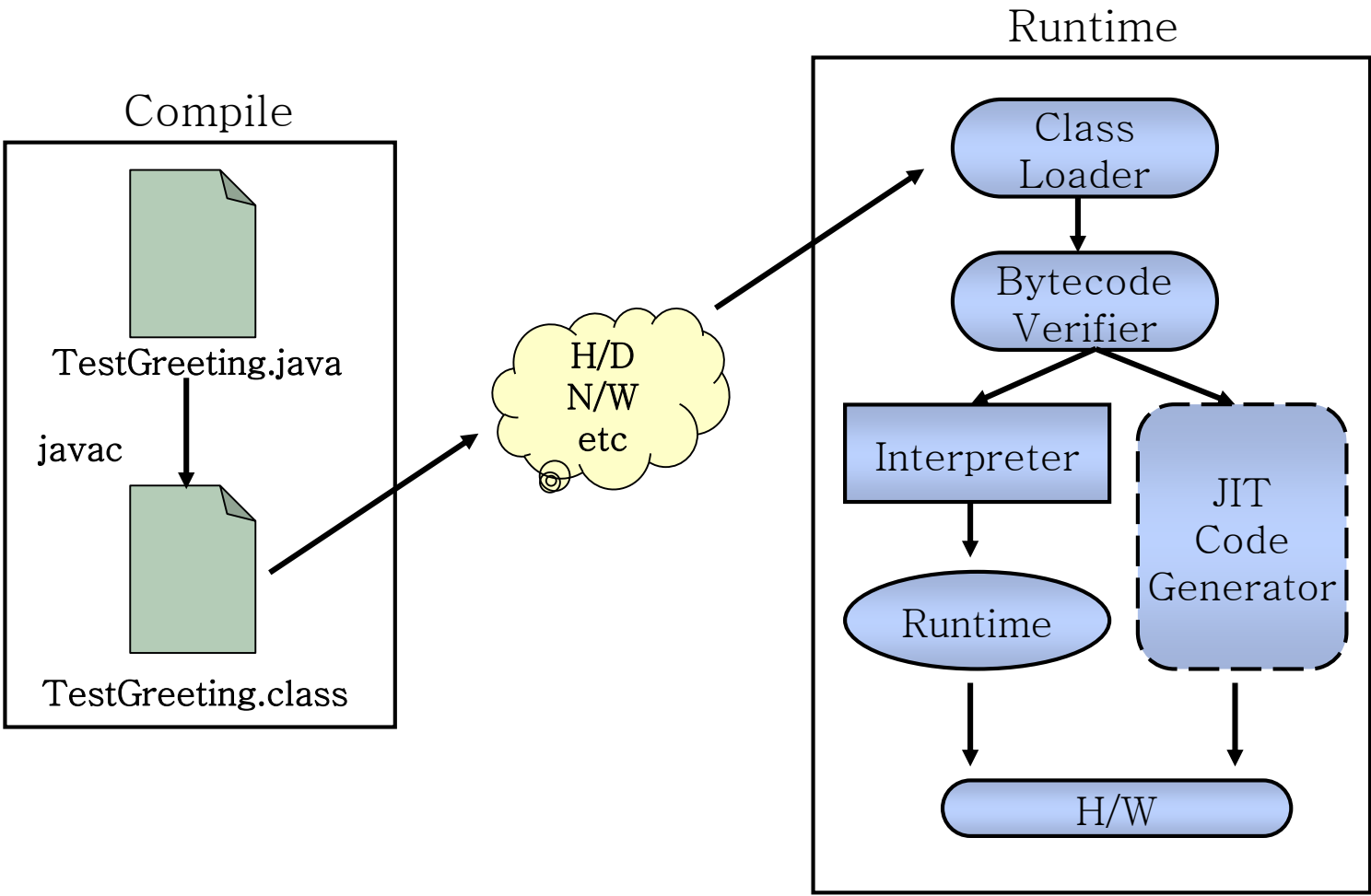
어느 위치에서부터
클래스를 찾을 것인가

③

확장자를 제외한
클래스명

자바 소스 작성에서부터 실행 단계까지의 과정



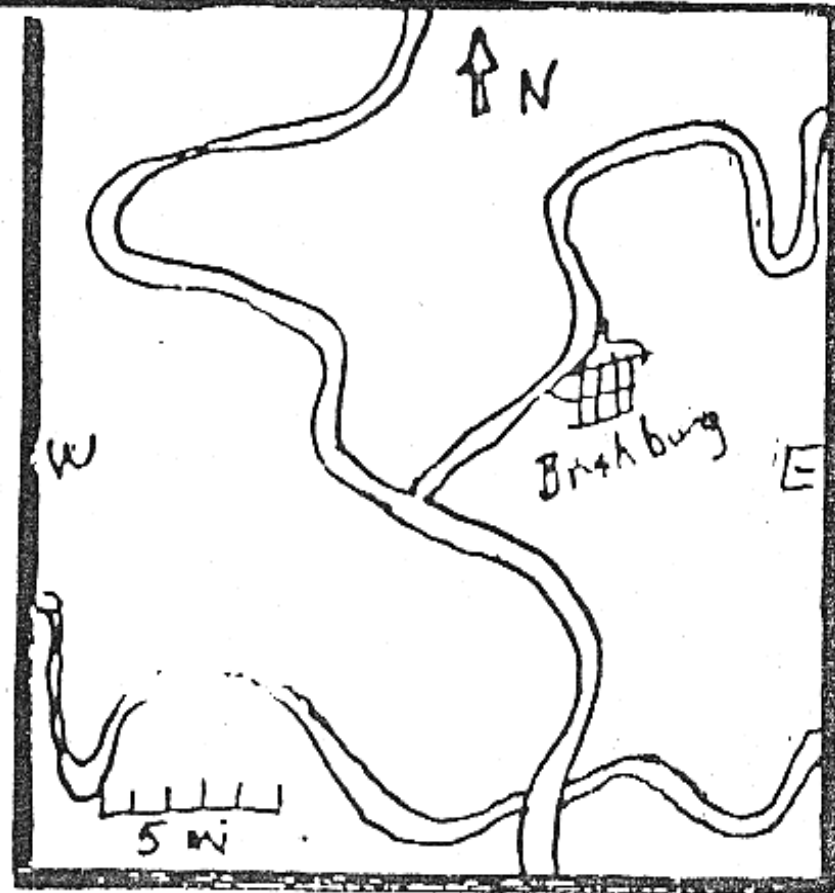


Chap 02. Object-Oriented Programming

1. abstraction
2. 기본 class작성
3. encapsulation
4. constructor
5. package / import



REAL WORLD



IMPERFECT REPRESENTATION

□ Java에서의 Abstraction

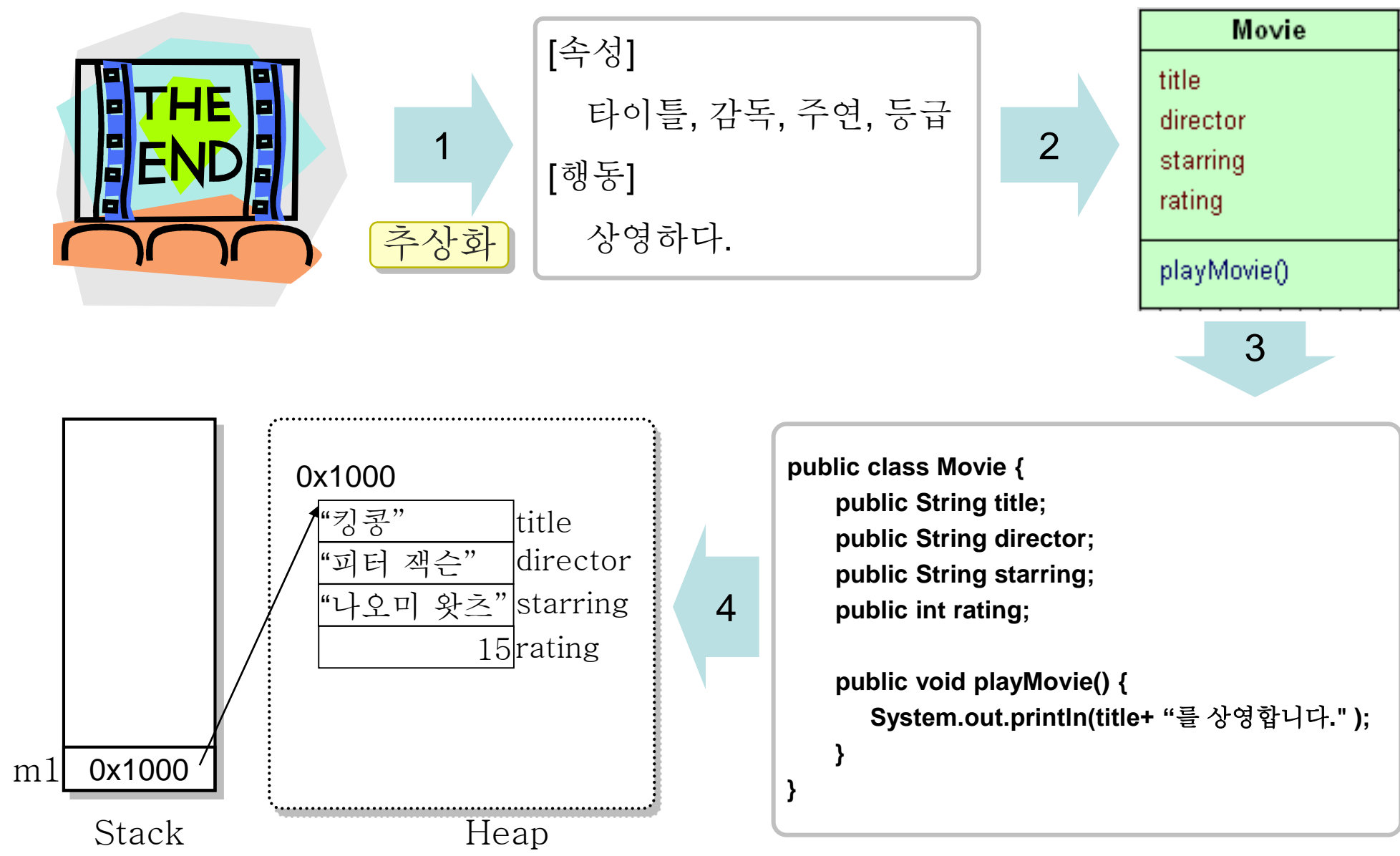
- 실 세계의 객체를 프로그래밍 관점에서 관심의 대상이 되는 속성과 행동을 추출해 내는 것을 말한다.

□ Abstraction의 결과 → Class

- Class : Member Variable + Member Method
- 속성 : Class의 Member Variable
- 행동 : Class의 Member Method

□ Class

- 클래스는 추상화 된 대상이 “이러이러한 속성과 행동을 가진다”라는 것을 정의한 설계이지, 실제로 값을 가진 객체(object/instance)가 아니다.
- 실제로 값을 갖는 객체(instance)는 new라는 키워드를 이용해서, 클래스 정의에 해당 하는 내용으로 메모리에 할당되어져 이용하게 된다.



□ 클래스 작성하기

패키지 선언

```
{ package chap02.entity;
```

클래스 선언

```
{ public class Movie {
```

멤버변수

```
    public String title;  
    public String director;  
    public String starring;  
    public int rating;
```

멤버메소드

```
    public void setTitle( String title ) {  
        title = newTitle;  
    }  
  
    public void playMovie() {  
        System.out.println( title + “를 상영합니다.”);  
    }  
}
```

□ 클래스 사용하기

패키지 선언

import 선언

main 메소드

```
package chap02.view;
```

```
import chap02.entity.*;
```

```
public class TestMovie {
```

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

```
        Movie m1 = new Movie();
```

```
        m1.setTitle( “킹콩” );
```

```
        System.out.println( m1.title );
```

```
    }
```

```
}
```

□ package

- 한 자바 소스에서 단 한 번만 쓸 수 있다.
- 관련 있는 클래스를 묶어 주는 역할을 한다.
- .java 파일에서 단 한 번만 사용되어 질 수 있다.

Ex) package shipping.reports;


□ import

- 서로 다른 package에 있는 클래스를 참조하기 위해, 어느 package에 있는 클래스인지 선언해 주는 역할을 한다.
- 한 자바 소스 내에 여러 번 쓸 수 있다.
- 같은 package 클래스를 사용하기 위해서는 필요 없다.

Ex) import shipping.reports.PrintReprot;

import shipping.reports.*;

```
public class Movie {  
    public String title = "킹콩";  
    public String director = "피터 잭슨";  
    public String starring = "나오미 왓츠";  
    public int rating ;  
      
    public void setTitle( String newTitle ) {  
        title = newTitle;  
    }  
  
    public String getTitle() {  
        return title;  
    }  
  
    public void playMovie() {  
        System.out.println( title + "를 상영합니다.");  
    }  
}
```



①` Access Modifier

②` Data Type

③` Variable Name

```
public class Movie {  
    public String title = "킹콩";  
    public String director = "피터 잭슨";  
    public String starring = "나오미 왓츠";  
    public int rating = 15;
```

```
    ⑤ {  
        ① ② ③ ④  
        public void setTitle( String newTitle ) {  
            title = newTitle;  
        }  
  
        public String getTitle() {  
            return title;  
        }  
  
        public void playMovie() {  
            System.out.println( title + "를 상영합니다.");  
        }  
    }
```

①` Access Modifier

② Return Type

③` Method Name

④` Parameter

⑤` Method Body

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

        Movie m1 = new Movie();
        m1.setTitle( "킹콩2" );
        m1.playMovie();
        String title = m1.getTitle();
        System.out.println( "영화 제목은 " +
                           s + "입니다.");
    }
}
```

```
public class Movie {
    public String title = "킹콩";
    public String director = "피터 잭슨";
    public String starring = "나오미 왓츠";
    public int rating = 15;

    public void setTitle( String newTitle ) {
        title = newTitle;
    }

    public String getTitle() {
        return title;
    }

    public void playMovie() {
        System.out.println( getTitle() +
                           "를 상영합니다.");
    }
}
```

```
public class TestDog {  
    public static void main( String[] args ) {  
        Dog d = new Dog(); // Dog 객체 생성  
        d.setWeight(42); // setWeight() 호출하며 42 전달  
        System.out.println( "Dog d's weight is" +  
                             d.getWeight() );  
        // getWeight() 통해서 weight 값 얻기  
    }  
}
```

```
public class Dog {  
    private int weight;  
  
    public void setWeight(int newWeight) {  
        weight = newWeight; // 42를 멤버변수인  
                             weight에 할당  
    }  
  
    public int getWeight() {  
        return weight; // 호출한 쪽으로 weight 리턴  
    }  
}
```

실행 결과 : Dog d's weight is 42

OOP (Object-Oriented Programming) 특징

Encapsulation

Inheritance

Polymorphism

MyDate
+day:int +month:int +year:int

```
public class MyDate {  
    public int day;  
    public int month;  
    public int year;  
}
```

```
public class MyDateTest {  
    public static void main( String[] args ) {  
        MyDate date = new MyDate();  
        date.day = 32;  
    }  
}
```

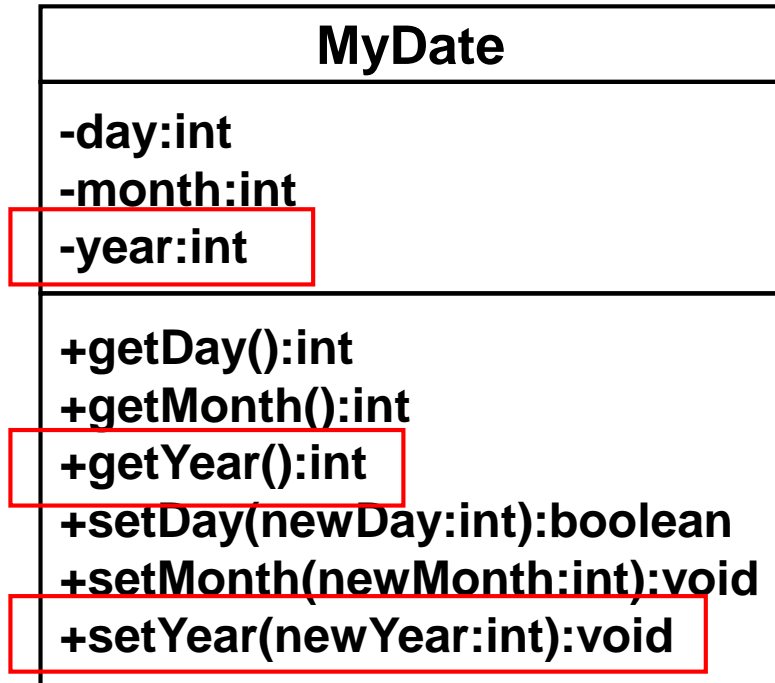
MyDate
-day:int -month:int -year:int
+getDay():int +getMonth():int +getYear():int +setDay(newDay:int):boolean +setMonth(newMonth:int):void +setYear(newYear:int):void

Verify days in month

```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public int getDay() {  
        return day;  
    }  
    public boolean setDay( int newDay ) {  
        //월별 해당일이 맞으면, day = newDay;  
        // true return, 아니면 false return  
    }  
    // month, year 관련 메소드 생략  
}
```

```
public class MyDateTest {  
    public static void main( String[] args ) {  
        MyDate date = new MyDate();  
        date.setDay( 31 );  
        date.setDay( date.getDay() + 1 );  
    }  
}
```

- 숨겨진 Data(private 멤버 변수) 에 대한 access는 getter/setter를 이용한다.



❑ Member Variable : **xxx**

```
private int year;
```

❑ Getter : **getXxx()**

```
public int getYear() {  
    return year;  
}
```

❑ Setter : **setXxx()**

```
public void setYear( int newYear ) {  
    year = newYear;  
}
```

MyDate
-day:int -month:int -year:int
+getDay():int +getMonth():int +getYear():int +setDay(newDay:int):void +setMonth(newMonth:int):void +setYear(newYear:int):void -validDay(checkDay:int):void

```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public int getDay() {  
        return day;  
    }  
  
    public void setDay( int newDay ) {  
        if ( validDay(newDay) )  
            day = newDay;  
    }  
  
    private boolean validDay( int newDay ) {  
        //각 해당월에 대한 일자가 맞는지 확인  
    }  
  
    // month, year 관련 메소드 생략  
}
```

❑ Information hiding

- Class 외부에서 주요 정보에 접근하지 못 하도록 숨기는 것
- 멤버변수는 private로 선언하여, 외부에서 직접 참조하지 못하도록
- 멤버변수는 getter/setter를 이용하여 참조

❑ Encapsulation

- Class 구현에 대한 상세내용을 숨기는 것
- Class사용자는 Class의 내부 구현에는 신경 쓸 필요 없이, 제공되는 Interface(method) 를 이용하기만 하면 됨
- 내부 로직은 private로 선언하여, 외부에서 보이지 않게 한다. 즉, 변화에 관계 없이 외부에서는 Interface만을 이용하므로, 사용자 관점에서는 유지보수가 편해 진다.
- Encapsulation은 Data보호 뿐 아니라, Class의 세부 구현도 숨긴다는 점에서 Information Hiding의 확장이라고 볼 수 있다.

- ❑ 역할 : 주로 멤버변수를 초기화 하는데 쓰인다.
- ❑ 특징
 - 1) 클래스 이름과 같다.
 - 2) Return 타입이 없다.
 - 3) 상속되지 않는다.
 - 4) 객체 생성시 new라는 키워드를 만났을 때 호출된다.

```
public class Dog {  
  
    private int weight;  
    public Dog() {  
        weight = 42;  
    }  
    ....  
}
```

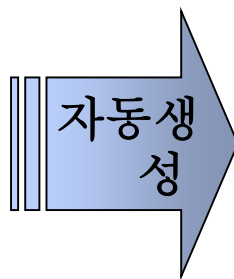
```
public class MyDate {
    private int day;
    private int month;
    private int year;

    public MyDate( int newDay, int newMonth, int newYear) {
        day = newDay;
        month = newMonth;
        year = newYear
    }
    // 메소드 생략
}
```

```
public class MyDateTest {
    public static void main( String[] args ) {
        MyDate d1 = new MyDate( 14, 2, 2006 );
        d1.setDay( 15 );
        System.out.println( m1.getDay() );
    }
}
```



```
public class Movie {  
    public String title;  
    public String director;  
    public String starring;  
    public int rating;  
  
    // 메소드 생략  
}
```



```
public class Movie {  
    public String title;  
    public String director;  
    public String starring;  
    public int rating;  
  
    public Movie() {  
    }  
  
    // 메소드 생략  
}
```


- ❑ 모든 클래스는 최소한 하나의 constructor가 있다.
- ❑ 만약, 아무런 constructor를 정의 하지 않았다면, 자동적으로 default constructor가 생성된다.
- ❑ default constructor는 아무런 argument가 없고, body에 내용이 없는 constructor를 말한다.

- 만약 constructor가 하나라도 있다면, default constructor는 자동 생성이 안 된다.

```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
  
    public MyDate( int newDay, int newMonth, int newYear) {  
        day = newDay;  
        month = newMonth;  
        year = newYear  
    }  
    // 메소드 생략  
}
```

```
public class MyDateTest {  
    public static void main( String[] args ) {  
        MyDate d1 = new MyDate(); //error  
    }  
}
```

□ 1개의 소스 파일에서



```
package chap02.entity; // 1번만 올 수 있다.  
  
import chap02.test.Account; // 필요에 따라 여러 개 가능  
import java.util.*;  
  
public class Test { // 단 하나의 public class만 올 수 있으며,  
    // 파일 이름(.java)과 같아야 한다.  
  
}
```

Chap 03. Identifiers, Keywords and Types

1. Identifier / Keywords
2. Data types
3. Constructing & Initializing Object
4. Pass by value
5. this keyword

```

/*
 * Class : Movie
 */
package chap02.entity;
/**
 * 영화정보를 담는 클래스
 *
 * @author 홍길동
 * @version 1.0, 2013/12/01
 */
public class Movie {
    // 영화 제목
    private String title;
    // 감독
    private String director;
    // 주연
    private String starring;
    // 등급
    private String rating;

    ~~~~~생략~~~~~

```

```

/**
 * 모든 정보를 갖는 Movie 클래스의 생성자
 * @param title 제목
 * @param director 감독
 * @param starring 주연
 * @param rating 등급
 */
public Movie(String title, String director,
             String starring, String rating) {
    this.title = title;
    this.director = director;
    this.starring = starring;
    this.rating = rating;
}

/**
 * 제목 정보에 대한 getter
 * @return 영화제목
 */
public String getTitle() {
    return title;
}

~~~~~생략~~~~~

```

- 한 문장(Statement)은 여러 라인에 걸쳐서 기술될 수 있으며, Semicolon(;)으로 그 끝을 표시한다.

```
totals = a + b + c + d + e + f;
```

```
totals = a + b + c +  
        d + e + f;
```

- Block은 여러 문장을 한 set으로 처리하도록 해주며, Brace({ })로 묶는다.

```
if ( x > y ) {  
    x = x + 1;  
    y = y + x;  
} else {  
    x = x = y;  
}
```

- ❑ 클래스, 메소드를 정의할 때 { } 로 묶는다.
- ❑ 중첩으로 block 지정이 가능하다.
- ❑ 공백 문자(space, tab, enter)는 연산자와 피연산자 사이에 얼마든지 올 수 있다.

```
totals = a + b + c + d + e + f;
```

```
totals = a + b + c +  
        d + e + f;
```


❑ Identifier 정의

- 클래스, 변수, 메소드에 주는 이름

❑ Identifier Naming Rule

- Unicode 문자, '_', '\$' 로 구성한다.
- 숫자로 시작할 수 없다.
- 대/소문자를 구분 한다.

❑ Identifier 정의시 주의점

- Keyword는 Identifier로 사용되어 질 수 없다.
- true, false, null 은 keyword가 아닌 literal(상수) 이지만,
예약된 값이므로 변수명으로 사용할 수 없다. goto, const는
자바 키워드로 정의는 되어있으나 사용되지 않는다.
- sizeof 라는 연산자는 자바에선 안 쓰인다. 키워드도 아님

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

□ Java Data Type

▪ Primitive Data Type

- 기본 데이터 타입으로 단일 값을 갖는다.
- 정수형/실수형/문자형/논리형 이 있다.

▪ Reference Data Type

- 객체에 대한 주소 값(reference)을 의미한다.
- Primitive Data Type을 제외한 모든 것이 Reference Data Type이다.

□ 논리형 : boolean

- 값 범위 : true / false
- default value : false

□ 문자형 : char

- 값 범위 : 16bit Uni-code 문자
- 표현법 : ' ' 으로 문자를 둘러 쓴다. ('A', 'Wn', 'WuAC00')
- default value : 'Wu0000'

□ 정수형 : byte / short / int / long

	byte	short	int	long
--	------	-------	-----	------

- 값 범위 : 1 2 4 8 bytes 내의 정수 값
- default value : 0 0 0 0L
- 표현법 : 10진수(10), 8진수(012), 16진수(0x0A) 로 표현 가능하다.
- 정수형 literal의 data type은 int 형이다.

□ 실수형 : float / double

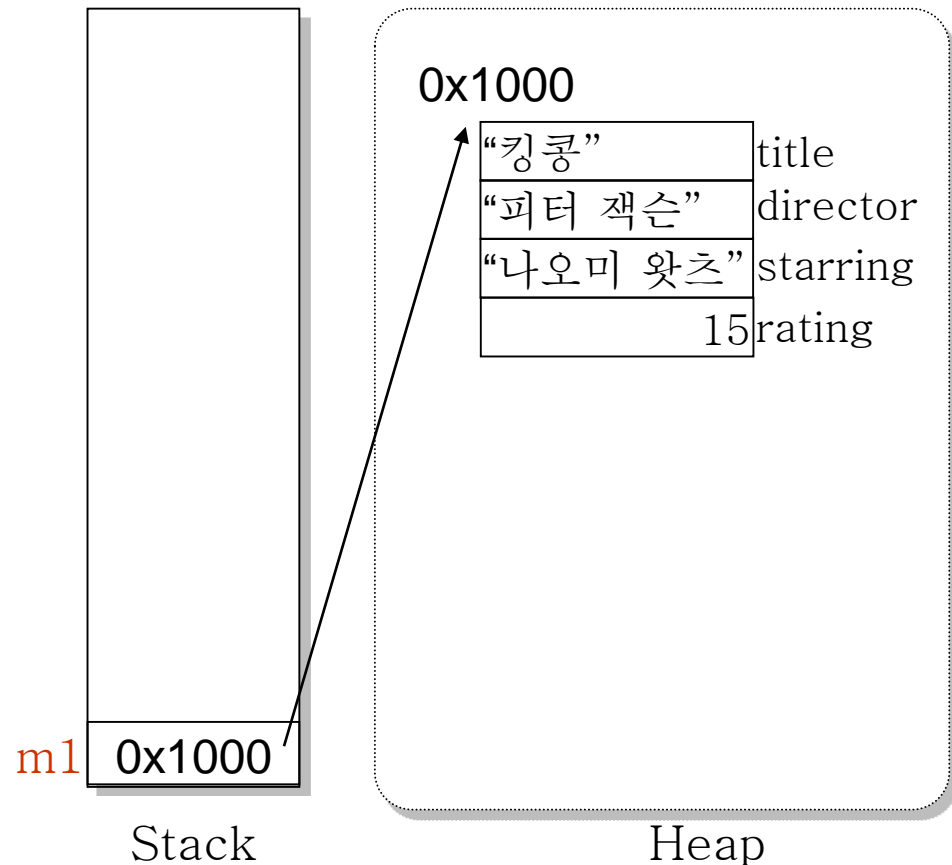
	float	double
--	-------	--------

- 값 범위 : 4 8 bytes 내의 실수 값
- default value : 0.0F 0.0D
- 실수형 literal의 data type은 double 형이다.

- ❑ Primitive type을 제외한 모든 Data Type은 Reference type이다.
- ❑ Reference type의 변수는 new를 이용해 생성한 객체의 주소 값을 가지고, 그 객체를 제어하는데 쓰인다.
- ❑ 객체의 생성은 new 키워드로서 이루어지며, 객체의 데이터 타입은 그 객체의 클래스로서 표현된다.
- ❑ default value : null

```
public class Movie {  
    public String title = "킹콩";  
    public String director = "피터 잭슨";  
    public String starring = "나오미 왓츠";  
    public int rating = 15;  
}
```

```
Movie m1 = new Movie();
```



□ 객체의 생성은 new 키워드로서 이루어진다.

ex) `Dog d = new Dog();`

□ 객체 생성시, 메모리가 할당되는 순서는 다음과 같다.

1) 메모리 할당 (reference 변수, 객체)

2) 객체의 멤버변수 초기화

i) default value로 초기화

ii) 명시적 초기화

iii) Constructor에 의한 초기화

3) 객체의 주소 값이, reference 변수에 할당

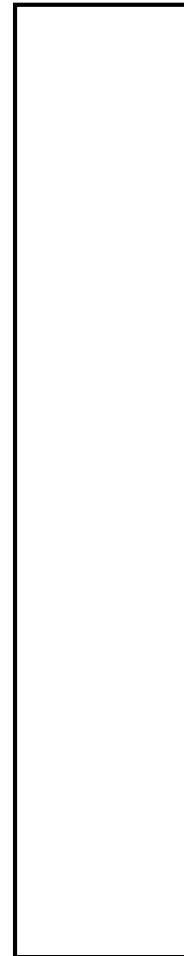
- ❑ 자기자신의 객체를 가리킨다.
- ❑ 용도
 - 자기자신이 멤버변수 참조 : `this.멤버변수_이름`
 - 자기자신의 멤버메소드 참조 : `this.멤버메소드_이름()`
 - 자기 자신의 생성자 호출 : `this(파라미터_리스트)`

```
public class MyDate {  
  
    private int day    = 1;  
    private int month  = 1;  
    private int year   = 2000;  
  
    public MyDate( int day, int month, int year ) {  
        this.day      = day;  
        this.month     = month;  
        this.year      = year;  
    }  
}
```

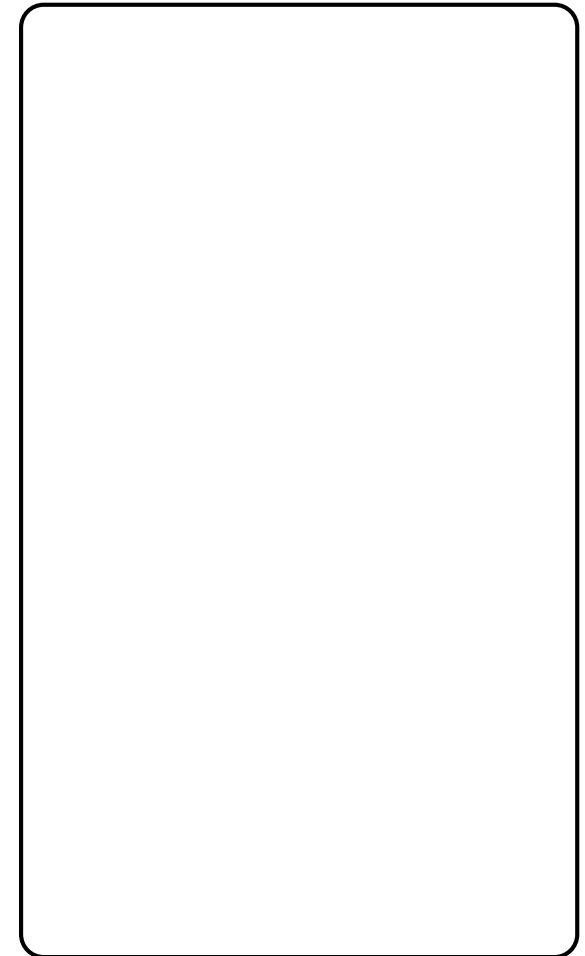


```
public class MyDate {  
  
    private int day  = 1;  
    private int month = 1;  
    private int year  = 2000;  
  
    public MyDate( int day, int month, int year ) {  
        this.day    = day;  
        this.month   = month;  
        this.year    = year;  
    }  
  
    public void print() {  
        System.out.println( "The Day is " +  
                             day + "-" + month + "-" + year );  
    }  
}
```

```
public class TestMyDate {  
    public static void main( String [ ] args ) {  
        MyDate today = new MyDate( 22, 7, 1964 );  
  
        today.print();  
    }  
}
```



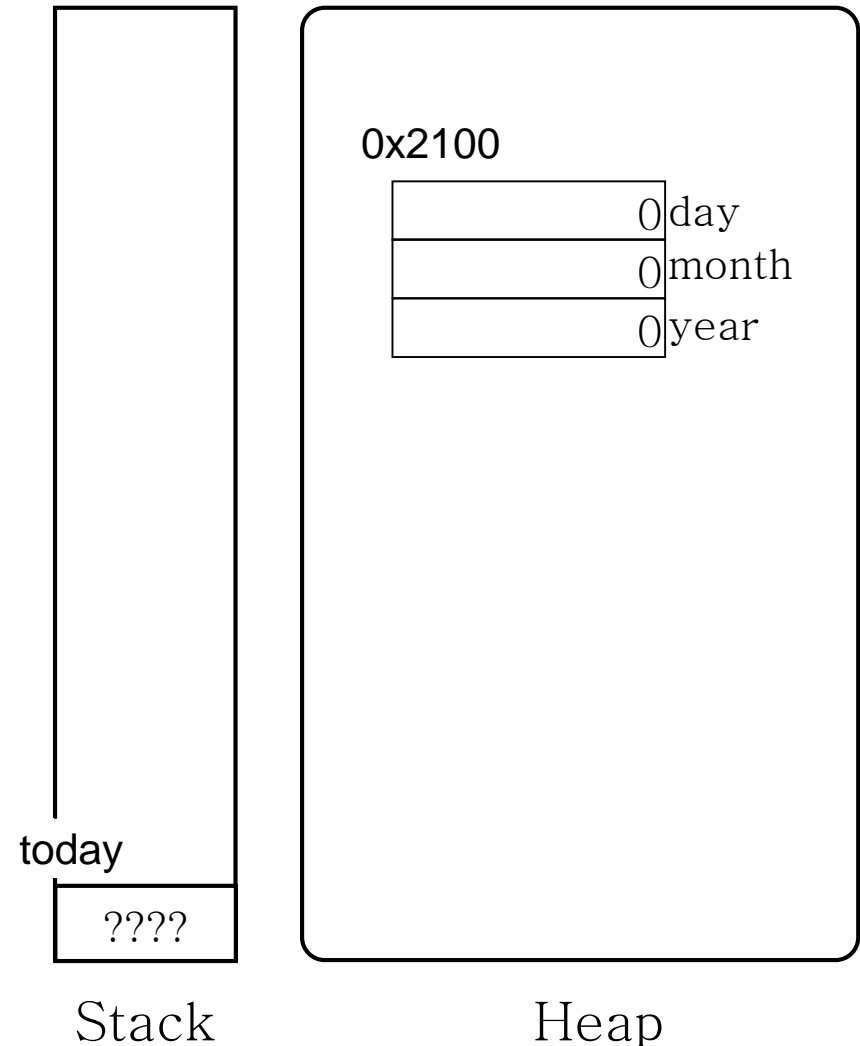
Stack



Heap

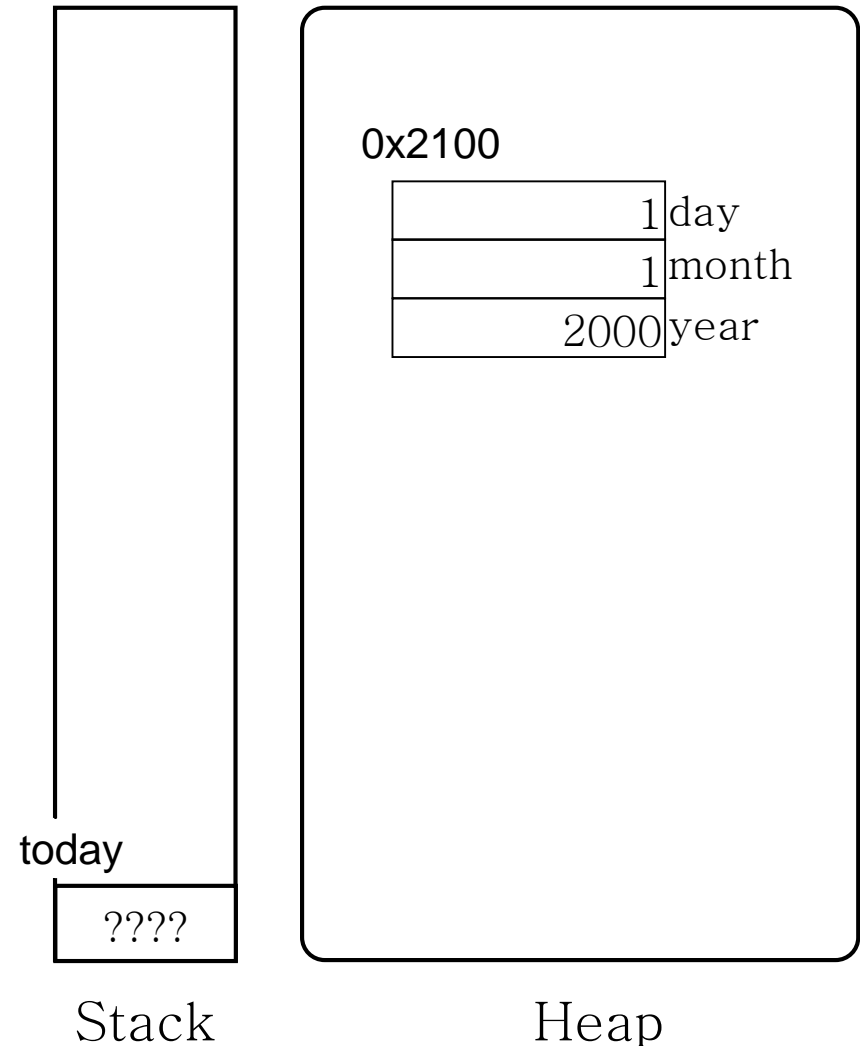
```
public class MyDate {  
  
    private int day  = 1;  
    private int month = 1;  
    private int year  = 2000;  
  
    public MyDate( int day, int month, int year ) {  
        this.day    = day;  
        this.month   = month;  
        this.year    = year;  
    }  
  
    public void print() {  
        System.out.println( "The Day is " +  
                             day + "-" + month + "-" + year );  
    }  
}
```

```
public class TestMyDate {  
    public static void main( String [ ] args ) {  
        MyDate today = new MyDate( 22, 7, 1964 );  
  
        today.print();  
    }  
}
```



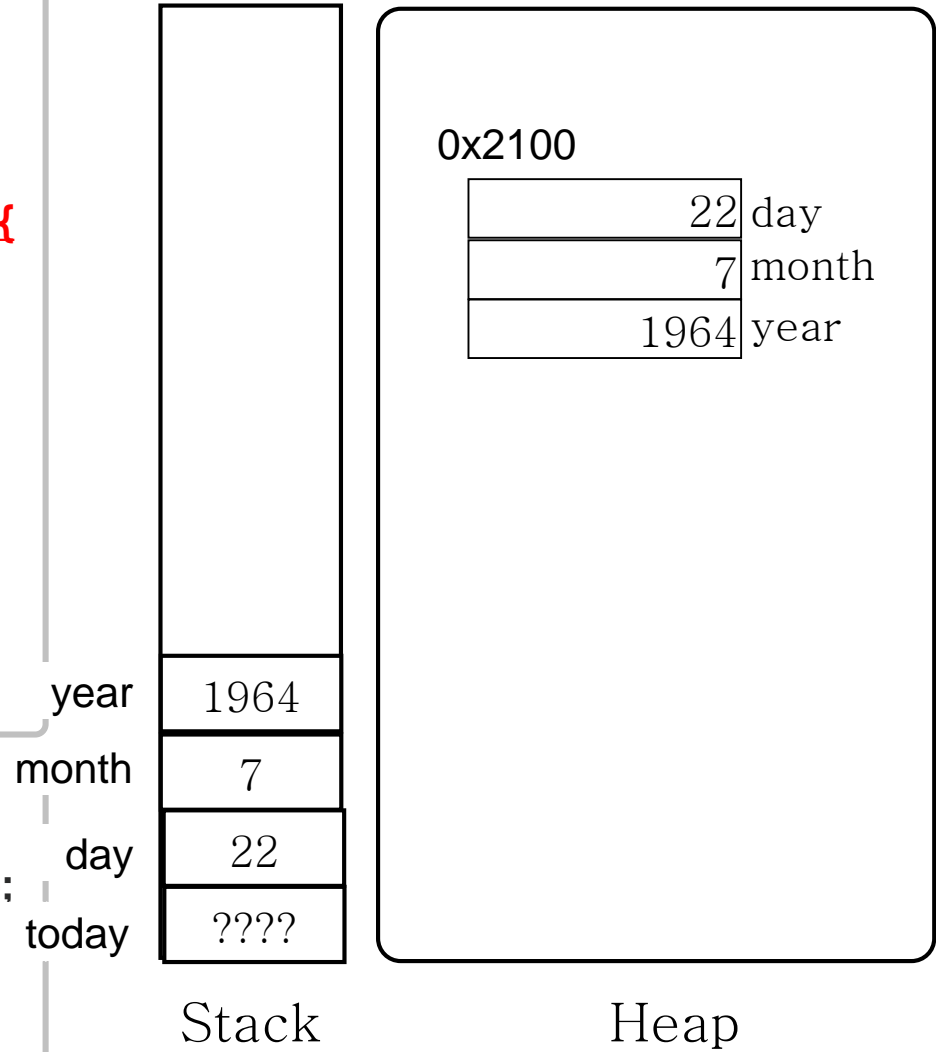
```
public class MyDate {  
  
    private int day = 1;  
    private int month = 1;  
    private int year = 2000;  
  
    public MyDate( int day, int month, int year ) {  
        this.day    = day;  
        this.month  = month;  
        this.year   = year;  
    }  
  
    public void print() {  
        System.out.println( "The Day is " +  
                             day + "-" + month + "-" + year );  
    }  
}
```

```
public class TestMyDate {  
    public static void main( String [ ] args ) {  
        MyDate today = new MyDate( 22, 7, 1964 );  
  
        today.print();  
    }  
}
```



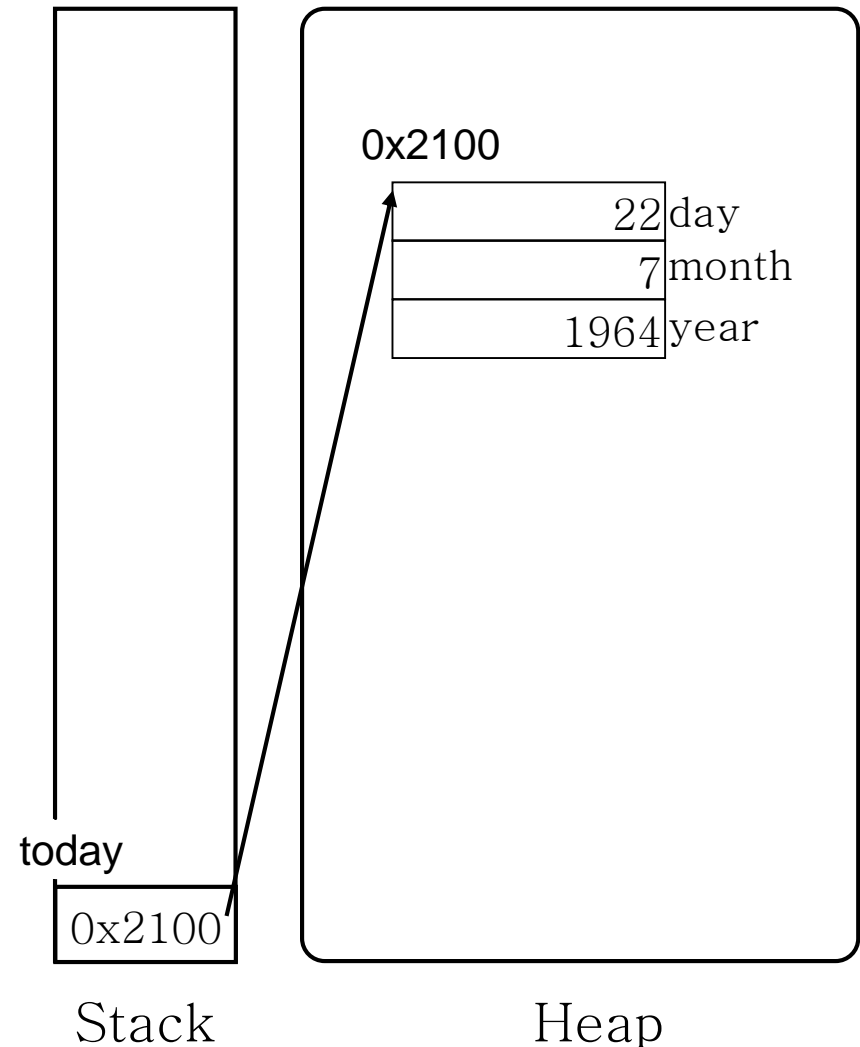
```
public class MyDate {  
  
    private int day  = 1;  
    private int month = 1;  
    private int year  = 2000;  
  
    public MyDate( int day, int month, int year ) {  
        this.day    = day;  
        this.month   = month;  
        this.year    = year;  
    }  
  
    public void print() {  
        System.out.println( "The Day is " +  
                             day + "-" + month + "-" + year );  
    }  
}
```

```
public class TestMyDate {  
    public static void main( String [ ] args ) {  
        MyDate today = new MyDate( 22, 7, 1964 );  
  
        today.print();  
    }  
}
```



```
public class MyDate {  
  
    private int day  = 1;  
    private int month = 1;  
    private int year  = 2000;  
  
    public MyDate( int day, int month, int year ) {  
        this.day    = day;  
        this.month   = month;  
        this.year    = year;  
    }  
  
    public void print() {  
        System.out.println( "The Day is " +  
                             day + "-" + month + "-" + year );  
    }  
}
```

```
public class TestMyDate {  
    public static void main( String [ ] args ) {  
        MyDate today = new MyDate( 22, 7, 1964 );  
  
        today.print();  
    }  
}
```



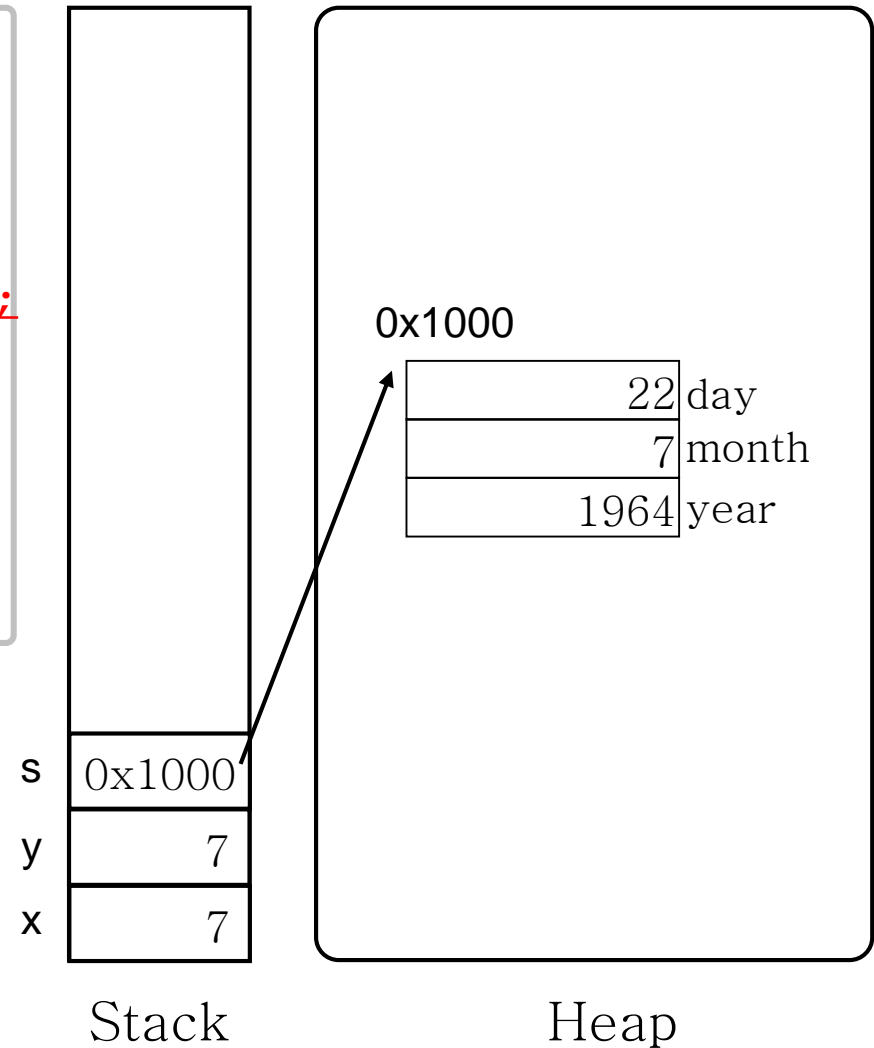
```
int x = 7;
```

```
int y = x;
```

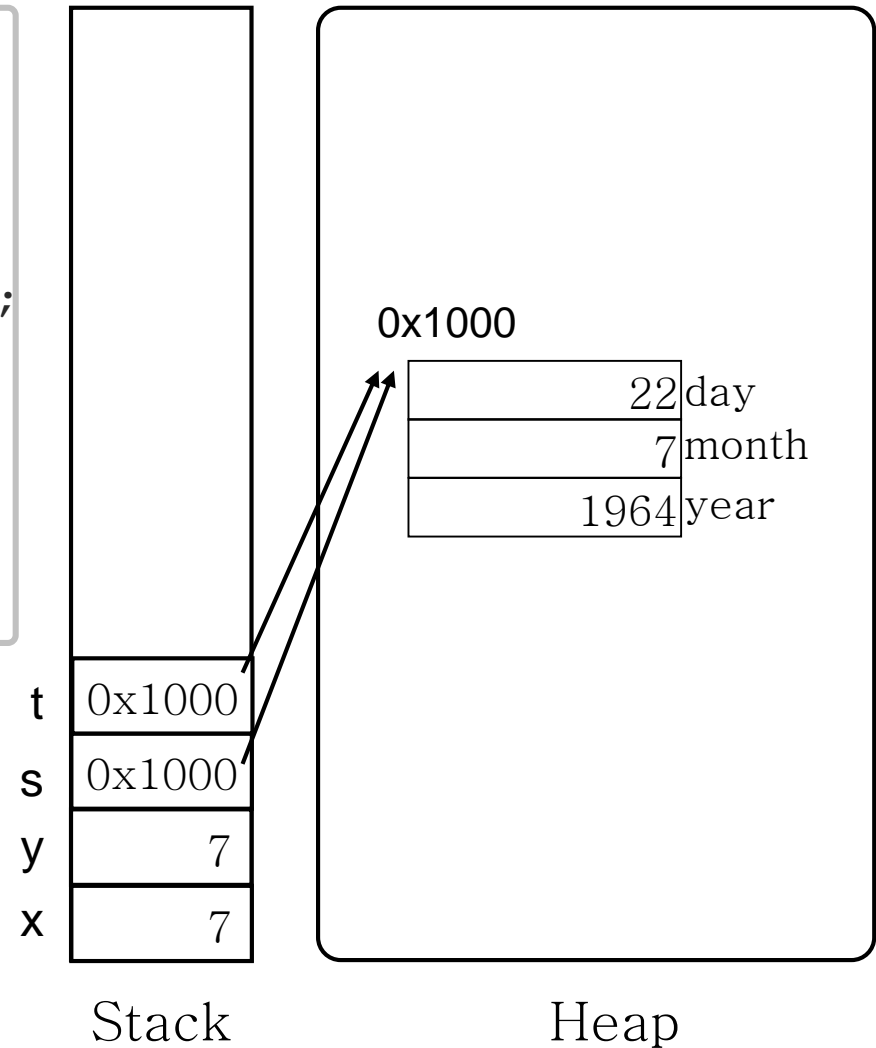
```
MyDate s = new MyDate( 22, 7, 1964 );
```

```
MyDate t = s;
```

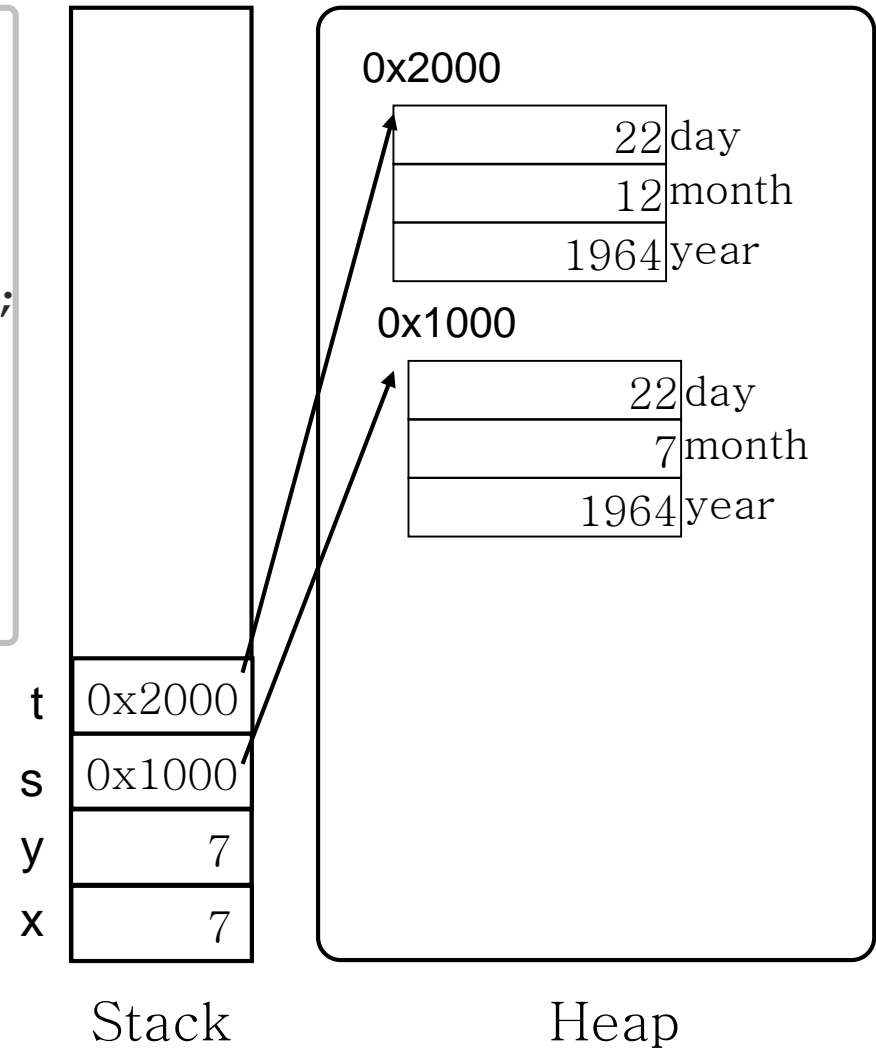
```
t = new MyDate( 22, 12, 1964 );
```



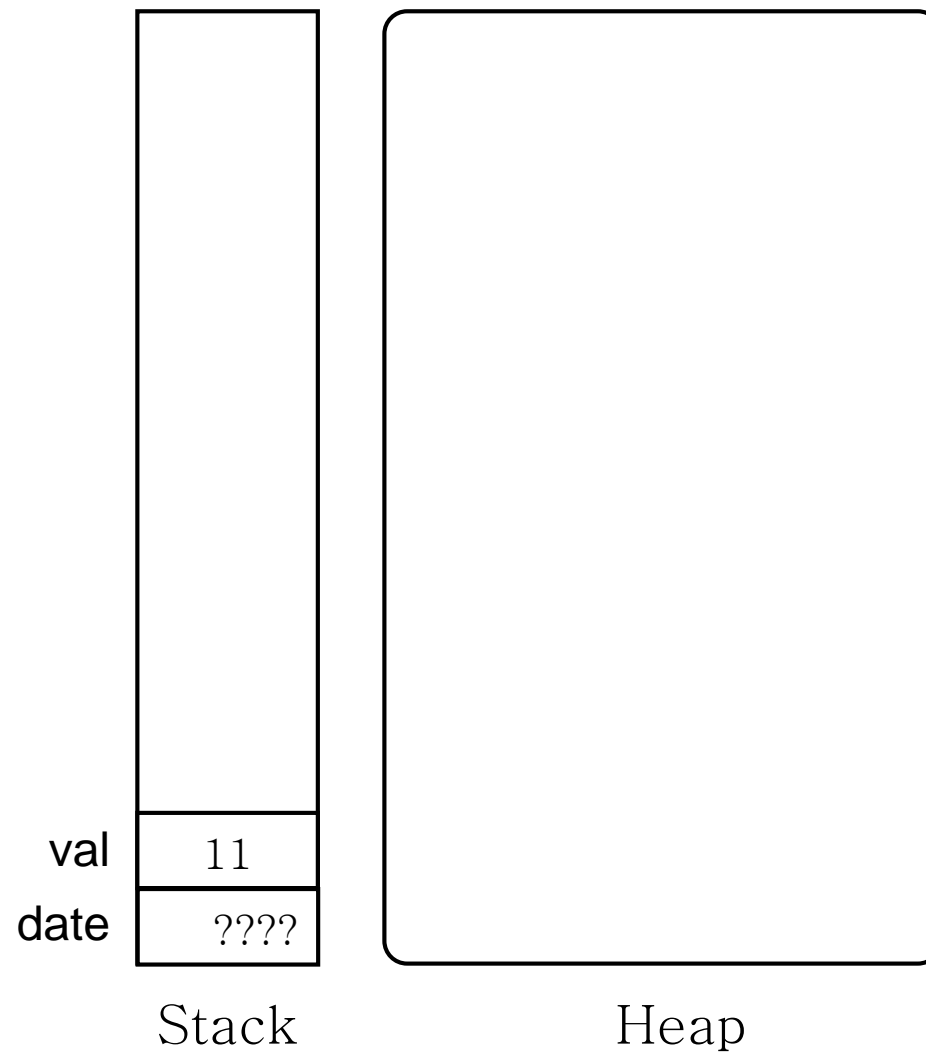
```
int x = 7;  
  
int y = x;  
  
MyDate s = new MyDate( 22, 7, 1964 );  
  
MyDate t = s;  
  
t = new MyDate( 22, 12, 1964 );
```

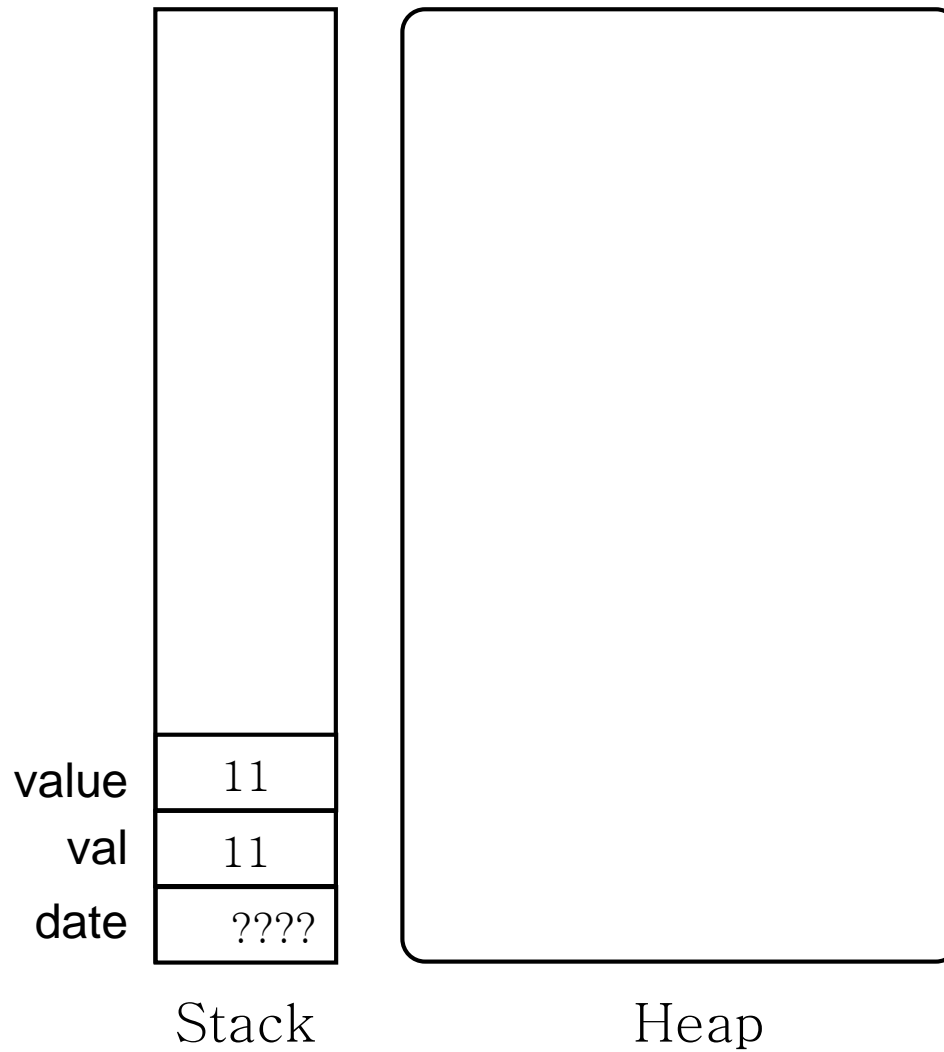


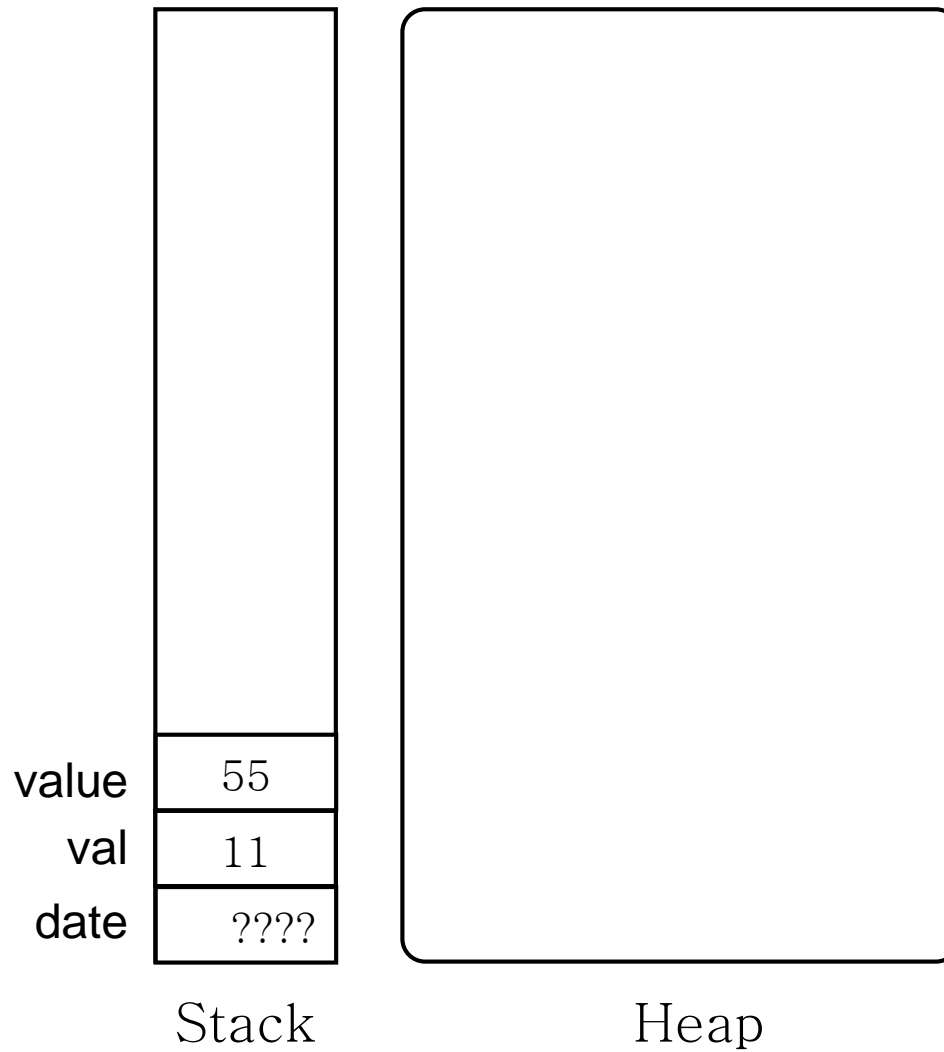
```
int x = 7;  
  
int y = x;  
  
MyDate s = new MyDate( 22, 7, 1964 );  
  
MyDate t = s;  
  
t = new MyDate( 22, 12, 1964 );
```

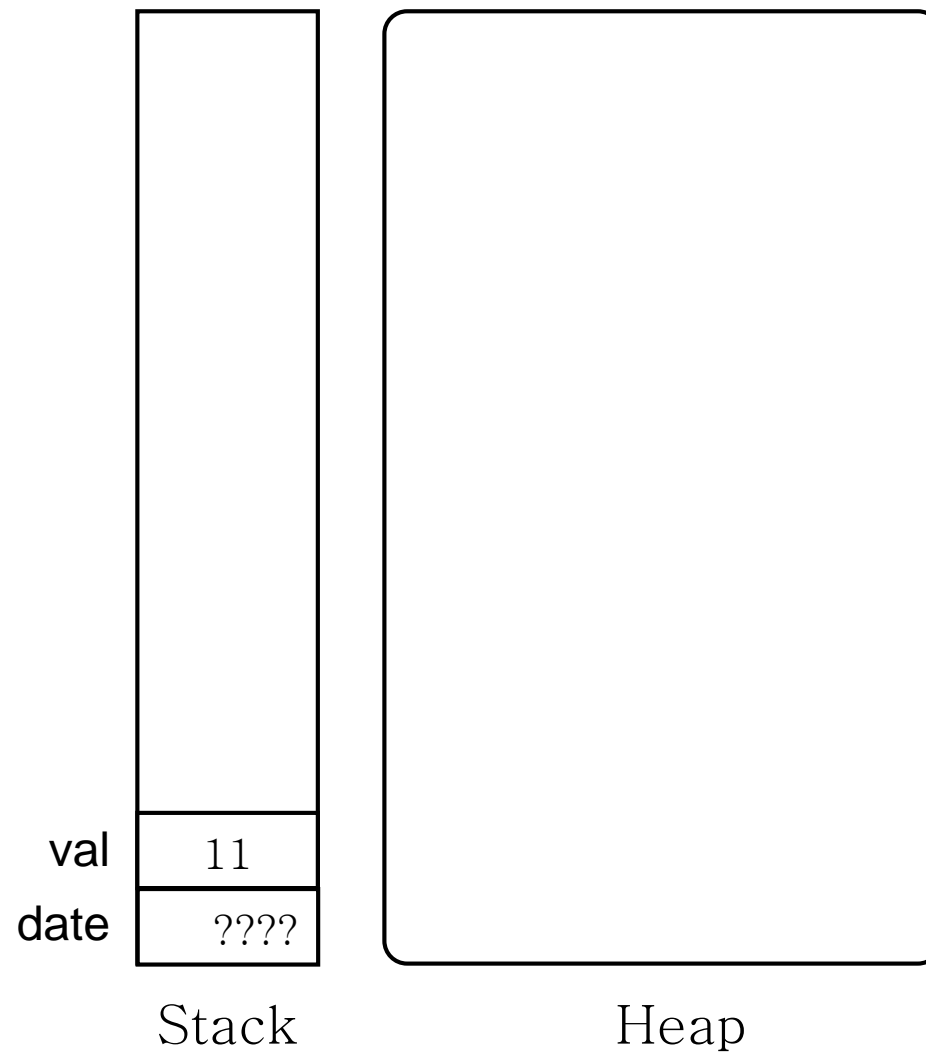


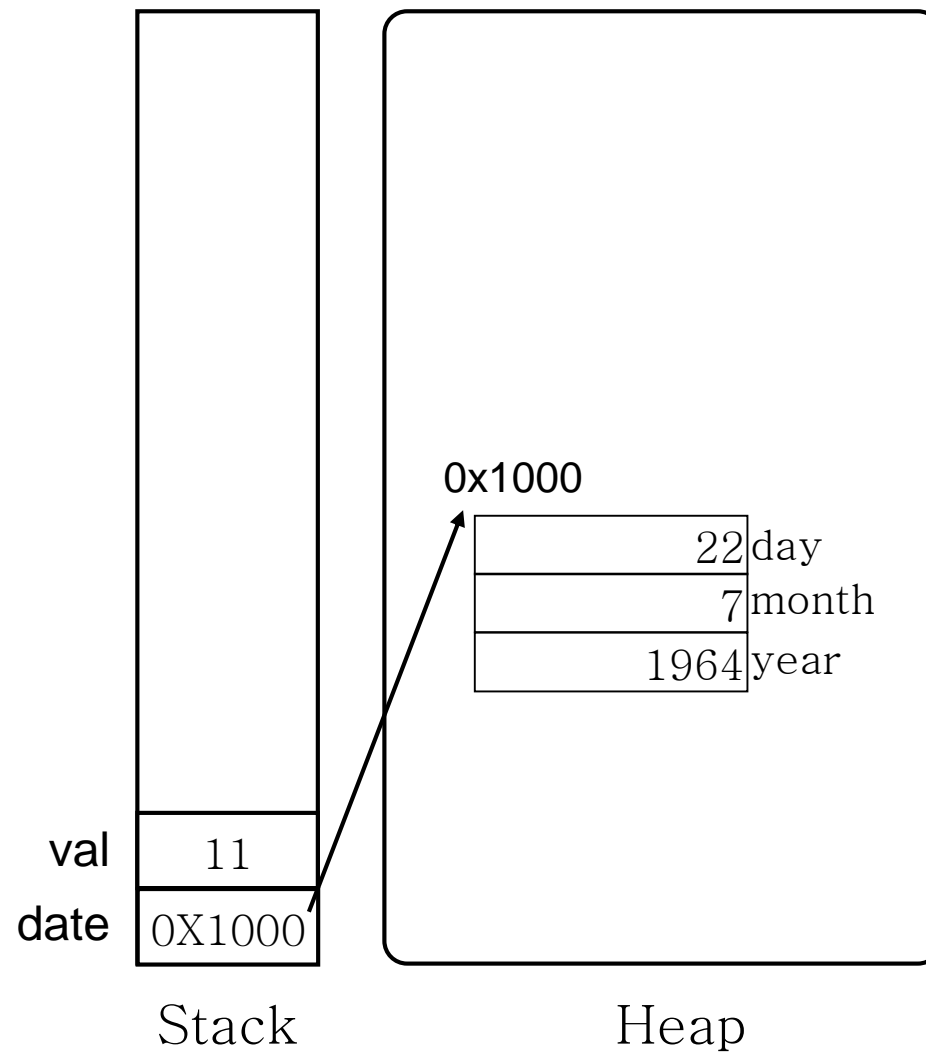

```
public class PassTest {  
    public static void changeInt(int value) { // value 공간에 11 전달되어 저장  
        value = 55;  
    } // 괄호가 닫히면, value 값은 stack 에서 사라짐  
  
    public static void changeObjectRef(MyDate ref ) { // ref 공간에 주소값 저장  
        ref = new MyDate(1,1,2000); // ref에 새로운 주소값이 할당됨  
    } // 괄호가 닫히면, ref 값은 stack에서 사라짐  
  
    public static void changeObjectAttr(MyDate ref) { // ref 공간에 주소값 저장  
        ref.setDay(4); // ref 가 가리키는 객체의 day값을 4로 변경  
    } // 괄호가 닫히면, ref 값은 stack에서 사라짐  
  
    public static void main(String []args) {  
        MyDate date;  
        int val;  
        val = 11;  
        changeInt(val); // val에 저장된 값 11이 전달됨  
        System.out.println("Int value is: " + val ); // val 값은 그대로 11 임  
        date = new MyDate(22,7,1964);  
        changeObjectRef(date); // date 객체의 주소값이 전달됨  
        date.print(); // date 객체는 여전히 같은 주소값을 가짐  
        changeObjectAttr(date); // date 객체의 주소값이 전달됨  
        date.print(); // date 객체의 day 값이 바뀌어져 있음  
    }  
}
```

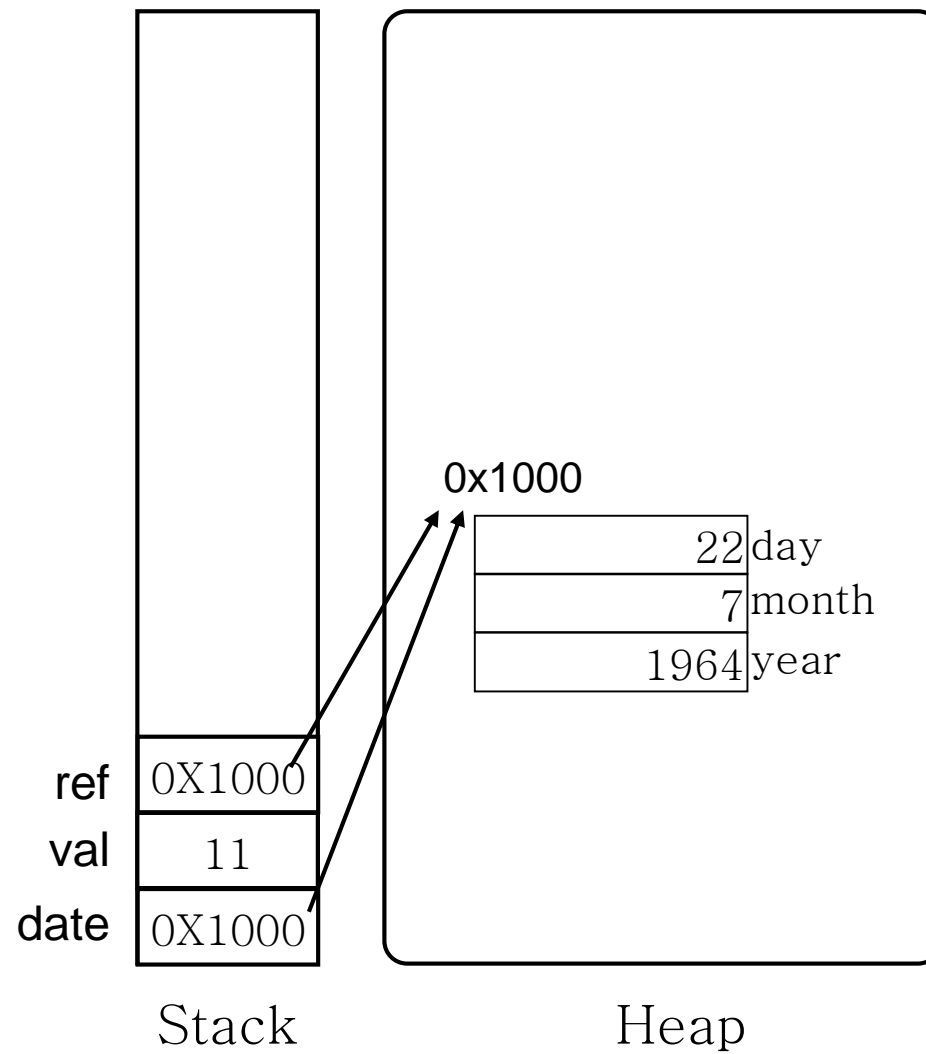


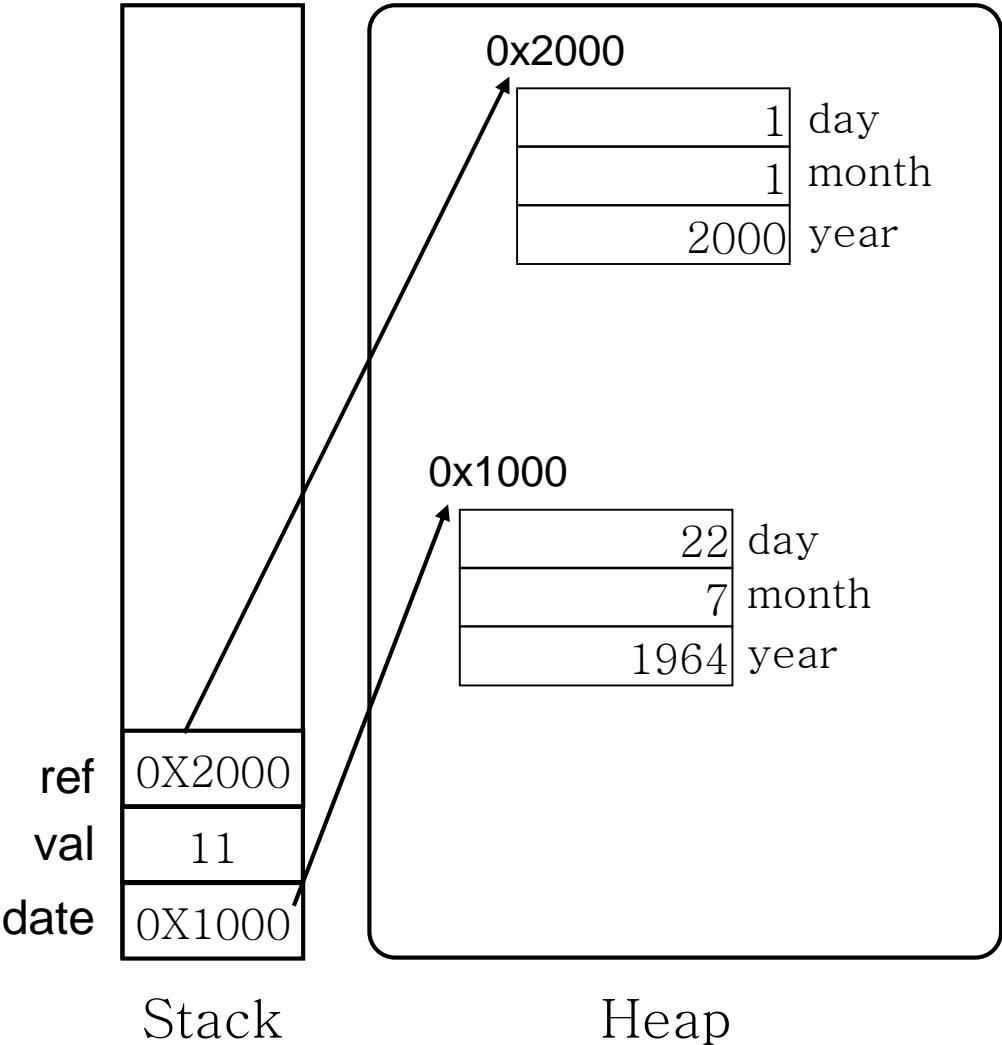


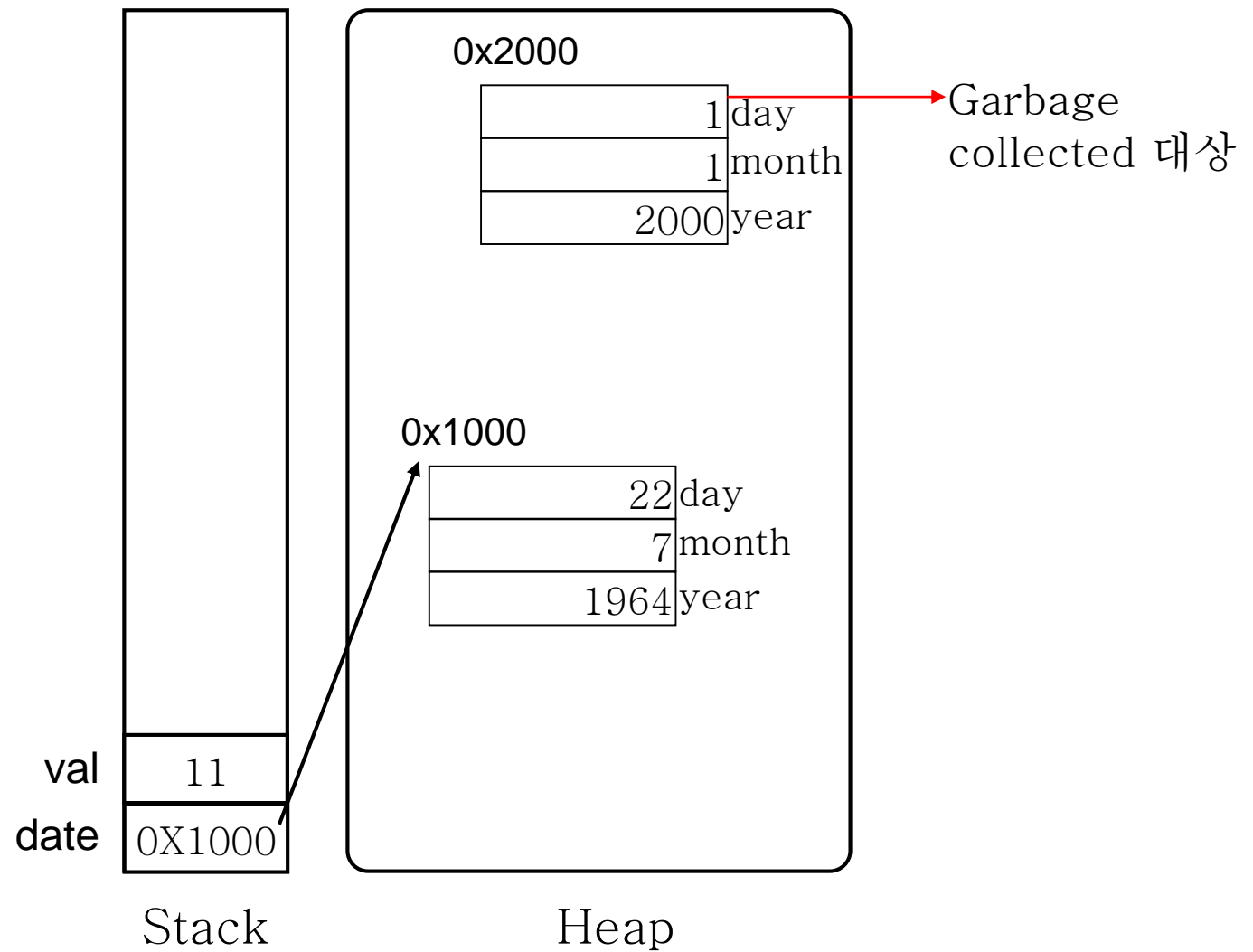


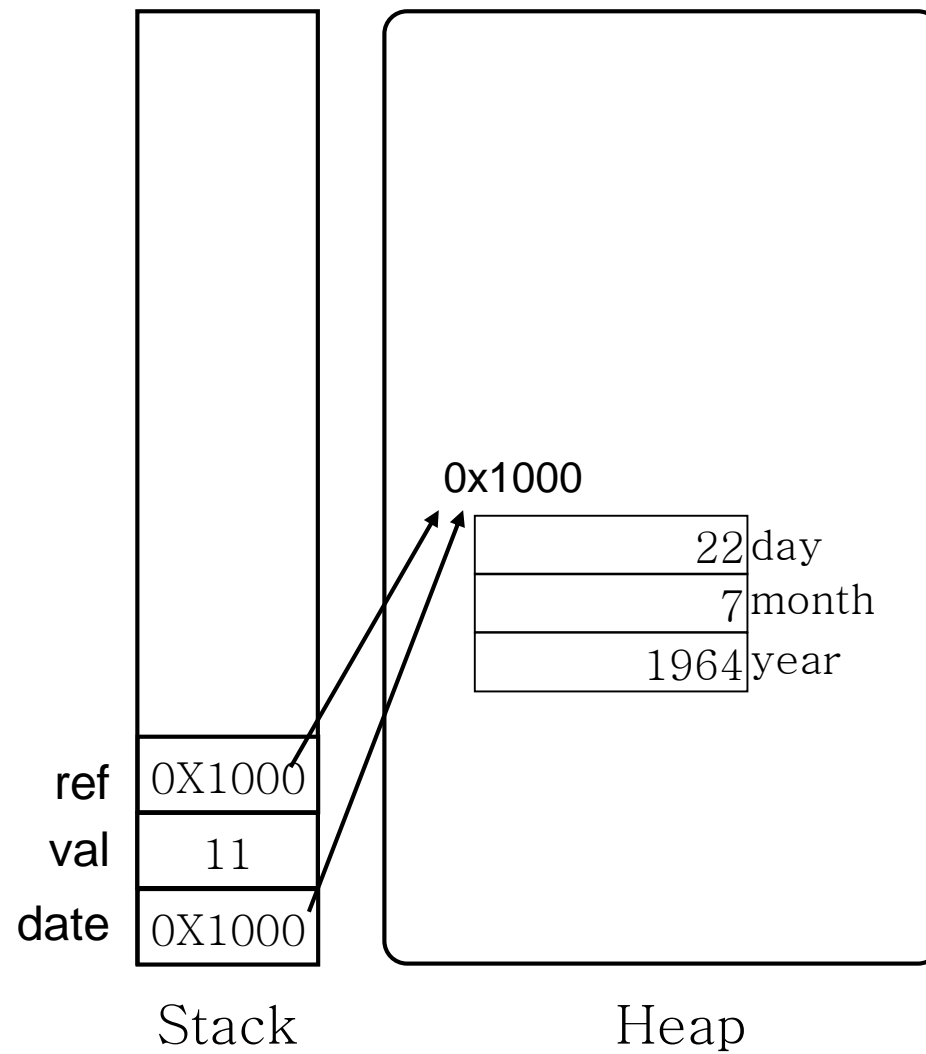


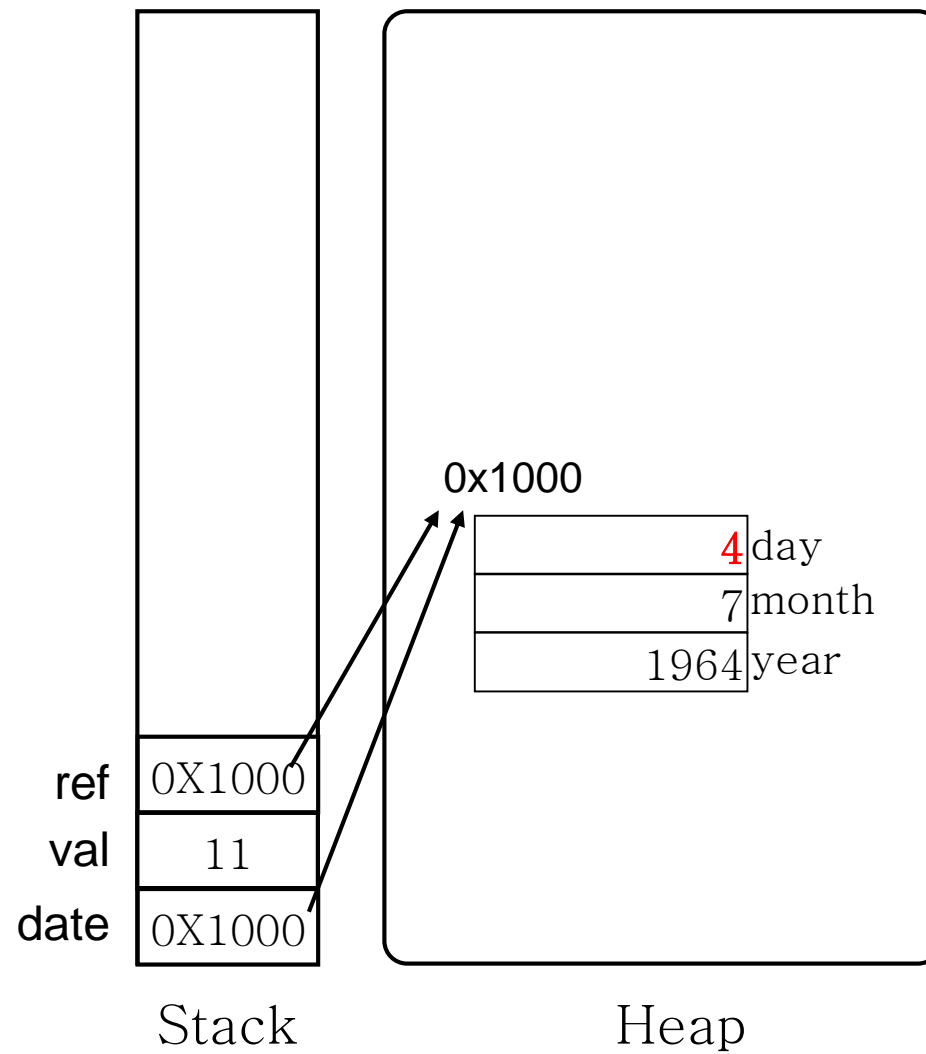


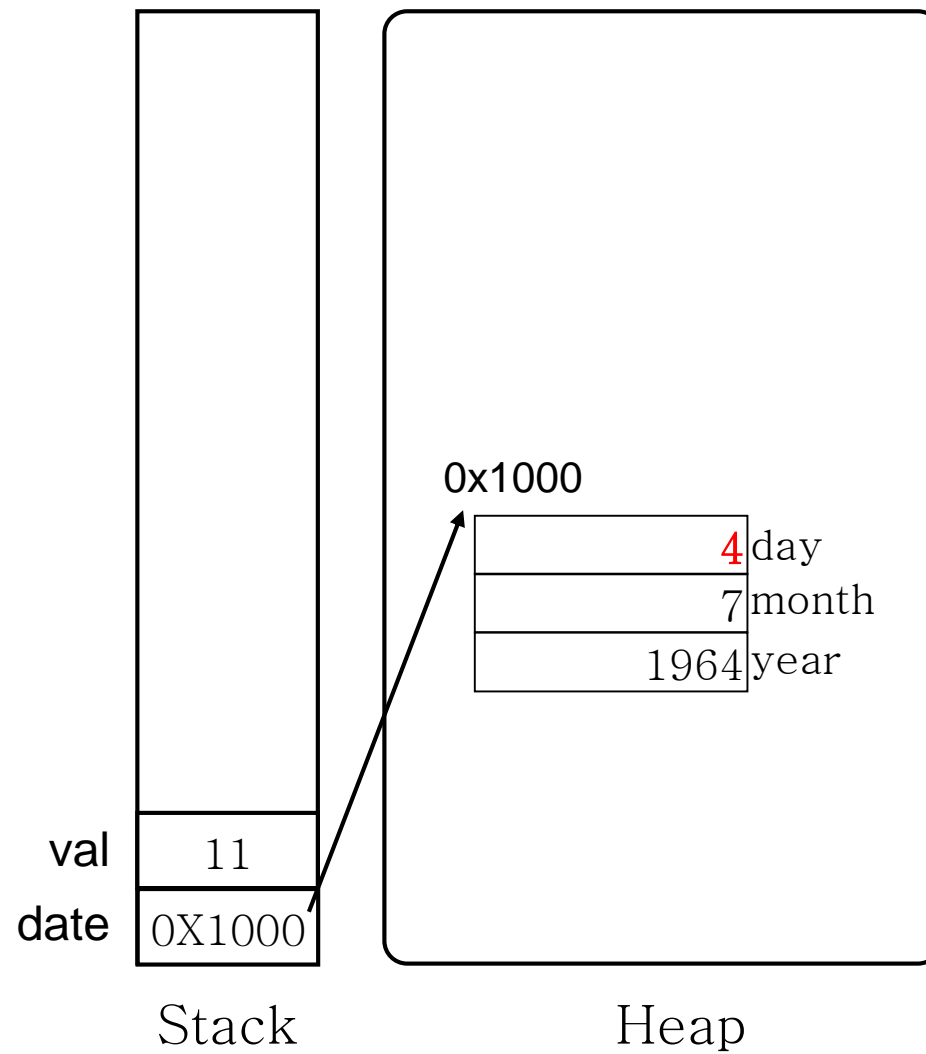












❑ 메소드 호출 시, parameter로 넘겨 지는 값은

primitive data type : 실제 값 (변수자체가 아니다.)

reference data type : 객체의 주소 값(객체 자체가 아니다.)

```
public class PassPrimitive {  
    public void passValue() {  
        int num = 10;  
  
        change( num );  
  
        System.out.println( num );  
    }  
  
    public void change( int num ) {  
        num = num + 10;  
    }  
  
    public static void main( String[] args ) {  
        PassPrimitive pp = new PassPrimitive();  
  
        pp.passValue();  
    }  
}
```

```
public class MyDate {
    private int day;
    private int month;
    private int year;

    public MyDate( int day, int month, int year ) {
        this.day = day;
        this.month = month;
        this.year = year;
    }

    public int getDay() {
        return day;
    }

    public void setDay( int day ) {
        this.day = day;
    }

    public int getMonth() {
        return month;
    }

    public void setMonth( int month ) {
        this.month = month;
    }
    ...
}
```

```
public class TestMyDate {
    public static void main( String [] args ) {
        MyDate today = new MyDate( 22, 7, 1964 );
        TestMyDate test = new TestMyDate();

        today.setDay( 30 );

        test.print( today );
    }

    public void print( MyDate day ) {
        System.out.println( "The Day is " +
            day.getDay() + "-" +
            day.getMonth() + "-" +
            day.getYear()
        );
    }
}
```

❑ Class : 대문자로 시작하고, 다음문자부터는 소문자.

여러 단어 조합인 경우, 각 단어의 시작 문자는 대문자.

ex) public class **MemberEntity** { }

❑ Interface : class와 동일

ex) public interface **ConnectionPool** { }

❑ Method : 소문자 시작, 다음 단어부터는 첫 문자만 대문자.

ex) public void **displayMovieInfo**() { }

❑ Variable (변수) : 소문자 시작, 다음 단어부터는 첫 문자만 대문자.

ex) String **memberName**;

❑ Constant (상수) : 모두 대문자, 단어 사이는 '_' (under bar) 기호로 구분한다.

ex) final int **MAX_NUMBER**;

Chap 04. Expressions and Flow Control

1. 변수의 scope
2. Logical Operators
3. String Concatenation
4. Casting 과 Promotion
5. 조건 분기문 (if, switch~case)
6. 순환문 (for, while, do~while)

□ 변수의 분류

- 변수의 내용에 따라
 - Primitive type
 - Reference type
- 선언위치에 따라
 - 메소드 내부 선언, 메소드 파라미터 : Local Variable
 - 클래스 안 & 메소드 밖 선언 : Member Variable, Class Variable

□ 변수의 생명 , 초기화 문제 , 참조 범위

- Local 변수
 - 메소드 시작시 생성, 메소드 종료시 해제 (stack)
 - 자동 초기화 안 된다. (반드시 사용전 명시적 초기화 필요)
 - 해당 메소드 내 참조
- 멤버 변수
 - 객체 생성시 생성, 객체 메모리 해제시 같이 해제 (heap)
 - 자동 초기화
 - 한 객체 내 참조
- 클래스 변수 (static)
 - 클래스 로드시 생성, Application 종료시 해제 (static 영역)
 - 자동 초기화
 - 전 클래스 객체 공통 참조

```
package chap04;

public class Initializing {

    public void doComputation() {

        int x = 10;
        int y;
        int z = 0;

        if ( x > 50 ) {
            y = 9;
        }

        z = y + x;
    }
}
```

```
int x = 7;

int y = x;

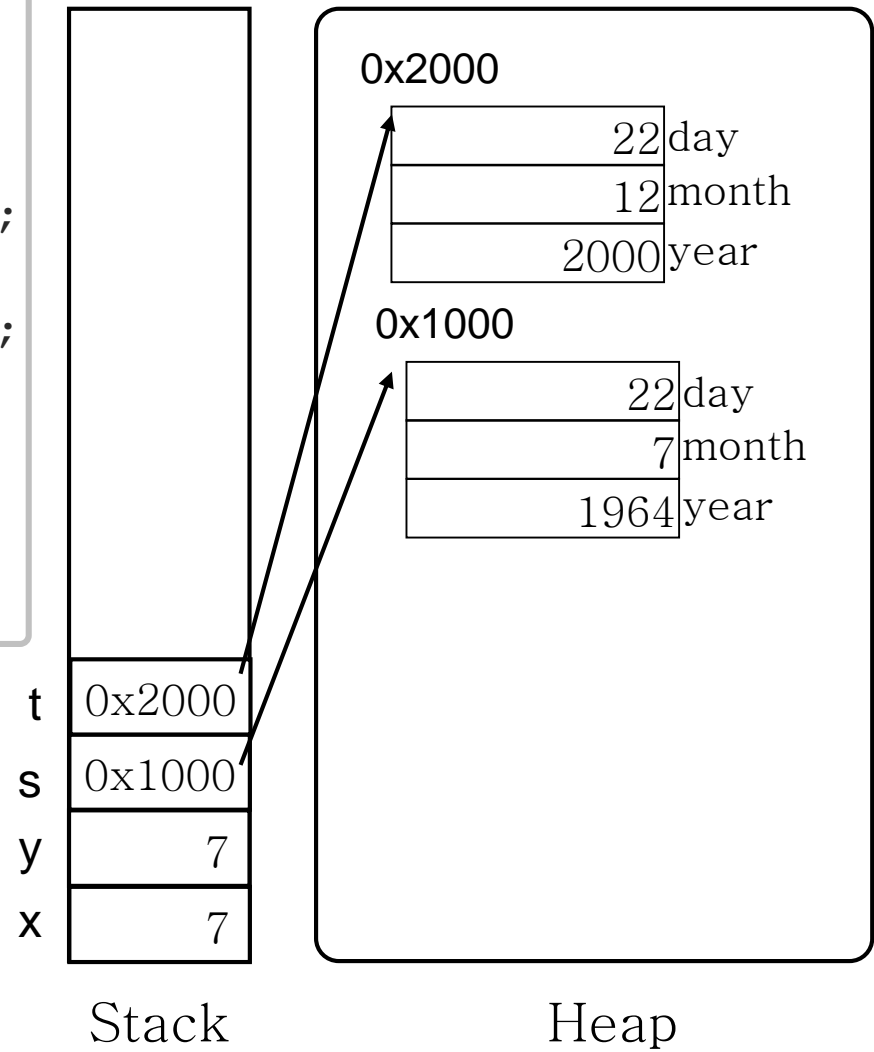
MyDate s = new MyDate( 22, 7, 1964 );

MyDate t = new MyDate( 22,12, 2000 );

s.setDay( 30 ); → 자신의 객체를 어떻게
                  찾을까?
```

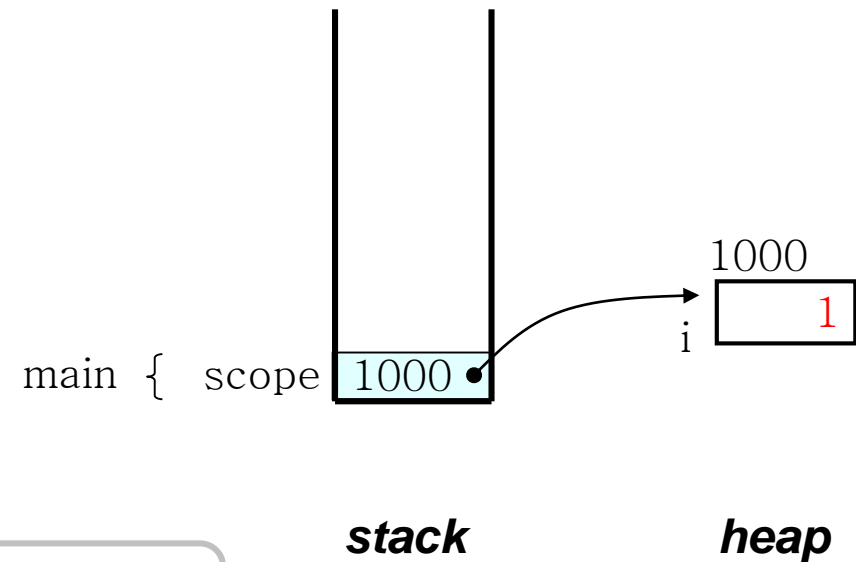
```
Code 들...
public void setDay( int day) {
    this.day = day;
}
```

Method
Area



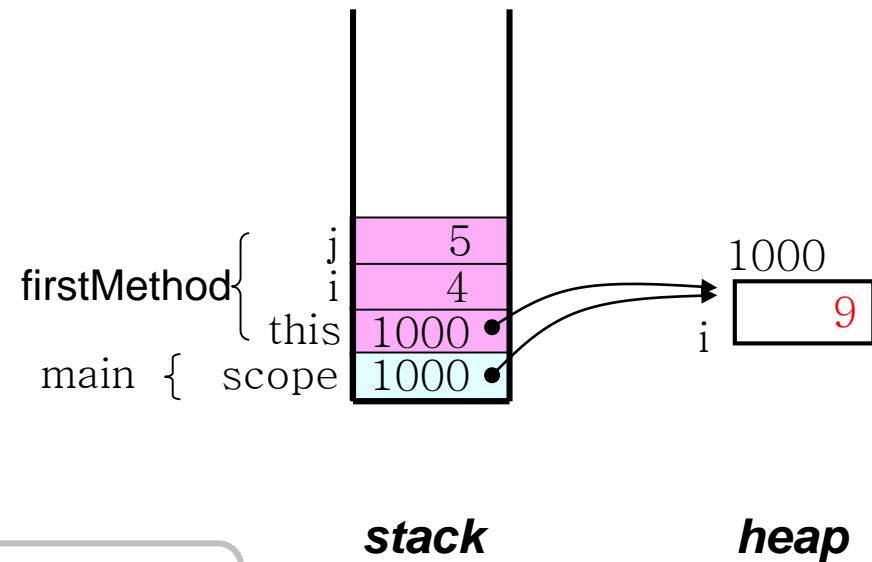
```
public class ScopeExample {  
    private int i = 1;  
  
    public void firstMethod() {  
        int i = 4, j = 5;  
  
        this.i = i + j;  
        secondMethod( 7 );  
    }  
  
    public void secondMethod( int i ) {  
        int j = 8;  
  
        this.i = i + j;  
    }  
}
```

```
public class TestScoping {  
    public static void main( String [] args ) {  
        ScopeExample scope = new ScopeExample();  
        scope.firstMethod();  
    }  
}
```



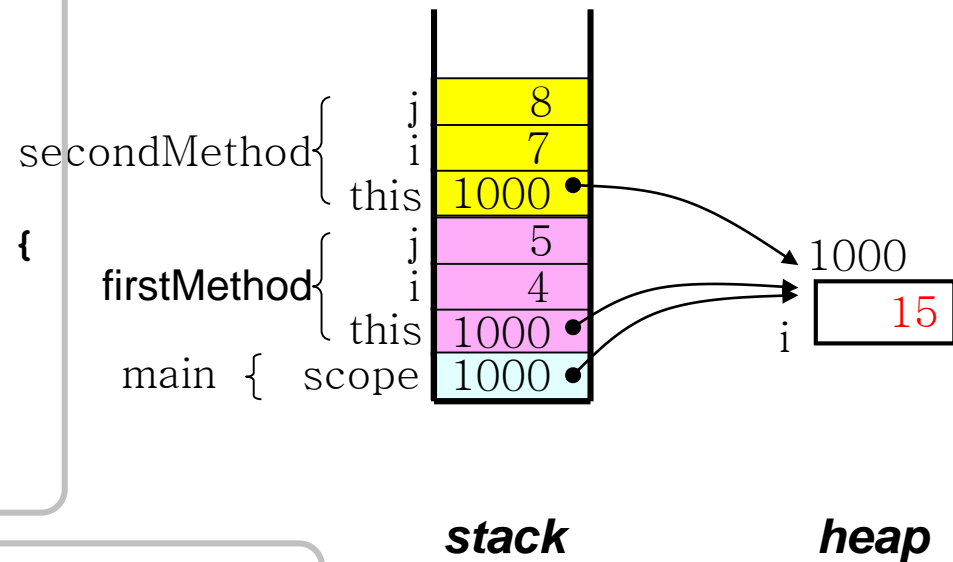
```
public class ScopeExample {  
    private int i = 1;  
  
    public void firstMethod() {  
        int i = 4, j = 5;  
  
        this.i = i + j;  
        secondMethod( 7 );  
    }  
  
    public void secondMethod( int i ) {  
        int j = 8;  
  
        this.i = i + j;  
    }  
}
```

```
public class TestScoping {  
    public static void main( String [] args ) {  
        ScopeExample scope = new ScopeExample();  
        scope.firstMethod();  
    }  
}
```



```
public class ScopeExample {  
    private int i = 1;  
  
    public void firstMethod() {  
        int i = 4, j = 5;  
  
        this.i = i + j;  
        secondMethod( 7 );  
    }  
  
    public void secondMethod( int i ) {  
        int j = 8;  
  
        this.i = i + j;  
    }  
}
```

```
public class TestScoping {  
    public static void main( String [] args ) {  
        ScopeExample scope = new ScopeExample();  
        scope.firstMethod();  
    }  
}
```



Operators	Associative	Precedence
<code>++ -- + - ~ ! (data_type)</code>	R to L	<div>High</div> <div>↑</div> <div>↓</div> <div>Low</div>
<code>* / %</code>	L to R	
<code>+ -</code>	L to R	
<code><< >> >>></code>	L to R	
<code>< > <= >= instanceof</code>	L to R	
<code>== !=</code>	L to R	
<code>&</code>	L to R	
<code>^</code>	L to R	
<code> </code>	L to R	
<code>&&</code>	L to R	
<code> </code>	L to R	
<code>? :</code>	R to L	
<code>= *= /= %= += -= <<=</code> <code>>>= >>>= &= ^= =</code>	R to L	

Java Operators

- ✓ 사칙 연산 : + , - , * , /
- ✓ 나머지 연산 : %
- ✓ 단항 증감 연산자 : ++ , --
- ✓ 같은지 비교 : ==
- ✓ 다른지 비교 : !=
- ✓ 크기 비교 : < , > , <= , >=
- ✓ 객체 비교 : instanceof
- ✓ 논리 연산 : & , | , && , || , !
- ✓ 타입 변환 : (data_type)

주의

```
int num = 10;
```

```
int result = 0;
```

```
1) result = num++;
```

```
→ result = num;
```

```
num = num + 1;
```

```
2) result = ++num;
```

```
→ num = num + 1;
```

```
result = num;
```

□ Logical Operators

- ! : NOT
- & : AND
- | : OR

A & B		Result
True	True	True
True	False	False
False	True	False
False	False	False

A && B		Result
True	True	True
True	False	False
False		False
False		False

A B		Result
True	True	True
True	False	True
False	True	True
False	False	False

A B		Result
True		True
True		True
False	True	True
False	False	False

□ Short-circuit Operators

- && : AND
- || : OR

! A	Result
True	False
False	True

```
int divisor = 0;  
int dividend = 100;
```

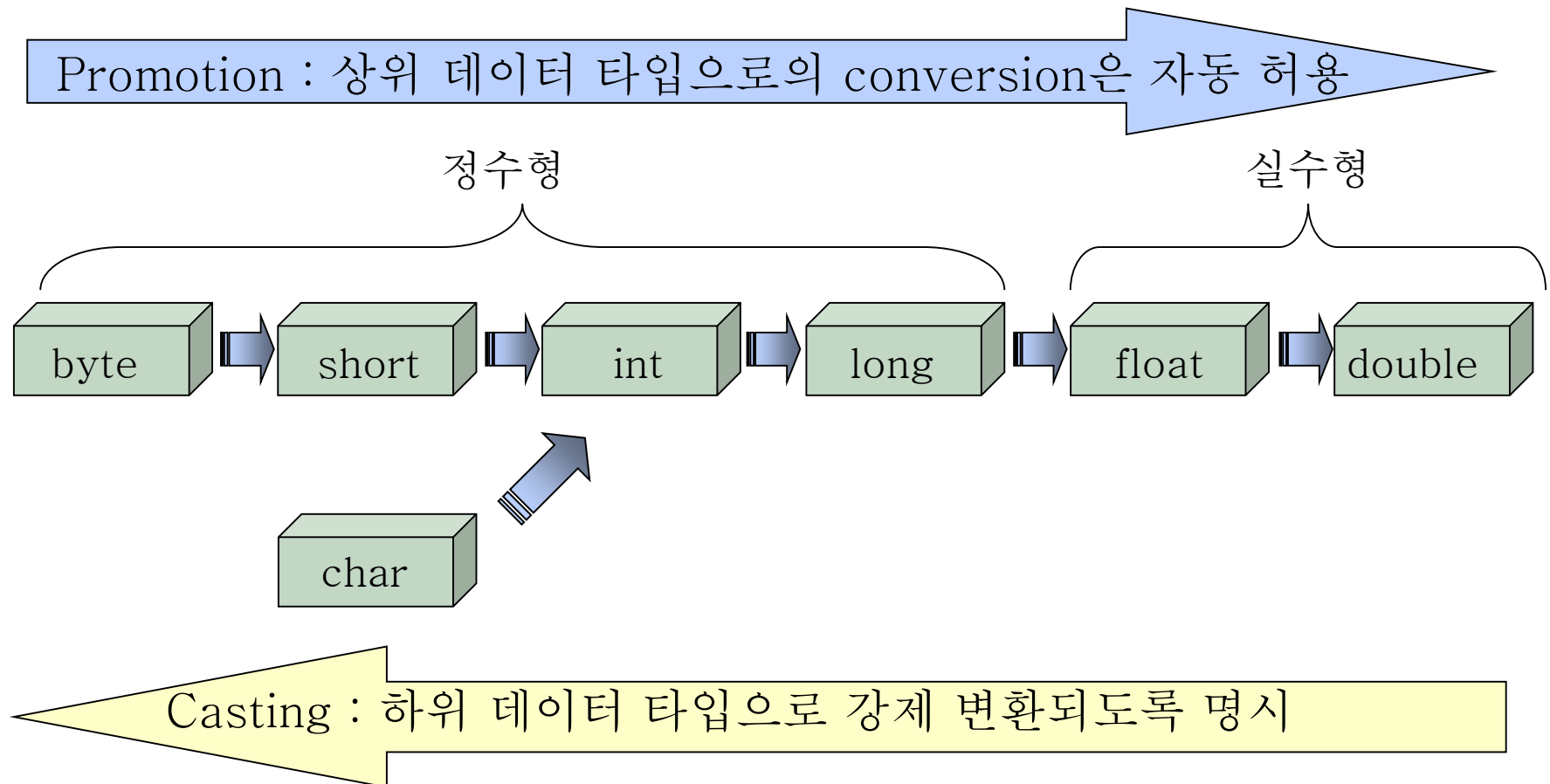
1)

```
if ( divisor != 0 & dividend/divisor > 10 ) {  
    System.out.println( "몫이 10보다 크다" );  
}
```

2)

```
if ( divisor != 0 && dividend/divisor > 10 ) {  
    System.out.println( "몫이 10보다 크다" );  
}
```

대원칙 >> 정보를 잃지 않는 방향으로 데이터 타입 간의 변환을 허용한다.



```
long longVal = 6;    // OK (Promotion)
int intVal = 99L;    // compile error
```

```
double doubleVal = 12.414F; // OK (promotion)
float floatVal = 12.141;    // compile error
```

```
long longVal = 99L;
int intVal1 = longVal;    // compile error
int intVal2 = (int)longVal; //Casting 했으므로 OK (정보도 잃지 않음)
```

[주의]

```
int intVal = 128;
byte byteVal = (byte)intVal; // byteVal에 저장될 값 보장 못함
```

```
long longVal = 99L;  
int intVal1 = longVal;           //error  
int intVal2 = (int)longVal;      //OK
```

[주의]

```
int intVal = 128;  
byte byteVal = (byte)intVal;
```

- ❑ 큰 데이터 타입을 작은 데이터 타입에 넣을 수 없다. → Compile Error
- ❑ Casting 연산자를 사용하여, 임시적으로 데이터 타입을 변환할 수 있다.
- ❑ Casting 연산자를 사용할 때에는, 적절한 범위의 값이 할당 되어야 하는 것을 주의한다.
- ❑ Primitive data type 뿐만 아니라, Reference data type도 Type Casting이 가능하다.

- ❑ `expr` 결과 값이 boolean인 문장들이나 식들이 올 수 있다.
- ❑ 중첩 if 문 사용시에는 대응 되는 `else` 문장을 주의 하여 작성한다.

```
if ( expr1 ) {  
    statements;  
}
```

```
if ( expr1 ) {  
    statements1;  
} else {  
    statements2;  
}
```

```
if ( expr1 ) {  
    statements1;  
} else if ( expr2 ) {  
    statements2;  
} else if ( expr3 ) {  
    statements3;  
} else {  
    statements4;  
}
```


Branching Statements – if Example

```
public class IfTest {  
    public static void main( String[] args ) {  
        int i = 200;  
        if ( i < 100 )  
            if ( i > 10 )  
                System.out.println( "10보다 크고 100보다 작다" );  
        else  
            System.out.println( "i는 100이상이다." );  
    }  
}
```

- ❑ `expr` 은 int형으로 표현 될 수 있는 값이어야 한다. (byte, short, char, int)
- ❑ `constant` 는 int형 상수여야 한다.
- ❑ `break` 문이 없으면 fall-through 현상이 발생한다.

```
switch( expr ) {  
    case constant1 :  
        statements1;  
        break;  
    case constant2 :  
        statements2;  
        break;  
    default :  
        statements3;  
        break;  
}
```

Branching Statements – switch Example

```
public class SwitchTest {  
    public static void main( String [] args ) {  
        int aaa = 10;  
        int bbb = 0;  
        switch( aaa ) {  
            case 5 :  
                bbb = bbb + 1;  
                break;  
            case 10 :  
                bbb = bbb + 2;  
                break;  
            case 15 :  
                bbb = bbb + 4;  
                break;  
            default :  
                bbb = bbb + 3;  
                break;  
        }  
        System.out.println( bbb );  
    }  
}
```

- ❑ Logic을 일정 조건 동안 반복적으로 수행하는 LOOP 구조에는 for, while, do~while 등 3가지가 있다.
- ❑ test_expr 의 결과는 boolean 타입 이어야 한다.

```
for ( init_expr ; test_expr ; alter_expr ) {  
    Statements;  
}
```

```
while ( test_expr ) {  
    Statements;  
}
```

```
do {  
    Statements;  
} while ( test_expr );
```

Looping Statements – Example

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

```
int inx = 0;  
  
while ( inx < 10 ) {  
    System.out.println( inx );  
    inx++;  
}
```

```
int inx = 0;  
  
do {  
    System.out.println( inx );  
    inx++;  
} while ( inx < 10 );
```

Chap 05. Arrays

1. 배열 선언, 생성, 초기화
 - Primitive type Array
 - Reference type Array
2. 배열 Access
3. System.arraycopy()
4. 2차원 배열

- 정의 : 동일한 타입의 여러 데이터를 하나의 이름으로 관리하는 것

- 종류

 - Primitive type array

 - Reference type array

- 배열은 객체이다.

- 배열의 이름은 배열객체의 주소 값을 갖는 Reference 변수이다.

- 선언 : 어느 type을 배열의 element로 가질지 선언한다.

```
int[ ] numbers; ⇔ int numbers[];
```

```
MyDate[ ] dates; ⇔ MyDate dates[];
```

- 생성 : 배열도 객체이므로 new 키워드로 생성하며, [] 안에 몇 개의 element를 가질지 size를 지정한다.

```
numbers = new int[10];
```

```
dates = new MyDate[10];
```

- 초기화 : 사용하기 전에, 처음 값을 할당하는 것을 말한다.

```
numbers[0] = 1;
```

```
dates[0] = new MyDate();
```

□ 선언시 주의점.

`int[] a, b; // a, b 둘다 Array에 대한 Reference`

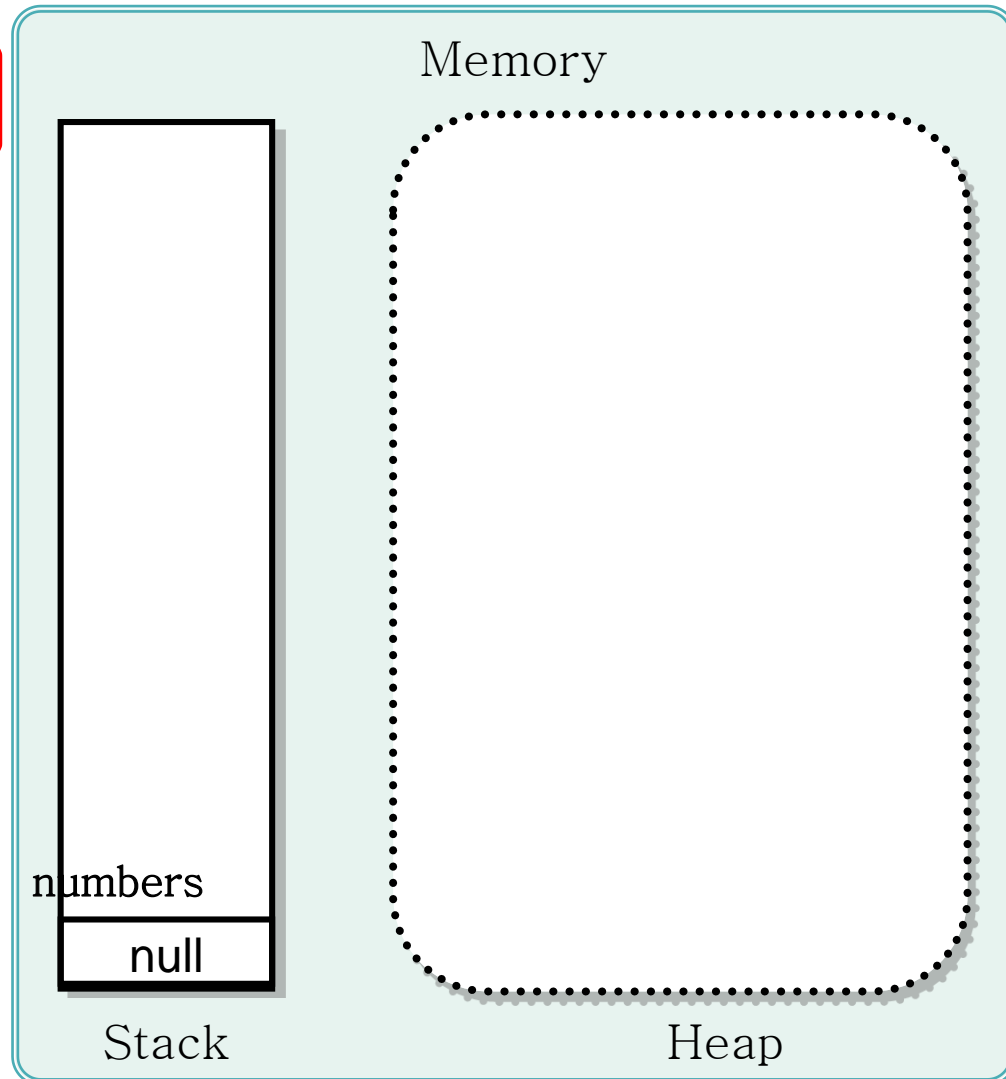
`int a[], b; // a는 Array에 대한 Reference, b는 int 형`

□ 선언

```
int[ ] numbers = null;
```

```
numbers = new int[10];
```

```
for ( int inx = 0 ; inx < numbers.length ; inx ++ ) {  
    numbers[inx] = inx + 1;  
    System.out.println( numbers[inx] );  
}
```

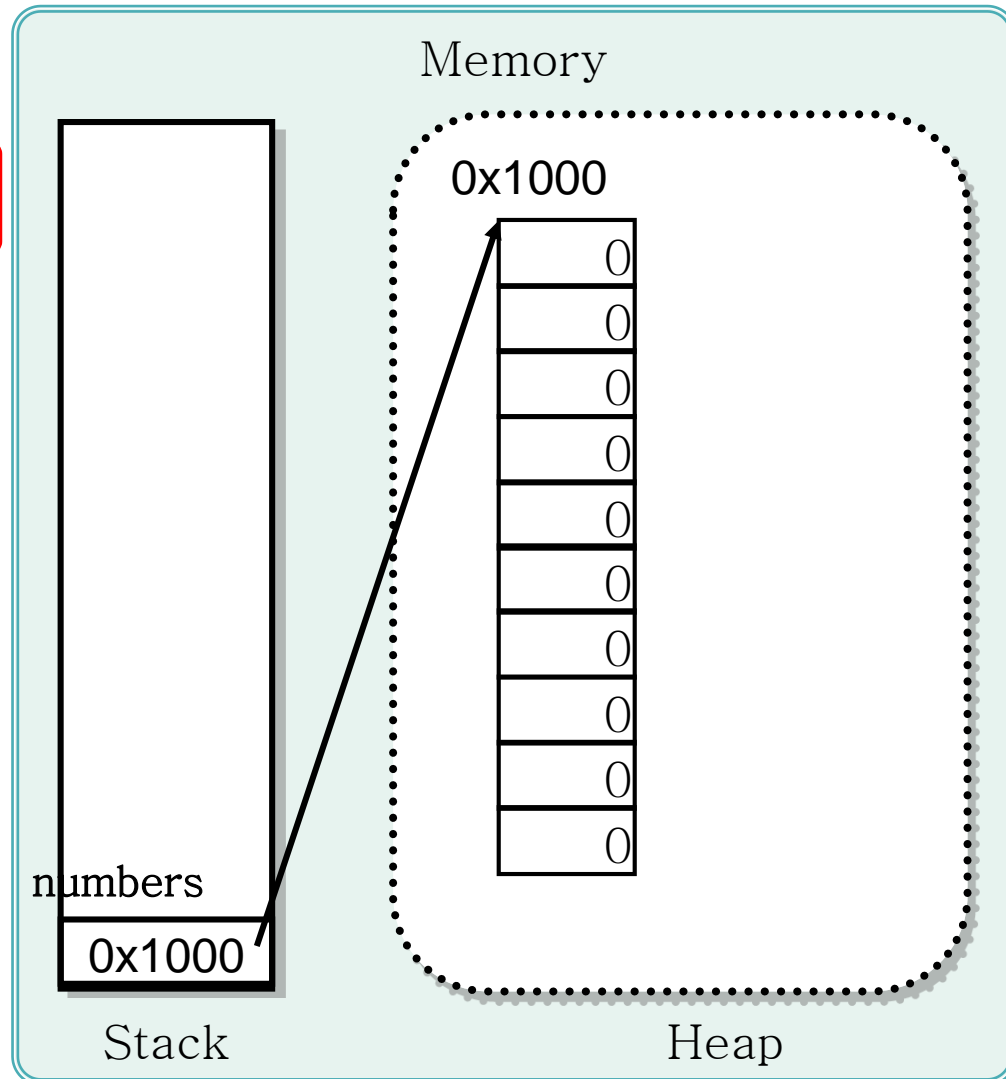


□ 생성

```
int[ ] numbers = null;
```

```
numbers = new int[10];
```

```
for ( int inx = 0 ; inx < numbers.length ; inx ++ ) {  
    numbers[inx] = inx + 1;  
    System.out.println( numbers[inx] );  
}
```

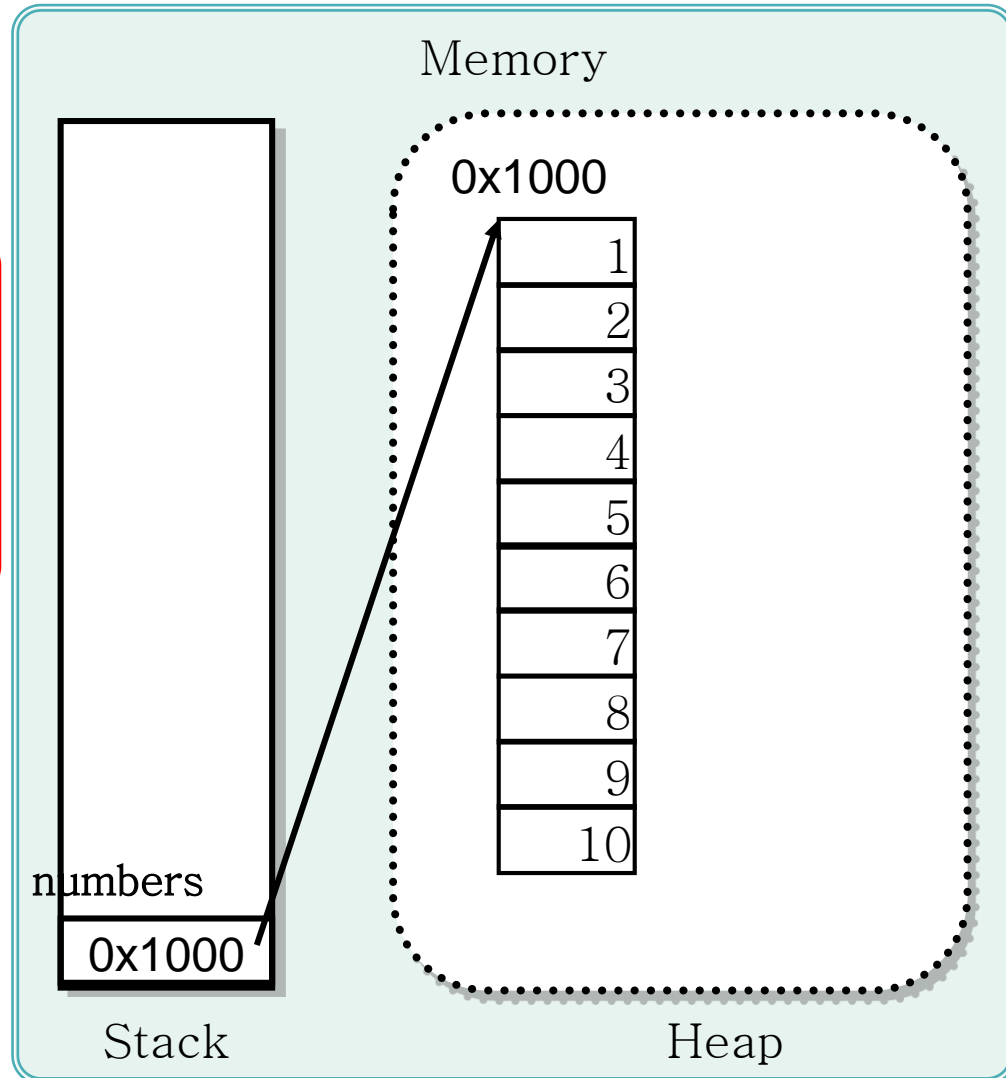


□ 초기화

```
int[ ] numbers = null;
```

```
numbers = new int[10];
```

```
for ( int inx = 0 ; inx < numbers.length ; inx ++ ) {  
    numbers[inx] = inx + 1;  
    System.out.println( numbers[inx] );  
}
```



□ 선언

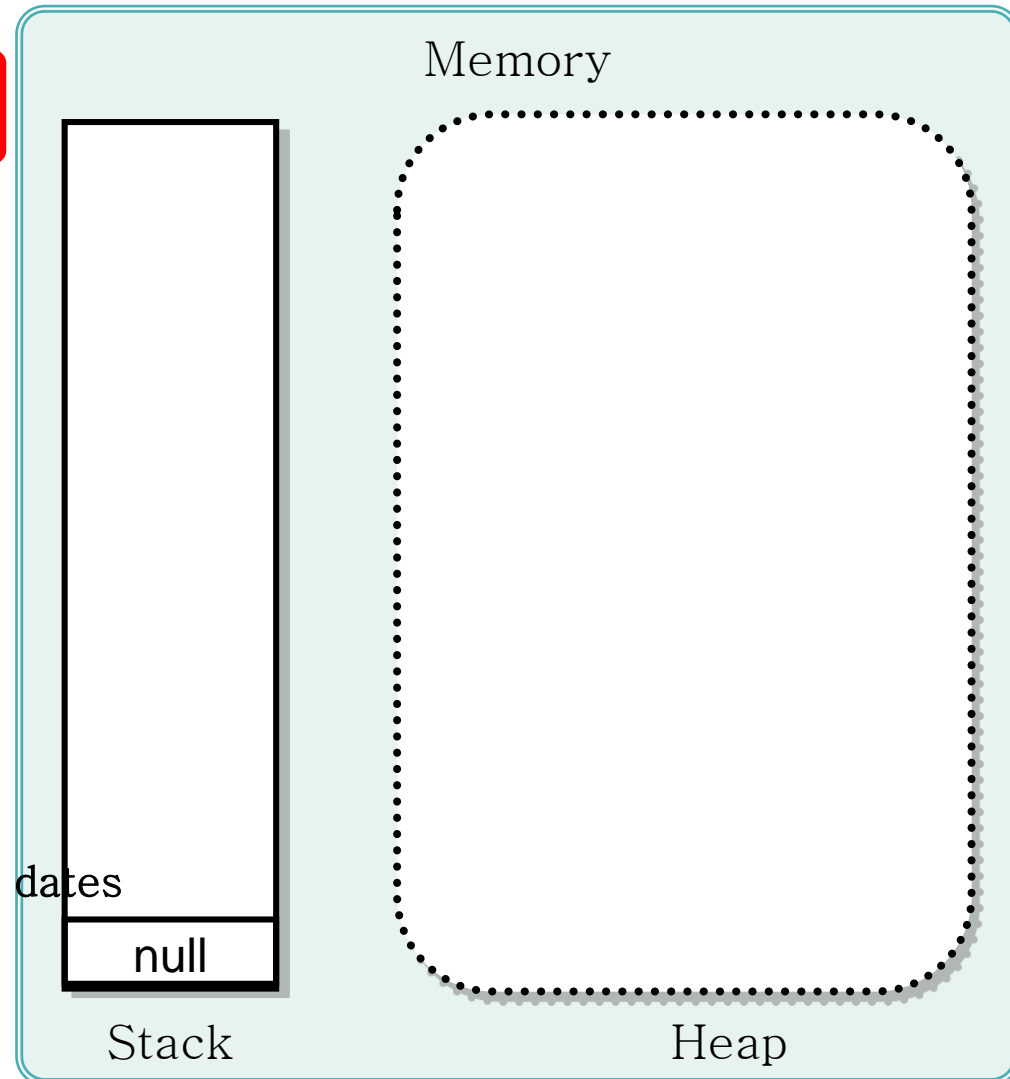
```
Date[ ] dates = null;
```

```
dates = new Date[10];
```

```
dates[0] = new Date( 22, 7, 1964 );
```

```
dates[1] = new Date( 12, 7, 2006 );
```

```
System.out.println( dates[0].getDay() );
```



□ 생성

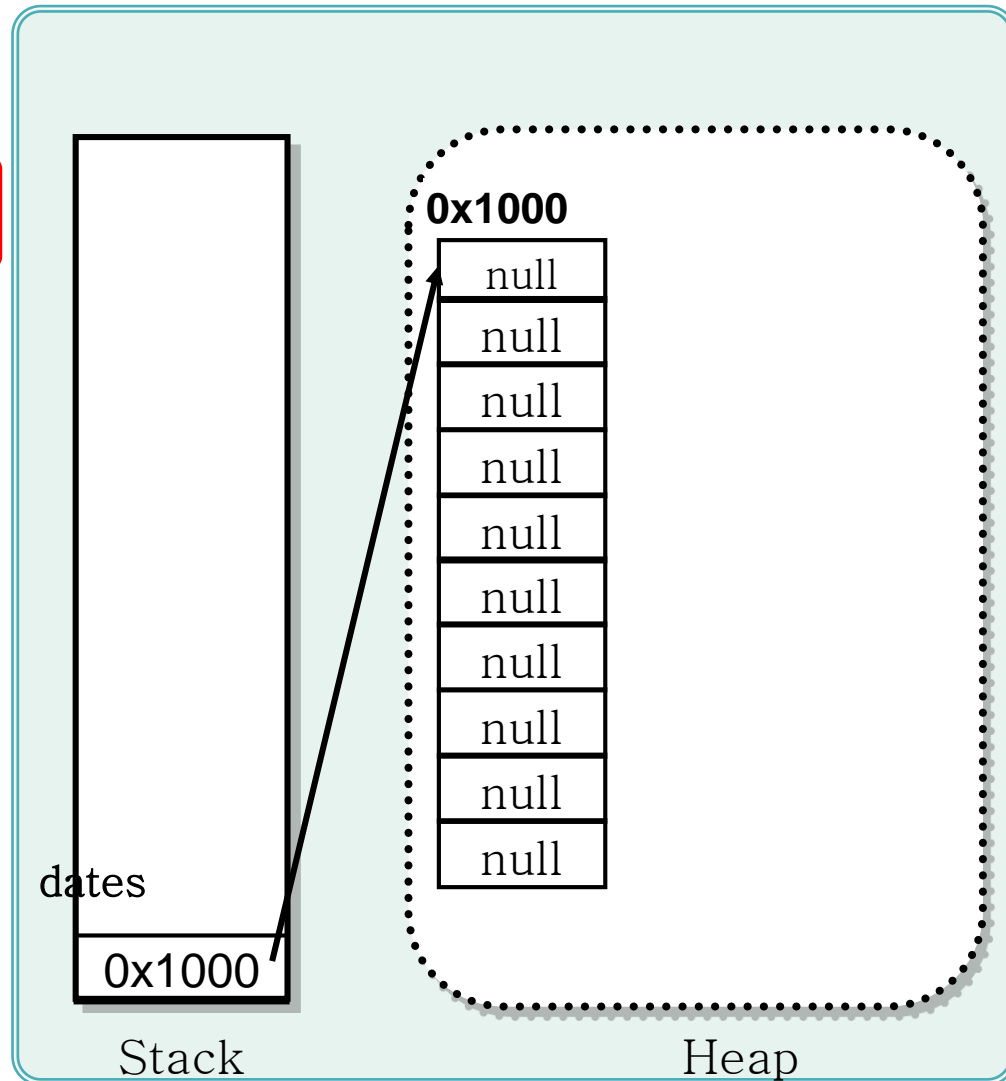
```
Date[ ] dates = null;
```

```
dates = new Date[10];
```

```
dates[0] = new Date( 22, 7, 1964 );
```

```
dates[1] = new Date( 12, 7, 2006 );
```

```
System.out.println( dates[0].getDay() );
```



□ 초기화

```
Date[ ] dates = null;
```

```
dates = new Date[10];
```

```
dates[0] = new Date( 22, 7, 1964 );
```

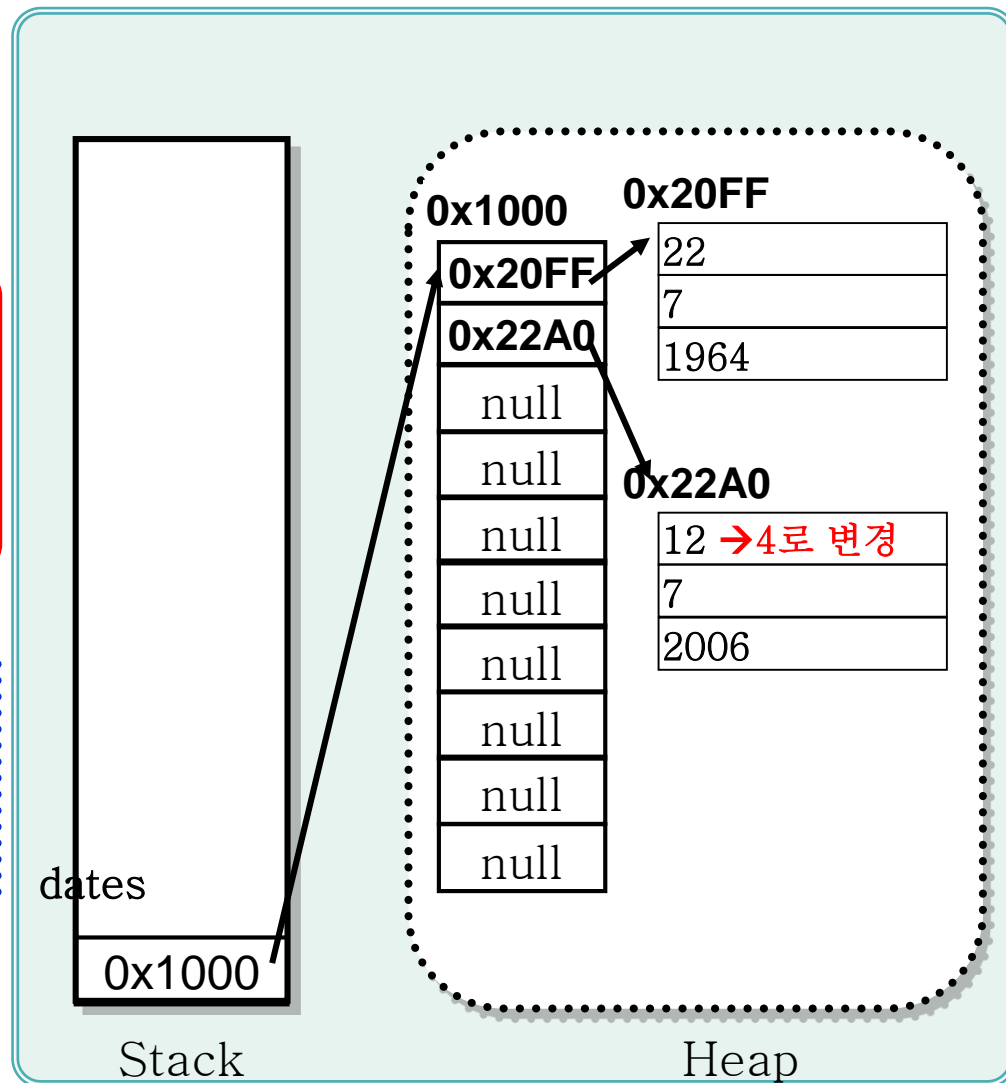
```
dates[1] = new Date( 12, 7, 2006 );
```

```
System.out.println( dates[0].getDay() );
```

```
// access
```

```
dates[1].setDay( 4 );
```


```
System.out.println( dates[1].getDay() );
```



□ 생성 후, 초기화

```
int[] numbers = new int[3];  
numbers[0] = 10;  
numbers[1] = 20;  
numbers[2] = 30;
```

```
MyDate[] date = new MyDate[3];  
date[0] = new MyDate(10, 08, 2005);  
date[1] = new MyDate(11, 08, 2005);  
date[2] = new MyDate(12, 08, 2005);
```



동 일

□ 생성과 동시에 초기화

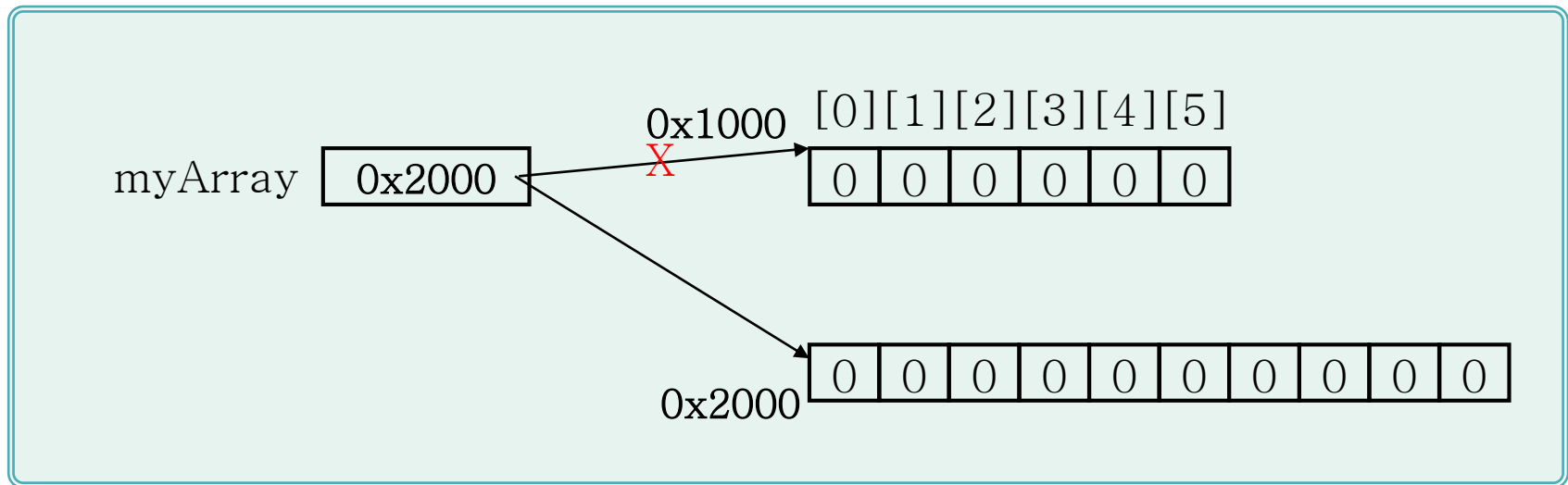
```
int[] numbers = { 1, 2, 3 };
```

```
MyDate[] date =  
    { new MyDate(10, 08, 2005),  
      new MyDate(11, 08, 2005),  
      new MyDate(12, 08, 2005)  
    };
```

- 배열은 크기를 바꿀 수 없다.

```
int[] myArray = new int[6];
```

```
myArray = new int[10]; //size가 늘어나는 것이 아니라, 새로운 배열  
이 생성
```



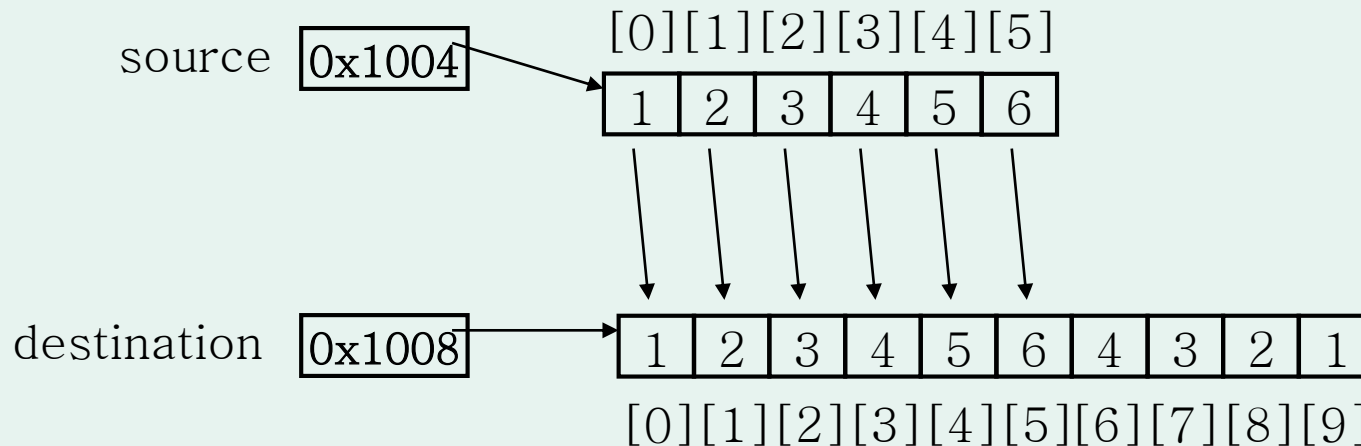
```
//original array
```

```
int[ ] source = { 1, 2, 3, 4, 5, 6 };
```

```
//new larger array
```

```
int[ ] destination = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
```

```
System.arraycopy( source, 0, destination, 0, source.length );
```



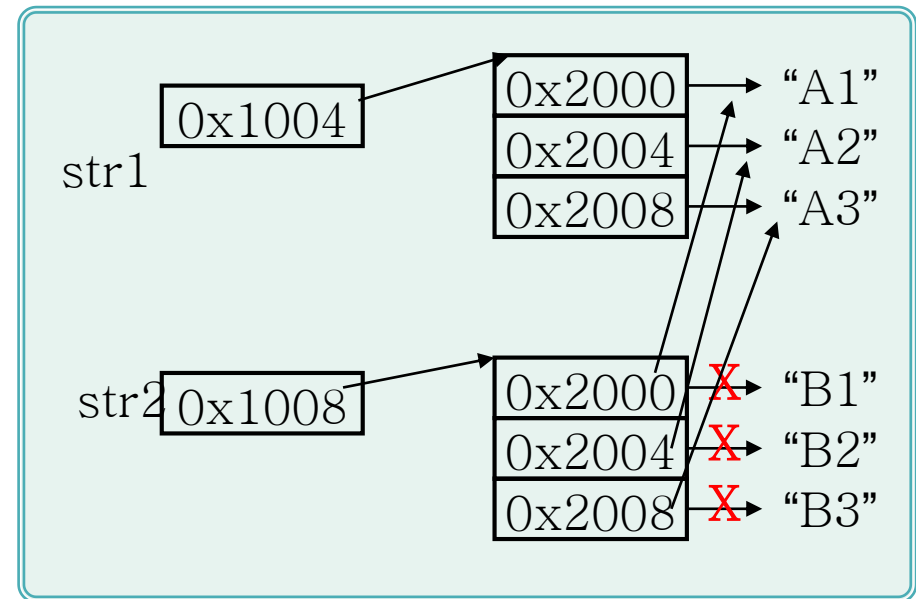
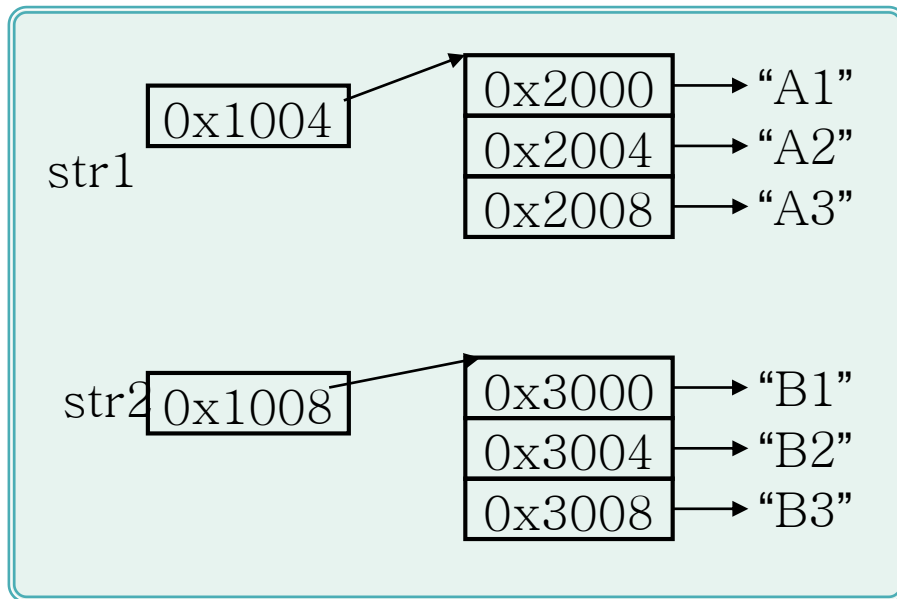
```
//original array
```

```
String[ ] str1 = { "A1", "A2", "A3" };
```

```
//new larger array
```

```
String[ ] str2 = { "B1", "B2", "B3" };
```

```
System.arraycopy( str1, 0, str2, 0, str1.length );
```



Chap 06. Class Design – Part 1

학습목표

1. Inheritance
2. Overriding / Overloading
3. super 키워드

OOP (Object-Oriented Programming) 특징

Encapsulation

Inheritance

Polymorphism

□ 현재 상황

Employee
+name : String = "" +salary : double +birthDate : MyDate
+getDetails() : String

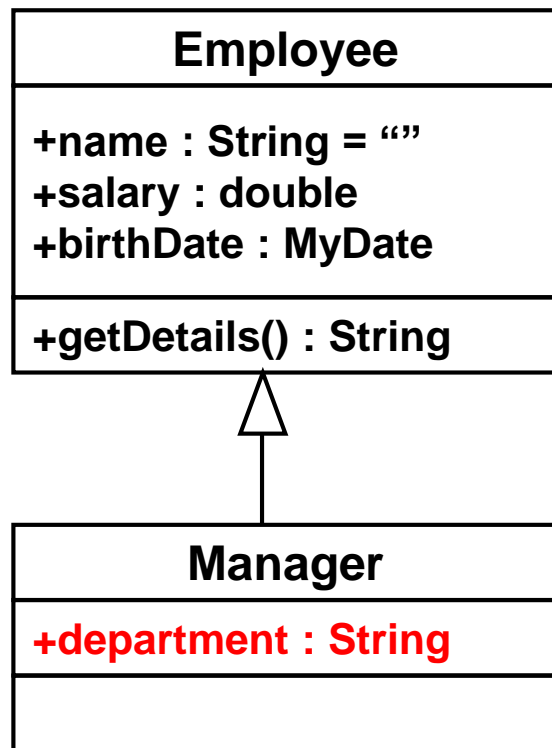
```
public class Employee {  
    public String name;  
    public double salary;  
    public MyDate birthDate;  
  
    public String getDetails() {  
        return "Name: " + name + "\n" +  
            "Salary: " + salary;  
    }  
}
```


□ 새로운 요구

Manager
+name : String = "" +salary : double +birthDate : MyDate +department : String
+getDetails() : String

```
public class Manager {  
    public String name;  
    public double salary;  
    public MyDate birthDate;  
    public String department;  
  
    public String getDetails() {  
        return "Name: " + name + "\n" +  
            "Salary: " + salary + "\n" +  
            "Manager of: " + department;  
    }  
}
```

□ 해결 : 기존 클래스 이용



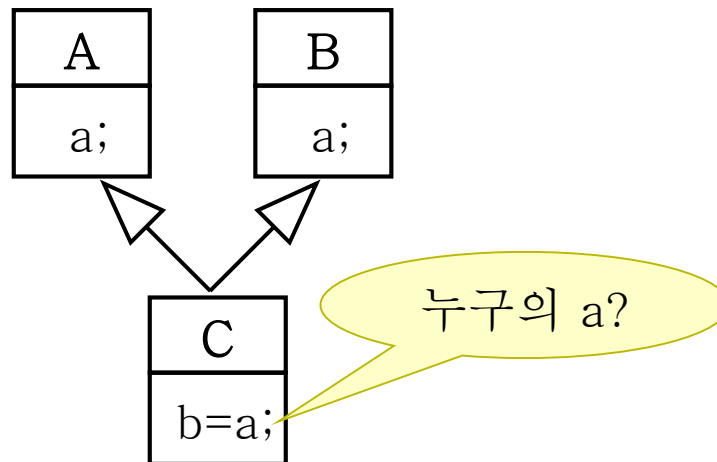
```
public class Employee {
    public String name;
    public double salary;
    public MyDate birthDate;

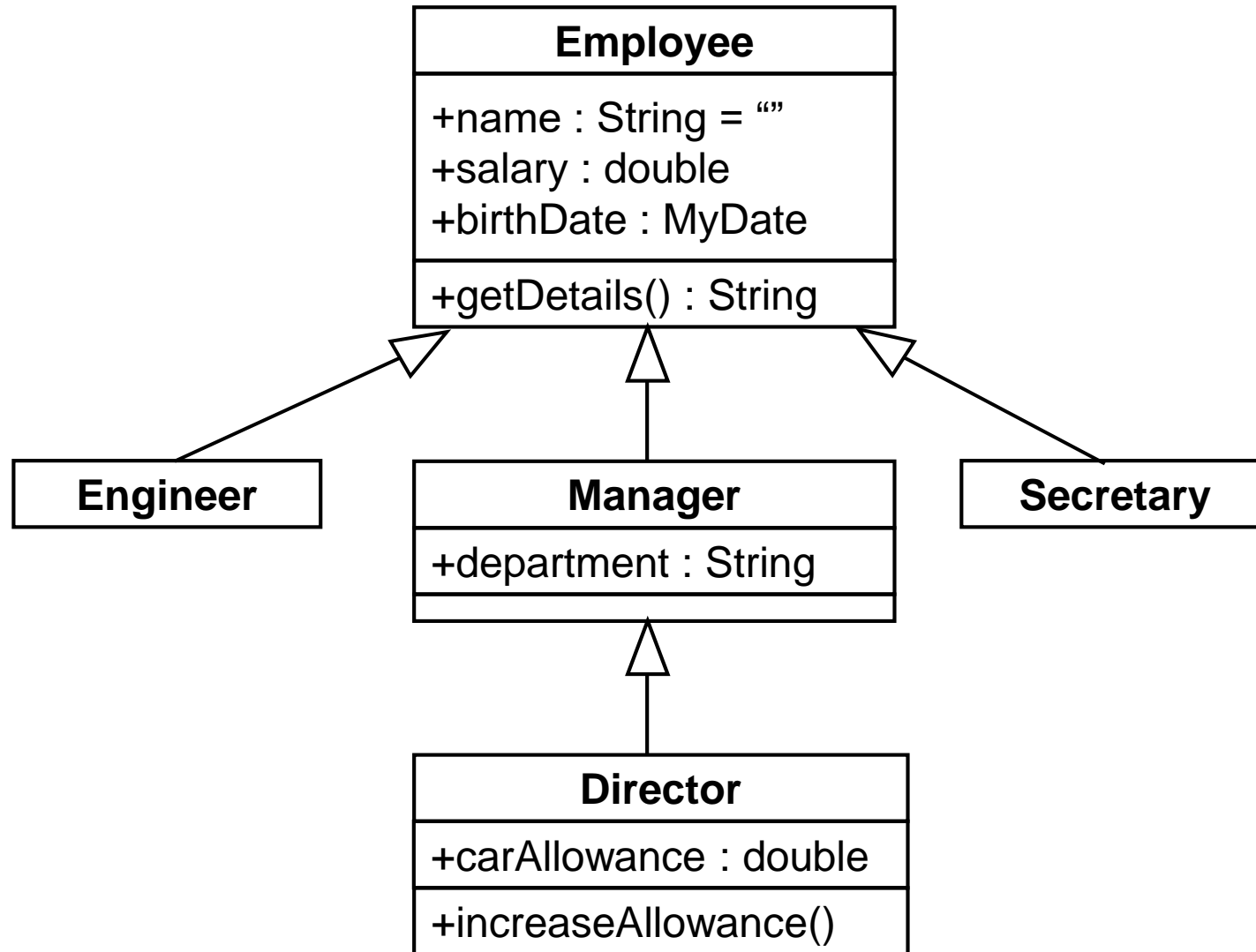
    public String getDetails() {
        return "Name: " + name + "\n" +
            "Salary: " + salary;
    }
}
```

```
public class Manager extends Employee {
    public String department;
}
```

□ Inheritance (상속)

- 문법 : `class 자식클래스 extends 부모클래스{ ... }`
- 자바에서의 상속 : 단일 상속만 지원
즉, 자식은 하나의 부모만 가질 수 있다.
- 단일 상속의 제한점 극복 : interface 이용
- 다중 상속을 허용하지 않는 이유: 코드의 모호성 배제

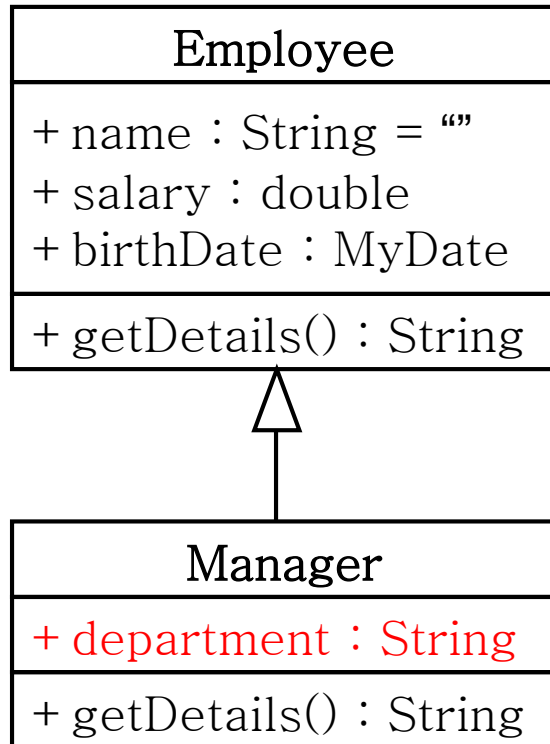




멤버 변수/ 멤버 메소드	private	클래스 내부에서만 참조 가능
	<i>default</i>	같은 디렉토리(같은 package)내의 다른 클래스에서 참조 가능
	protected	default 참조 범위 및 상속 관계에 있을 때 Package관계없이 참조가능
	public	어디서나 참조 가능

클래스는 public, *default* 만 가능

- ❑ 자식 클래스에서 부모 메소드의 기능을 자신의 기능에 맞게 메소드 Body를 새롭게 정의 하는 것.



```
public class Employee {
    public String name;
    public double salary;
    public MyDate birthDate;

    public String getDetails() {
        return "Name: " + name + "\n" +
            "Salary: " + salary;
    }
}
```

```
public class Manager extends Employee {
    public String department;

    public String getDetails() {
        return "Name: " + name + "\n" +
            "Salary: " + salary +
            "\nDepartment: " + department;
    }
}
```

- ❑ 부모 객체를 가리킨다.
- ❑ 용도 - 부모의 멤버변수 참조 : `super.멤버변수_이름`
 - 부모의 멤버메소드 참조 : `super.멤버메소드_이름()`
 - 부모의 생성자 호출 : `super(파라미터_리스트)`

```
public class Employee {  
    private String name;  
    private double salary;  
    private MyDate birthDate;  
  
    public String getDetails() {  
        return "Name: " + name + "\n" +  
            "Salary: " + salary;  
    }  
}
```

```
public class Manager extends Employee {  
    private String department;  
  
    public Manager() {  
    }  
  
    public String getDetails() {  
        return super.getDetails() +  
            "\nDepartment: " + department;  
    }  
}
```

□ 상속의 장점

- 비슷한 유형의 코드 재사용
- 검증된 코드를 사용 → 오류의 최소화
- 관련된 여러 클래스들의 공통점 통일화

PrintStream Class의 println() 메소드

```
public void println( char c ) { }  
public void println( boolean b ) { }  
public void println( int i ) { }  
public void println( long l ) { }  
public void println( double d ) { }  
public void println( float f ) { }  
public void println( String s ) { }  
public void println( Object obj ) { }
```

...

```
System.out.println( 10 );
```

```
System.out.println( "Java Programming" );
```

- Overloading 의 장점 : 같은 기능에 대해, 같은 메소드 이름을 사용하는 것이 가능하다.
(일관성 유지)

- ❑ 메소드 뿐만 아니라, 생성자도 Overloading 이 가능하다.
- ❑ 생성자에서 자신의 다른 생성자 호출 : this() 이용

```
public Employee() {  
}  
public Employee( String name, double salary ) {  
    this( name, salary, null );  
}  
public Employee( String name, double salary, MyDate birthDate ) {  
    this.name = name;  
    this.salary = salary;  
    this.birthDate = birthDate;  
}
```

- ❑ 생성자는 상속이 안 된다.
- ❑ 자식 클래스에서 부모클래스의 멤버변수를 초기화 : super() 이용

```
public Manager() {  
}  
public Manager( String name, double salary ) {  
    this( name, salary, null, null );  
}  
public Manager( String name, double salary, MyDate birthDate, String department ) {  
    super( name, salary, birthDate );  
    this.department = department;  
}
```

- ❑ **주의** : this 나 super 키워드가 생성자를 호출하는데 쓰일 경우, 반드시 생성자의 첫 line에 기술되어야 한다. 또한, super(), this() 를 같이 호출하지 못 한다.

```
public Manager() {  
}  
  
public Manager( String name, double salary ) {  
    super( name, salary ); //error  
    this( name, salary, null, null );  
}  
  
public Manager( String name, double salary, MyDate birthDate, String department ) {  
    this.department = department; // error !!!  
    super( name, salary, birthDate );  
}
```

- ❑ **주의** : super, this 를 사용하여 생성자를 호출하지 않는 경우, 묵시적으로 super(); 이 호출 된다. 부모 클래스에 default 생성자가 없으면 컴파일 에러
→ 클래스 작성시, 항상 default 생성자를 기술하자!!!

```
public Manager() {  
    super(); //묵시적으로 자동 호출  
}  
  
public Manager( String name, double salary ) {  
    this( name, salary, null, null );  
}  
  
public Manager( String name, double salary, MyDate birthDate, String department ) {  
    super( name, salary, birthDate );  
    this.department = department;  
}
```

Overriding vs Overloading

Overriding(재정의)	Overloading(다중정의)
<ul style="list-style-type: none">• 메소드를 하위 클래스에서 정의	<ul style="list-style-type: none">• 메소드를 같은 클래스에서 정의
<ul style="list-style-type: none">• 메소드 이름 동일• Parameter(개수 및 데이터 타입) 동일• Return 타입 동일	<ul style="list-style-type: none">• 메소드 이름 동일• Parameter(개수나 데이터 타입) 다름• Return 타입 다를 수 있음
<ul style="list-style-type: none">• 접근 제한자 : 하위 메소드의 접근 범위가 상위 메소드의 접근 범위 보다 넓거나 같아야 한다.	<ul style="list-style-type: none">• 접근 제한자 : 관계 없음
<ul style="list-style-type: none">• 예외 처리 : 예외 발생시 같은 예외 형식이거나, 더 구체적인 예외 형식이어야 한다.	<ul style="list-style-type: none">• 예외 처리 : 관계없음

Method Overriding 규칙 – 접근 제한자

- 하위 메소드의 접근 범위가 상위 메소드의 접근 범위 보다 넓거나 같아야 한다.

```
public class Example {  
    public static void main(String[] args) {  
        SuperClass superClass = new SubClass();  
  
        //Runtime시 타입이 private임. 호출 되어질 수 없음.  
        superClass.method();  
    }  
}  
class SuperClass {  
    public void method() {  
    }  
}  
class SubClass extends SuperClass {  
    private void method() { //컴파일 오류가 안난다고 가정  
    }  
}
```

Chap 06. Class Design – Part 2

1. Polymorphism

- 개념
- Virtual Method Invocation
- Why Polymorphism
- instanceof keyword

2. Object Class

3. Wrapper Class

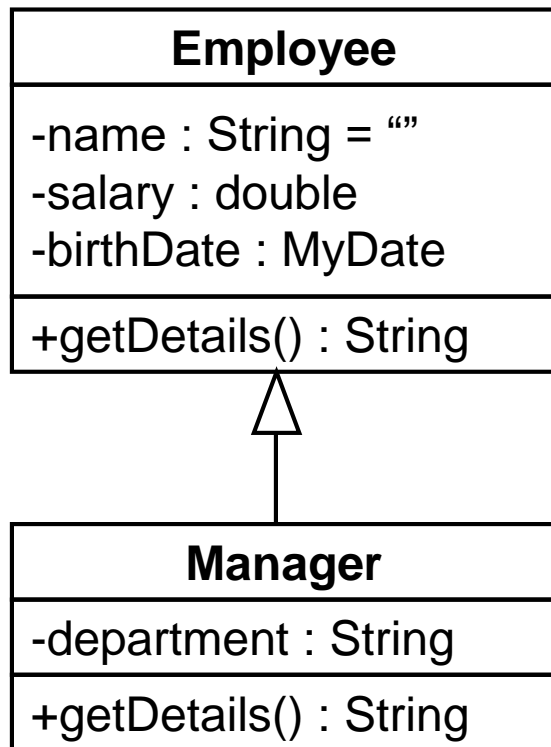
OOP (Object-Oriented Programming) 특징

Encapsulation

Inheritance

Polymorphism

- ❑ Polymorphism(다형성) : 다양한 형태를 가짐을 말한다.
- ❑ 자바에서는 한 reference변수가 다른 형태의 객체를 참조 할 수 있음을 말한다.
(부모 reference 변수 → 자식 객체)
- ❑ 즉, reference 변수를 polymorphic 하다고 할 수 있다.



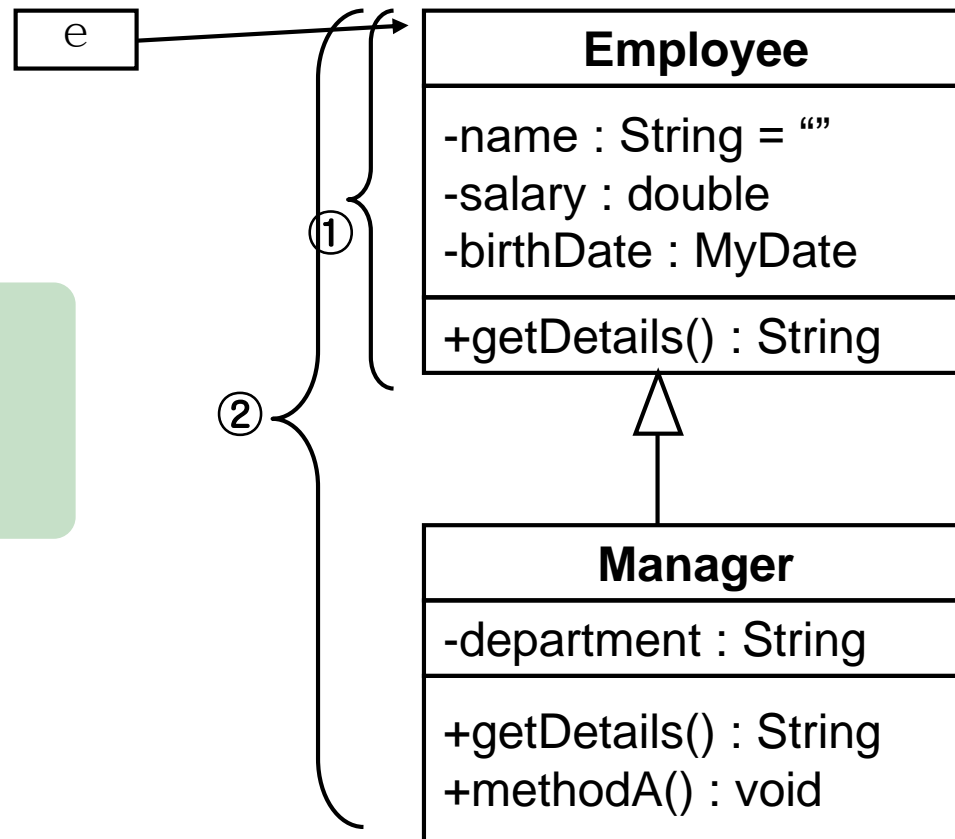
```
Employee e = new Manager(); //OK
Manager m = new Employee(); //error
```

```
Employee e = new Manager();  
e.methodA();    // 컴파일 결과는 ?  
e.getDetails(); // Employee 의 메소드? Manager 의 메소드?
```

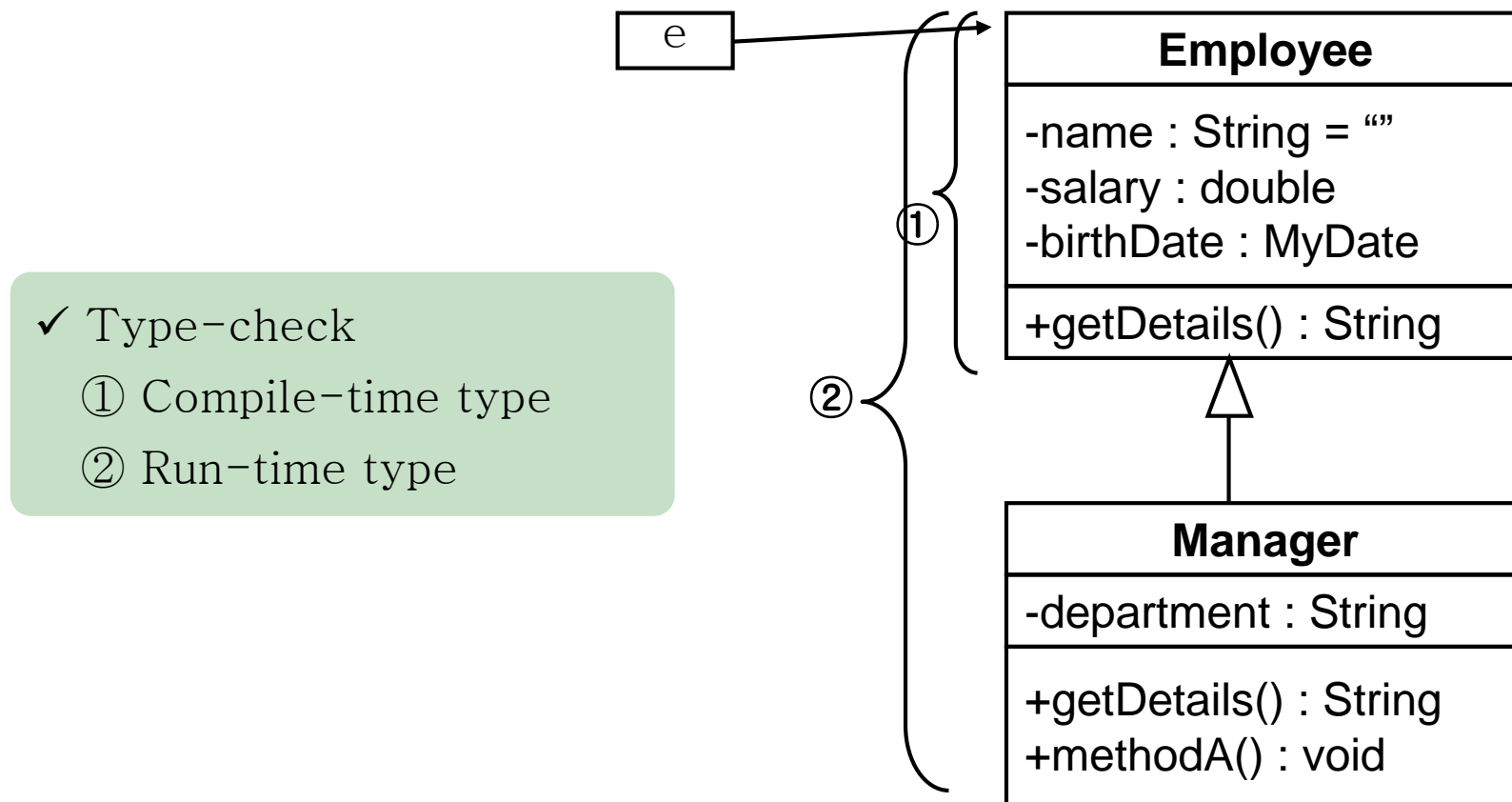
✓ Type-check

① Compile-time type

② Run-time type



- ❑ Compile 시에는 Reference type 만을 가지고, compile 여부 결정
- ❑ 실행 시에는 실제 new를 통해서 생성 되는 객체의 메소드 실행



❑ Homogenous collection

동일한 Class의 객체로 이루어진 집합

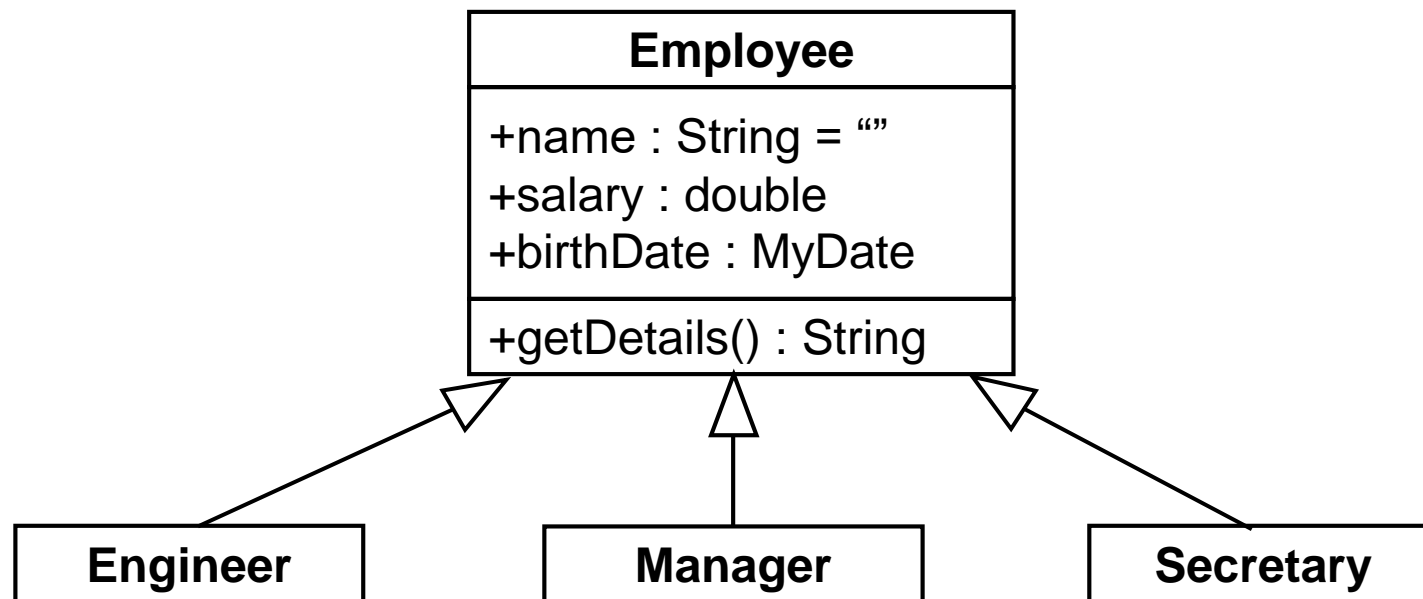
```
MyDate[] dates = new MyDate[2];  
dates[0] = new MyDate( 22, 12, 1964 );  
dates[1] = new MyDate( 22, 7, 1964);
```

❑ Heterogeneous Collection

다른 Class의 객체로 이루어진 집합

```
Employee[] staff = new Employee[1024];  
staff[0] = new Manager();  
staff[1] = new Employee();  
staff[2] = new Engineer();
```

- 사원에 대한 세금을 계산필요 → 메소드로 구현
직군(관리자, 엔지니어, 비서)에 따라 세금이 달라진다.



- ✓ 방법 1 : 메소드 오버로딩을 이용하여 각각의 직군에 따른 세금계산 메소드 작성

```
Employee[] staff = new Employee[1024];
```

```
public int getTax( Employee e ) {  
    return 100;  
}
```

```
public int getTax( Manager m ) {  
    return 300;  
}
```

```
public int getTax( Engineer e ) {  
    return 200;  
}
```


- ✓ 방법2 : Polymorphism 이용 (Polymorphic Arguments)

```
Employee[] staff = new Employee[1024];
```

```
public double getTax( Employee e ) {  
    }  
}
```

- 문제점 : 부모 reference로 입력 받은 객체가 실제로 어떠한 객체인지
판별 필요

- ✓ 방법2 : Polymorphism 이용 (Polymorphic Arguments)

```
Employee[] staff = new Employee[1024];
```

```
public double getTax( Employee e ) {  
    if ( e instanceof Manager )  
        ....  
}
```

❑ A instanceof B

- A가 B의 자식이거나 같은 class 타입이면 true
- A가 B의 부모이면 false
- 부모/자식 관계에 속하지 않는 것끼리 instanceof 하면 컴파일 에러

✓ 주의

```
public double getTax( Employee e ) {  
    if ( e instanceof Employee )  
        ....  
}
```

❑ Casting Operator

```
public double getTax( Employee e ) {  
    double tax = 0.0;  
    if ( e instanceof Manager ) {  
        Manager m = (Manager)e;  
        tax = m.getSalary() * 0.30;  
    } else if ( e instanceof Engineer ) {  
        Engineer eng = (Engineer)e;  
        tax = eng.getSalary() * 0.20;  
    } else {  
        tax = e.getSalary() * 0.10;  
    }  
  
    return tax;  
}
```

- ❑ Upward Casting : Type Casting 불필요

```
Employee e = new Manager();
```

- ❑ Downward Casting : Compile Ok, Runtime Exception 발생

```
Employee e = new Employee();  
Manager m = (Manager)e;
```

- ❑ 이렇게 하자...

```
Employee e = new Manager();  
Manager m = (Manager)e;
```

- ▶ 모든 클래스의 부모클래스이다.
- ▶ extends 키워드가 안 쓰인 클래스는 extends Object를 한 것이다.

□ Object 클래스의 메소드들

1) equals

- ref.변수를 '==' 연산자로 비교한다는 것은, 주소 값을 비교하는 것이다.
- equals() 의 내용은 비교되는 두 reference의 값을 '==' 연산자로 비교한다.
- 주소비교가 아닌 객체 내용을 비교하려는 경우에는 equals() 를 override 한다.

2) toString()

- 객체를 String 으로 변환한다.
- String Concatenation시 사용된다.
- toString() 내용은 '클래스이름@hashCode값' 이다.
- 필요하면 적절하게 override하여 사용한다.
- Primitive 타입인 경우 String으로 변환하기 위해, Wrapper 클래스의 toString()을 이용한다.

Object 클래스

```
public boolean equals(Object obj) {  
    return (this == obj);  
}
```

String 클래스

```
public boolean equals(Object anObject) {  
    if (this == anObject) {  
        return true;  
    }  
    if (anObject instanceof String) {  
        String anotherString = (String)anObject;  
        int n = count;  
        if (n == anotherString.count) {  
            char v1[] = value;  
            char v2[] = anotherString.value;  
            int i = offset;  
            int j = anotherString.offset;  
            while (n-- != 0) {  
                if (v1[i++] != v2[j++])  
                    return false;  
            }  
            return true;  
        }  
    }  
    return false;  
}
```

Object 클래스

```
public String toString() {  
    return getClass().getName() +  
        "@" + Integer.toHexString(hashCode());  
}
```

String 클래스

```
public String toString() {  
    return this;  
}
```


❑ Primitive Data Type을 객체화 해 주는 클래스이다.

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

❑ Primitive type을 객체화 하기

```
int pInt = 500;  
Integer wInt = new Integer( pInt );  
int p2 = wInt.intValue();
```

```
int pInt = 500;  
Integer wInt = pInt;    //Boxing  
int p2 = wInt;          //Unboxing
```

❑ String 값을 숫자로 바꾸기

```
1) int x = Integer.valueOf( "2" ).intValue();  
2) int x = Integer.parseInt( "2" );
```

Chap 07. Advanced Class Features

1. static keyword
2. final keyword
3. abstract class
4. interface

- ❑ Member Variable, Member Method 앞에 사용
- ❑ 일반적으로 멤버변수나 멤버메소드는 한 객체에서 의미가 있음
- ❑ static 키워드를 사용하면, 한 객체가 아닌 클래스 자체와 연관
- ❑ Class 변수, Class 메소드라고 불리운다.
- ❑ Class 변수, Class 메소드는 객체 생성 없이 사용한다.

- 객체생성 없이 호출될 수 있다.
- 클래스이름.메소드명() 으로 호출
- static 메소드 안에서는 this, super, non-static 멤버들은 사용할 수 없다.
(local 변수는 가능)

```
public class Count2 {  
    private int serialNumber;  
    private static int counter = 0;  
  
    public static int getTotalCount() {  
        return counter;  
    }  
  
    public Count2()  
    {  
        counter++;  
        serialNumber = counter;  
    }  
}
```

```
public class TestCount2 {  
    public static void main( String [] args ) {  
        System.out.println( "Number of counter is "  
                               + Count2.getTotalCount() );  
  
        Count2 count1 = new Count2();  
        System.out.println( "Number of counter is "  
                               + Count2.getTotalCount() );  
  
        Count2 count2 = new Count2();  
        System.out.println( "Number of counter is "  
                               + Count2.getTotalCount() );  
    }  
}
```

- ❑ final class - 상속 못 하게 함
- ❑ final method - override 못 하게 함
- ❑ final variable - 상수

- ▶ final variable
 - 선언시 초기화 안 할 수도 있다. (blank final variable)
 - 단, 선언시 초기화 안 한 경우 모든 생성자에서 초기화 해주어야 한다.
 - 사용되기 전에 초기화가 되어 저야 한다. (local 변수와 동일)

- ❑ Radio, TV, MP3 세 개의 클래스를 살펴 본 결과, 공통 되는 멤버 변수와 멤버 메소드가 존재한다.
- ❑ 이 공통 되는 부분들을 상위 클래스로 작성 하면 어떨까?

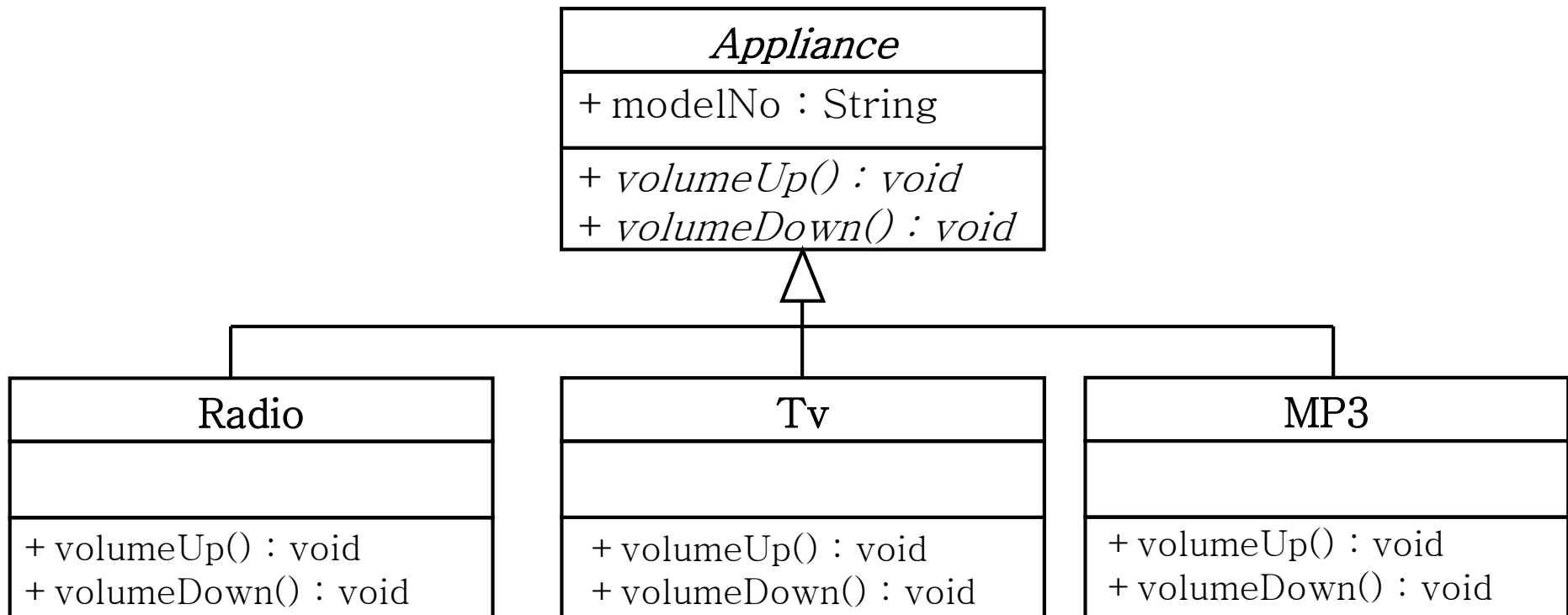
Super Class ?

Radio
+ modelNo : String
+ volumeUp() : void + volumeDown() : void

Tv
+ modelNo : String
+ volumeUp() : void + volumeDown() : void

MP3
+ modelNo : String
+ volumeUp() : void + volumeDown() : void

상속인 경우 문제점은?



```
abstract public class Appliance {  
    private String modelNo;  
  
    public Appliance() {  
    }  
  
    public Appliance( String modelNo ) {  
        this.modelNo = modelNo;  
    }  
  
    public String getModelNo() {  
        return modelNo;  
    }  
  
    public void setModelNo( String modelNo ) {  
        this.modelNo = modelNo;  
    }  
  
    abstract public void volumeUp();  
  
    abstract public void volumeDown();  
}
```

```
public class Radio extends Appliance {  
    public Radio( String modelNo ) {  
        super( modelNo );  
    }  
  
    public void volumeUp() {  
        System.out.println( "라디오 볼륨업" );  
    }  
  
    public void volumeDown() {  
        System.out.println( "라디오 볼륨다운" );  
    }  
}
```

□ Abstract Class

- 미완성 클래스 (abstract 키워드 사용)
- abstract 메소드가 포함 된 클래스 → 반드시 abstract 클래스
- 자체적으로 객체 생성 불가 → 반드시 상속 통하여 객체 생성
- 일반적인 메소드, 멤버변수도 포함할 수 있다.
- abstract 메소드가 없어도, abstract 클래스 선언 가능
- 객체 생성은 안되나, Ref 변수로서는 가능하다.
ex) `Appliance app = new Radio();`

□ Abstract Method 선언

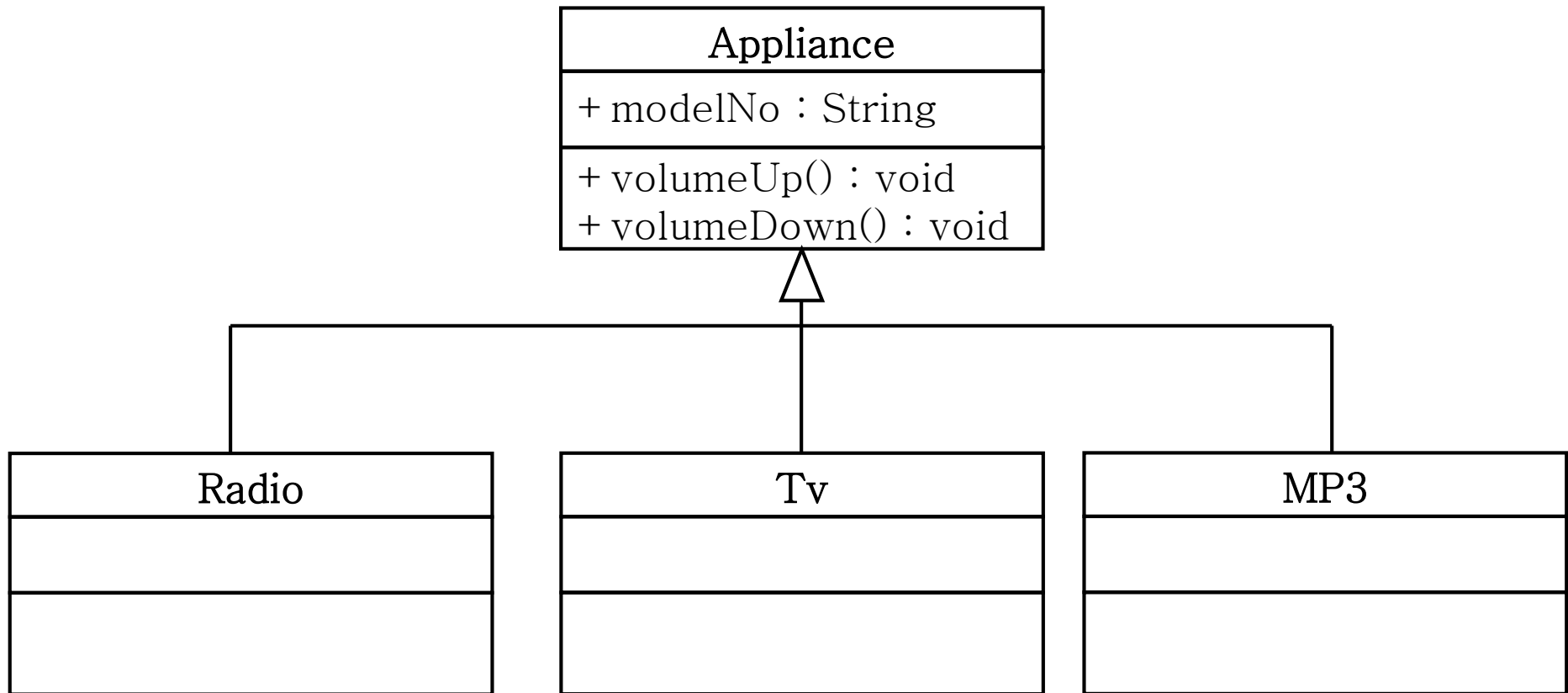
- { } (메소드 body)가 없는 메소드 , ‘;’ 으로 끝나야 한다.

□ Abstract Class 장점

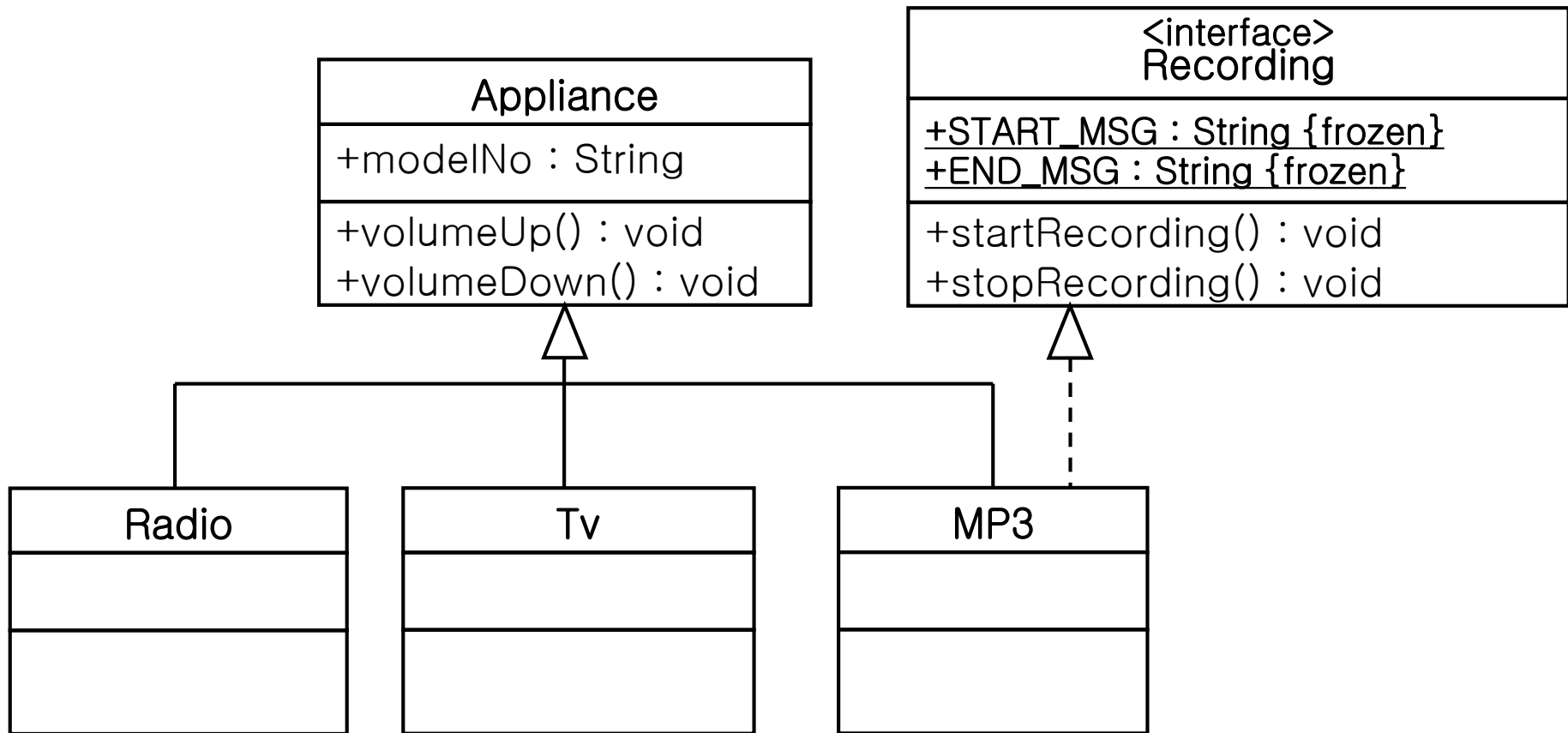
- 일관된 인터페이스 제공
- 꼭 필요한 기능 강제함 (공통적이긴 하나, 자식클래스에서 특수화 되는 기능)

□ 추가 요구 사항 도출

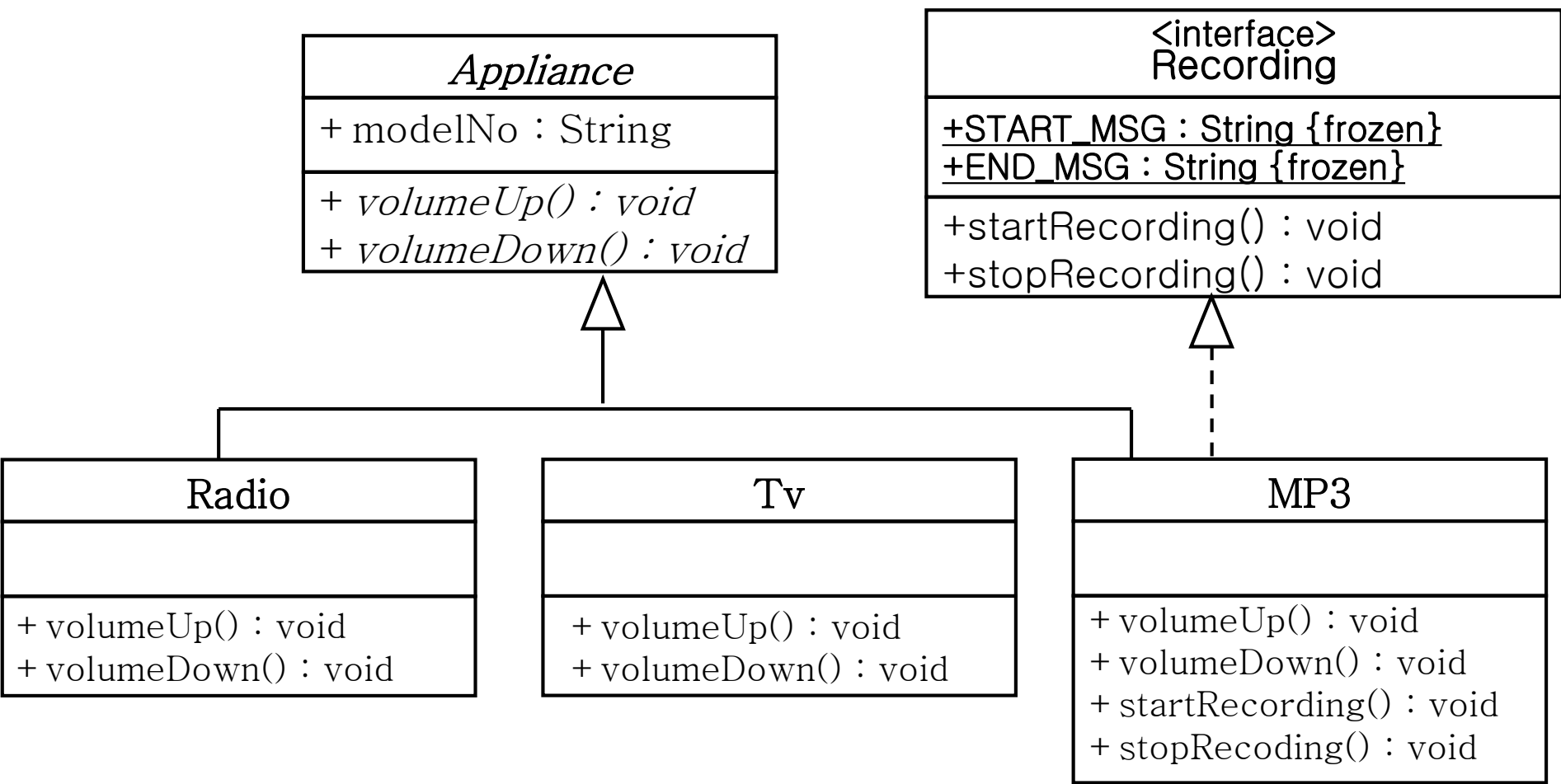
- 1) MP3 Player는 레코딩 기능이 있음
- 2) 추후 새로 개발되는 가전 제품에는 레코딩 기능 제공 예정



- 해결 : 레코딩에 관계 되는 메소드 형식만 모아 놓은 클래스 제공



□ 해결 : 레코딩에 관계 되는 메소드 형식만 모아 놓은 클래스 제공



```
public interface Recording {  
  
    public static final String START_MSG  
        = "Recording Started";  
    String END_MSG  
        = "Recording Ended";  
  
    public abstract void startRecord();  
    void stopRecord();  
}
```

```
public class MP3 extends Appliance  
    implements Recording {  
  
    public MP3( String modelNo ) {  
        super( modelNo );  
    }  
  
    public void volumeUp() {  
        System.out.println( "MP3 볼륨업" );  
    }  
  
    public void volumeDown() {  
        System.out.println( "MP3 볼륨다운" );  
    }  
  
    public void startRecord() {  
        System.out.println( START_MSG );  
    }  
  
    public void stopRecord() {  
        System.out.println( END_MSG );  
    }  
}
```


□ Interface

- 모든 메소드가 abstract 메소드인 클래스
- 메소드는 묵시적으로 public abstract
- 멤버변수는 묵시적으로 public static final
- implements 키워드로 구현
- 다중 구현 가능 → 단일 상속 극복
- 객체 생성은 안되나, Ref 변수로서는 가능하다.
ex) Recording app = new MP3();

□ Interface의 장점

- 공통 기능상의 일관성 제공
- 공동 작업을 위한 인터페이스 제공

Chap 08. Exceptions

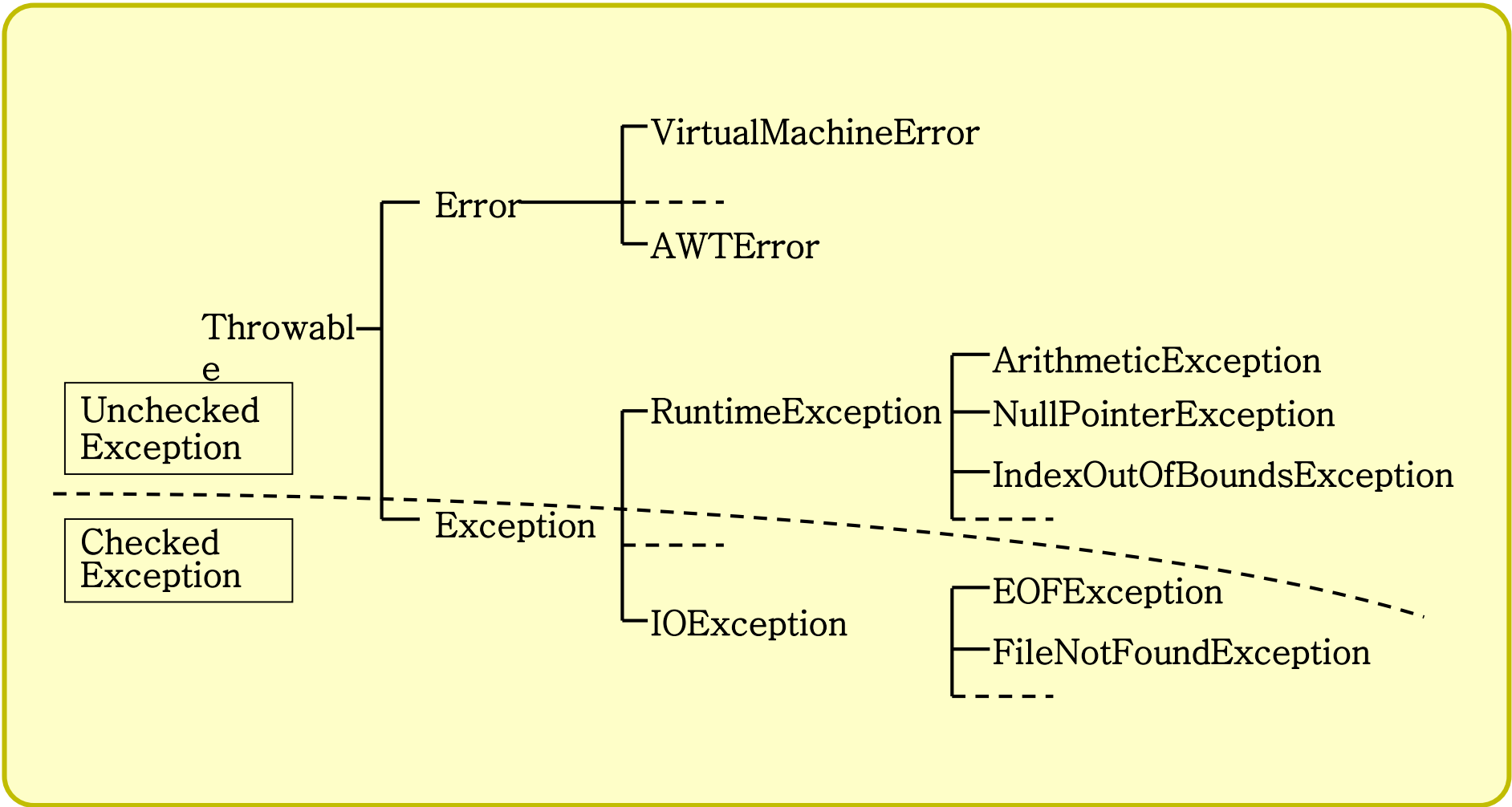
1. Exception Category
2. Exception 처리방법
3. Method overriding & Exception
4. 사용자 정의 Exception

□ Error

- 프로그램 수행시, 치명적 상황 발생
- 처리가 불가능
- 예) 메모리 부족 등

□ Exception

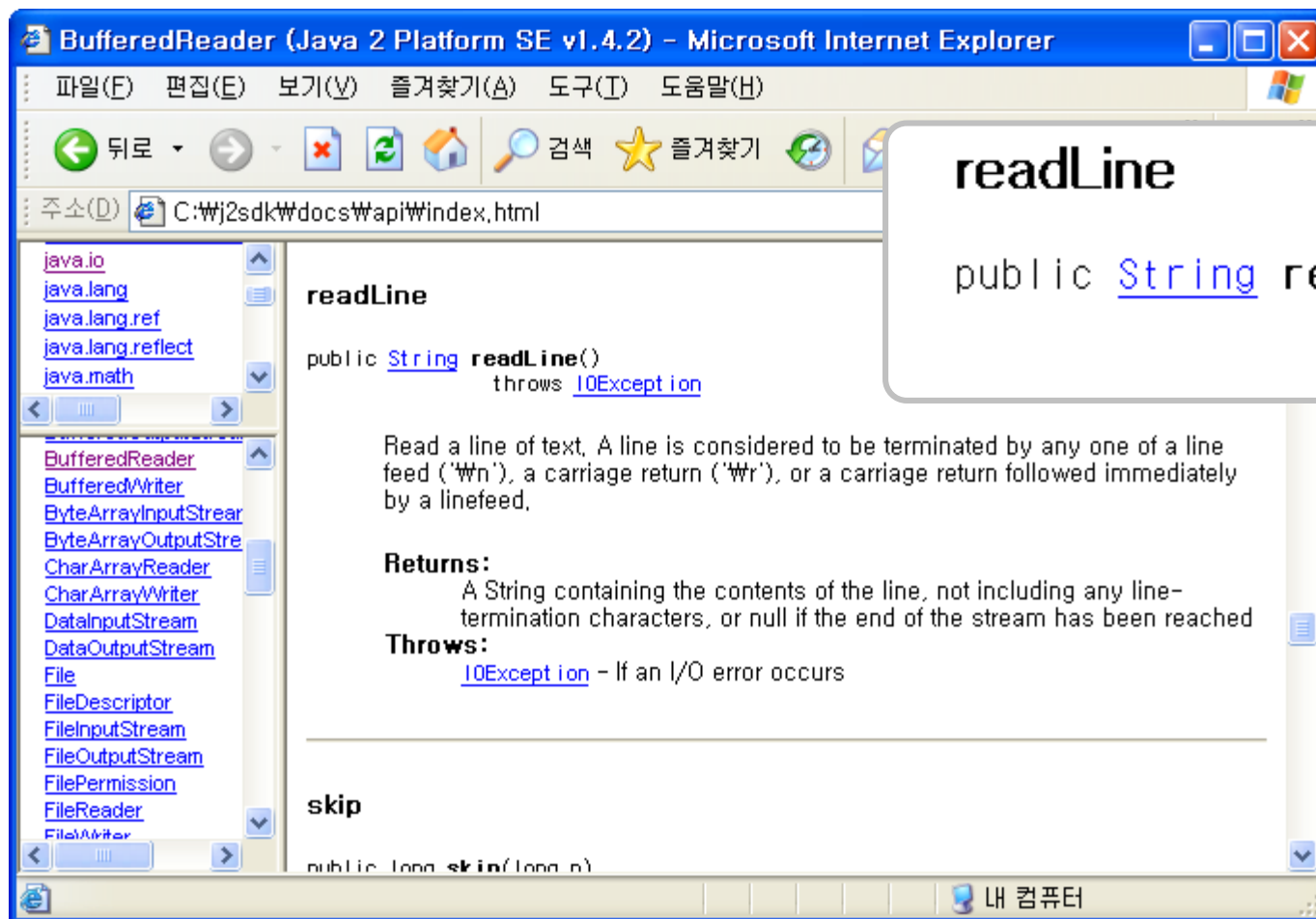
- 비교적 가벼운 에러
- 미리 예상하여 처리 가능
- 예) 파일 access시 파일 없음,
부적절한 parameter에 의한 메소드 호출 등



Exception 확인하기

API Document에서 해당 클래스에 대한 생성자나 메소드를 검색하면, 그 메소드가 어떠한 Exception을 발생시킬 가능성이 있는지 확인할 수 있다.

java.io.BufferedReader 의 readLine() 메소드 경우

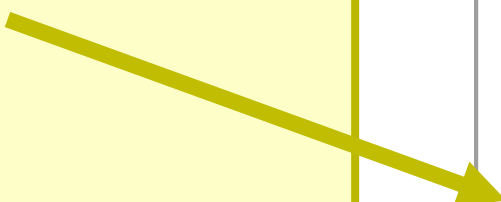


readLine

```
public String readLine()  
           throws IOException
```


- ❑ Unchecked Exception이며, 주로 프로그래머의 부주의로 인한 Bug인 경우가 많기 때문에 Exception 처리보다는 코드를 수정해야하는 경우가 많다.

- ArithmeticException
- NullPointerException
- NegativeArraySizeException
- ArrayIndexOutOfBoundsException
- ClassCastException



```
int sum = 0;
int total = 100;
int count = 0;

sum = total / count;
```



```
int sum = 0;
int total = 0;
int count = 0;

if ( count != 0 )
    sum = total / count;
```

- `ArithmeticException` : 0으로 나누는 경우 발생
→ if 문으로 먼저 나누는 수가 0인지를 검사.
- `NullPointerException` : null 인 ref.변수로 객체 멤버 참조 시도
→ 객체를 사용하기 전에 ref. 변수가 null인지 먼저 확인.
- `NegativeArraySizeException` : 배열 크기를 음수로 준 경우
→ 배열 size를 0보다 크게 지정.
- `ArrayIndexOutOfBoundsException` : 배열의 index 범위를 넘어서서 참조 하는 경우
→ 배열이름.length 를 써서 배열의 범위를 확인.
- `ClassCastException` : Cast 연산자 사용시 타입오류
→ instanceof 연산자를 이용하여, 먼저 객체 타입을 확인하고 Cast 연산자를 사용.


```
1 public class ExceptionTest {  
2     public static void main(String[] args) {  
3         String str = null;  
4  
5         String str2 = str.concat( "This is displayed" );  
6  
7         System.out.println( "Contents of str : " + str2 );  
8     }  
9 }
```




NullPointerException

실행결과

```
java.lang.NullPointerException  
    at ExceptionTest.main(ExcetionTest.java:5)  
Exception in thread "main"
```

```
1 public class ExceptionTest {
2     public static void main(String[] args){
3         String str = null;
4         String str2 = null;
5
6         try {
7             str2 = str.concat( "This is displayed" );
8         } catch ( NullPointerException e ) {
9             System.out.println( "null값입니다." );
10        }
11
12        System.out.println( "Contents of str2 : " + str2 );
13    }
14 }
```

NullPointerException



finally 구문에는 오류 발생 여부에 관계 없이 처리 되어져야 하는 로직을 기술한다.

```
public String readFile() {  
  
    try {  
        파일을 연다.  
        파일을 읽는다.  
  
        if ( 파일오픈상태 == true )  
            파일을 닫는다.  
    } catch ( Exception e ) {  
        System.out.println( “개발자 문의” );  
    }  
  
    읽은 것을 return한다.  
}
```



수정

```
public String readFile() {  
  
    try {  
        파일을 연다.  
        파일을 읽는다.  
    } catch ( Exception e ) {  
        System.out.println( “개발자 문의” );  
    } finally {  
        if ( 파일오픈상태 == true )  
            파일을 닫는다.  
    }  
  
    읽은 것을 return한다.  
}
```

발생한 곳에서 직접 처리

- try

Exception 발생할 가능성이 있는 코드를 try 구문 안에 기술

- catch

try 구문에서 Exception 발생시 해당하는 Exception에 대한 처리 기술.

여러 Exception 처리 가능하나, Exception간 상속 관계 고려 해야 함

(자식 → 부모 순 기술, 부모 먼저 기술하면 컴파일 에러)

- finally

Exception 발생 여부에 관계 없이, 꼭 처리해야 하는 logic은 finally 에서 구현한다.

중간에 return문을 만나도, finally 구문은 실행한다.

단, System.exit(); 를 만나면 무조건 프로그램 종료한다.

주로 java.io , java.sql 패키지에 있는 메소드 처리시 많이 이용


Exception Handling 방법 1 – finally example

```
1  import java.io.*;
2  public class ReadFile {
3      public static void main ( String args[] ) {
4          BufferedReader in = null;
5          try {
6              in = new BufferedReader( new FileReader( "c:/data/test.txt" ) );
7              String s;
8              while ( ( s = in.readLine() ) != null ) {
9                  System.out.println( s );
10             }
11         } catch ( FileNotFoundException e ) {
12             System.out.println("파일이 없습니다.");
13         } catch ( IOException e) {
14             e.printStackTrace();
15         } finally {
16             try {
17                 if ( in != null )
18                     in.close();
19             }
20             catch ( IOException e ) { }
21         }
22     }
23 }
```

□ Exception 처리를 호출한 메소드에게 위임 : Call Stack Mechanism

- 메소드 선언시 throws *Exception_Name* 문을 추가하여 호출한 상위 메소드에게 처리 위임
- 계속적으로 위임하면, main() 까지 위임하게 되고 main() 까지 가서 try catch 없으면 비정상 종료된다.

```
public class ThrowTest {  
    public static void main( String[] args ) {  
        ThrowTest t = new ThrowTest();  
  
        try {  
            t.methodA();  
            System.out.println( "정상수행" );  
        } catch (IOException e) {  
            System.out.println( "IOException이 발생" );  
        }  
    }  
  
    public void methodA() throws IOException {  
        methodB();  
    }  
  
    public void methodB() throws IOException {  
        methodC();  
    }  
  
    public void methodC() throws IOException {  
        throw new IOException();  
    }  
}
```



- ❑ Overriding 시 throws하는 Exception은 같거나, 더 구체적인 것(자식)이어야 한다.

```
public class TestA {  
    public void methodA() throws IOException {  
        ...  
    }  
}  
  
public class TestB1 extends TestA {  
    public void methodA() throws EOFException { // → OK  
        ...  
    }  
}  
  
public class TestB2 extends TestA {  
    public void methodA() throws Exception { //→ compile error  
        ...  
    }  
}
```



```
public class Example {
    public static void main(String[] args) {
        SuperClass superClass = new SubClass();

        try {
            superClass.method(); //자식클래스의 Exception 발생
        } catch ( IOException ioe ) { // Exception을 잡을 수 없다.
            System.out.println( "IOException occurred!" ) ;
        }
    }
}

class SuperClass {
    public void method() throws IOException {
        throw new IOException();
    }
}

class SubClass extends SuperClass {
    //컴파일 오류가 안난다고 가정
    public void method() throws Exception {
        throw new Exception(" Exception");
    }
}
```

- ❑ Exception 클래스를 상속하여 작성
- ❑ Exception 발생할 곳에서, `throw new UserDefineException()`

```
public class MyException  
  
    extends Exception {  
  
    public MyException( String msg ) {  
        super( msg );  
    }  
}
```

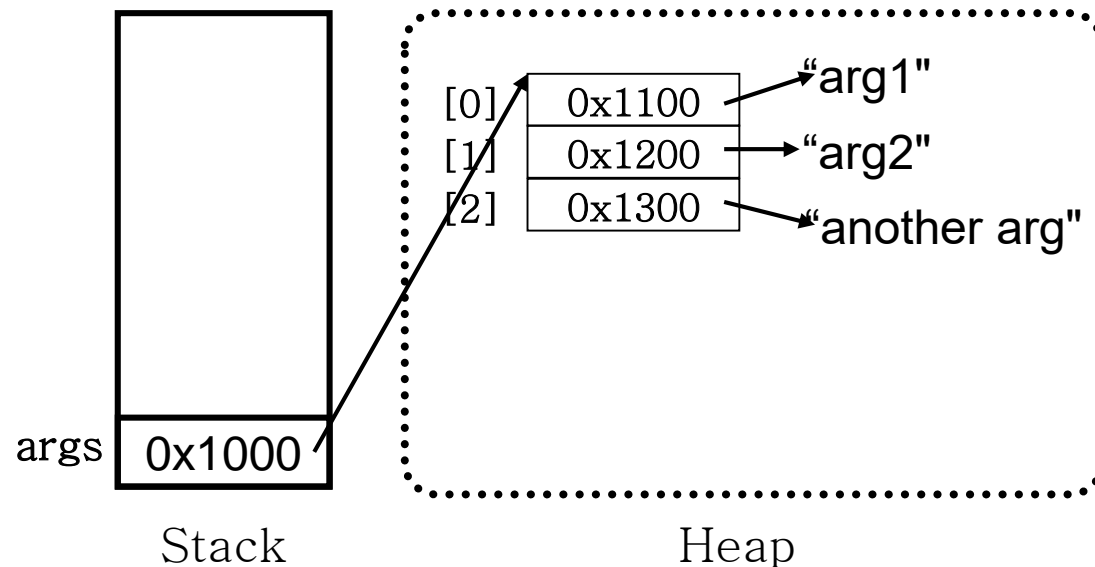
```
public class MyExceptionTest {  
    public static void main( String[] args ) {  
        int age = 20;  
  
        try {  
            if ( age < 19 ) {  
                throw new MyException( "좀 더 크고 오세요" );  
            } else {  
                System.out.println( "즐감" );  
            }  
        } catch ( MyException me ) {  
            System.out.println( me );  
        }  
    }  
}
```

Chap 09. Text-Base Application

1. Command-Line Arguments
2. System Properties & Properties Class
3. File Class
4. File Stream I/O → Chap 14
5. String/StringBuffer Class
6. Collection API

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

```
java TestArgs arg1 arg2 "another arg"
```



```
import java.util.*;
public class TestProperties {
    public static void main( String[] args ) {
        Properties props = System.getProperties();
        Enumeration prop_names = props.propertyNames();

        while ( prop_names.hasMoreElements() ) {
            String prop_name = (String)prop_names.nextElement();
            String property = props.getProperty( prop_name );
            System.out.println( "property '" + prop_name + "' is '" +
                               property + "'" );
        }
    }
}
```

새로운 Property 추가

```
java -Dname=value Class_Name
```

□ String

- String 객체는 immutable(변경할 수 없는) 하다.
- String 변경 메서드를 호출한다는 것은 새로운 객체를 생성한다는 뜻

□ StringBuffer

- StringBuffer 객체는 mutable(변경할 수 있는) 하다.
- Buffer를 두어 문자열 연산을 하여, 빠르다.

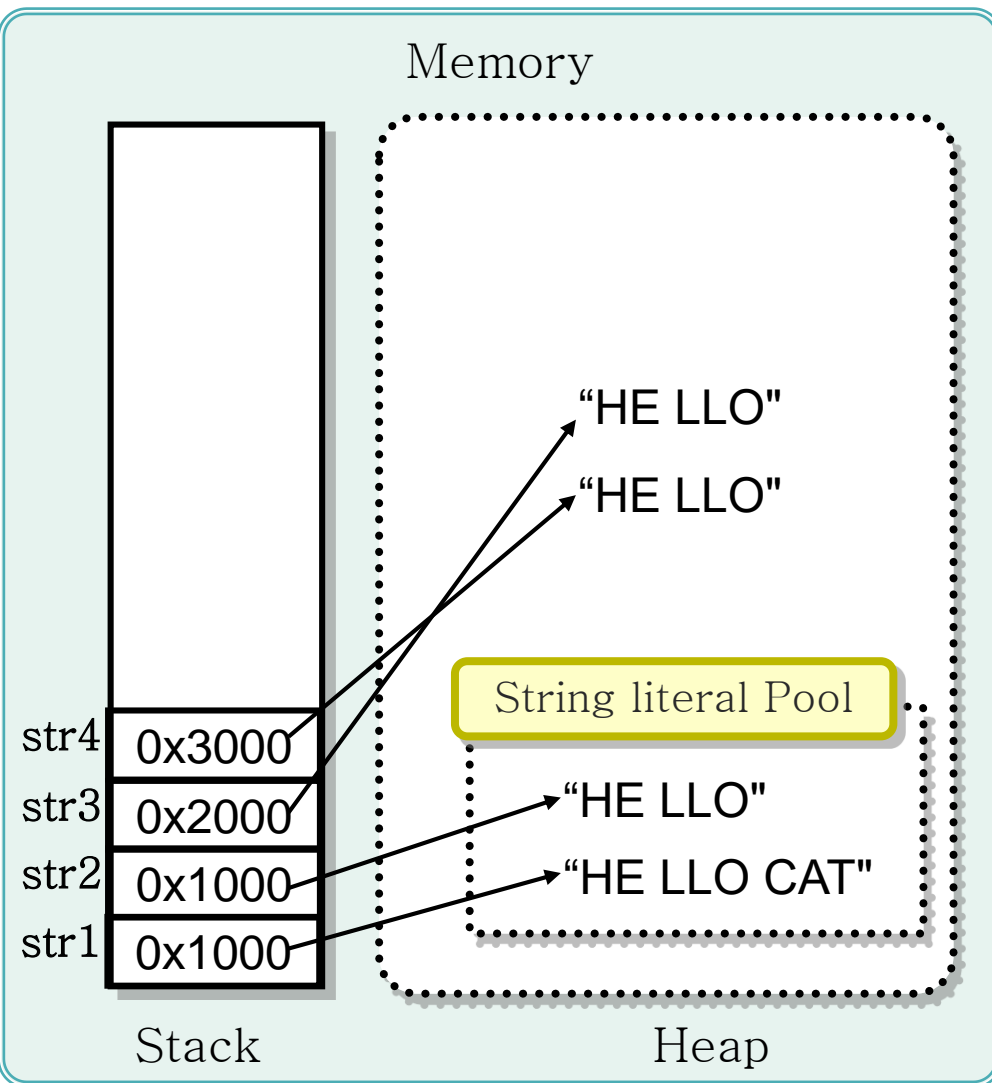
```
public class TestString {  
    public static void main( String[] args ) {  
        //          012345678901234567  
        String s = "HE LLO JAVA World";  
  
        System.out.println( s.substring( 3, 6 ) );  
        System.out.println( s.indexOf( "J", 4 ) );  
        System.out.println( s.charAt(14) );  
        s.concat( " Student" );  
        System.out.println( s );  
    }  
}
```



```
public class TestStringBuffer {  
    public static void main( String[] args ) {  
        StringBuffer sb = new StringBuffer( "ROM" );  
        System.out.println( sb );  
        System.out.println( sb.append( "A" ) );  
        System.out.println( sb.insert( 3, "R" ) );  
        System.out.println( sb.reverse() );  
    }  
}
```

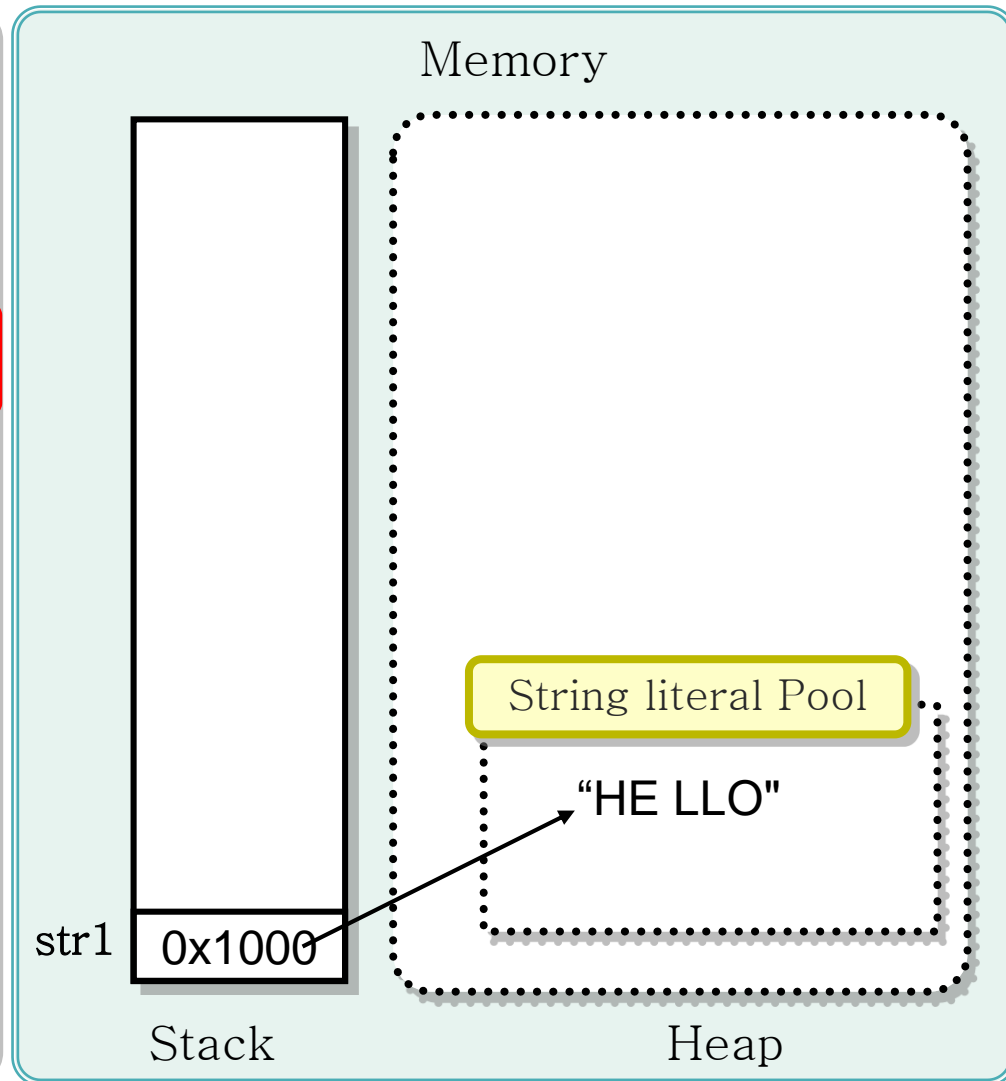
String 객체 메모리 할당 영역

```
public class StringStringBuffer {  
    public static void main( String[] args ) {  
        String str1 = "HE LLO";  
        String str2 = "HE LLO";  
        String str3 = new String( "HE LLO" );  
        String str4 = new String( "HE LLO" );  
        String str1 = str1 + " CAT";  
    }  
}
```



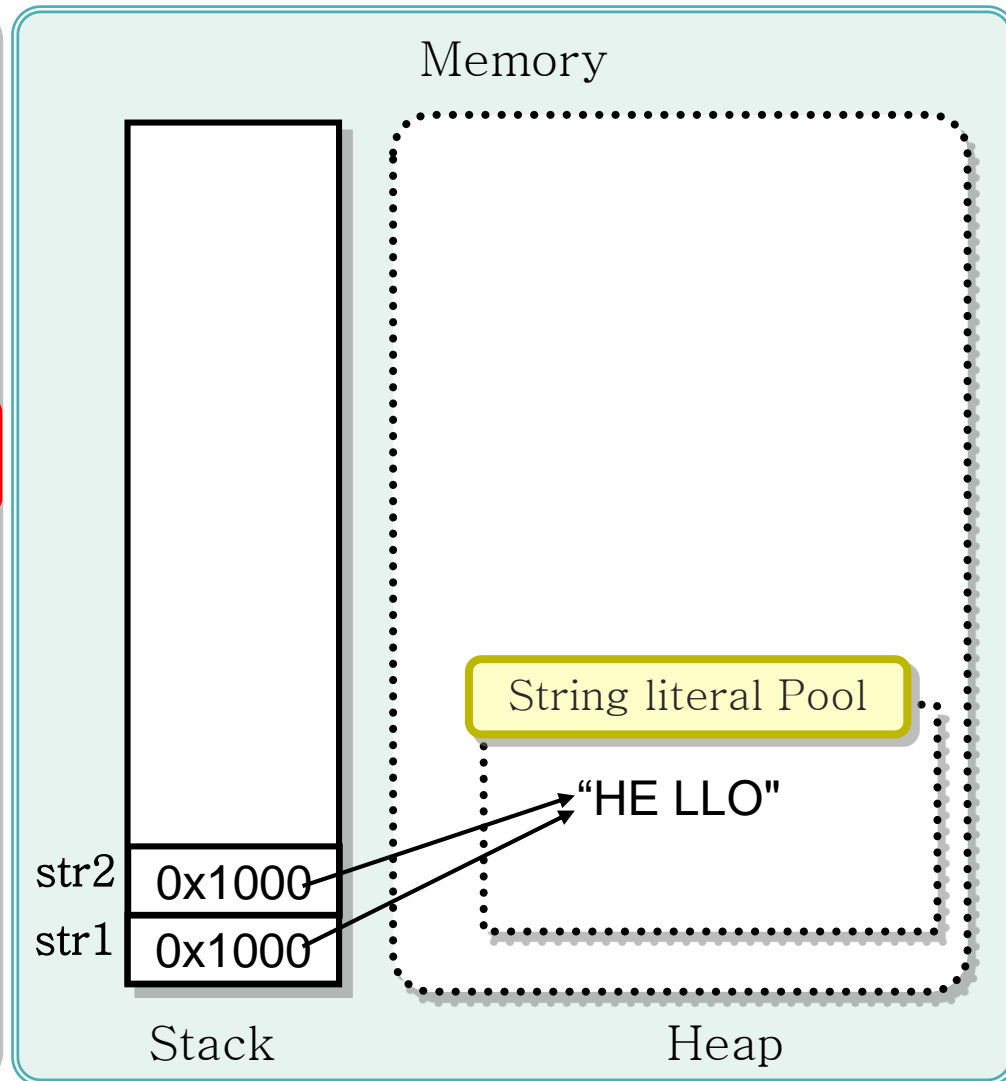
String 객체 메모리 할당 영역

```
public class StringStringBuffer {  
    public static void main( String[] args ) {  
        String str1 = "HE LLO";  
        String str2 = "HE LLO";  
        String str3 = new String( "HE LLO" );  
        String str4 = new String( "HE LLO" );  
        String str1 = str1 + " CAT";  
    }  
}
```



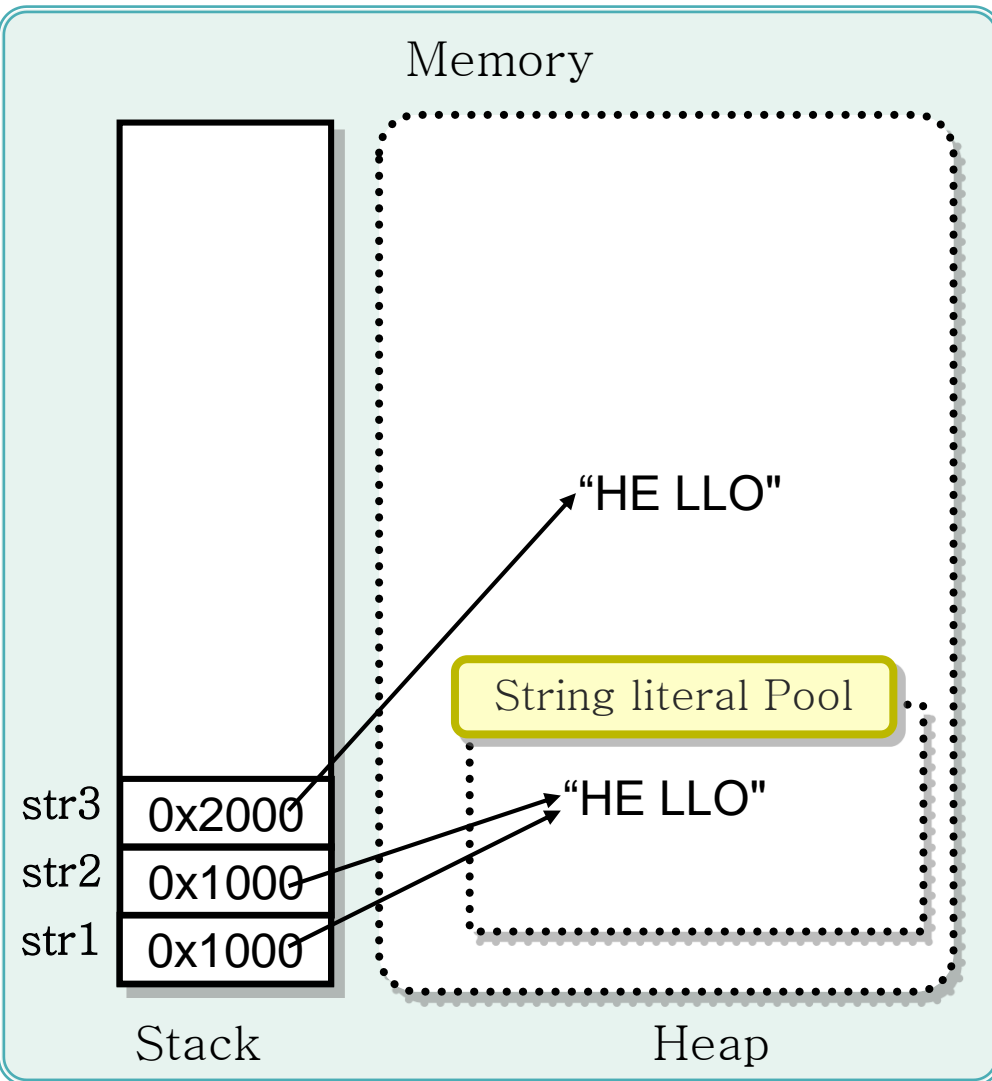
String 객체 메모리 할당 영역

```
public class StringStringBuffer {  
    public static void main( String[] args ) {  
        String str1 = "HE LLO";  
        String str2 = "HE LLO";  
        String str3 = new String( "HE LLO" );  
        String str4 = new String( "HE LLO" );  
        String str1 = str1 + " CAT";  
    }  
}
```



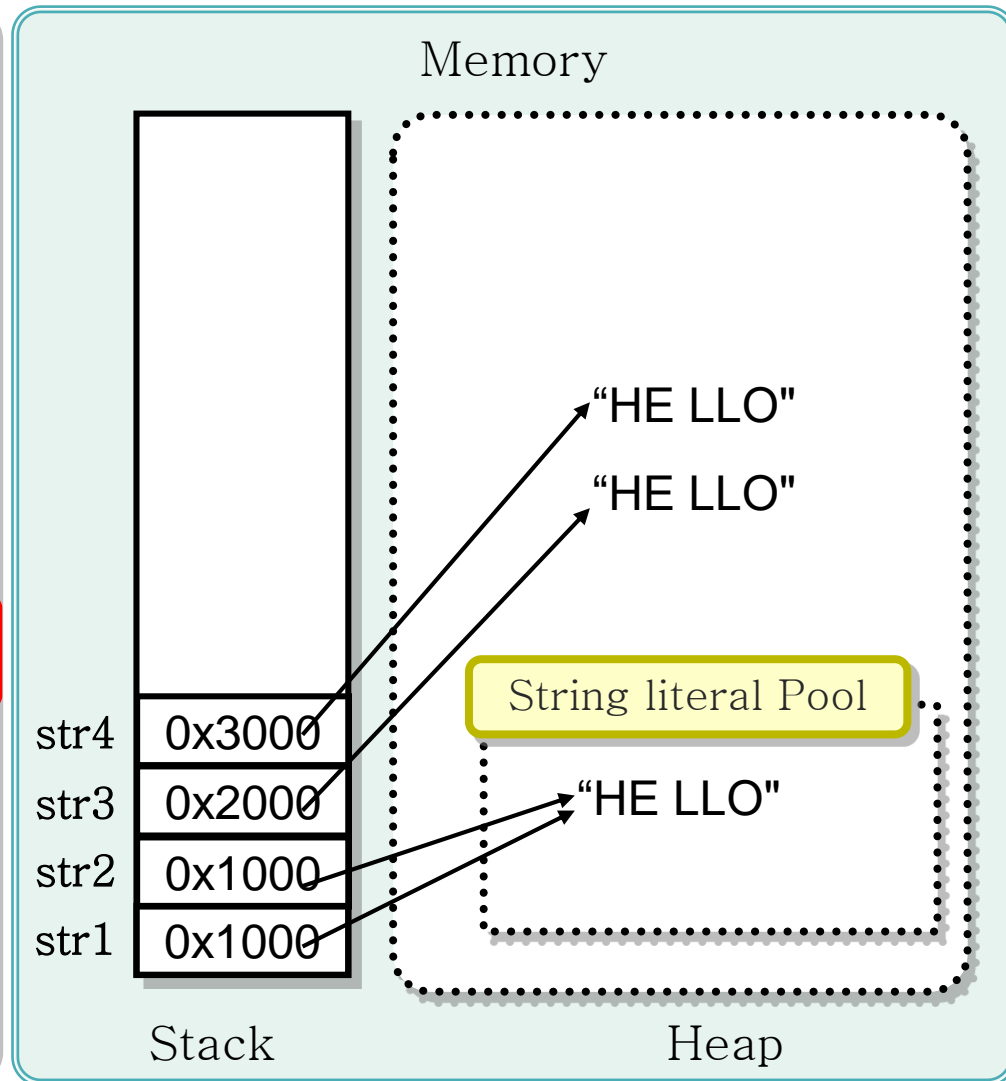
String 객체 메모리 할당 영역

```
public class StringStringBuffer {  
    public static void main( String[] args ) {  
        String str1 = "HE LLO";  
        String str2 = "HE LLO";  
        String str3 = new String( "HE LLO" );  
        String str4 = new String( "HE LLO" );  
        String str1 = str1 + " VCC";  
    }  
}
```



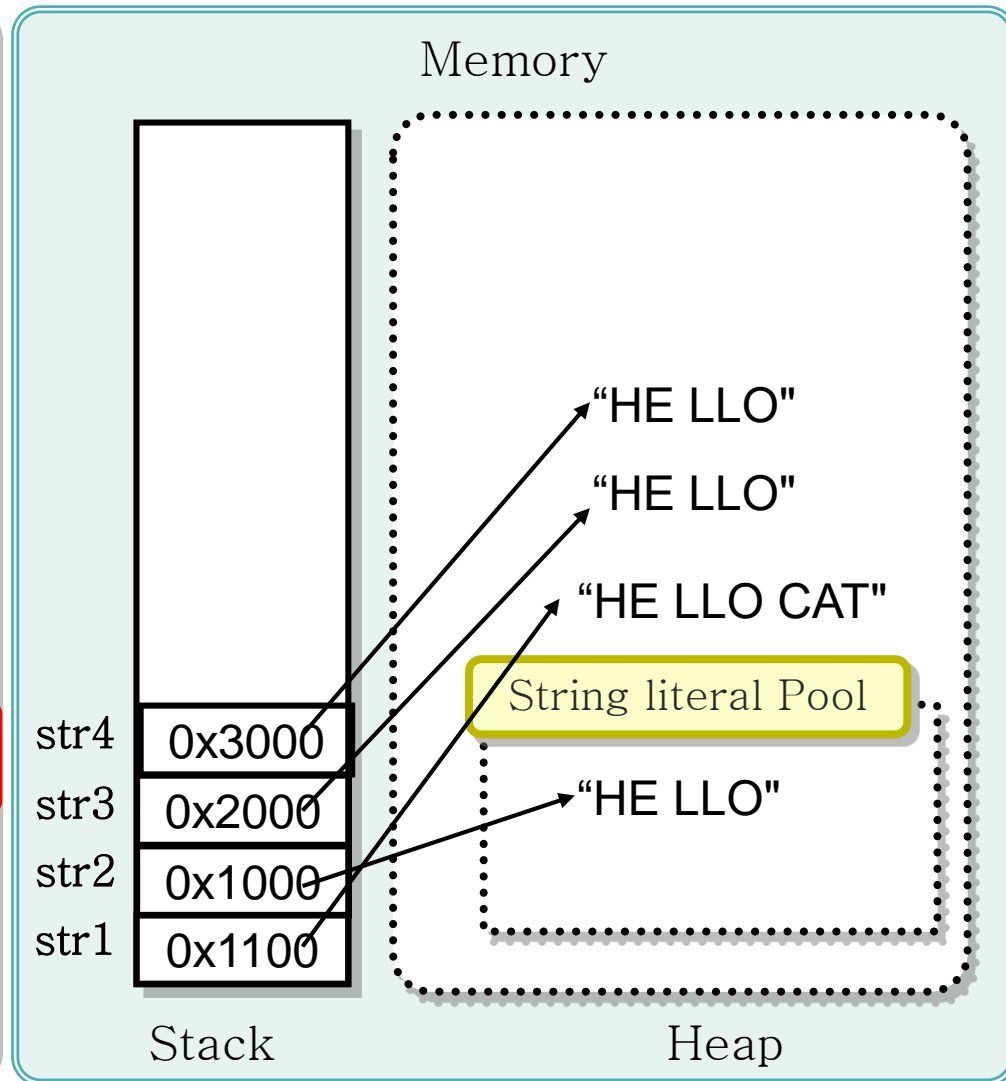
String 객체 메모리 할당 영역

```
public class StringStringBuffer {  
    public static void main( String[] args ) {  
        String str1 = "HE LLO";  
        String str2 = "HE LLO";  
        String str3 = new String( "HE LLO" );  
        String str4 = new String( "HE LLO" );  
        String str1 = str1 + " CAT";  
    }  
}
```



String 객체 메모리 할당 영역

```
public class StringStringBuffer {  
    public static void main( String[] args ) {  
        String str1 = "HE LLO";  
        String str2 = "HE LLO";  
        String str3 = new String( "HE LLO" );  
        String str4 = new String( "HE LLO" );  
        String str1 = str1 + " CAT";  
    }  
}
```



String 문자열 비교

```
String str1 = "HELLO";
String str2 = "HELLO";
String str3 = new String("HELLO" );
String str4 = new String( "HELLO" );

if ( str1 == str2 ) {
    System.out.println( "str1 과 str2는 같은 객체이다" );
} else {
    System.out.println( "str1 과 str2는 다른 객체이다" );
}

if ( str2 == str3 ) {
    System.out.println( "str2 과 str3는 같은 객체이다" );
} else {
    System.out.println( "str2 과 str3는 다른 객체이다" );
}

if ( str3 == str4 ) {
    System.out.println( "str3 과 str4는 같은 객체이다" );
} else {
    System.out.println( "str3 과 str4는 다른 객체이다" );
}

if ( str1.equals( str4 ) ) {
    System.out.println( "str1 과 str4는 같은 문자열이다" );
} else {
    System.out.println( "str1 과 str4는 다른 문자열이다" );
}
```


문자열 비교시 주의점

- ❑ 문자열 비교는 반드시 equals() 메소드를 사용할 것
- ❑ == 연산자 비교의 사용은 하지 말 것!!!
- ❑ 문자열 비교는 String ref변수가 null 이 아닌 경우에 equals 메소드를 호출 해야 하기 때문에, null 체크를 해 주는 습관을 들이자.

```
if ( str1 != null && str1.equals( str2 ) ) {  
    ...  
}
```

StringBuffer 문자열 비교

```
StringBuffer sb1 = new StringBuffer("HELLO" );
StringBuffer sb2 = new StringBuffer( "HELLO" );

if ( sb1.equals( sb2 ) ) {
    System.out.println( "sb1 와 sb2은 같은 문자열이다" );
} else {
    System.out.println( "sb1 와 sb2은 다른 문자열이다" );
}

if ( sb1.toString().equals( sb2.toString() ) ) {
    System.out.println( "sb1 와 sb2은 같은 문자열이다" );
} else {
    System.out.println( "sb1 와 sb2은 다른 문자열이다" );
}
```

StringBuffer와 String 문자열 비교

```
StringBuffer sb1 = new StringBuffer( "HELLO" );  
StringBuffer sb2 = new StringBuffer( "HELLO" );  
String str = "HELLO";
```

```
if ( str.equals( sb1 ) ) {  
    System.out.println( "sb 와 str은 같은 문자열이다" );  
} else {  
    System.out.println( "sb 와 str은 다른 문자열이다" );  
}
```

```
if ( str.equals( sb1.toString() ) ) {  
    System.out.println( "sb 와 str은 같은 문자열이다" );  
} else {  
    System.out.println( "sb 와 str은 다른 문자열이다" );  
}
```

```
if ( str.contentEquals( sb1 ) ) {  
    System.out.println( "sb 와 str은 같은 문자열이다" );  
} else {  
    System.out.println( "sb 와 str은 다른 문자열이다" );  
}
```

StringBuffer 문자열 비교시 주의점

- ❑ 문자열 비교는 반드시 toString()를 이용하여, String으로 변환 후, equals() 사용한다.

```
if ( sb1.toString().equals(sb2.toString()) ) {  
  
}
```

```
import java.util.*;

public class StringTokenizerTest {

    public static void main( String[] args ) {

        String str = "하하|호호|후후";

        StringTokenizer st = new StringTokenizer( str, "|" );

        while ( st.hasMoreElements() ) {
            System.out.println( st.nextToken() );
        }
    }
}
```

```
import java.util.Date;
import java.text.SimpleDateFormat;

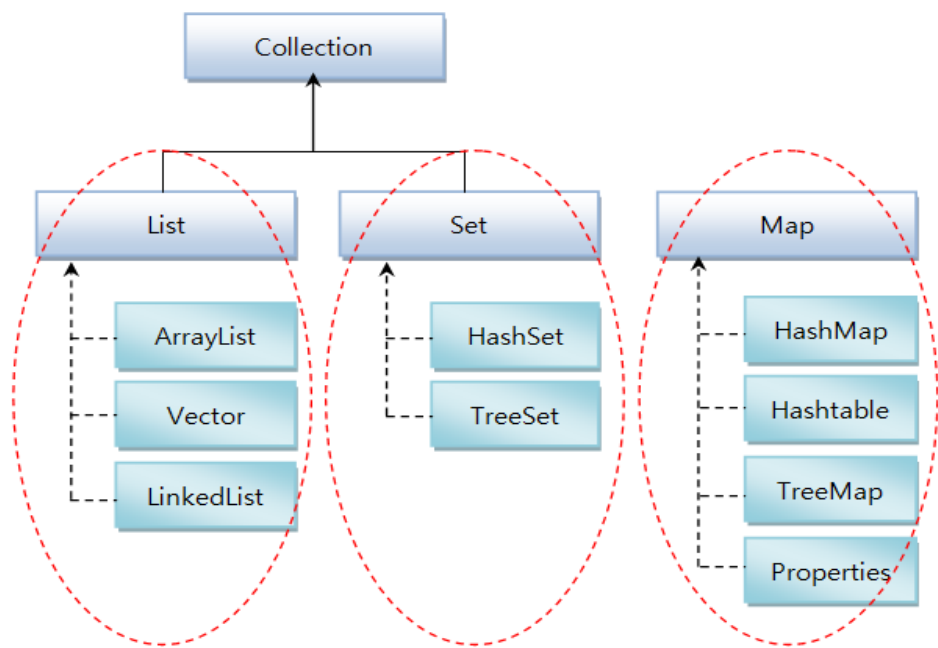
public class ForamttingDate {

    public static void main( String[] args ) {
        Date today = new Date();
        System.out.println( today );

        SimpleDateFormat ft = new SimpleDateFormat( "yyyy-MM-dd" );
        String formattedDate = ft.format(new Date());
        System.out.println( formattedDate );
    }
}
```

Collection 프레임워크 소개

❖ 컬렉션 프레임워크의 주요 인터페이스



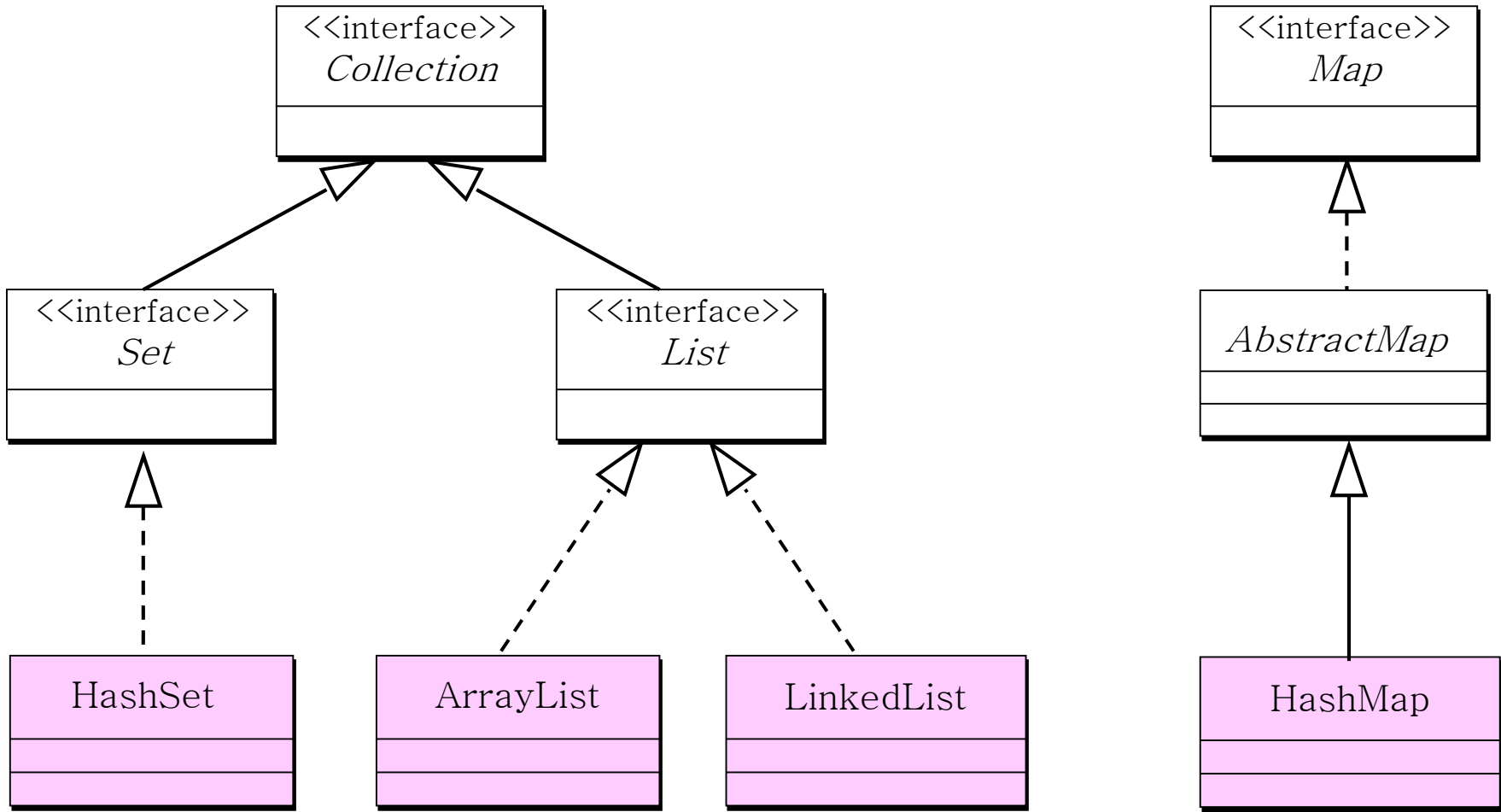
인터페이스 분류		특징	구현 클래스
Collection	List 계열	- 순서를 유지하고 저장 - 중복 저장 가능	ArrayList, Vector, LinkedList
	Set 계열	- 순서를 유지하지 않고 저장 - 중복 저장 안됨	HashSet, TreeSet
Map 계열		- 키와 값의 쌍으로 저장 - 키는 중복 저장 안됨	HashMap, Hashtable, TreeMap, Properties

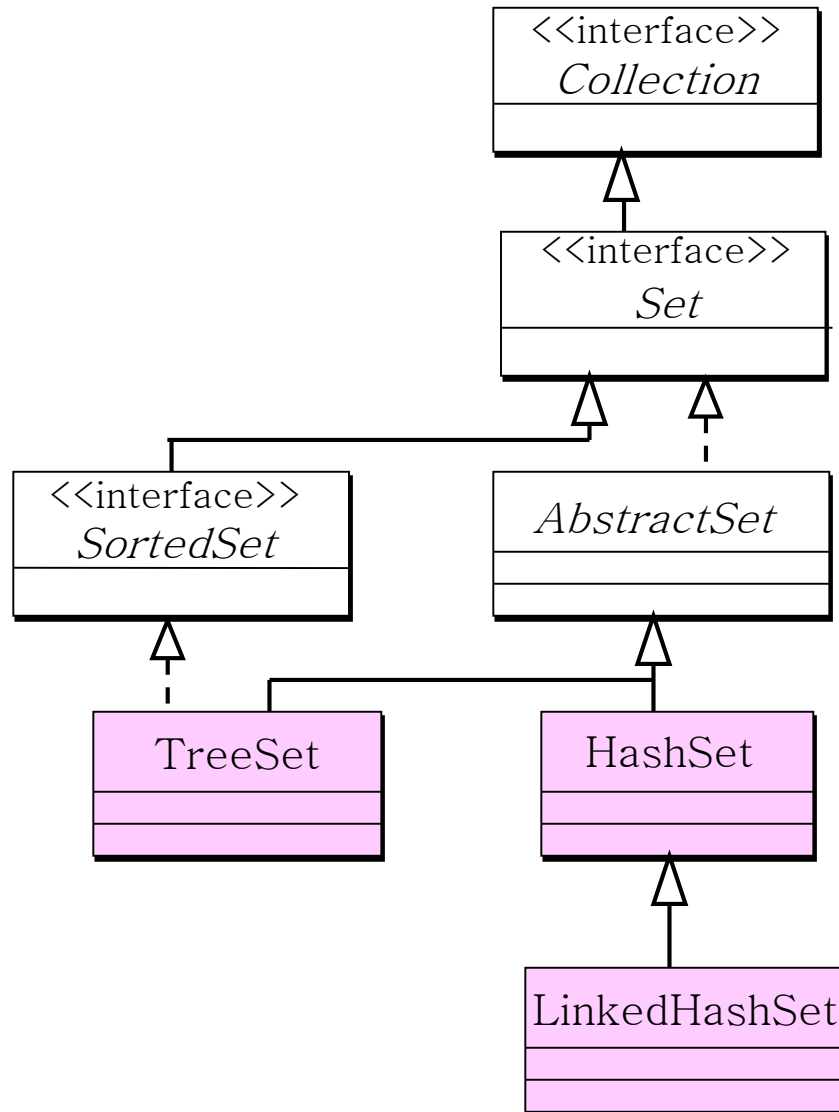
□ 배열

- 장점 : 객체를 저장하고, 검색하는데 가장 효율적이다, 사용하기 편하다.
- 단점 : 사이즈 변경이 불가능하다.

□ Collection

- 객체를 저장할 때 마다, 크기를 자동으로 늘려 준다.
- Set 계열 : 중복을 허용하지 않고, 추가되는 순서를 유지하지 않는다.
- List 계열 : 중복을 허용하고, 추가되는 순서를 유지한다.
- Map 계열 : 키와 값의 쌍으로 저장 된다. (키와 값 모두 객체여야 한다.)
- java.util package에 있다.





❑ HashSet

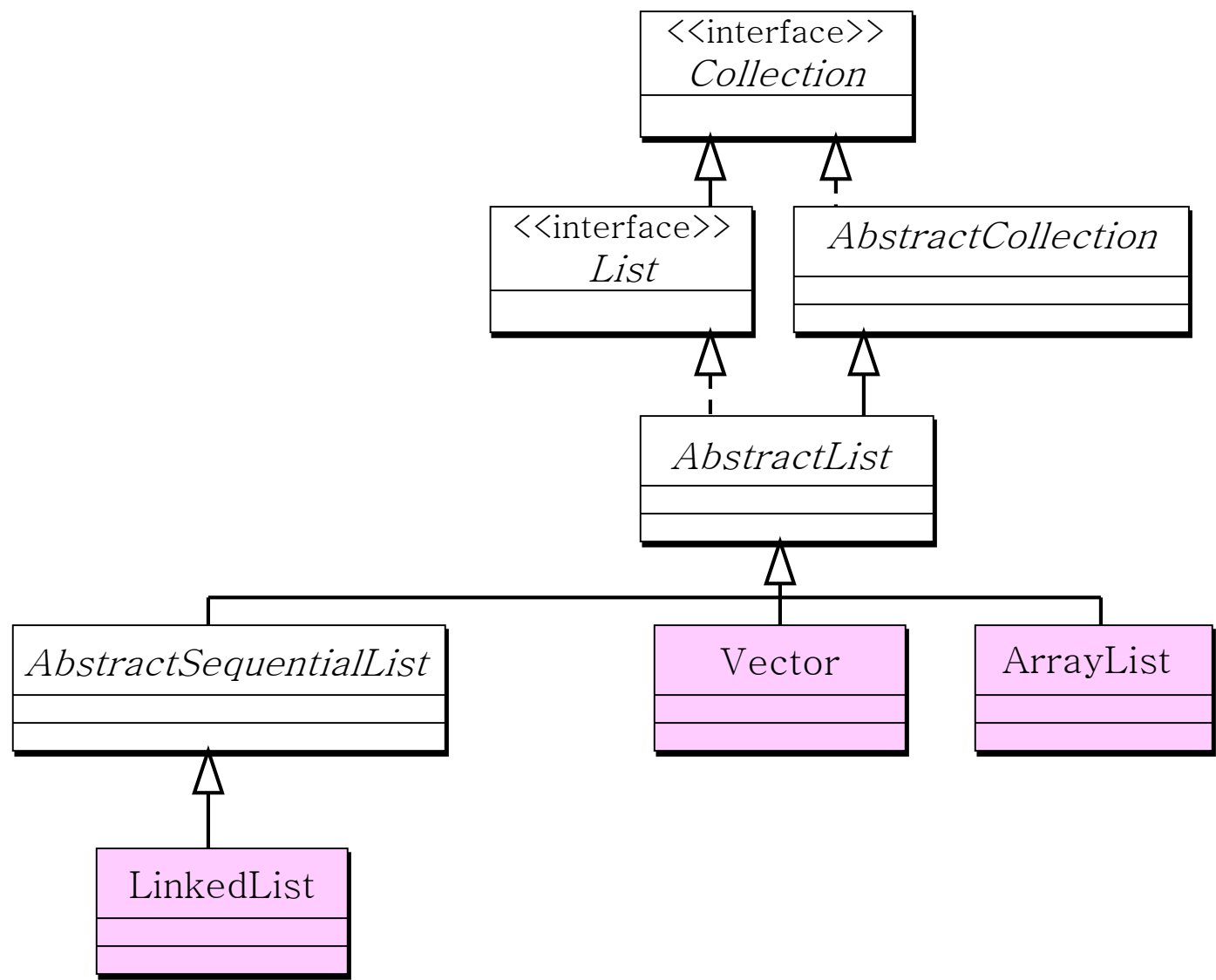
- Set에 객체를 저장하는데 Hash를 사용하여 처리 속도가 빠르다.
- hashCode() 및 equals()를 재정의 해야 한다.
- 오직 하나의 Null객체를 가질 수 있다.

❑ LinkedHashSet

- HashSet과 거의 같다. 차이점은 Set에 추가되는 순서를 유지한다는 점.
- JDK1.4에서 추가

❑ TreeSet

- 객체의 Hash값에 의한 오름차순의 정렬 유지





ArrayList 객체를 얻어내는데 효율적

□ LinkedList

- List에서 데이터를 삽입하거나 삭제하는데 효율적

□ Vector

- List객체들 중에서, 가장 성능이 좋지 않다.
- 동기화(Synchronization)를 제공한다.

List 컬렉션

❖ List 컬렉션의 특징 및 주요 메소드

■ 주요 메소드

기능	메소드	설명
객체 추가	<code>boolean add(E e)</code>	주어진 객체를 맨끝에 추가
	<code>void add(int index, E element)</code>	주어진 인덱스에 객체를 추가
	<code>set(int index, E element)</code>	주어진 인덱스에 저장된 객체를 주어진 객체로 바꿈
객체 검색	<code>boolean contains(Object o)</code>	주어진 객체가 저장되어 있는지 여부
	<code>E get(int index)</code>	주어진 인덱스에 저장된 객체를 리턴
	<code>isEmpty()</code>	컬렉션이 비어 있는지 조사
	<code>int size()</code>	저장되어있는 전체 객체수를 리턴
객체 삭제	<code>void clear()</code>	저장된 모든 객체를 삭제
	<code>E remove(int index)</code>	주어진 인덱스에 저장된 객체를 삭제
	<code>boolean remove(Object o)</code>	주어진 객체를 삭제

List 컬렉션 – ArrayList

❖ ArrayList

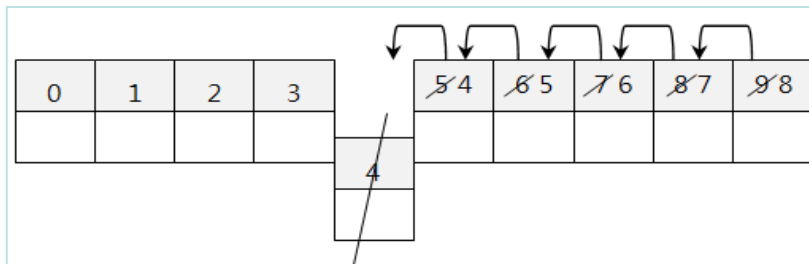
■ 저장 용량(capacity)

- 초기 용량 : 10 (따로 지정 가능)
- 저장 용량을 초과한 객체들이 들어오면 자동적으로 늘어남. 고정도 가능



■ 객체 제거

- 바로 뒤 인덱스부터 마지막 인덱스까지 모두 앞으로 1씩 당겨짐



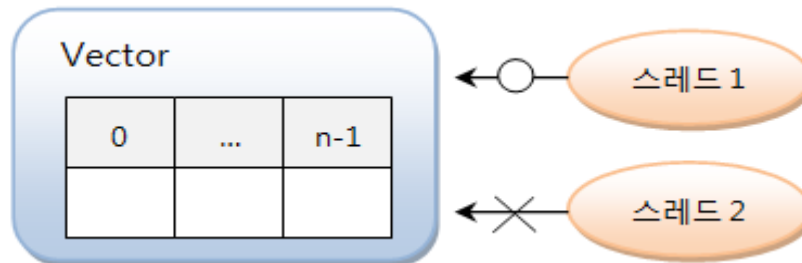
List 컬렉션 – Vector

❖ Vector

```
List<E> list = new Vector<E>();
```

■ 특징

- Vector는 스레드 동기화(synchronization)
 - 복수의 스레드가 동시에 Vector에 접근해 객체를 추가, 삭제하더라도 스레드에 안전(thread safe)



List 컬렉션 – LinkedList

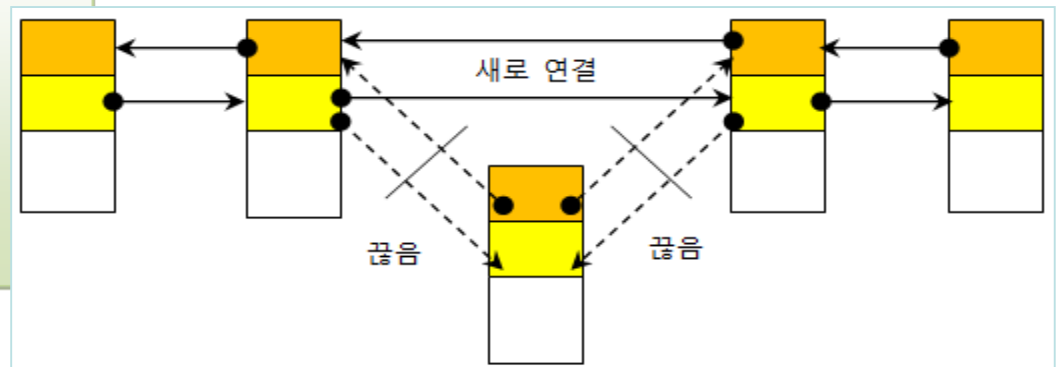
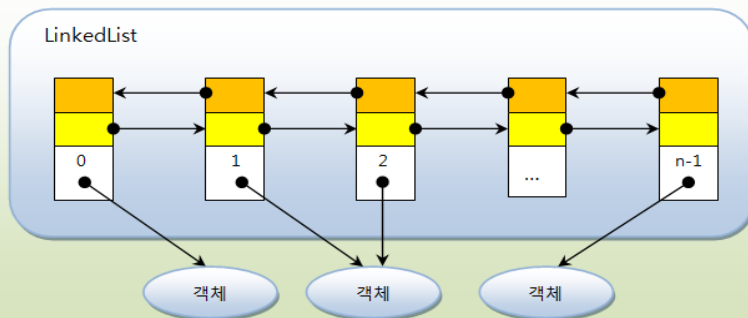
❖ LinkedList

```
List<E> list = new LinkedList<E>();
```

■ 특징

- 인접 참조를 링크해서 체인처럼 관리
- 특정 인덱스에서 객체를 제거하거나 추가하게 되면 바로 앞뒤 링크만 변경
- 빈번한 객체 삭제와 삽입이 일어나는 곳에서는 ArrayList보다 좋은 성능

힙 영역



```
import java.util.*;
public class ListExample {
    public static void main( String[] args ) {
        List list = new ArrayList();
        list.add( "add" );
        list.add( "second" );
        list.add( "3rd" );
        list.add( new Integer(4) );
        list.add( new Float(5.0F) );
        list.add( "second" );
        list.add( new Integer(4) );
        System.out.println( list );

        list.remove( "second" );
        list.remove( new Float(5.0F) );
        System.out.println( list );
    }
}
```

```
import java.util.ArrayList;

public class ListTest {
    public static void main( String[] args ) {
        List list = new ArrayList();
        MyDate d1 = new MyDate(2006,03,01 );
        MyDate d2 = new MyDate(2006,05,01 );

        System.out.println( "d1:" + d1 );
        System.out.println( "d2:" + d2 );

        list.add(d1);
        list.add(d1);
        list.add(d2);

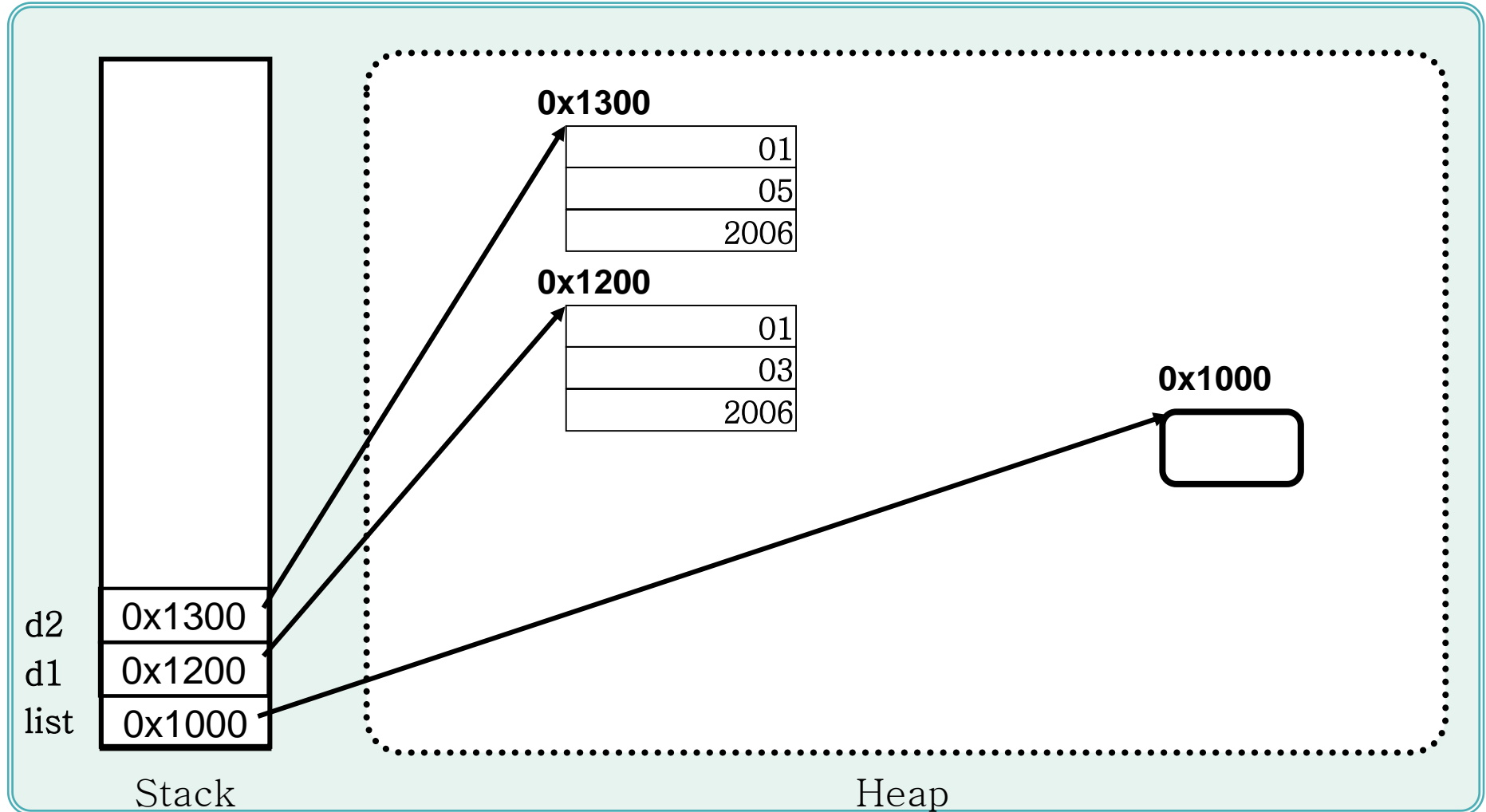
        d1.setDay(30);

        MyDate d11 = (MyDate)list.get(0);
        MyDate d12 = (MyDate)list.get(1);

        System.out.println( "d11.getDay(): "+d11.getDay() );
        System.out.println( "d12.getDay(): "+d12.getDay() );
        System.out.println( "d2:" + (MyDate)list.get(2));
    }
}
```

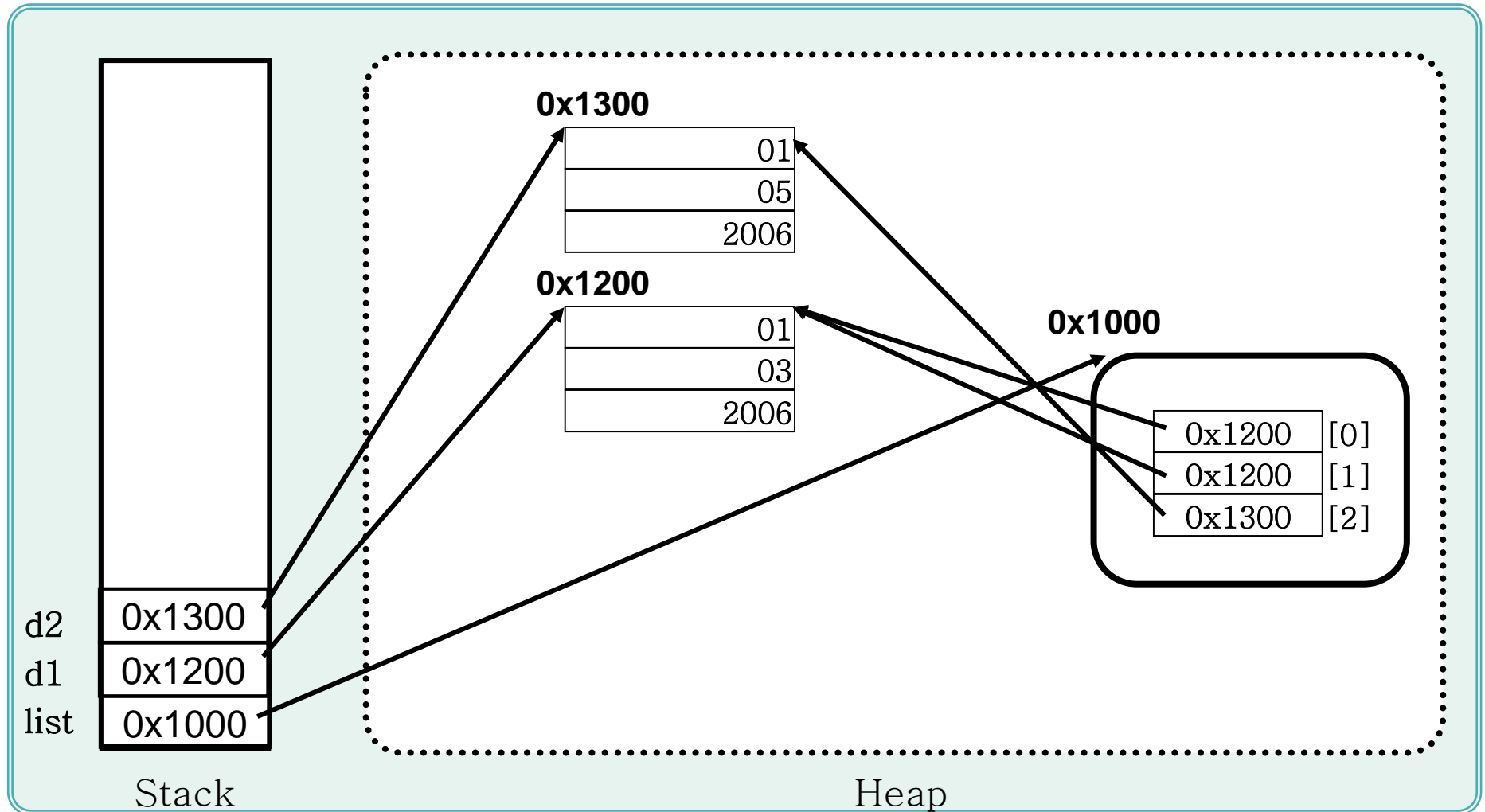
Collections – Vector

```
List list = new ArrayList();  
MyDate d1 = new MyDate(2006,03,01 );  
MyDate d2 = new MyDate(2006,05,01 );
```



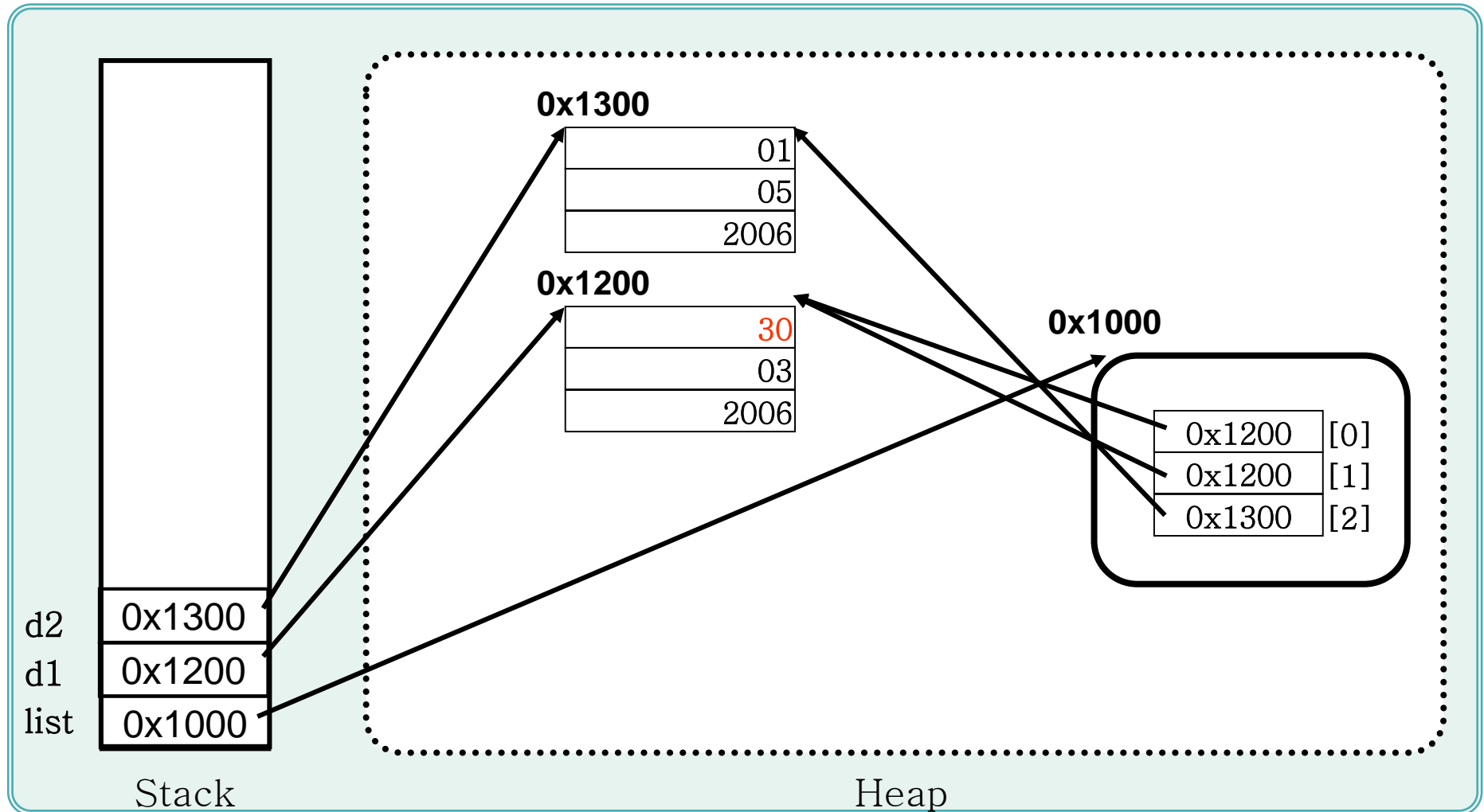
Collections – Vector

```
list.add(d1);  
list.add(d1);  
list.add(d2);
```



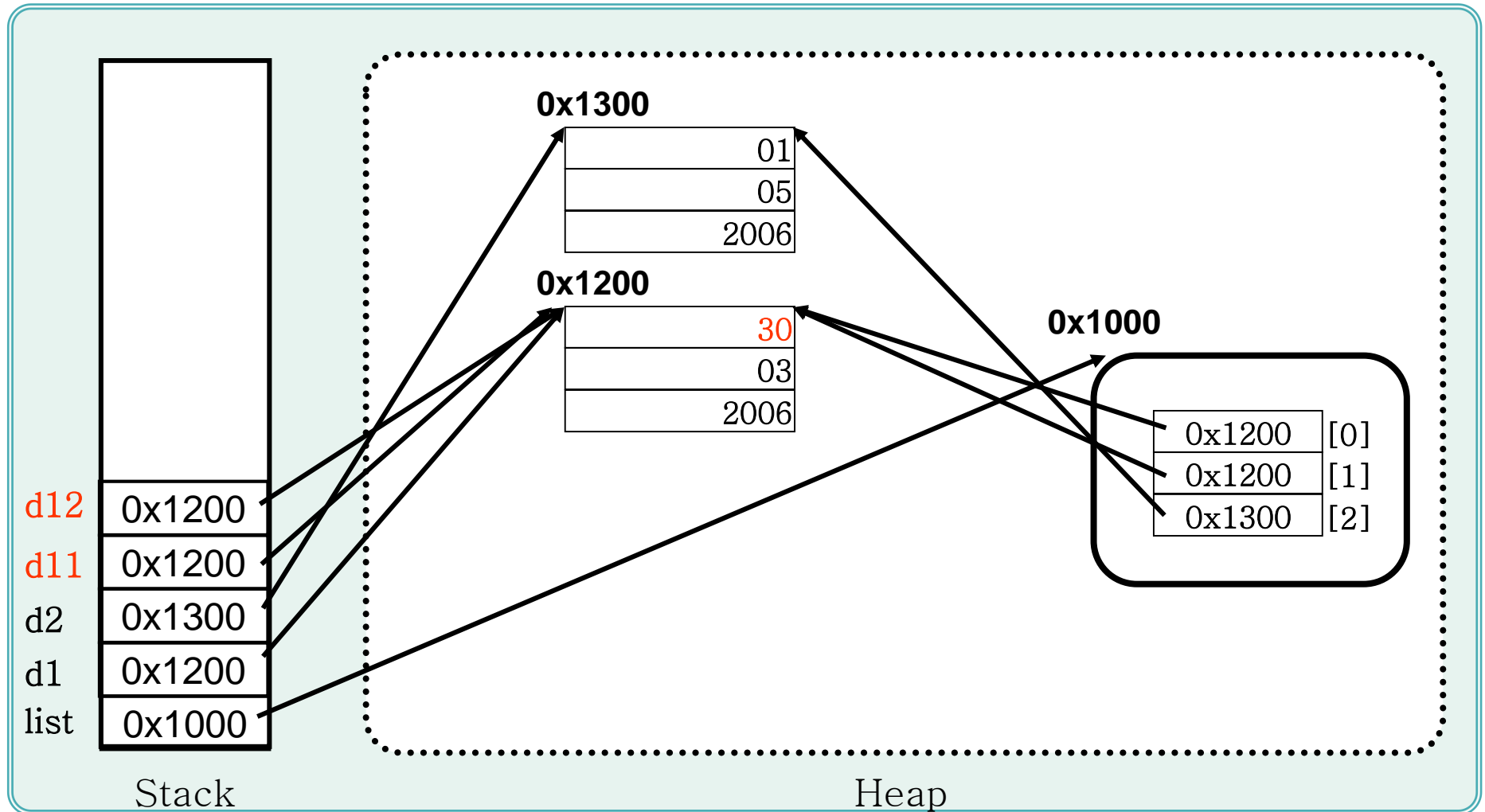
Collections – Vector

```
d1.setDay(30);
```



Collections – Vector

```
MyDate d11 = (MyDate)vec.get(0);  
MyDate d12 = (MyDate)vec.get(1);
```



Collections – Set

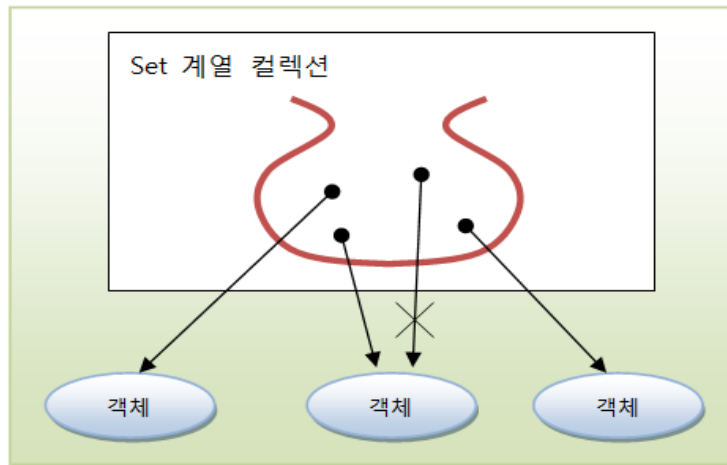
❖ Set 컬렉션의 특징 및 주요 메소드

■ 특징

- 수학의 집합에 비유
- 저장 순서가 유지되지 않음
- 객체를 중복 저장 불가
- 하나의 null만 저장 가능

■ 구현 클래스

- HashSet, LinkedHashSet, TreeSet



Collections – Set

❖ Set 컬렉션의 특징 및 주요 메소드

■ 주요 메소드

기능	메소드	설명
객체 추가	<code>boolean add(E e)</code>	주어진 객체를 저장, 객체가 성공적으로 저장되면 <code>true</code> 를 리턴하고 중복 객체면 <code>false</code> 를 리턴
객체 검색	<code>boolean contains(Object o)</code>	주어진 객체가 저장되어 있는지 여부
	<code>isEmpty()</code>	컬렉션이 비어 있는지 조사
	<code>Iterator<E> iterator()</code>	저장된 객체를 한번씩 가져오는 반복자 리턴
	<code>int size()</code>	저장되어있는 전체 객체수 리턴
객체 삭제	<code>void clear()</code>	저장된 모든 객체를 삭제
	<code>boolean remove(Object o)</code>	주어진 객체를 삭제

- 전체 객체 대상으로 한 번씩 반복해 가져오는 반복자(iterator) 제공
 - 인덱스로 객체를 검색해서 가져오는 메소드 없음

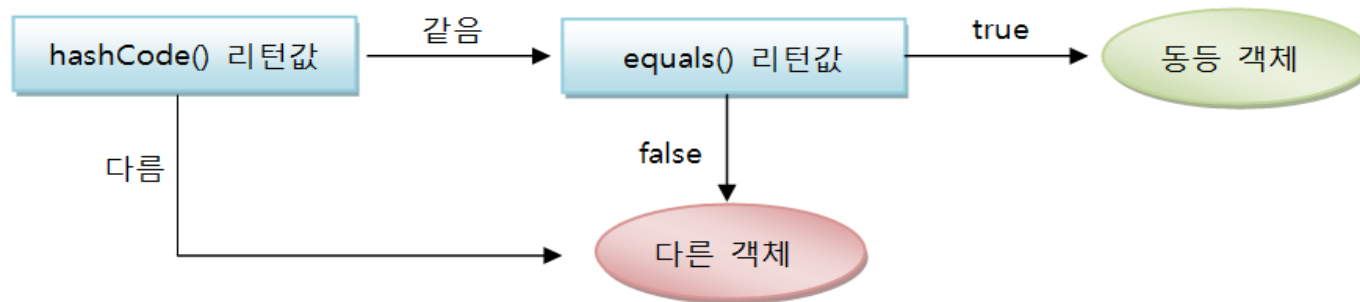
Collections – Set

❖ HashSet

```
Set<E> set = new HashSet<E>();
```

■ 특징

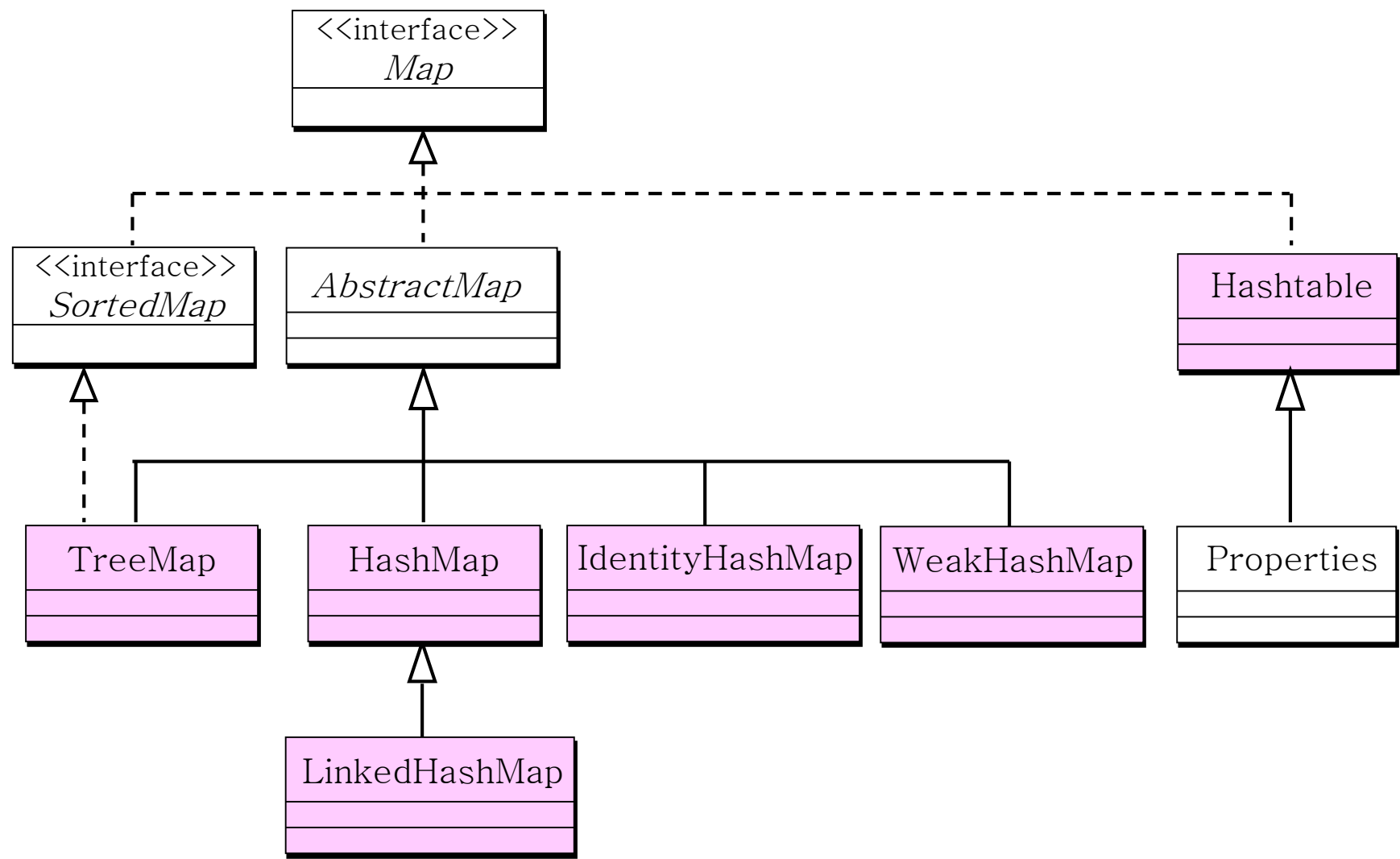
- 동일 객체 및 동등 객체는 중복 저장하지 않음
- 동등 객체 판단 방법



```
import java.util.*;
public class SetExample {
    public static void main( String[] args ) {
        Set set = new HashSet();
        set.add( "add" );
        set.add( "second" );
        set.add( "3rd" );
        set.add( new Integer(4) );
        set.add( new Float(5.0F) );
        set.add( "second" );
        set.add( new Integer(4) );
        System.out.println( set );

        set.remove( "second" );
        set.remove( new Float(5.0F) );
        System.out.println( set );
    }
}
```

Collections – Map 계열



Collections – Map

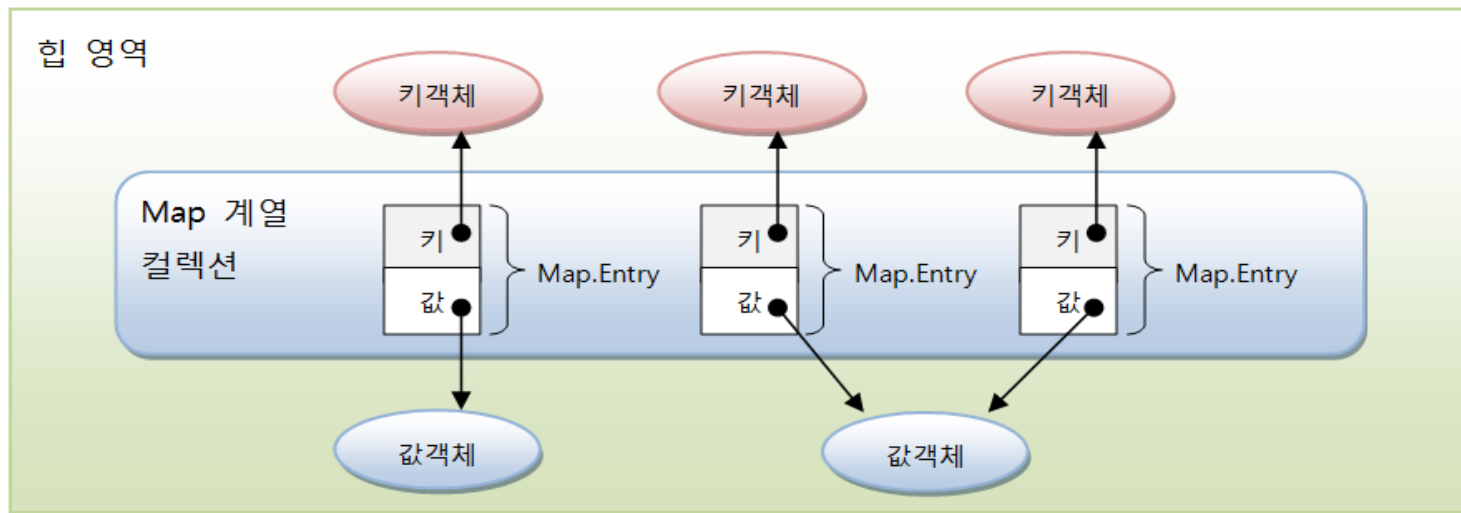
❖ Map 컬렉션의 특징 및 주요 메소드

■ 특징

- 키(key)와 값(value)으로 구성된 Map.Entry 객체를 저장하는 구조
- 키와 값은 모두 객체
- 키는 중복될 수 없지만 값은 중복 저장 가능

■ 구현 클래스

- HashMap, Hashtable, LinkedHashMap, Properties, TreeMap



Collections – Map

❖ Map 컬렉션의 특징 및 주요 메소드

■ 주요 메소드

기능	메소드	설명
객체 추가	<code>V put(K key, V value)</code>	주어진 키와 값을 추가, 저장이 되면 값을 리턴
객체 검색	<code>boolean containsKey(Object key)</code>	주어진 키가 있는지 여부
	<code>boolean containsValue(Object value)</code>	주어진 값이 있는지 여부
	<code>Set<Map.Entry<K,V>> entrySet()</code>	키와 값의 쌍으로 구성된 모든 <code>Map.Entry</code> 객체를 <code>Set</code> 에 담아서 리턴
	<code>V get(Object key)</code>	주어진 키의 값을 리턴
	<code>boolean isEmpty()</code>	컬렉션이 비어있는지 여부
	<code>Set<K> keySet()</code>	모든 키를 <code>Set</code> 객체에 담아서 리턴
	<code>int size()</code>	저장된 키의 총 수를 리턴
	<code>Collection<V> values()</code>	저장된 모든 값 <code>Collection</code> 에 담아서 리턴
객체 삭제	<code>void clear()</code>	모든 <code>Map.Entry</code> (키와 값)를 삭제
	<code>V remove(Object key)</code>	주어진 키와 일치하는 <code>Map.Entry</code> 삭제, 삭제가 되면 값을 리턴

Collections – Map 컬렉션

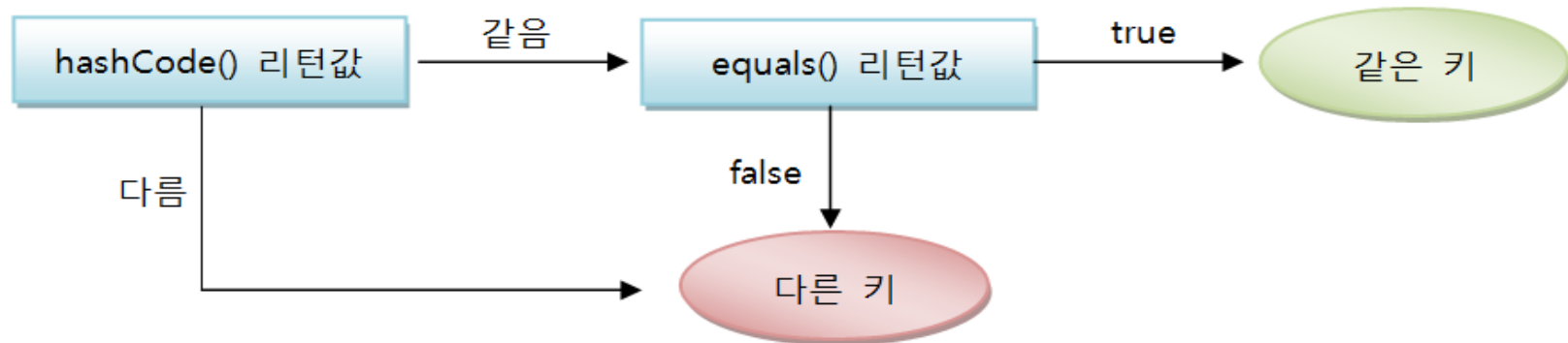
❖ HashMap

■ 특징

```
Map<K, V> map = new HashMap<K, V>();
```

키 타입 값 타입 키 타입 값 타입

- 키 객체는 hashCode()와 equals() 를 재정의해 동등 객체가 될 조건을 정해야



- 키 타입은 String 많이 사용
 - String은 문자열이 같을 경우 동등 객체가 될 수 있도록 hashCode()와 equals() 메소드가 재정의되어 있기 때문

Collections – Map 컬렉션

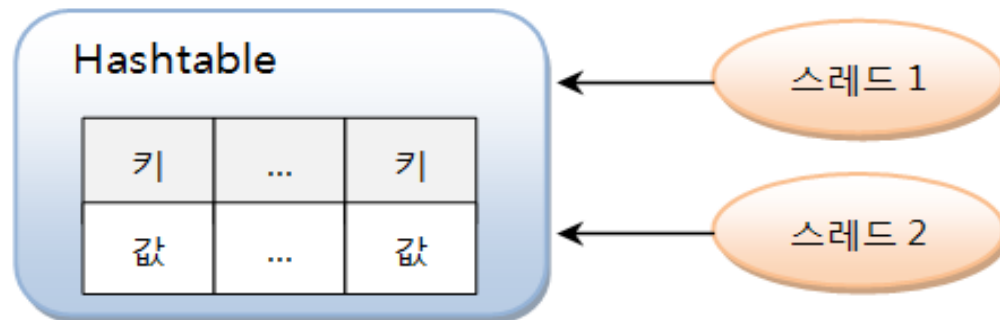
❖ Hashtable

```
Map<K, V> map = new Hashtable<K, V>();
```

키 타입 값 타입 키 타입 값 타입

■ 특징

- 키 객체 만드는 법은 HashMap과 동일
- Hashtable은 스레드 동기화(synchronization)가 된 상태
 - 복수의 스레드가 동시에 Hashtable에 접근해서 객체를 추가, 삭제하더라도 스레드에 안전(thread safe)



스레드 동기화 적용됨

Collections – Map 계열

❑ HashMap

- Map에 키를 저장하는데 hash를 사용하여 성능이 좋다.
- 저장되는 순서가 유지 되지 않는다.
- 오직 하나의 Null 키를가질 수 있다.

❑ LinkedHashMap

- HashMap과 거의 같다. 차이점은 Map에 추가되는 순서를 유지한다는 점.
- JDK1.4에서 추가

❑ Hashtable

- 동기화(Synchronization)를 제공한다.
- null키와 null 값을 저장할 수 없다.

Collections – Map 계열

```
import java.util.*;
public class MapExample {
    public static void main( String[] args ) {
        HashMap hMap = new HashMap();

        String key1 = "1";
        String key2 = "2";

        String value1 = "Hello Java";
        String value2 = "Java World";

        hMap.put( key1, value1 );
        hMap.put( key2, value2 );

        System.out.println( hMap.get( key1 ) );
        System.out.println( hMap.get( "2" ) );

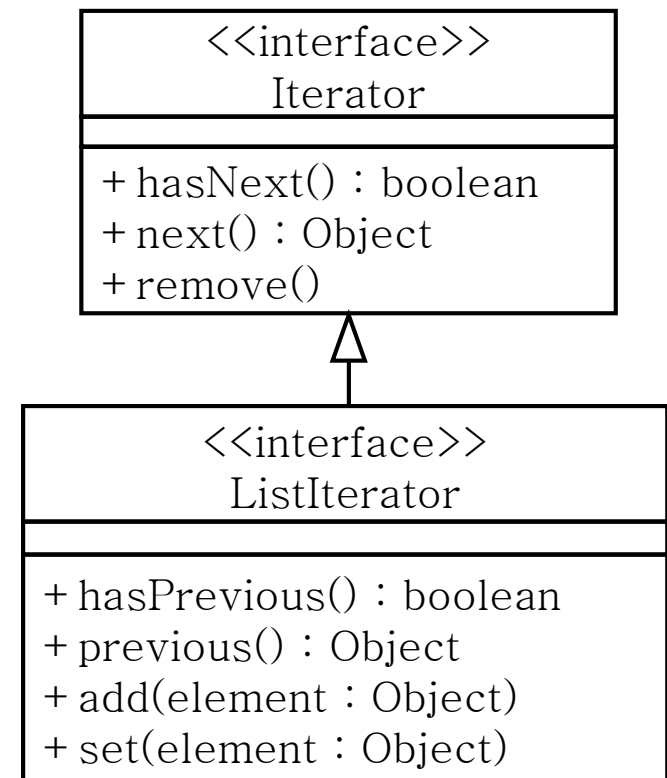
        hMap.put( key1, value2 );
        System.out.println( hMap.get( key1 ) );
    }
}
```

❑ Iterator

- Collection 객체들의 내용을 순차적으로 접근 하는 기능 제공
- Set, List 계열에서 사용가능

❑ ListIterator

- Collection 객체들의 내용을 순차적 및 역방향으로도 접근 하는 기능 제공
- List 계열에 사용 가능



```
import java.util.*;

public class IteratorTest {
    public static void main( String[] args ) {
        List list = new ArrayList();
        list.add( "add" );
        list.add( "second" );
        list.add( "3rd" );

        Iterator it = list.iterator();

        System.out.println( "=>Iterator 사용 " );

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

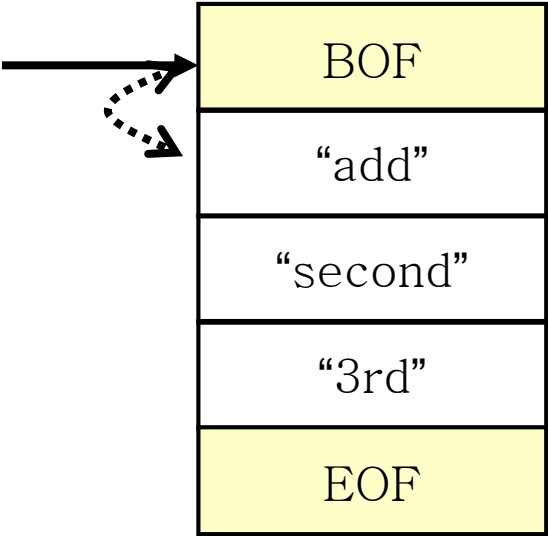
        ListIterator listIt = list.listIterator();

        System.out.println( "=>ListIterator 사용 " );

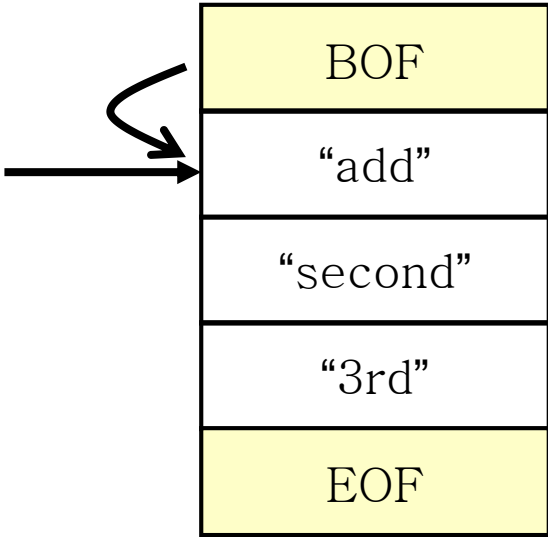
        if ( listIt.hasNext() ) {
            System.out.println( listIt.next() );
        }

        if ( listIt.hasPrevious() ) {
            System.out.println( listIt.previous() );
        }
    }
}
```

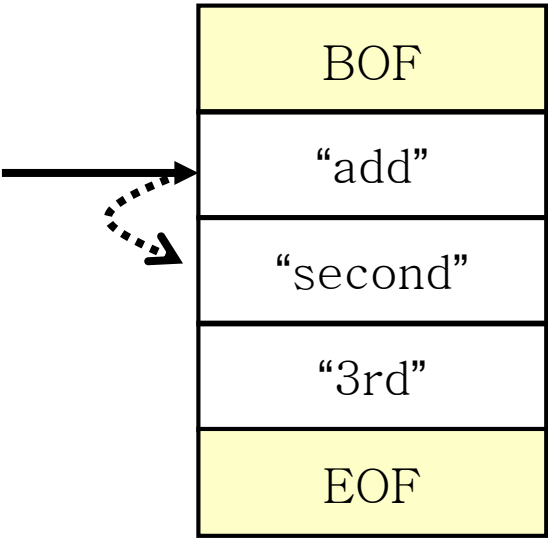
현재 가리키는 포인터
`it.hasNext()` → true



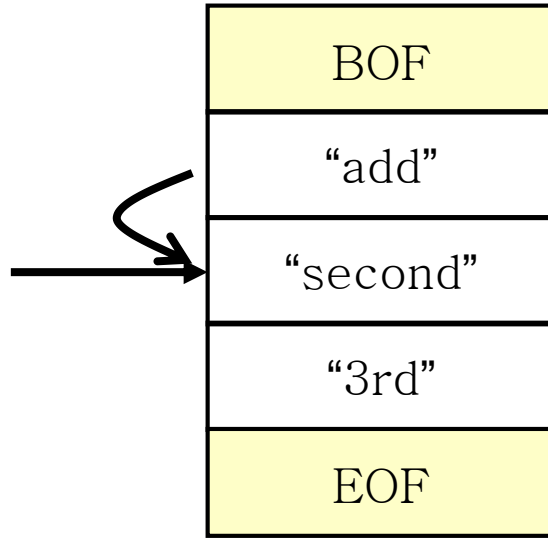
`it.next()`



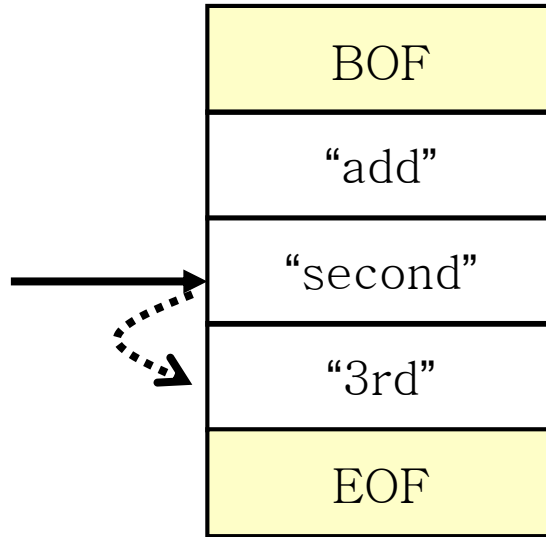
현재 가리키는 포인터
`it.hasNext()` → true



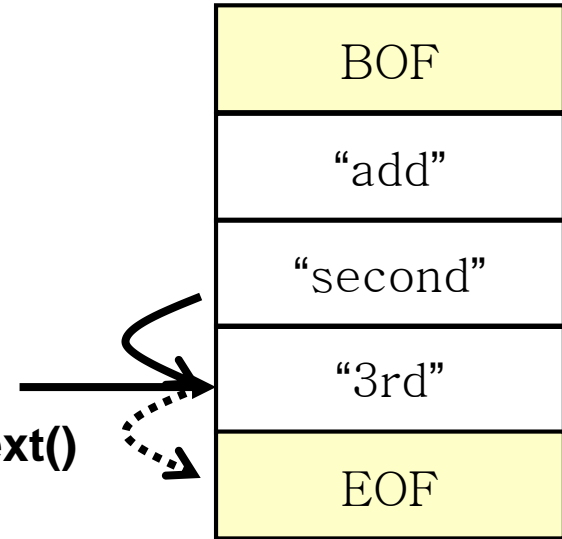
`it.next()`



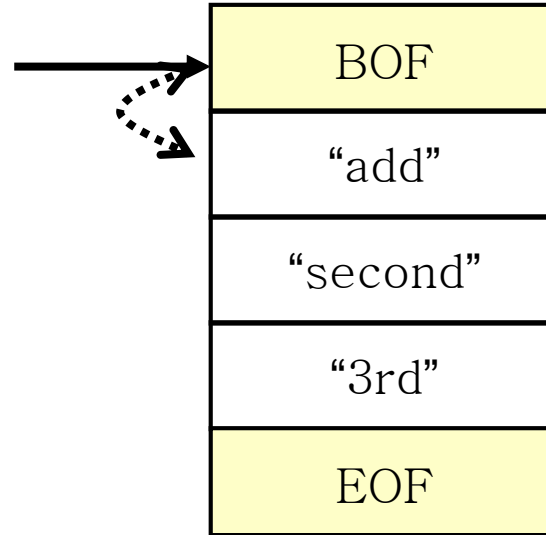
현재 가리키는 포인터
it.hasNext() → true

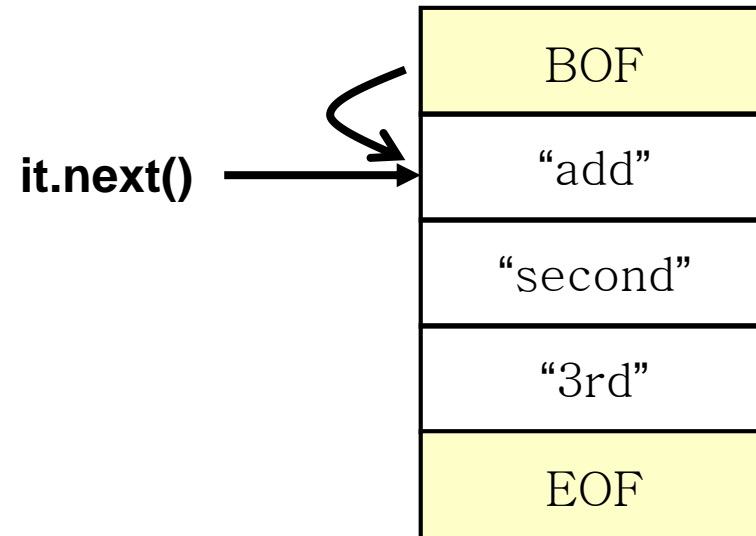


it.next()
it.hasNext()
→ false

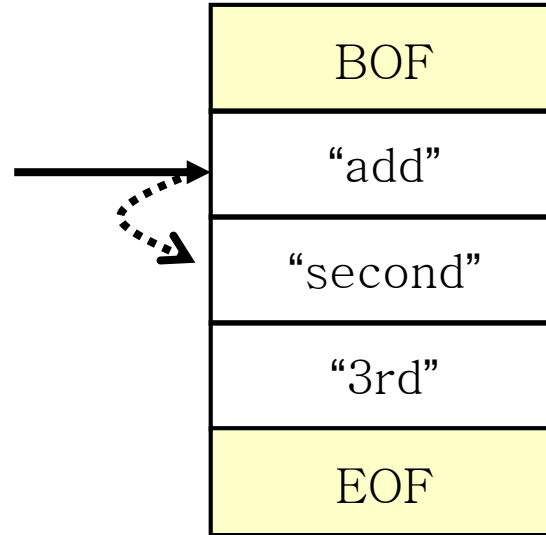


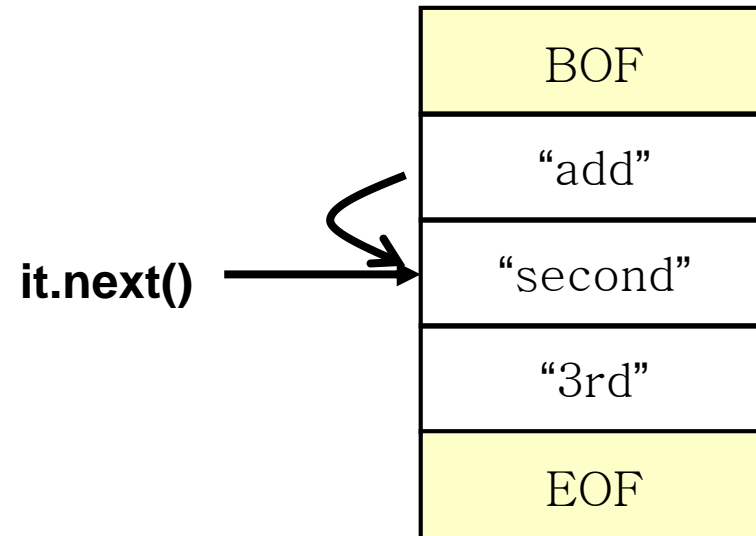
현재 가리키는 포인터
`it.hasNext() → true`

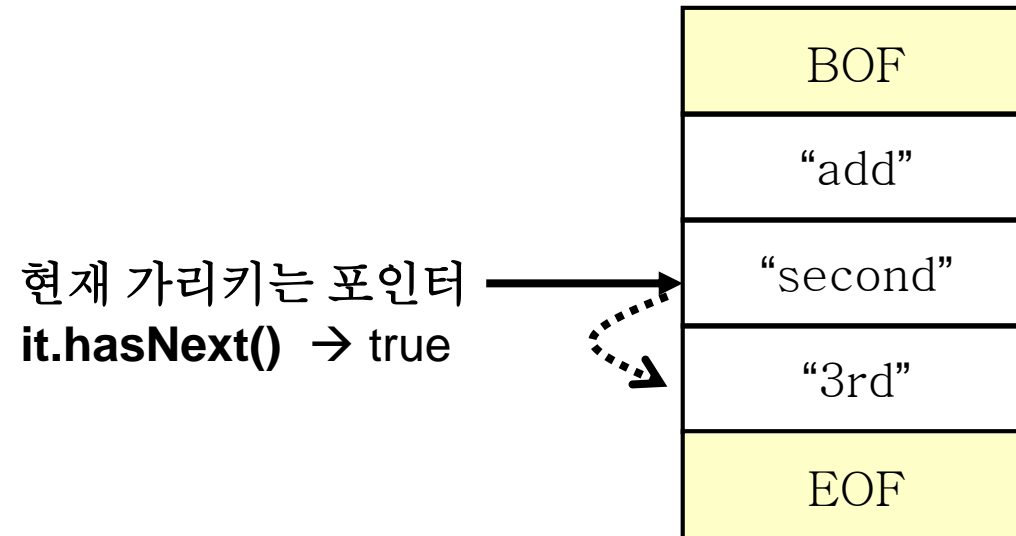


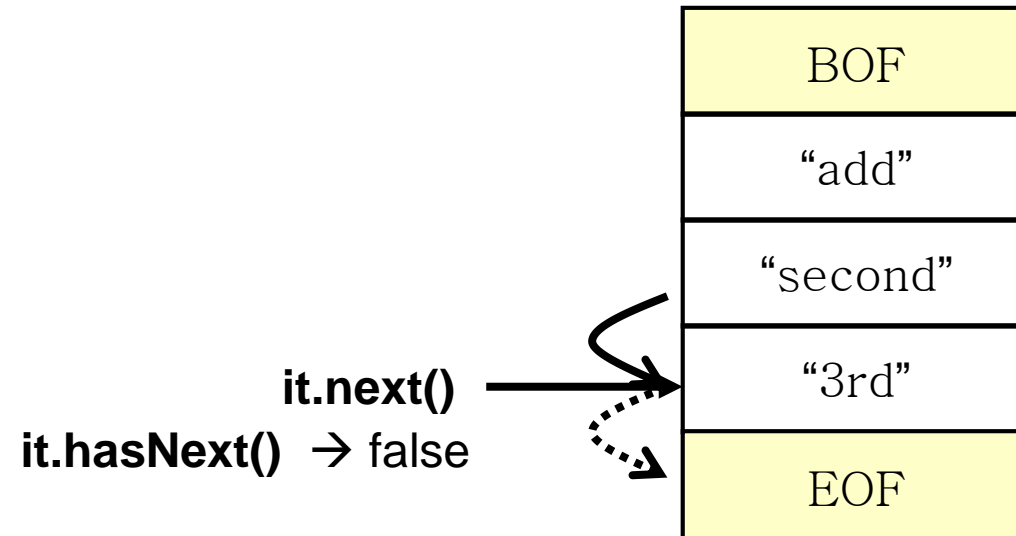


현재 가리키는 포인터
`it.hasNext() → true`









- ❑ Iterator Interface와 마찬가지로, 자바에서 제공하는 Collection 객체들에 대해, 각 Collection의 항목들을 순차적으로 접근하는 방법을 제공
- ❑ 자바 초창기 버전에서, 제공 되었던 것으로 Vector, HashTable 지원
- ❑ java.util Package에 있음
- ❑ 제공 메소드

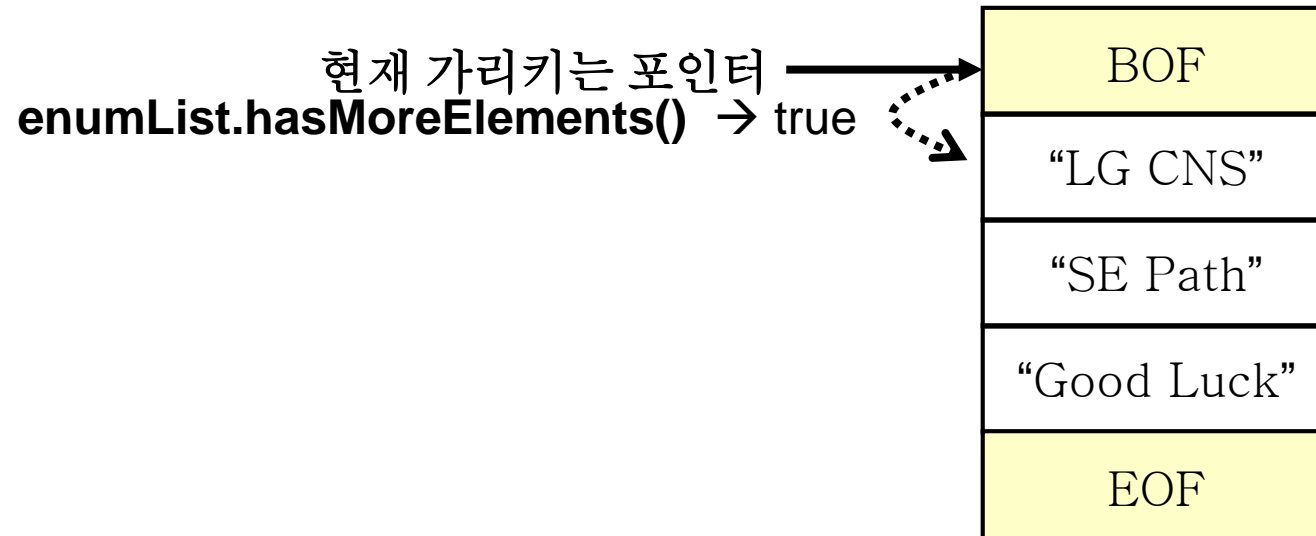
```
public boolean hasMoreElements ()
```

- 다음 항목이 있는지 검사. 있다면 true, 없으면 false return

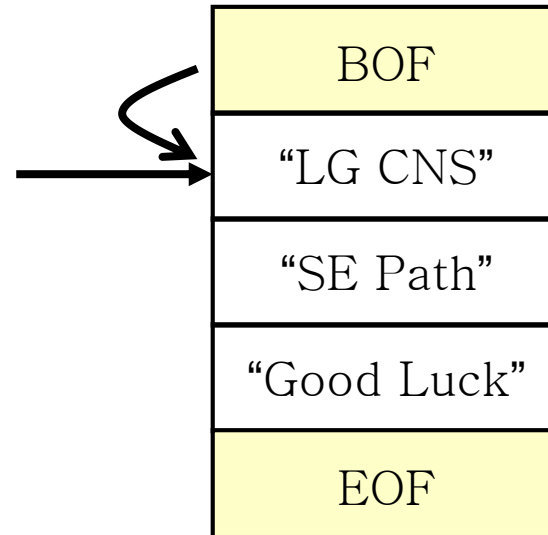
```
public Object nextElement ()
```

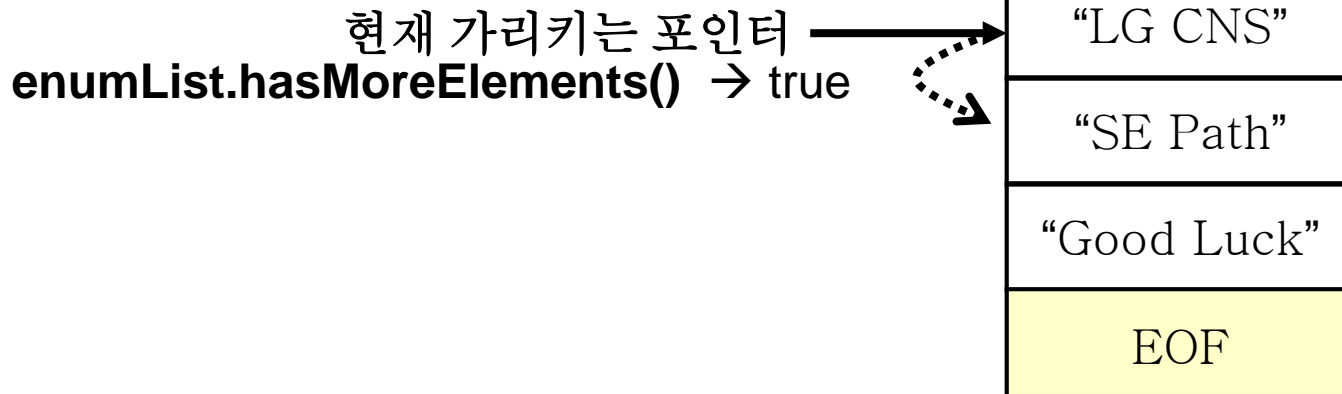
- Collection 객체에서 다음 항목을 Object 타입으로 return

```
1 import java.util.*;
2
3 public class EnumerationTest {
4
5     public static void main( String[] args ) {
6         Vector vec = new Vector();
7
8         vec.add( "LG CNS" );
9         vec.add( "SE Path" );
10        vec.add( "Good Luck!!!" );
11
12        Enumeration enumList = vec.elements();
13
14        String str = null;
15
16        while ( enumList.hasMoreElements() ) {
17            str = (String)enumList.nextElement();
18            System.out.println( str );
19        }
20    }
21 }
```

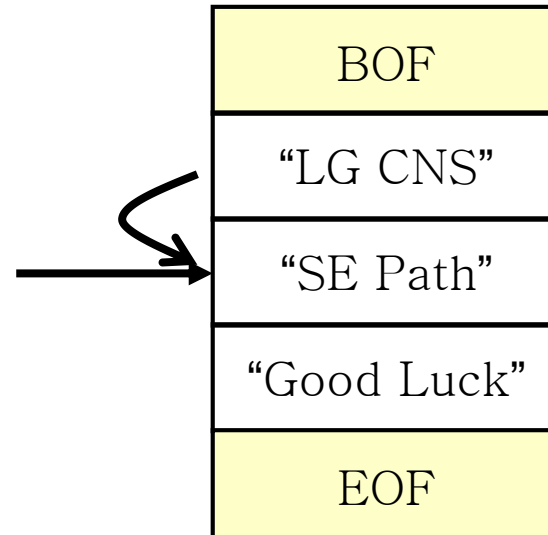


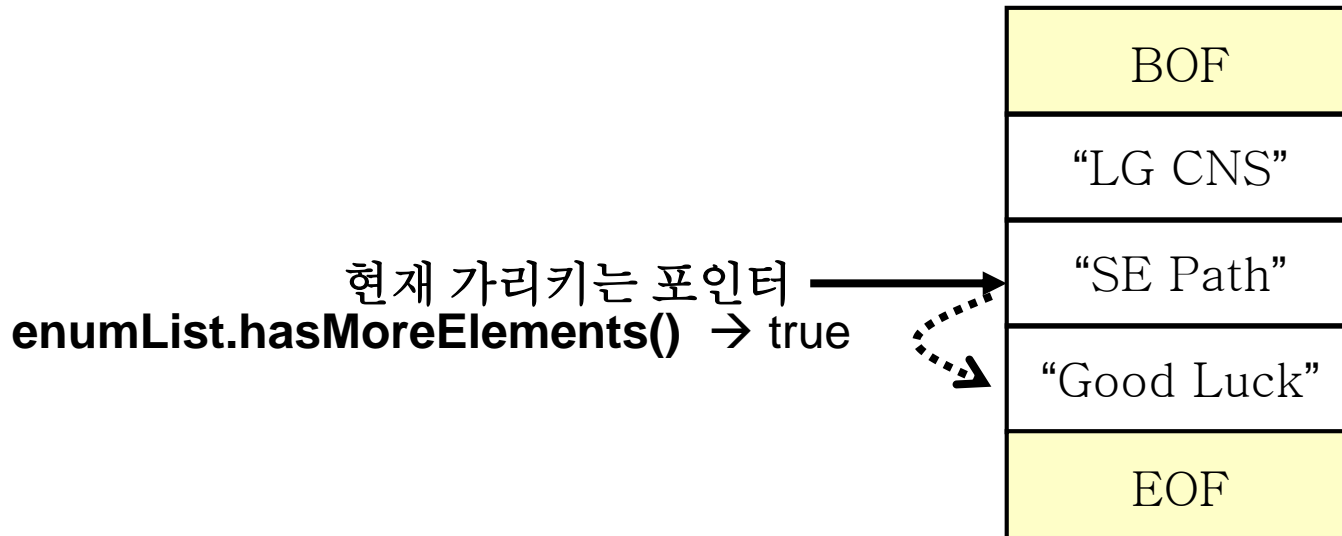
`enumList.nextElement()`

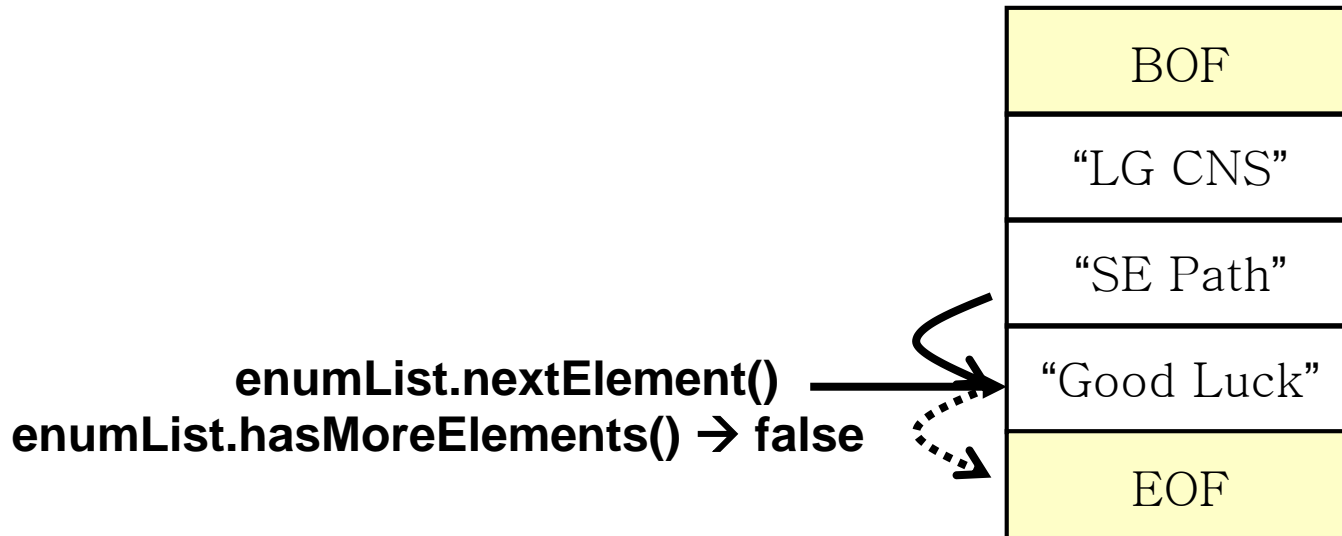




`enumList.nextElement()`





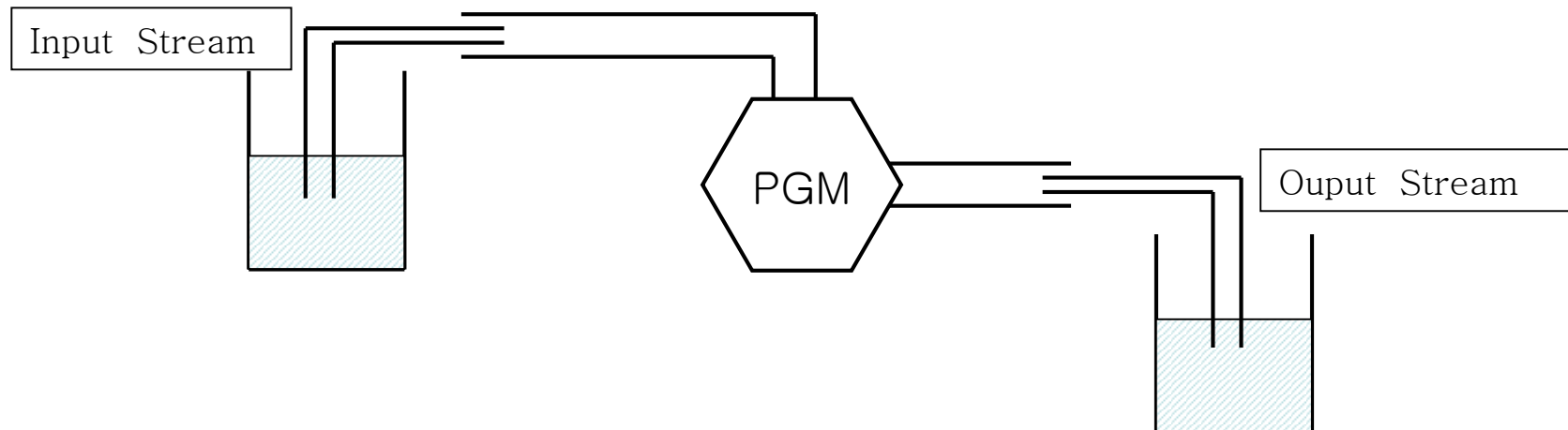


Chap 10. Advanced I/O Streams

1. Stream
2. Node Stream / Processing Stream
3. File Read/Write
4. Consol I/O
5. Object Serialization

□ Stream

- data의 연속 된 흐름을 말한다.
- data 시작(source stream/input stream)과 data의 끝 (sink stream/output stream)을 말한다.
특히 이것을 Node Stream이라고 한다.
- Node stream : file, memory, 쓰레드나 프로세스 사이의 pipe 등등
- 단방향이다.



- ❑ Byte 단위 입/출력 처리 : InputStream, OutputStream
이진 데이터 처리시 이용
- ❑ Character 단위 입/출력 처리 : Reader, Writer
text 처리시 이용

	Character Streams	Byte Streams
Source Streams	Reader	InputStream
Sink Streams	Writer	OutputStream

- Data의 근원이 되는 물리적 장치와 직접 연결하는 stream

Type	Character Streams	Byte Streams
File	FileReader FileWriter	FileInputStream FileOutputStream
Memory Array	CharArrayReader CharArrayWriter	ByteArrayInputStream ByteArrayOutputStream
Memory String	StringReader StringWriter	
Pipe	PipedReader PipedWriter	PipedInputStream PipedOutputStream

□ Data를 가공해 주는 stream

Type	Character Streams	Byte Streams
Buffering	BufferedReader BufferedWriter	BufferedInputStream BufferedOutputStream
Filtering	FilterReader FilterWriter	FilterInputStream FilterOutputStream
Converting between bytes and character	InputStreamReader OutputStreamWriter	
Object serialization		ObjectInputStream ObjectOutputStream
Data conversion		DataInputStream DataOutputStream
Counting	LineNumberReader	LineNumberInputStream
Peeking ahead	PushbackReader	PushbackInputStream
Printing	PrintWriter	PrintStream

“c:\data\text.txt”에서 문자열을 읽기 위한 스트림 작성

File 지정

```
String fileName = "c:/data/text.txt";
```

Stream 생성

3가지 방법
모두 동일

```
1) FileReader fr = new FileReader(fileName);
   BufferedReader br = new BufferedReader( fr );
2) BufferedReader br
   = new BufferedReader( new FileReader(fileName) );
3) BufferedReader br = new BufferedReader(
   new InputStreamReader(new FileInputStream(fileName)) );
```

READ

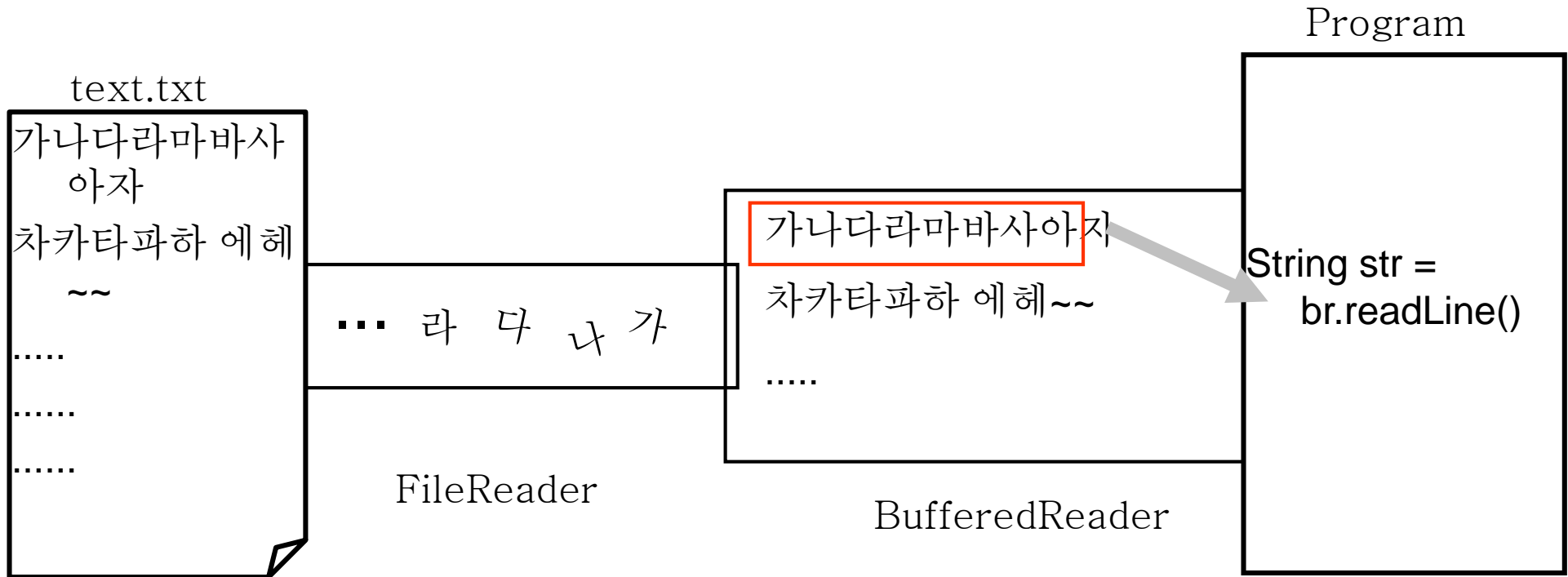
```
s = br.readLine(); // 결과의 null여부로 파일을 다 읽었는지 확인
```

Stream 닫기

```
br.close();
```

File I / O – Read

```
BufferedReader br = new BufferedReader( new FileReader("text.txt") );
```



```
import java.io.*;
public class ReadFile {
    public static void main ( String args[] ) {

        try {
            BufferedReader in
                = new BufferedReader(new FileReader("c:/data/text.txt"));
            String s;

            while ( ( s = in.readLine() ) != null ) {
                System.out.println( s );
            }

            in.close();
        } catch ( FileNotFoundException e1 ) {
            System.err.println( "File not found: " + "c:/data/text.txt" );
        } catch ( IOException e2 ){
            e2.printStackTrace();
        }
    }
}
```

“c:\Wdata\Wtext2.txt”에 문자열을 쓰기 위한 스트림 작성

File 지정

```
String file = "c:/data/text2.txt";
```

Stream 생성

2가지 방법
모두 동일

```
1) FileWriter fw = new FileWriter(fileName);
```

```
    PrintWriter pw = new PrintWriter ( fw );
```

```
2) PrintWriter pw
```

```
    = new PrintWriter( new FileWriter(fileName) );
```

READ

```
pw.println( msg[inx] );
```

Stream 닫기

```
pw.close();
```

```
import java.io.*;
public class WriteFile {
    public static void main ( String args[] ) {
        try {
            BufferedReader in
                = new BufferedReader(new FileReader("c:/data/test.txt"));
            PrintWriter out
                = new PrintWriter(new FileWriter("c:/data/test2.txt"));
            String s;
            while ( ( s = in.readLine() ) != null ) {
                out.println( s );
            }
            in.close();
            out.close();
        } catch ( IOException e ) {
            e.printStackTrace();
        }
    }
}
```

System.out

PrintStream 객체로 Terminal 참조

System.in

InputStream 객체로 키보드 참조

System.err

PrintStream 객체로 Terminal 참조

- ❑ 표준 출력은 콘솔(모니터) 출력이며, System.out의 메소드를 사용한다.
 - println() : 출력 후, new line
 - print() : new line 없이 출력
 - print() 와 println()은 다양한 타입을 화면에 출력하기 위해 overload 되어 있음

```
import java.io.*;
public class KeyboardInput {
    public static void main ( String [] args ) {
        String s;
        InputStreamReader ir = new InputStreamReader( System.in );
        BufferedReader in = new BufferedReader( ir );
        System.out.println( "Type 'exit' to exit." );
        try {
            while ( !(s = in.readLine()).equals("exit") ) {
                System.out.println( "Read: " + s );
            }
            in.close();
        } catch ( IOException e ) {
            e.printStackTrace();
        }
    }
}
```

Serialization (객체 직렬화)

- ❑ 객체 메모리를 한 줄로 늘어난 byte 형태로 만드는 것
- ❑ 객체 직렬화 대상
 - 멤버 변수
 - Class variable(static), 메소드, 생성자는 Serialization 대상에서 제외
- ❑ 장점
 - 완전한 객체 데이터 보장
 - 객체를 얻는 즉시 사용
- ❑ 관련 Stream (Processing Stream)
 - ObjectOutputStream : Unserialization
 - ObjectOutputStream : Serialization

Serialization 대상 클래스 선언

- ❑ Serialization 대상이 되는 객체는 반드시 implements Serializable

```
import java.io.Serializable;
public class EmpInfo implements Serializable {
    private String empId;
    private String name;

    public EmpInfo( String empId, String name ) {
        this.empId = empId;
        this.name = name;
    }
    public String getEmpId() {
        return empId;
    }
    public void setEmpId( String empId ) {
        this.empId = empId;
    }
    public String getName() {
        return name;
    }
    public void setName( String name ) {
        this.name = name;
    }
}
```

Serialization – Example

```
import java.io.*;

public class SerializeTest {
    public static void main( String[] args ) {
        FileOutputStream empOutputFile = null;
        ObjectOutputStream empOutputStream = null;
        EmpInfo[] empList = { new EmpInfo( "11111", "홍길동" ),
                               new EmpInfo( "22222", "고길동" ),
                               new EmpInfo( "33333", "김길동" ), };

        try {
            empOutputFile = new FileOutputStream("d:/data/emplist.ser");
            empOutputStream = new ObjectOutputStream( empOutputFile );
            empOutputStream.writeObject( empList );
            empOutputStream.close();
        } catch ( IOException ioe ) {
            ioe.printStackTrace();
        }
    }
}
```

Unserialization – Example

```
import java.io.*;
public class SerializeTest {
    public static void main( String[] args ) {
        FileInputStream empInputFile = null;
        ObjectInputStream empInputStream = null;
        try {
            empInputFile = new FileInputStream("d:/data/emplist.ser");
            empInputStream = new ObjectInputStream( empInputFile );
            EmpInfo[] readList = (EmpInfo[])empInputStream.readObject();
            empInputStream.close();

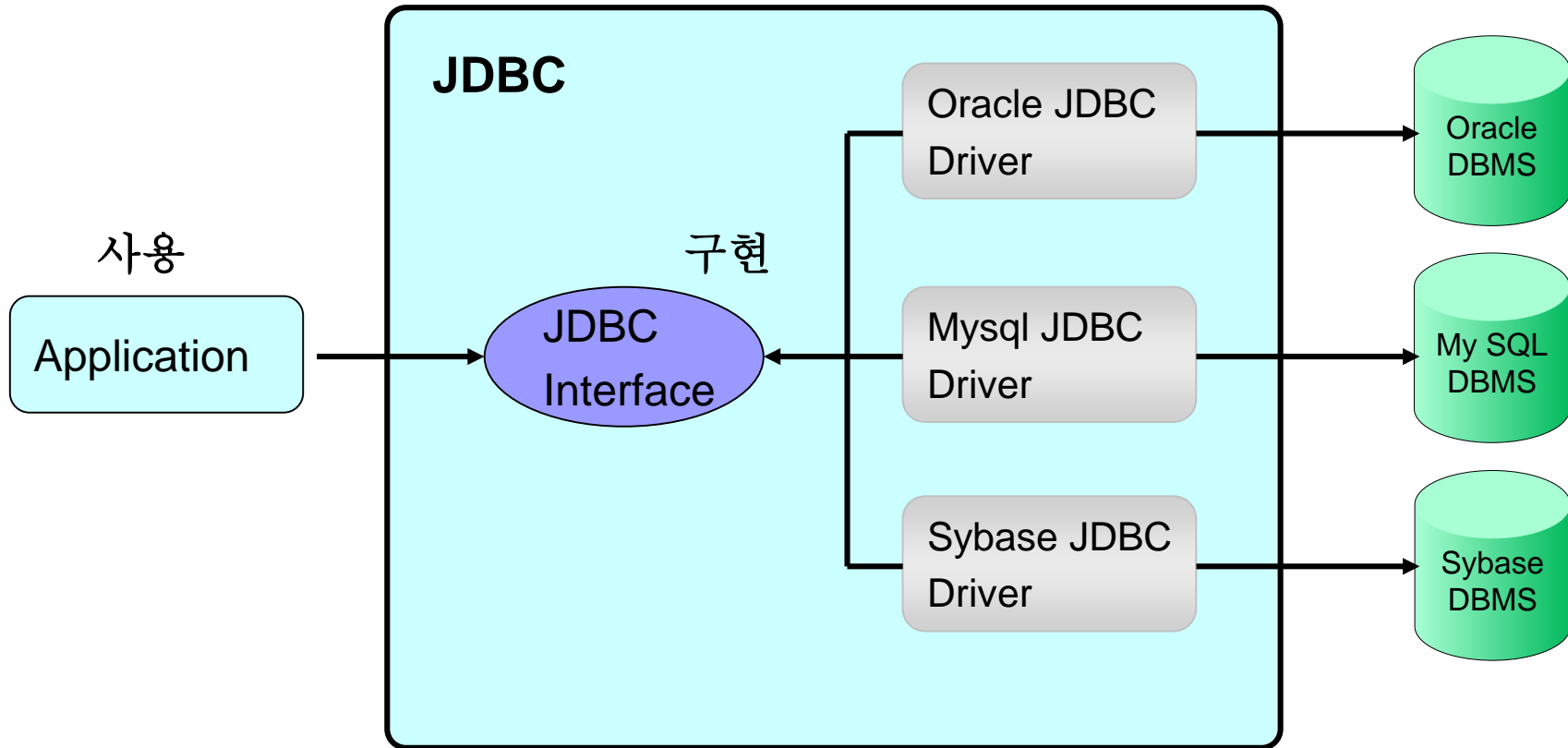
            for ( int inx = 0 ; inx < readList.length ; inx++ ) {
                System.out.println( "사번 : " + readList[inx].getEmpId() +
                                    " 이름 : " + readList[inx].getName() ) ;
            }
        } catch ( ClassNotFoundException ce ) {
            ce.printStackTrace();
        } catch ( IOException ioe ) {
            ioe.printStackTrace();
        }
    }
}
```

11. Java Programming with JDBC

1. JDBC 개요
2. JDBC Coding 절차
3. SELECT / UPDATE
4. Statement/ PreparedStatement

JDBC (Java Database Connectivity)

- 자바 언어에서 Database에 접근할 수 있게 해주는 Programming API (java.sql, javax.sql 패키지)



JDBC Driver Download – Oracle (www.oracle.com)

Database 12c

Database In-Memory

Multitenant

More Key Features

Application Development

Big Data Appliance

Cloud Database Services

Private Database Cloud

Data Warehousing & Big Data

Database Appliance

Exadata Database Machine

High Availability

Manageability

Migrations

Security

Unstructured Data

Upgrades

Windows

Database Technology Index

Oracle Database 11g Release 2 JDBC Drivers

Thank you for accepting the OTN License Agreement; you may now download this software.

Oracle Database 11g Release 2 (11.2.0.4) JDBC Drivers

SimpleFAN

↓ [simplefan.jar](#) (20,365 bytes) - (SHA1 Checksum:
307a7e203d7e141964158d181ca849d512d7e710)
Classes for subscribing to RAC events via ONS; simplefan policy and javadoc

JDBC Thin for All Platforms

↓ [ojdbc.policy](#) (10,591 bytes) - Sample security policy file for Oracle Database JDBC drivers

↓ [JavaDoc](#) (6,415,512 bytes)

↓ [README](#)

↓ [ojdbc6.jar](#) (2,739,670 bytes) - (SHA1 Checksum:
a483a046eee2f404d864a6ff5b09dc0e1be3fe6c)
Certified with JDK 8, JDK 7 and JDK 6: It contains the JDBC driver classes except classes for NLS support in Oracle Object and Collection types.

↓ [ojdbc6_g.jar](#) (4,494,956 bytes) - (SHA1 Checksum:
bf50af31967911af63058a6e1e5249c2dae34823)
Same as ojdbc6.jar except compiled with "javac -g" and contains tracing code.

↓ [ojdbc6dms.jar](#) (3,350,769 bytes) - (SHA1 Checksum:
d268a890a9a681cf498a9fe9c47e92ca06ac26f0)
Same as ojdbc6.jar, except that it contains instrumentation to support DMS and limited java.util.logging calls.



Critical Capabilities for
Operational Database
Management Systems

[Read Gartner's
Report >](#)



Oracle Live SQL

Learn and Share SQL

[Try Now >](#)

❑ ojdbc6.jar 를 클래스패스에 추가해 주어야 합니다.

JDBC Driver Download – MySql (www.mysql.org)

The screenshot shows the MySQL website's Developer Zone. The 'Downloads' link in the navigation menu is highlighted with a red box. Below it, the 'Download Connector/J 5.0' link is also highlighted with a red box. The page content includes a note about the GPL license, a link to the latest production version, and a table of download links for Source and Binaries (tar.gz and zip) for version 5.0.0-beta. The table also includes MD5 checksums and links to pick a mirror and view signatures. On the right side, there are related pages and a section to learn about new MySQL releases.

http://www.mysql.org/ – Microsoft Internet Explorer

파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)

주소(D) http://www.mysql.org/ 이동

MySQL.com MySQL Network Developer Zone Partners Online Shop

Downloads Documentation Forums Lists Bugs Events User Groups Guilds Blogs Support Resources Books

FAQ

Download Connector/J 5.0

Overview Database Server Cluster MaxDB Migration Toolkit Administrator Query Browser Co

Mirror Sites

NOTE: By downloading the software from this page, you acknowledge that the software available from here is licensed under the GPL. We advise that you review the [GPL](#) before downloading.

If you need commercial, non-GPL, licenses, you can order them [online](#).

[MySQL Connector/J](#) is the official JDBC driver for MySQL. On this page are downloads of the current development release, and [the latest production version](#) is available on another page. A [list of changes](#) is available in the documentation.

We suggest that you [use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download](#).

Source and Binaries (tar.gz)	5.0.0-beta	8.0M	Pick a mirror
MD5: 8b673dde79ba5539f49183d2a410b1fb Signature			
Source and Binaries (zip)	5.0.0-beta	8.1M	Pick a mirror
MD5: 3b557d6f53b3534305f400110c00a1 Signature			

Related pages:

- [Product Information](#)
- [Documentation](#)
- [Connector/J 3.1 Downloads](#)
- [Connector/J 3.0 Downloads](#)

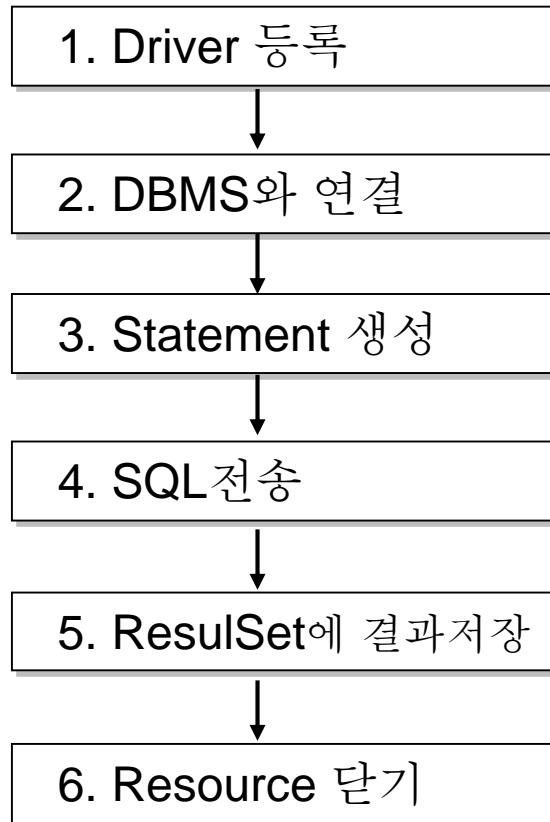
Check out the [Pogo Linux DataWare 2600](#): a premium out-of-the-box database solution with superior performance at a low price-point.

Learn about new MySQL releases, technical articles, events and more.

Subscribe to the monthly MySQL Newsletter

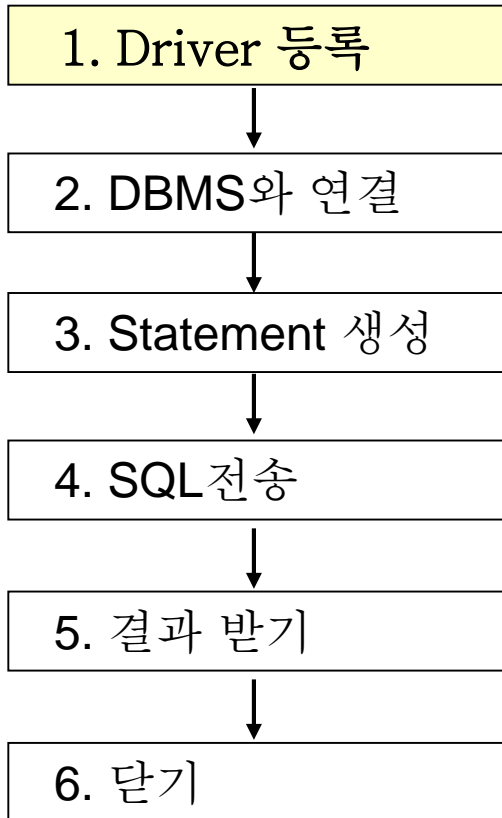
알 수 없는 영역 (혼합)

JDBC – JDBC Coding 절차



JDBC – JDBC Coding 절차

1. DriverManager에 해당 DBMS Driver 등록



```
Class.forName( "oracle.jdbc.driver.OracleDriver" );
```

```
cf)  
Class.forName( "com.microsoft.jdbc.sqlserver.SQLServerDriver" );  
  
Class.forName( "org.gjt.mm.mysql.Driver" );
```

JDBC – JDBC Coding 절차

2. 해당 Driver로부터 Connection instance를 획득

1. Driver 등록



2. DBMS와 연결



3. Statement 생성



4. SQL전송



5. 결과 받기



6. 닫기



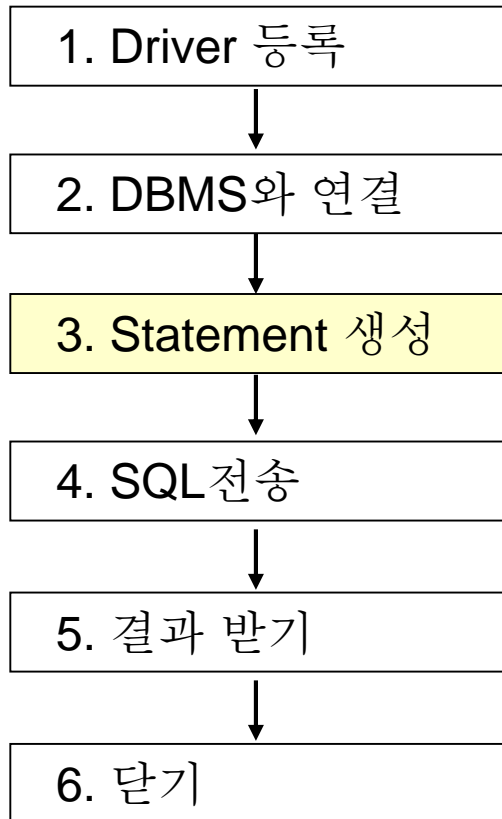
```
public static Connection getConnection( String url,  
                                       String user,  
                                       String password )
```

throws SQLException

```
Connection conn =  
    DriverManager.getConnection(  
        "jdbc:oracle:thin:@192.168.0.200:1521:XE",  
        "SEXXXXX",  
        "SEXXXXX" );
```

JDBC – JDBC Coding 절차

3. Connection instance로부터 Statement instance 획득

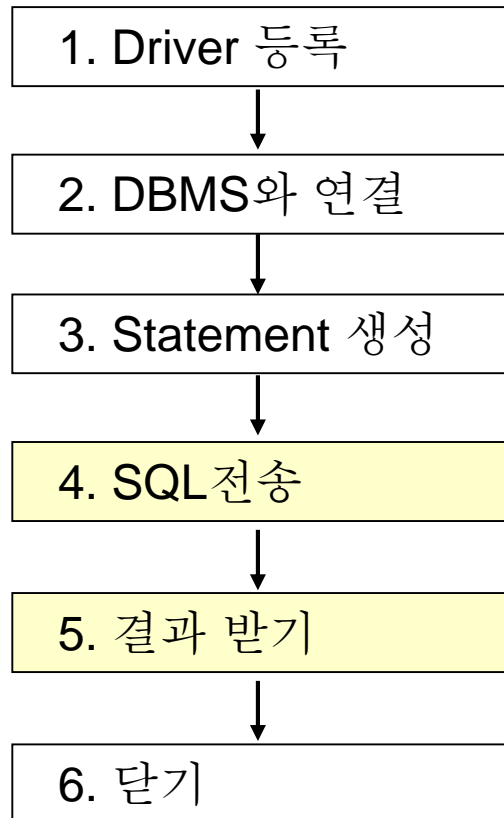


```
Statement stmt = conn.createStatement();
```

JDBC – JDBC Coding 절차

4. Statement method를 이용하여 SQL 실행

5. 실행후 결과를 ResultSet(SELECT) 혹은 int형 변수(DML)로 받아 처리



DriverManager

Connection

Statement

ResultSet

Select

```
String query = "SELECT ID, LAST_NAME FROM EMP";  
ResultSet rset = stmt.executeQuery( query );  
  
while ( rset.next() ) {  
    System.out.println( rset.getString( "ID" ) + "\t" +  
                        rset.getString( 2 ) );  
}
```

DML

```
String query = "UPDATE EMP " +  
              " SET LAST_NAME = 'KIM' "+  
              " WHERE ID = '100000' ";  
int result = stmt.executeUpdate( query );
```


JDBC – JDBC Coding 절차

6. 사용한 자원 반납

1. Driver 등록



2. DBMS와 연결



3. Statement 생성



4. SQL전송



5. 결과 받기



6. 닫기

Select

```
rset.close();  
stmt.close();  
conn.close();
```

DML

```
stmt.close();  
conn.close();
```

JDBC – SELECT

```
package jdbc;

import java.sql.*;

public class EmpList {

    public static void main( String[] args )
        throws SQLException, ClassNotFoundException {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rset = null;
        String url = "jdbc:oracle:thin:@192.168.0.200:1521:XE";
        String query = null;

        // 1. DBMS Driver 로딩
        Class.forName( "oracle.jdbc.driver.OracleDriver" );

        // 2. Connection 객체 획득
        conn = DriverManager.getConnection( url , "SEXXXXXX" , "SEXXXXXX" );

        // 3. Statement 객체 생성
        stmt = conn.createStatement();

        // 4. SQL 실행
        query = "SELECT ID      " +
            "      ,LAST_NAME " +
            " FROM EMP " ;
```

JDBC – SELECT


```
rset = stmt.executeQuery( query.toString() );

System.out.println( "IDWtWtWtLAST_NAMEWn" );
System.out.println( "===== Wn" );

// 5. ResultSet을 이용한 결과 처리
while( rset.next() ){
    System.out.println( rset.getString( "ID" ) + "WtWtWt" +
        rset.getString( 2 ) );
}

// 6. 사용할 Resource 반납
rset.close();
stmt.close();
conn.close();
}
}
```

JDBC – SELECT



BOF	ID	LAST_NAME
ROW 1	10001	
ROW 2	10002	BOSS
ROW 3	10003	JACKSON
		HITE
...
EOF		

JDBC – SELECT

rs.next() // true 리턴



```
String id = rset.getString( "ID" );  
String lastName = rset.getString( 2 );
```

ID = 10001
LAST_NAME = BOSS

	1	2
BOF	ID	LAST_NAME
ROW 1	10001	
ROW 2	10002	BOSS
ROW 3	10003	JACKSON
		HITE
...
EOF		

JDBC – SELECT

rs.next() // true 리턴

```
String id = rset.getString( "ID" );  
String lastName = rset.getString( 2 );
```

ID = 10002
LAST_NAME = JACKSON

	1	2
BOF	ID	LAST_NAME
ROW 1	10001	
ROW 2	10002	BOSS
ROW 3	10003	JACKSON
		HITE
...
EOF		

JDBC – SELECT

rs.next() // false 리턴

BOF	ID	LAST_NAME
ROW 1	10001	
ROW 2	10002	BOSS
ROW 3	10003	JACKSON
		HITE
...
EOF		

□ StringBuffer Class 사용

```
StringBuffer query = new StringBuffer();
try{
    // 1. DBMS Driver 로딩
    Class.forName( "oracle.jdbc.driver.OracleDriver" );

    ....
    // 4. SQL 실행
    query.append( "SELECT ID          " )
           .append( "          , LAST_NAME " )
           .append( "FROM EMP          " );
    rs = stmt.executeQuery( query.toString() );
    ....
} catch( ClassNotFoundException ce){
    ce.printStackTrace();
} catch( SQLException se){
    se.printStackTrace();
} finally {
    // 6. 사용할 Resource 반납
    try {
        rs.close();
        stmt.close();
        conn.close();
    } catch ( SQLException e ) {
        e.printStackTrace();
    }
}
```


❑ Exception Handling 로직 추가

```
StringBuffer query = new StringBuffer();
try{
    // 1. DBMS Driver 로딩
    Class.forName( "oracle.jdbc.driver.OracleDriver" );

    ....
    // 4. SQL 실행
    query.append( "SELECT ID          " )
           .append( "          , LAST_NAME " )
           .append( "FROM EMP          " );
    rs = stmt.executeQuery( query.toString() );
    ....
} catch( ClassNotFoundException ce){
    ce.printStackTrace();
} catch( SQLException se){
    se.printStackTrace();
} finally {
    // 6. 사용할 Resource 반납
    try {
        rs.close();
        stmt.close();
        conn.close();
    } catch ( SQLException e ) {
        e.printStackTrace();
    }
}
```

JDBC – UPDATE Example

```
public class UpdateTest {  
    public static void main( String[] args ) throws SQLException, ClassNotFoundException {  
        Connection conn = null;  
        Statement stmt = null;  
        String url = "jdbc:oracle:thin:@192.168.0.200:1521:XE";  
        StringBuffer query = new StringBuffer();  
        int updateCount = 0;  
  
        Class.forName( "oracle.jdbc.driver.OracleDriver" );  
        conn = DriverManager.getConnection( url , "SEXXXXXX" , "SEXXXXXX" );  
        conn.setAutoCommit( false );  
  
        query.append( "UPDATE EMP          " )  
            .append( "SET LAST_NAME = 'HITE'    " )  
            .append( "WHERE ID = '10004'        " );  
        stmt = conn.createStatement();  
  
        updateCount = stmt.executeUpdate( query.toString() );  
        System.out.println( "업데이트된 행의 갯수 : " + updateCount );  
  
        if( updateCount == 1 ){  
            conn.commit();  
        }else{  
            conn.rollback();  
        }  
        stmt.close();  
        conn.close();  
    }  
}
```

JDBC – PreparedStatement

```
public class PreparedUpdateTest {  
    public static void main( String[] args ) throws SQLException, ClassNotFoundException {  
        Connection conn = null;  
        PreparedStatement pstmt = null;  
        String url = "jdbc:oracle:thin:@192.168.0.200:1521:VCC";  
        StringBuffer query = new StringBuffer();  
        int updateCount = 0;  
  
        Class.forName( "oracle.jdbc.driver.OracleDriver" );  
        conn = DriverManager.getConnection( url , "SEXXXXX" , "SEXXXXX" );  
        conn.setAutoCommit( false );  
        query.append( "UPDATE EMP " )  
            .append( "SET LAST_NAME = ? " )  
            .append( "WHERE ID = ? " );  
  
        pstmt = conn.prepareStatement( query.toString() );  
        pstmt.setString( 1, "HITE2" );  
        pstmt.setString( 2, "10005" );  
  
        updateCount = pstmt.executeUpdate();  
        if( updateCount == 1 ){  
            conn.commit();  
        }else{  
            conn.rollback();  
        }  
        pstmt.close();  
        conn.close();  
    }  
}
```



JDBC – Statement vs PreparedStatement

	Statement	PreparedStatement
장점	원하는 Query를 직접 넣어주기 때문에 직관적이고 사용하기 쉽다.	같은 Query를 반복 수행해야 하는 경우 성능이 좋다. (loop 이용이 용이)
단점	실행시마다 SQL문을 해석해서 오버헤드가 크다.	코드가 길어질 수 있다.
Sample	<pre>Statement stmt = conn.createStatement(); stmt.executeUpdate("Insert into emp values ('21421', 'Kim')"); stmt.executeUpdate("Insert into emp values ('32211', Hong)"); ...</pre>	<pre>PreparedStatement pstmt = conn.prepareStatement(" Insert into emp values (?, ?) "); pstmt.setString(1, "21421"); pstmt.setInt(2, "Kim"); pstmt.executeUpdate(); pstmt.setString(1, "32211"); pstmt.setInt(2, "Hong"); pstmt.executeUpdate(); ...</pre>