

# JAVA Basic, Fundamental 과정

자바/스프링 기반 디지털 융합 웹 개발자 양성과정

지도교수: 박인상

# 자바/스프링 기반 디지털 융합 웹 개발자 양성과정

---

1. JDK, Eclipse, Tomcat
  2. JAVA 1 (data type, operator, 제어문, array, class, object, OOP, package)
  3. JAVA 2 (access control, static, inheritance, abstract class, interface, API)
  4. JAVA 3 (exception handling, collections, I/O, multi-thread, Network)
  5. JSP & Servlet (DB, member, EL, JSTL, controller, DO(VO), DAO, pattern, forwarding)
  6. Spring (DI, life cycle & scope, AOP, MVC, form data, JDBC, transaction, Mybatis)
  7. 프로젝트
-

# 자바의 특징

---

- 자바가 프로그래밍 언어임에도 단순한 언어의 차원을 넘어 자바 컴퓨팅(Java Computing)으로 까지 불려지고 있는 것은 막강한 API(Application Programming Interface)와 오픈소스 라이브러리가 있기 때문이다.
  - 이식성이 높은 언어
  - 객체지향 언어
  - 함수적 스타일(functional-style) 코딩을 지원
  - 메모리 자동관리
  - 풍부한 기능을 제공하는 오픈 소스 - 자바 개발 키트(JDK)
  - 멀티 스레드(Multi-Thread)를 쉽게 구현
-

# 자바 언어 학습 로드맵

자바 기초



자바 중급



자바 활용



자바 실무

- 자바 개발 환경 구축
- 변수명 만들기
- 자료형 이해
- 연산자의 종류와 우선순위
- 제어문 이해
- 배열 이해
- 컬렉션 프레임워크
- 열거형 Enum 이해
- 예외처리 이해

- 기본 API 클래스
- 정렬 알고리즘 구현
- 검색 알고리즘 구현
- 객체지향 개념 터득
- 클래스
- this, this() 차이점
- super, super() 차이점
- 열거형 Enum 응용
- 인터페이스
- 랴다식 이해
- 스트림 이해
- 멀티 스레드 이해

- 랴다식 응용, 활용
- 스트림 응용, 활용
- 웹 크롤링, 파싱
- IO 기반 입출력 및 네트워킹
- NIO 기반 입출력 및 네트워킹
- JDBC API 활용

- 자바 어플리케이션 개발
- 고객관리 시스템 구현
- 자바 오라클 연동
- 자바를 Web에서 활용법
- 자바를 적용한 안드로이드 앱 개발

## FRONT - END

User Device



**Maven**<sup>™</sup>

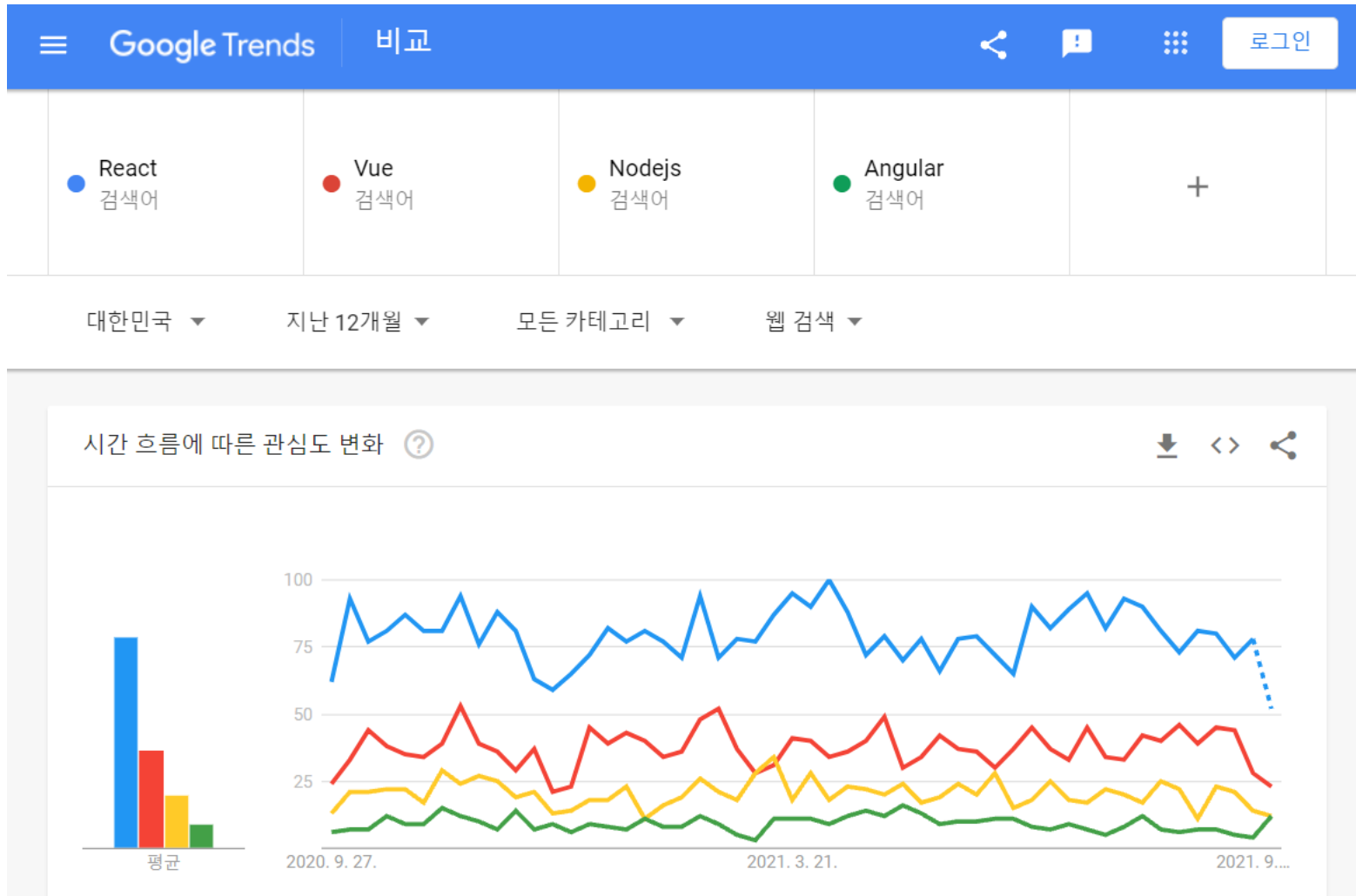


React

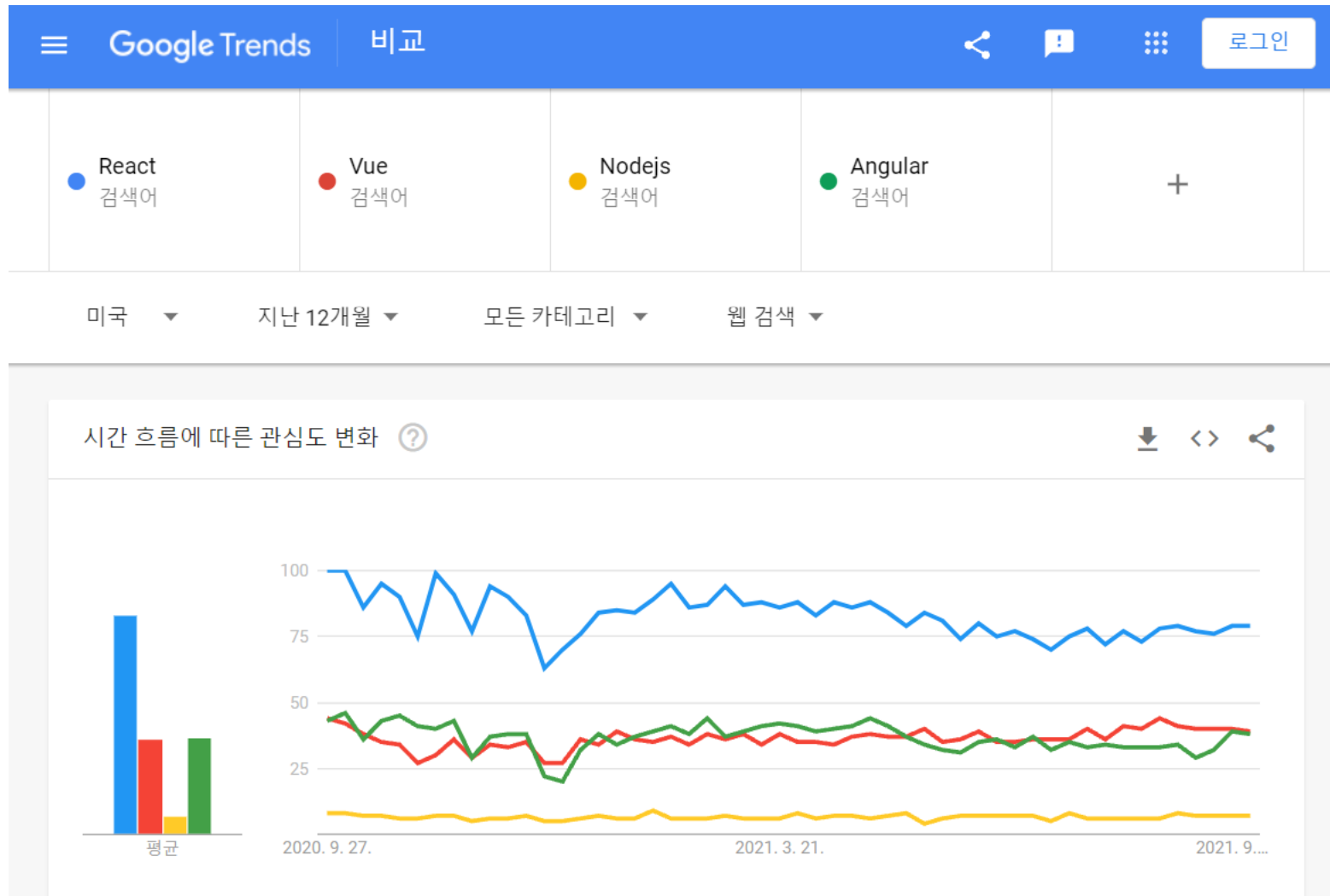


Vue.js

<https://trends.google.co.kr/trends/explore?geo=KR&q=React,Vue,Nodejs,Angular>



<https://trends.google.co.kr/trends/explore?geo=US&q=React,Vue,Nodejs,Angular>



## BACK - END

Admin Device



**MyBatis**

**ORACLE®**

**DATABASE**



**JSOUP**



Apache Tomcat

**Maven™**

**You Tube**

Data API v3



## 4차 산업혁명 시대의 생존법

문제만 찾아내면 답은 어떤 식으로든  
구할 수 있다!



## 4차 산업혁명 시대의 생존법

**‘남의 플랫폼에 올라타라!!!!’**

[사례]

- IBM 왓슨 IoT 플랫폼, AWS IoT 기반 아키텍처 구현 활용

사물 인터넷(IoT) 디바이스에서 가치를 간편하게 도출하도록 설계된 관리형 클라우드 서비스이다.

- 구글 클라우드 플랫폼을 활용

코랩(Colab) : 학생이든, 데이터 과학자든, AI 연구원이든 Colab으로 업무를 더욱 간편하게 처리할 수 있다.

- 삼성 SDS의 AI 기반 분석 플랫폼을 활용

Brightics Studio : (마케팅 산업, 산업 동향, 금융 산업) 빅데이터 분석

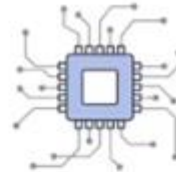
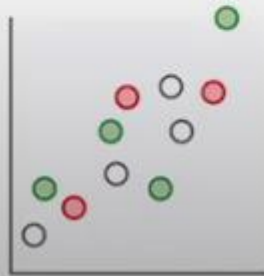
# 데이터 생성과 접근이 빠르게 변화



사람이 생성한 데이터

산발적인 상호작용

세상에 대한 **부분적인** 시각



기계가 생성한 데이터

지속적인 상호작용

세상에 대한 **전체적인** 시각



# 자바 개발 환경 구축

---

1. 자바 JDK 설치
2. Eclipse 설치

[참고]

**JDK**(자바 개발 키트)는 자바 프로그램을 개발하는 필요한 라이브러리와 플랫폼이 포함되어 있다. 자바 프로그램을 만들기 위해서는 반드시 JDK가 있어야 한다.

자바 프로그램을 실행하기 위해서는 **JRE**(자바 실행 환경)가 필요하다.

**이클립스**(Eclipse)는 통합 개발 환경(IDE)으로 자바 프로그래밍을 하는 데 꼭 필요하다.

---

# 자바 버전업

---

## 자바5 (JDK 1.5) -----> 진화계

- 향상된 for문 제공
- 가변인자(Varargs) 사용 가능
- 제네릭(Generic) 타입 추가

## 자바8 (JDK 1.8) -----> 혁명계

- 인터페이스에 - 디폴트 메소드  
- 정적 메소드 선언이 가능
  - 랴다식(Lambda Expression) 지원 → 즉 함수적-스타일(functional-style) 지원  
- "함수적 프로그래밍"을 위해 랴다식을 지원하면서 기존의 코드 패턴이 많이 달라졌다.
  - 스트림(Stream) 추가  
- 컬렉션(배열 포함)의 저장 요소를 하나씩 참조해서 랴다식으로 처리할 수 있도록 해주는 반복자이다.
-

# 자바의 데이터 타입(data type)

---

## □ 기본 타입 : 8개

- 정수 타입 – byte(1), short(2), int(4), long(8)
  - 실수 타입 – float(4), double(8)
  - 문자 타입 – char(2byte, Unicode)
  - 논리 타입 – boolean(1byte, true 또는 false)
- \* 단, 기본 타입으로는 자바에서 문자열을 표현할 수 없다  
String 클래스를 이용하여 문자열을 표시할 수 있다

## □ 참조 타입 : 1개

참조 타입은 한 가지이지만 용도는 4가지이다

- 배열 타입
  - 클래스 타입
  - 인터페이스 타입
  - 열거 타입
-

# 자바 변수의 종류와 특징

\* 변수의 종류를 결정짓는 요소는 '변수의 선언된 위치'가 된다.

변수의 종류	선언위치	생성시기
클래스 변수 (공통 속성)	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스 변수 (개별 속성)		인스턴스가 생성되었을 때
지역 변수	클래스 영역 이외의 영역 (메소드, 생성자, 블록 { } 내부)	변수 선언문이 수행되었을 때

- 클래스 변수 선언 방법은 인스턴스 변수 앞에 static을 붙이면 된다. 클래스 내에 Static 키워드로 선언된 변수
- 클래스 변수는 모든 인스턴스가 공통된 저장공간(변수)을 공유하게 된다.
- 클래스 변수는 인스턴스를 생성하지 않고도 언제라도 바로 사용 가능
- 그리고 public을 앞에 붙이면 프로그램 내에서 어디서나 접근할 수 있는 '전역 변수(global variable)의 성격을 갖는다.
- 인스턴스마다 고유한 상태를 유지해야 하는 속성의 경우, 인스턴스 변수로 선언한다.

# 포커(POKER) 카드 클래스로 정의

---



## ○ 클래스의 속성 발견?


어떤 속성을 **클래스 변수**로 선언할 것이며, 또 어떤 속성들을 **인스턴스 변수**로 선언할 것인지 생각해봅시다.

크기(넓이, 높이), 무늬, 숫자 ?

---



# 연산자의 종류와 우선순위



종류	연산자	연산 방향
단항 연산자	증감(++/--), (dataType)	←
이항 연산자	산술(*, /, %)	→
	산술(+, -)	→
	쉬프트(<<, >>)	→
	비교(<, >, <=, >=, instanceof)	→
	비교(==, !=)	→
	논리곱(&&)	→
	논리합(  )	→
삼항 연산자	조건(?:)	→
대입 연산자	대입(=, +=, -=, *=, /=, %=)	←

# 제어문

---

유형	제어문
조건문	if문, if-else문, 중첩 if문, switch문
반복문	for문, while문, do-while문
분기문	break문, continue문

# 기본 API 클래스 or 인터페이스

<https://docs.oracle.com/javase/8/docs/api/> 접속

---

패키지	클래스 or 인터페이스들
java.lang 패키지	Object System <b>String</b> <b>StringBuffer</b> Wrapper(Byte, Short, Integer, Long, Character, Float, Double, Boolean)
java.util 패키지	<u><b>List, Set, Map,</b></u> Scanner, <b>ArrayList, Vector, HashMap, Hashtable, Properties</b> , Date, Calender, Random, Arrays, StringTokenizer
java.util.regex 패키지	<b>Pattern</b> , Matcher
Java.text 패키지	<b>DecimalFormat</b> , MessageFormat
java.util.stream 패키지	BaseStream, <b>Stream</b> , IntSream, LongStream. DoubleStream
java.time 패키지	LocalDate, LocalTime, LocalDateTime, ZonedDateTime, Instant
java.security 패키지	<b>MessageDigest</b> : getInstance(), update(), digest() 메소드

---

## String 클래스의 사용 빈도수가 높은 메소드

리턴 타입	메소드명(매개변수)	기능(역할)
int	<b>compareTo</b> (String anotherString)	문자열의 사전적 값을 비교하여 int값 리턴
boolean	<b>equals</b> (Object anObject)	두 문자열을 비교하여 boolean값 리턴
char	<b>charAt</b> (int index)	특정 위치의 문자 리턴
String	<b>replaceAll</b> (String regex, String replacement)	정규표현식과 대체 문자열과 일치하는, 문자의 모든 시퀀스와 대체되는 스트링 반환
String	<b>substring</b> (int beginIndex)	beginIndex 위치에서 끝까지 잘라낸 새로운 문자열 리턴
String	<b>substring</b> (int beginIndex, int endIndex)	beginIndex 위치에서 endIndex 전까지 잘라낸 새로운 문자열 리턴
int	<b>length</b> ()	총 문자의 수를 리턴
byte[]	<b>getBytes</b> (String charsetName)	특정 문자셋으로 인코딩된 바이트 배열을 리턴
boolean	<b>matches</b> (String regex)	정규표현식으로 글자의 포함여부를 체크

## String 클래스의 사용 빈도수가 높은 메소드

---

리턴 타입	메소드명(매개변수)	기능(역할)
String	<b>replace</b> ( <a href="#">CharSequence</a> target, <a href="#">CharSequence</a> replacement)	리터럴 대상 시퀀스와 일치하는이 문자열의 각 하위 문자열을 지정된 리터럴 교체 시퀀스로 바꾼다.
String[]	<b>split</b> (String regex)	입력 받은 정규표현식 또는 특정문자(문자열)를 기준으로 문자열을 나누어(쪼개어) 배열(Array)에 저장하여 리턴하는 메소드.
String	<b>format</b> (String format, Object... args)	지정된 위치에 값을 대입해서 문자열을 만들어 내는 용도로 사용
int	<b>lastIndexOf</b> (String str)	문자열에 지정한 문자가 마지막 몇 번째에 있는 문자인지 위치를 int로 반환
boolean	<b>contains</b> (CharSequence s)	특정 문자열이 포함되어 있는지 확인하는 기능을 한다. 특정 문자열이 포함되어 있다면 true를 없다면 false를 반환한다.

# 자료구조(data structure)

---

- 기본 자료형을 근간으로 여러가지 구조를 만들 수 있는데 이러한 구조를 통칭하여 자료구조라 한다
  - 자료의 접근과 처리가 용이하도록 잘 조직화된 자료의 집단
  - 데이터 값의 모임, 또 데이터 간의 관계, 그리고 데이터에 적용할 수 있는 함수나 명령을 의미한다.
-

# 알고리즘(Algorithm)

---

- 어떤 문제를 해결하는 논리적인 절차를 알고리즘이라 한다.
  - 복잡한 문제를 쉽고 효율적으로 해결하기 위해 정의된 방법과 절차
  - 주어진 문제를 해결하기 위한 동작과 순서를 정의하는 것
  
  - 중요 알고리즘
    - 정렬 알고리즘
      - 선택(selection) 정렬 알고리즘
    - 검색 알고리즘
      - 순차(sequential) 검색 알고리즘
      - 이분(binary) 검색 알고리즘
-

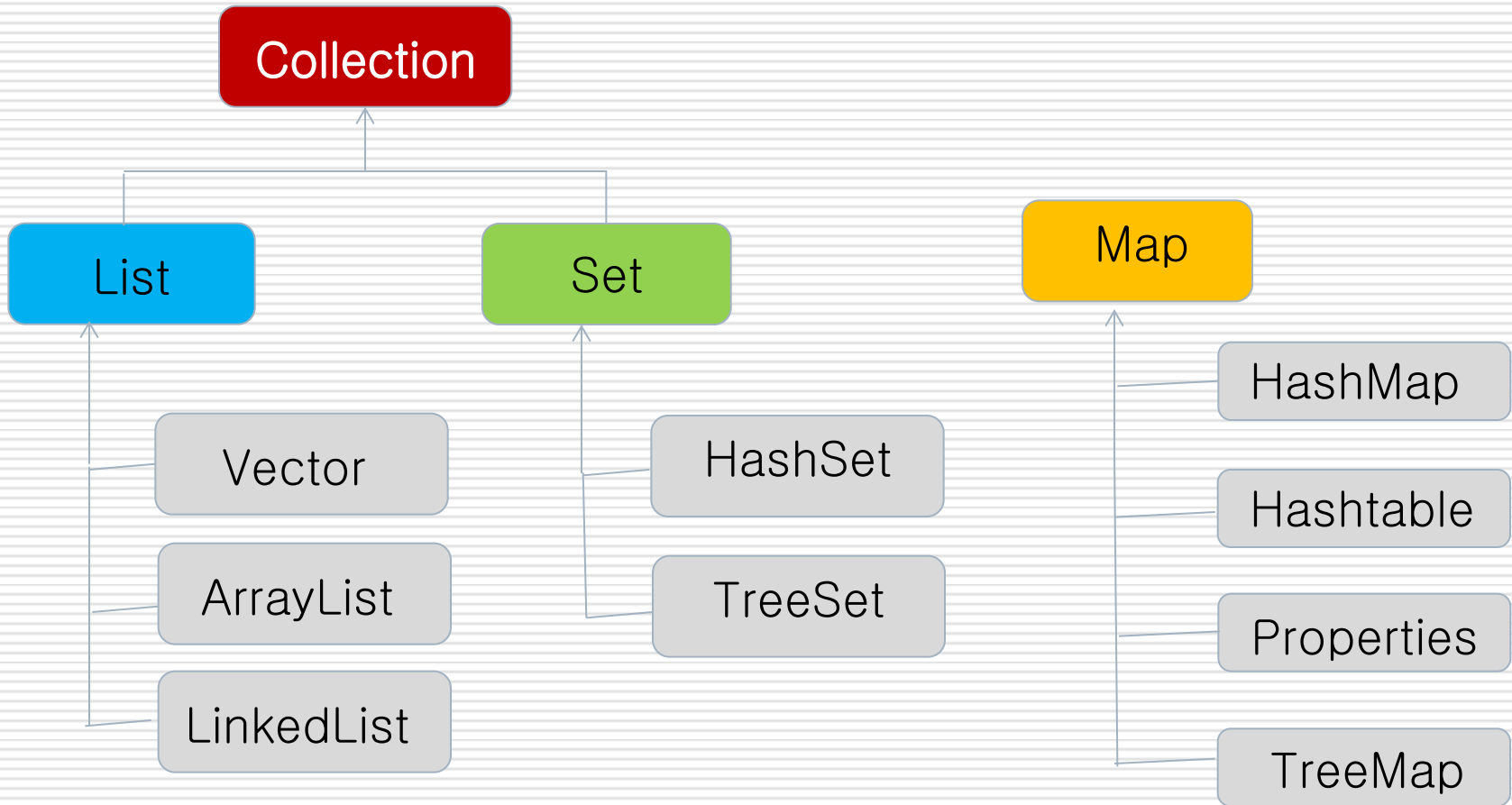
# JAVA 컬렉션 개요

---

- 컴퓨터공학에서는 '자료구조' 분야가 있다.  
자료구조는 많은 양의 데이터를 효율적으로 관리  
(정렬, 검색, 추가, 수정, 삭제)하고자 데이터를 저장하는 방법이다.
  - 자바는 여러 가지 자료구조 알고리즘을 API로 제공한다.  
덕분에 우리는 복잡한 알고리즘을 몰라도 **API를 사용해 자료구조를 구현**할 수 있다.  
이처럼 자료구조를 통해 데이터 그룹을 효율적으로 처리할 수 있도록 지원하는  
자바 API들을 '**자바 컬렉션 프레임워크**'라고 한다.
  - 여러 가지 자료구조 알고리즘을 미리 구현하여 데이터를 효율적으로  
처리하는 자바 API다.
-



# 컬렉션(Collection) 프레임워크



- 순서를 유지하고 저장
- 중복 저장 가능

- 순서를 유지하지 않고 저장
- 중복 저장 안 됨

- 키와 값의 쌍으로 저장
- 키는 중복 저장 안 됨

# 컬렉션(collection) Framework

---

- ❑ 고정 크기의 배열이 가지는 단점을 극복하고
  - ❑ 가변 개수의 객체들을 쉽게 삽입,삭제,검색을 할 수 있는 가변 크기의 컨테이너이다
  - ❑ 따라서 컬렉션은 요소(element)라고 불리는 가변 개수의 객체들의 모임이다
  - ❑ 컬렉션은 요소들을 관리하는 자료구조로서 요소의 추가,삭제,검색 등의 기능을 제공한다
-

# 컬렉션을 구현한 클래스들

---

- java.util 패키지는 컬렉션의 개념을 구현한 핵심적인 다양한 클래스를 제공한다
  - Vector
  - **ArrayList**
  - Hashtable
  - HashMap
  - LinkedList 등

[참고]

컬렉션을 구현한 모든 클래스들은 Object를 상속받는 객체들만 요소로 받아들인다. 즉 byte, char, short, int, long, float, double, boolean 등 8종류의 기본타입은 원칙적으로 사용할 수 없다

---

# 배열과 컬렉션의 개념 차이

---

## □ 배열(array)

- 고정 크기 이상의 객체를 관리할 수 없다
- 배열의 중간에 객체가 삭제되면 응용 프로그램에서 자리를 옮겨야 한다

## □ 컬렉션(collection)

- 가변 크기로서 객체의 개수를 염려할 필요 없다
  - 컬렉션 내의 한 객체가 삭제되면 컬렉션이 자동으로 자리를 옮겨준다
-

# 배열 객체 생성 방법

---

- (1) 데이터타입[] 변수 = { 값0,값1,값2,값3 };
- (2) 데이터타입[] 변수 = null;  
변수 = new 데이터타입[] { 값0,값1,값2,값3 };

[예]

- (1) String[] names = {“김진명”,“조정래”,“이외수”};
  - (2) String[] names = null;  
names = new String[] {“김진명”,“조정래”,“이외수”};
-

# 가변인자(Varargs) = 비정형 인자

---

- 메소드 파라미터의 인자가 변하는 형태(개수의 제한이 없다)
- 같은 타입의 인자만 전달 가능하다.
- 전달받은 데이터는 변수명에 해당하는 **1차원 배열로** 자동으로 생성된다.
- 메소드 선언부의 파라미터 부분 선언 시 **가변인자 기호(...)**을 사용하여 표현한다.

## <기본 문법>

```
접근제한자 리턴타입 메소드명(데이터타입 ... 변수명) {  
    실행문;  
}
```

# Collections 클래스 활용

---

- 이 클래스는 컬렉션에 대해 연산을 수행하고 **결과로 컬렉션을 리턴하는** 유용한 유틸리티 메소드를 제공하며 모든 멤버 메소드는 static 메소드이다
  - 따라서 Collections 클래스를 사용하기 위해 객체를 생성할 필요는 없다
  - 주요 메소드
    - shuffle()
    - sort()
-

# 자바 클래스 라이브러리

---

- 패키지는 연관된 클래스와 인터페이스의 그룹
  - 자바 API는 그 자체가 패키지의 그룹으로 구성
-



# java.lang 패키지

---

- 자바 language 패키지는 자바 언어의 핵심을 구성한다
- 일반적으로 자바 애플리케이션을 수행하기 위하여 기본적으로 필요한 클래스들이 java.lang 패키지에 선언되어 있다
- java.lang 패키지는 import 하지 않아도 항상 자바 컴파일러에 의하여 바이트 코드가 만들어질 때 자동적으로 포함된다

# String 클래스

---

- C나 C++와는 다르게 자바의 String은 문자의 배열이 아니라 클래스이다
  - C나 C++에서의 스트링은 null로 끝나는 문자의 배열인데 반하여 자바의 스트링은 String 클래스의 객체이다
  - 모든 스트링은 `java.lang.String` 클래스로부터 상속받으며 이 클래스에 있는 메소드를 적절하게 사용할 수 있게 된다
-

# 스트링 객체 생성 방법 두 가지

---

(1) 스트링 리터널로 객체 생성

(예) `String str1 = "손흥민";`

(2) String 클래스의 생성자를 이용하여 객체 생성

(예) `String str2 = new String("이강인");`

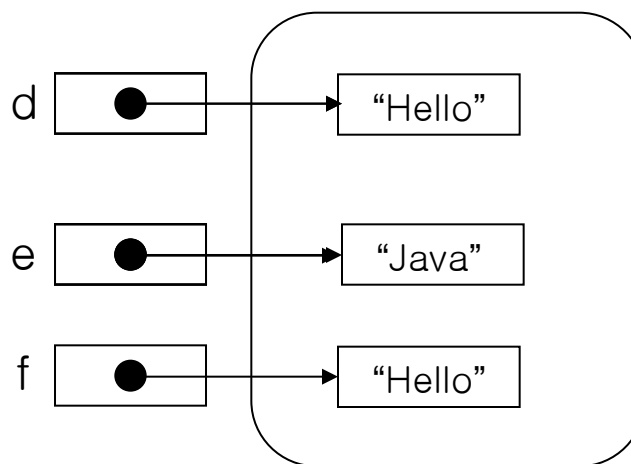
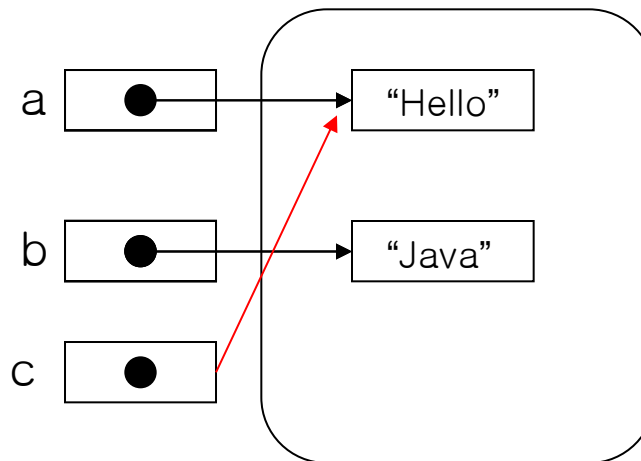
---

## <차이점>

```
String a = "Hello";  
String b = "Java";  
String c = "Hello";
```

```
String d = new String("Hello");  
String e = new String("Java");  
String f = new String("Hello");
```

## 자바 가상 기계의 스트링 리터널 테이블



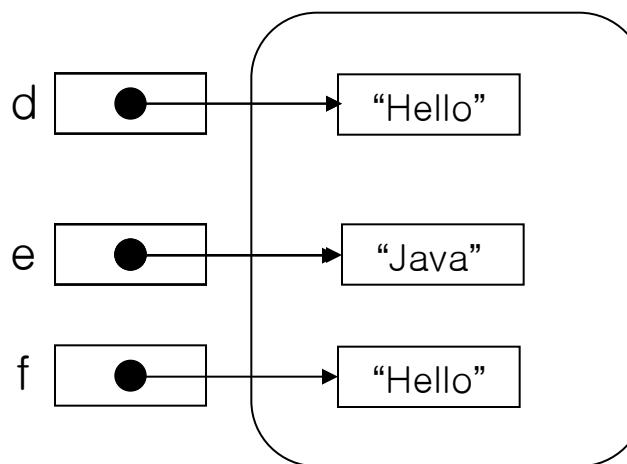
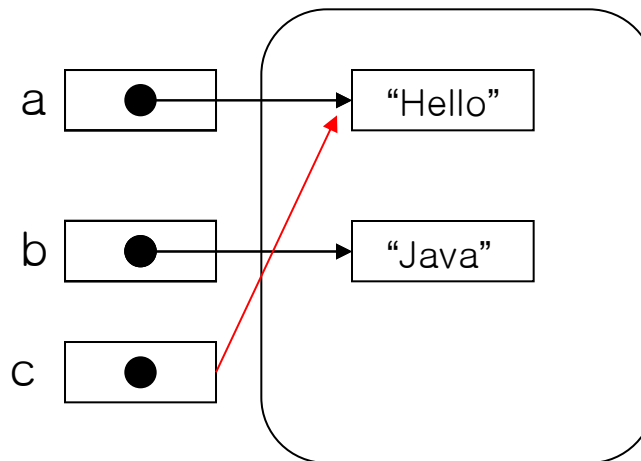
힙(Heap) 메모리

## <차이점>

```
String a = "Hello";  
String b = "Java";  
String c = "Hello";
```

```
String d = new String("Hello");  
String e = new String("Java");  
String f = new String("Hello");
```

## 자바 가상 기계의 스트링 리터널 테이블



힙(Heap) 메모리

# StringTokenizer 클래스

---

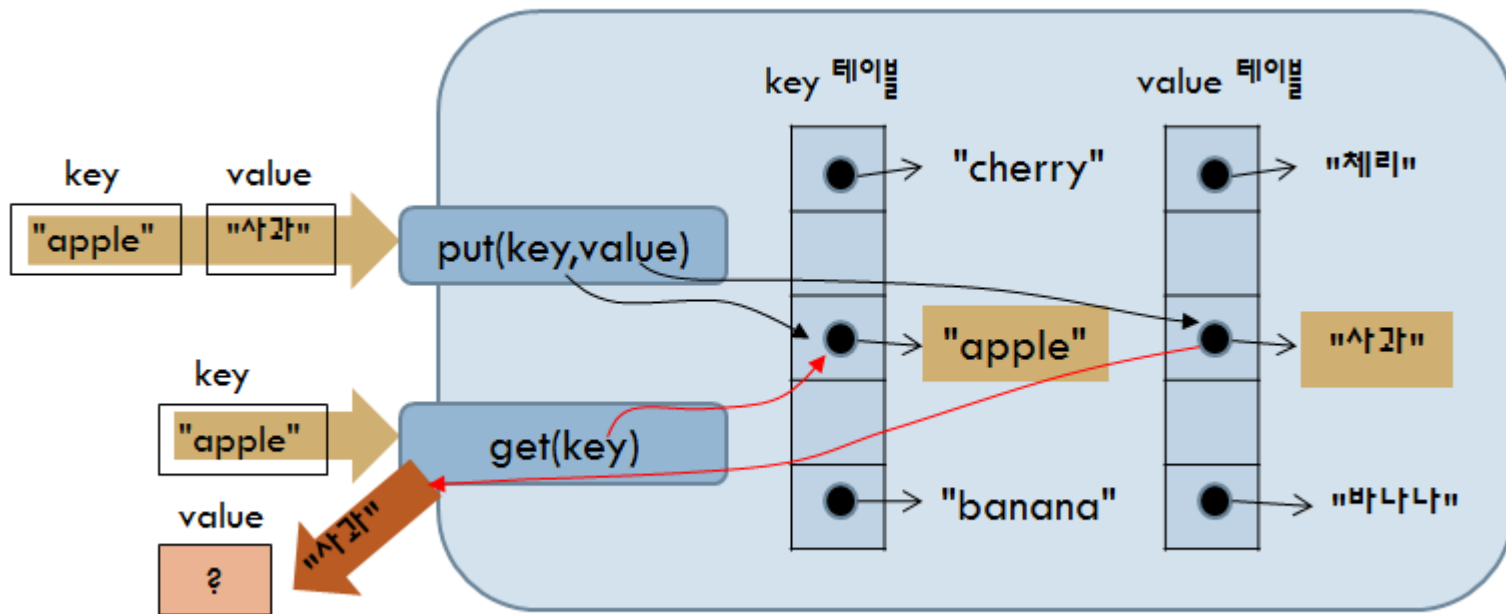
- java.util 패키지에 포함되어 있다
  - 문자열을 분리하기 위해 사용된다
  - 문자열을 사용할 때 사용되는 문자들을 **구분문자(delimiter)**라고 하고, 구분문자로 분리된 문자열을 **토큰(token)**이라고 한다
-

# Hashtable 클래스

---

- ❑ java.util 패키지에 포함되어 있다
  - ❑ Hashtable이 다루는 요소 객체는 항상 **키(key)와 값(value)**의 쌍으로 구성되며 키를 특정 값에 매핑시키는 해시 테이블 기능을 제공한다
  - ❑ 키와 값은 모두 객체만 사용 가능하며 int, char 등과 같은 기본 데이터 타입은 불가능하다
  - ❑ Hashtable은 내부에 키와 값을 저장하는 **자료구조를 각각 가지고 있으며** 키를 입력으로 받는 해시 함수를 통해 내부 자료구조에서의 키와 값의 위치를 결정한다
-

# Hashtable의 내부구성과 put(), get() 메소드





# 예외가 발생하는 경우

---

- ❑ 정수를 0으로 나누는 경우
    - ▶ **ArithmeticException**
  - ❑ 배열의 크기보다 큰 인덱스로 배열의 원소를 접근하는 경우
    - ▶ **IndexOutOfBoundsException**
  - ❑ 유효하지 않은 형변환
  - ❑ 파일을 열어서 읽거나 쓸 때 파일이 존재하지 않은 경우
  - ❑ 레퍼런스 변수가 null인 상태에서 객체를 참조할 경우
    - ▶ **NullPointerException**
  - ❑ 문자열이 나타내는 숫자와 일치하지 않는 타입의 숫자로 변환시 발생
    - ▶ **NumberFormatException**
-

# 예외처리, try-catch-finally문

---

```
try {  
    예외가 발생할 가능성이 있는 실행문(try 블록)  
}  
catch (처리할 예외 타입 선언) {  
    예외 처리문(catch 블록)  
}  
finally { //finally는 생략 가능  
    예외 발생 여부와 상관없이 무조건 실행되는 문장(finally 블록)  
}
```

# 클래스

---

- 자바의 가장 핵심적인 내용
  - 자바 프로그램을 작성하는 기본단위
-

# 객체지향

---

- 데이터와 이를 처리하는 기능이 하나로 이루어져 있는 객체를 모델링하고 이들간의 관계를 정의하는 것
  - 즉 객체 자신 안에 데이터와 데이터 처리 기능의 모든 것이 포함되어 있는 것
-

# 객체지향의 세 가지 기본 요소

---

- ▶ 클래스(Class) – 개념적인 의미
  - ▶ 객체(Object) - 구체적인 의미
  - ▶ 메시지(Message) – 객체 간의 상호작용 수단
-

# 객체지향의 중요 특징

---

- ▶ 추상화(abstraction)
  - ▶ 캡슐화(encapsulation)
  - ▶ 상속(inheritance)
  - ▶ 다형성(polymorphism)
-

# 객체지향의 중요 특징

---

## □ ▶ 추상화(abstraction)

현실 세계에 존재하는 어떤 것을 있는 그대로 표현하면 너무 복잡해진다.  
이때 특정 측면을 강조하여 나타내는 것을 추상화라고 한다.  
(예) 사람, 자동차, 색상

추상화를 통해서 실세계 상황을 간결하고 명확하게 모델링해서 프로그램으로 구현한다. 이러한 구현 과정을 UML(통합 모델링 언어; Unified Modeling Language)에서는 실체화라고 표현한다.

객체 지향에서는 클래스를 이용하여 실세계에 대응하는 추상 모델을 만든다.

[참고]

UML이란? 소프트웨어 공학에서 사용되는 표준화된 범용 모델링 언어이다.

---

# 객체지향의 중요 특징

---

## □ ▶ 캡슐화(encapsulation)

데이터와 이 데이터를 처리하는 오퍼레이션이 한 틀 안에서 결합되어 객체라는 단위로 묶어 사용되는 것을 말한다.

객체 외부에서는 객체 내 정보에 직접 접근하거나 조작할 수 없고, 외부에서 접근할 수 있도록 정의된 오퍼레이션을 통해서만 관련 데이터에 접근할 수 있다.

캡슐화하면 서로 관련 있는 데이터와 오퍼레이션을 객체 단위로 처리해주기 때문에 현실 세계를 단순화하면서도 명확하게 표현할 수 있다.

클래스의 멤버변수나 메소드들을 외부에서 사용가능한것과 그렇지 않은 것을 구분할 수 있게 하는 것으로, 정보은폐를 구현하는 개념이다. 이러한 개념을 구현하기 위해, 자바에서는 **private, protected, public** 등의 세가지로 구분하여 가시성을 나누고 있다.

---



# 객체지향의 중요 특징

---

## □ ▶ 상속(inheritance)

상속은 객체 지향에서 가장 핵심이 되는 개념으로, 프로그램을 **확장**할 수 있도록 도와주는 수단이다. 상속은 객체 지향 패러다임에서만 구현할 수 있다.

상속은 **일반화**(Generalization)와 **상세화**(Specialization)라는 두 가지 개념으로 나누어 표현되기도 한다.

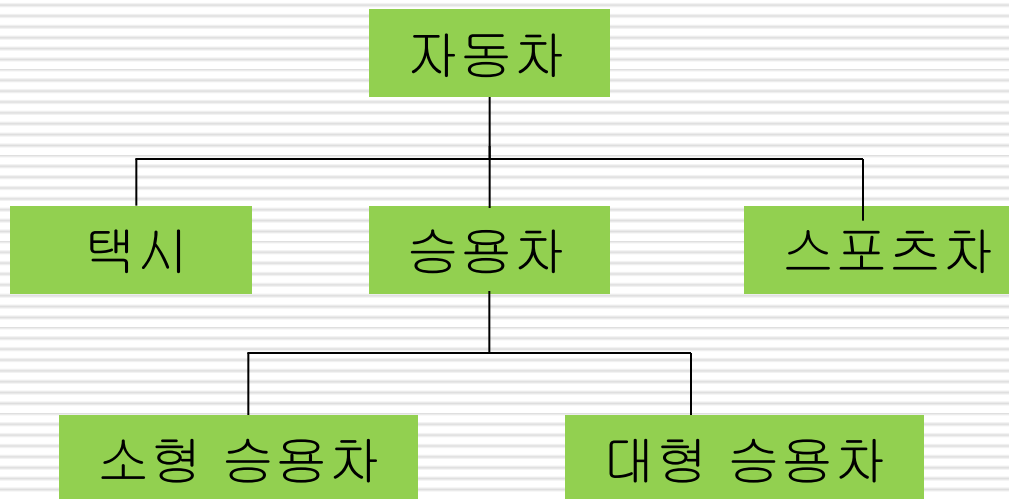
상속은 부모 클래스의 모든 특성을 자식 클래스가 이어받음으로써 이미 정의한 클래스를 **재사용**하고 **확장**할 수 있도록 지원하는 개념이다.

코드를 재사용(reuse)하여 개발의 효율성을 증대하기 위한 중요한 개념이다. 자바에서 자식 클래스를 정의 할 때, **extends**라는 키워드를 함께 상위 클래스 이름을 써주면 간단하게 상속이 이루어진다.

---

# 일반화와 상세화로 표현된 상속

---



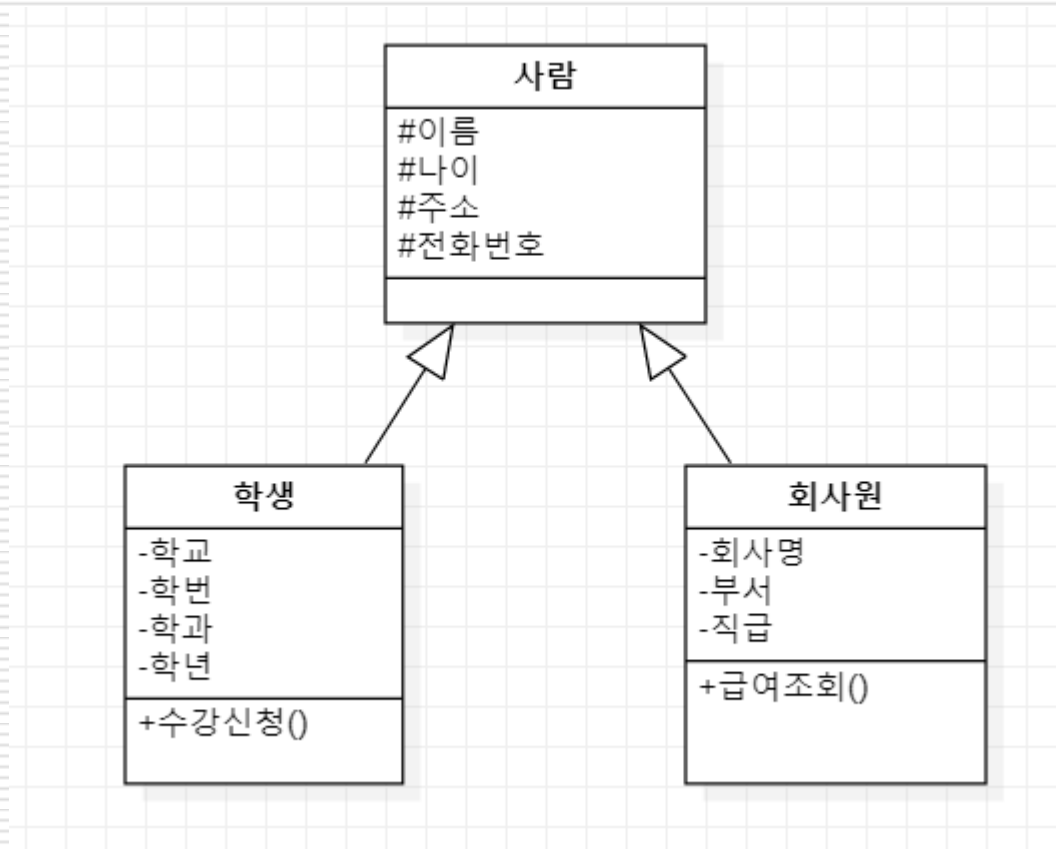
일반화



상세화

# 상속을 표현한 클래스 다이어그램(일반화)

---



# 상속(inheritance) 예제

---

- 상속을 사용하여 '고객 관리' 프로그램 구현하기  
*고객의 등급은 일반 고객과 VIP 고객으로 구분한다.*

## [실행결과]

홍길동님의 고객 등급은 GENERAL이며, 보너스 포인트는 1000입니다.  
제품 지불 금액: 20000

손흥민님의 고객 등급은 VIP이며, 보너스 포인트는 50000입니다.  
제품 할인율 적용 지불 금액: 18000  
VIP 고객 담당 상담원 아이디: 12345

# 객체지향의 중요 특징

---

## □ ▶ 다형성(polymorphism)

여러 클래스에 같은 이름의 메소드가 존재하지만 동작은 다르게 수행하는 것을 의미한다.

같은 메소드 이름을 중복해서 사용할 수 있게 함으로써, 프로그램 개발자에게 편리함을 주는 것이다.

자바에서는 메소드 오버로딩과 메소드 오버라이딩을 통해 다형성을 구현하고 있다.

---

# 자바에서의 객체 설계 방법(3단계)

---

객체 모델링

프로그래밍 하고자 하는 객체의 속성과 필요한 기능을 정리하는것



클래스 정의

객체를 실제로 사용하기 위해서 클래스라는 형태로 객체를 표현한다



클래스 생성과 사용

정의된 클래스를 이용해서, 메모리상에 객체(Object)를 생성하고 사용한다

---

# 1단계 : 객체 모델링

---

## □ 자동차의 속성과 기능

속성	기능
현재속도	속도를 올린다
바퀴의 수	속도를 내린다
자동차 이름	멈춘다

---

## 2단계 : 클래스 정의 => 멤버변수 정의

---

### 1) 클래스 이름 결정

```
class Car {  
  
}
```

### 2) 멤버변수의 정의

```
class Car {  
    private int  speed;           //현재속도  
    private int  wheelNum;       //바퀴의수  
    private String carName;      //자동차이름  
}
```

---



## 2단계 : 클래스 정의 => 메소드 정의

---

```
class Car {  
    private int  speed;           //현재속도  
    private int  wheelNum;       //바퀴의수  
    private String  carName;     //자동차이름  
  
    // 속도를 올린다  
    public void speedUp( ) {  
        speed = speed + 1;  
    }  
    // 속도를 내린다  
    public void speedDown( ){  
        speed = speed - 1;  
        if(speed < 0)  
            speed = 0;  
    }  
    // 멈춘다  
    public void stop( ){  
        speed = 0;  
    }  
}
```

---

## 3단계 : 클래스 생성과 사용

---

- 클래스는 new 연산자를 이용해서 생성한다  
<예> `Car myCar = new Car("제너시스");`  
`Car yourCar = new Car("소나타");`
  - new 연산자의 결과로서 클래스 "객체"를 가리키는 레퍼런스 변수를 반환하게 된다
  - 이 레퍼런스 변수로 해당 클래스를 마음대로 사용할 수 있다  
<사용 예>  
레퍼런스변수.멤버변수 => `myCar.carName`  
레퍼런스변수.메소드( ) => `myCar.speedUp( )`
-

# 접근 지정자

---

□ 자바에서는 4가지 접근 지정자 방식 정의

- default - 접근지정자 생략
- public - **공개**
- private - **비공개**
- protected - **보호된 공개**

default	• 같은 패키지 내에서 접근 가능
public	• 패키지 내부, 외부 클래스에서 접근 가능
private	• 정의된 클래스 내에서만 접근 가능 • 상속 받은 하위 클래스에서도 접근 불가
protected	• 같은 패키지 내에서 접근 가능 • 다른 패키지에서 접근은 불가하나 <b>상속을 받은 경우 하위 클래스에서는 접근 가능</b>

---

# 클래스 접근 지정자

---

## (1) `public`

- 어떤 다른 클래스에서도 사용 가능

## (2) `default`

- 같은 패키지 내에 있는 클래스만이 접근 허용
  - 클래스를 선언할 때 `public`을 생략했다면 클래스는 `default` 접근 제한을 가진다.
-

# 멤버 접근 지정자

---

- 클래스의 멤버인 필드나 메소드의 접근 지정자

멤버에 접근하는 클래스	멤버의 접근 지정자			
	default	private	protected	public
같은 패키지의 클래스	○	X	○	○
다른 패키지의 클래스	X	X	X	○

# 클래스 용도별 분류

---

## (1) 라이브러리 클래스

자체적으로 실행되지 않으나, 다른 클래스에서 이용할 목적으로 만든 클래스

## (2) 실행 클래스

main() 메소드를 가지고 있는 클래스로 실행할 목적으로 만든 클래스

1개의 애플리케이션 = n개의 라이브러리 클래스 + 1개의 실행 클래스

[예] 간단한 공연 예약 시스템을 만든다고 하면, 세 개의 클래스를 생성한다.

(1) Seat 클래스 → 좌석 클래스

(2) SeatType 클래스 → 좌석타입 클래스

(3) Reserve 클래스 → 예약 클래스 ▶ 실행 클래스(즉, main( ) 메소드가 포함된 클래스)

---

# 메소드 오버로딩(Method Overloading)

---

## (1) 정의

“같은 클래스내에서 같은 이름을 가진 메소드를 여러 개 정의 (즉, 구현)할 수 있는 것”

즉, 인자가 다를지라도 같은 동작을 해야 하는 메소드를 같은 이름을 가질 수 있도록 지원해 주는 것을 의미한다.

## (2) 오버로딩의 필요성

“생성자의 이름은 클래스의 이름과 반드시 같아야 한다”라는 규칙이 있었다.

이 규칙대로라면 생성자는 하나밖에 만들 수 없다.

이런 문제를 해결하기 위해서, 자바에는 메소드 오버로딩이라는 방식이 제공된다.

메소드 오버로딩이 가능하면 자동차의 생성자가 여러 개 필요한 경우에는 생성자를 여러 개 정의하고, 그때 그때 필요한 생성자를 호출하면 된다.

또한 생성자 오버로딩을 이용하면, 해당 객체에 가장 적합한 초기화를 할 수 있다.

---

# 메소드 오버로딩(Method Overloading)

---

## (3) 메소드 오버로딩 사용시 주의할 점

메소드의 구분은 인자의 개수와 데이터형(인자의 개수가 같은 경우)에 의해서만 구분된다는 점이다

(예) String 클래스의 `valueOf()` 메소드 경우 - 9개 존재

`valueOf`(boolean b)

`valueOf`(char c)

`valueOf`(char[] data)

`valueOf`(char[] data, int offset, int count)

`valueOf`(double d)

`valueOf`(float f)

`valueOf`(int i)

`valueOf`(long l)

`valueOf`(Object obj)

---



# 메소드 오버라이딩(Method Overriding)

---

- 상속과 관련하여 상위 클래스의 메소드를 하위 클래스에서 재정의 하는 것

# 디폴트 생성자와 상속관계

---

## □ 디폴트 생성자란 ?

클래스 안에 아무런 생성자가 없는 경우에, 컴파일러가 자동으로 생성해주는 생성자를 의미한다.

이것 때문에, 클래스 정의시 특별한 생성자를 코딩하지 않아도 괜찮았던 것이다  
하지만, 클래스 상속을 사용하는 경우에는 생성자 문제를 조금 고려해야 한다

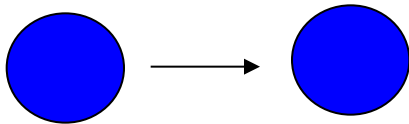
특별히 문제가 되는 경우는 상위 클래스에서 인자가 있는 생성자만이 정의되어  
을 때 문제가 발생하게 된다.

부모 클래스에서 인자가 있는 생성자만을 정의하고 있으면, 컴파일러는 그  
클래스에게 디폴트 생성자를 제공하지 않는다  
결과적으로 인자가 없는 생성자는 자동으로 만들어지지 않는다

---

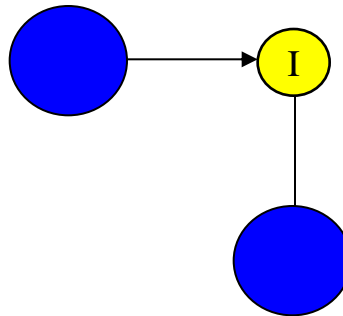
# 자바에서 애플리케이션 설계 방식

sample1



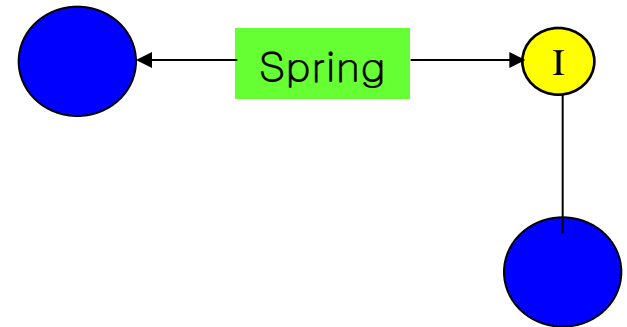
객체가 서로  
강한 결합으로  
묶여 있다

sample2



객체가 서로  
약한 결합으로  
연결된다

sample3

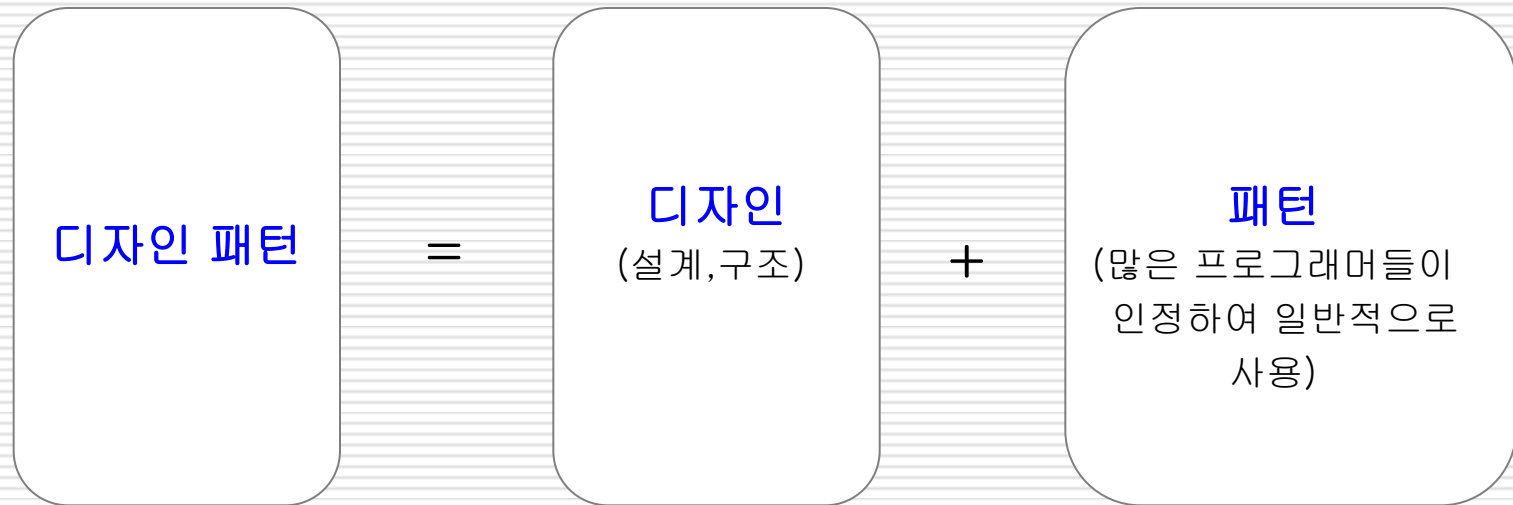


객체가 서로  
좀더 약한 결합으로  
연결된다

# 디자인 패턴(Design Pattern)이란?

---

- 객체 지향 프로그램을 어떻게 구현해야 좀 더 유연하고 재사용성이 높은 프로그램을 만들수 있는지를 정리한 내용이 디자인 패턴이다.



# 디자인 패턴(Design Pattern)의 장점 및 종류

---

1. 코딩이 명확하고 단순하며 소프트웨어 구조 파악이 용이
2. 재사용성이 높아 개발 시간 단축
3. 유지보수를 쉽게하여 요구사항의 변화에 대한 빠른 대처
4. 불필요한 리소스 낭비 방지

## 종류

### ■ 생성 패턴

#### Singleton Pattern ( 싱글톤 패턴 )

클래스의 인스턴스가 하나임을 보장하고 접근할 수 있는 전역적인 접근점을 제공하는 패턴

### ■ 행위 패턴

#### Template Method Pattern ( 템플릿 메소드 패턴 )

알고리즘의 구조는 변경하지 않고 알고리즘의 각 단계를 서브클래스에서 재정의 하는 패턴

---

# Java 디자인 패턴

## 템플릿 메소드 패턴(Template Method Pattern)의 역할

---

- 메소드 '실행순서'와 '시나리오'를 정의 하는것

### 장점

- 코드 중복 감소
- 자식 클래스의 역할(롤)을 감소시키면서 핵심로직 관리 용이
- 객체 추가 및 확장 쉽게 가능

### 단점

- 추상메소드가 너무 많아지면 클래스 관리가 복잡
  - 추상클래스와 구현클래스간 복잡성 증대
-

# 추상 클래스란?

---

1. 실체 클래스의 공통적인 부분(변수, 메소드)를 추출해서 선언한 클래스이다.
2. 추상 클래스는 객체를 생성할 수 없다. 아직은 실체성이 없고, 구체적이지 않기 때문에
3. 추상 클래스와 실체 클래스는 상속 관계이다.

※ 추상 클래스는 왜 사용할까? (용도)

1. 공통된 필드와 메소드를 통일할 목적
2. 실체 클래스 구현시, 시간 절약
3. 규격에 맞는 실체 클래스 구현

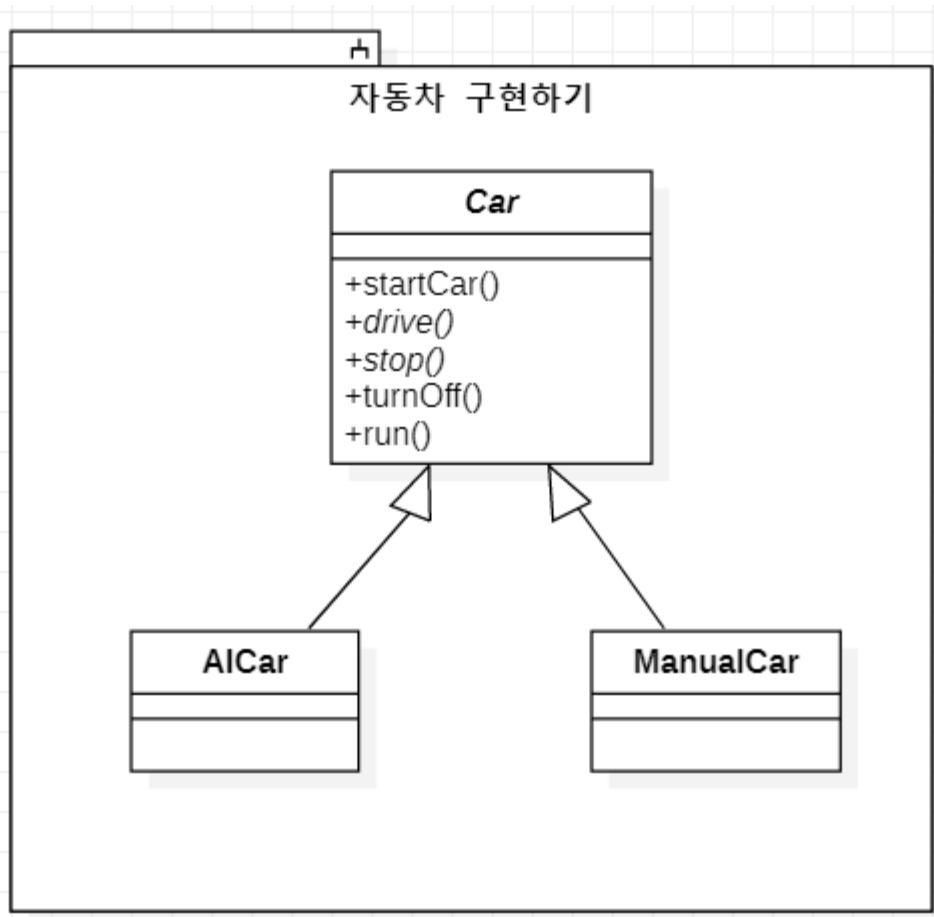
*Car* 추상 클래스를 상속받는 두 클래스는

- 자율 주행 자동차(AICar)
- 일반 자동차(ManualCar) 이다.

자동차 시동을 켜고 끄는 방법은 어느 차나 동일하다.

그러나 달리고, 브레이크로 멈추는 것은 차종에 따라 다른 방식으로 움직일 수 있다.

\* StarUML 툴을 이용하여 클래스 다이어그램 작성





# 내부 클래스(Inner Class)

---

- 내부 클래스를 사용하는 경우  
이벤트 핸들러를 만드는데 많이 사용되는 방식 – **안드로이드 APP 개발시**  
(즉 이벤트 핸들링을 효율적으로 처리하기 위해서)
- <내부 클래스 작성 형식>

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

# 내부 클래스의 주요 특징

---

1. 내부 클래스도 사용시 반드시 클래스 생성
2. 외부 클래스의 멤버변수와 메소드를 마음대로 사용 가능
3. 내부 클래스는 외부에서 단독으로 사용 불가
4. 따로 컴파일하지 않아도 자동으로 생성
5. 클래스 파일이 외부\_클래스\_이름\$내부\_클래스\_이름.class로 생성됨

## 내부 무명 클래스(inner anonymous class)

---

- 무명 클래스는 어떤 class나 interface를 상속/구현 해야만 그 instance를 사용할 수 있는 것이다
- 이처럼 무명 클래스를 사용하면 어떤 절차(수행)를 다른 method의 인수로 건네줄 수 있게 된다. 하지만 간단한 로직만 구현 처리해야 한다.
- 무명 클래스는 조금만 복잡해져도 급격히 소스의 '가독성'이 떨어지게 되므로 남용하지 않는 것이 바람직하다"

# this 변수 , this() 차이점

---

\* this 변수 => 현재 객체의 주소(레퍼런스)  
즉 '객체자신'을 가리킨다.  
this는 컴파일러에 의해 자동으로 생성되며  
별도로 this를 선언할 필요 없이 사용하면 된다.

<사용 형식>

**this.객체내의멤버필드변수 = 매개변수;**

\* this() => 생성자에서 동일한 클래스 내의 다른 생성자를 호출할 때 사용한다.  
즉 한 클래스 내에서 한 생성자에서 다른 생성자를 호출할 때 사용

<사용 형식>

**this(멤버필드변수 or 데이터값, ... );**

---

# super 변수, super() 차이점

---

\* super 변수 => 부모 객체를 참고하고 있기 때문에 부모 메소드를 직접 접근할 수 있다.

<사용 형식>

**super.부모메소드( );**

\* super() => 자식 클래스의 생성자에서 부모 클래스 생성자 호출을 명시적으로 선택할 때 사용된다.

<사용 형식>

**super(부모 클래스의 필드변수);**

**super(부모 추상클래스의 필드변수);**

[주의할점]

super()의 사용은 반드시 생성자 코드의 '첫 라인'에 와야 한다.

그렇지 않으면 컴파일 에러 발생함!

# 열거형 Enum

---

1. 클래스 처럼 보이게 하는 상수
2. 서로 관련 있는 상수들끼리 모아 상수들을 대표할 수 있는 이름으로 타입을 정의하는 것
3. Enum 클래스 형을 기반으로 한 클래스형 선언

Enum을 사용하면서 우리가 얻을 수 있는 이점

1. 코드가 단순해지며, 가독성이 좋다.
  2. 인스턴스 생성과 상속을 방지하여 상수값의 타입 안정성이 보장된다.
  3. enum class를 사용해 새로운 상수들의 타입을 정의함으로 정의한 타입이외의 타입을 가진 데이터값을 컴파일시 체크한다.
  4. 키워드 enum을 사용하기 때문에 구현의 의도가 열거형임을 분명하게 알 수 있다.
-

## 결제/정산 시스템 개발 사례 ► 열거형 Enum 응용

결제라는 데이터는 '**결제 종류**'와 '**결제 수단**'이라는 2가지 형태로 표현된다.  
예를 들어 신용카드 결제는 신용카드 결제라는 결제 수단이며, 카드라는 결제 종류에 포함된다.

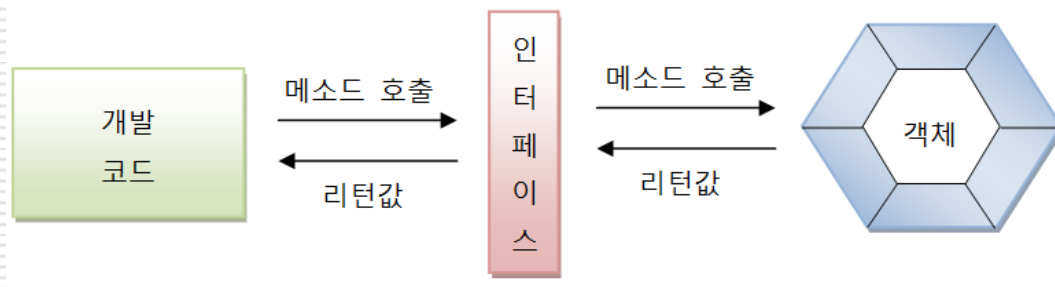


# 인터페이스(interface)의 역할

---

## \*인터페이스란?

- 추상 클래스의 특별한 형태
- 개발 코드와 객체가 서로 통신하는 접점
- 개발 코드는 인터페이스의 메소드만 알고 있으면 OK





# 인터페이스(interface)

- 자바에서는 다중 상속을 가능하게 하기 위해서 인터페이스를 제공한다

<형식>

<pre>interface 인터페이스이름{     상수;     추상 메소드(매개변수);     디폴트 메소드(매개변수)     정적 메소드( ) }</pre>	<pre>interface 인터페이스이름{     public static final 자료형 상수이름 = 값;     public abstract 추상메소드(매개변수);     default void 디폴트메소드(매개변수);     static void 정적메소드( ); }</pre>
-------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------

- interface는 예약어로 선언하며 상수 또는 추상 메소드, **디폴트 메소드, 정적 메소드** (**자바 8에 추가**)를 포함할 수 있다
- interface 내의 메소드는 서브 클래스에서 어떻게 동작해야 할지를 구현(새롭게 정의)해 주어야 한다
- interface는 추상 메소드를 멤버로 갖기 때문에 인스턴스를 생성할 수는 없지만 레퍼런스 변수는 선언이 가능하다

# 인터페이스 구현

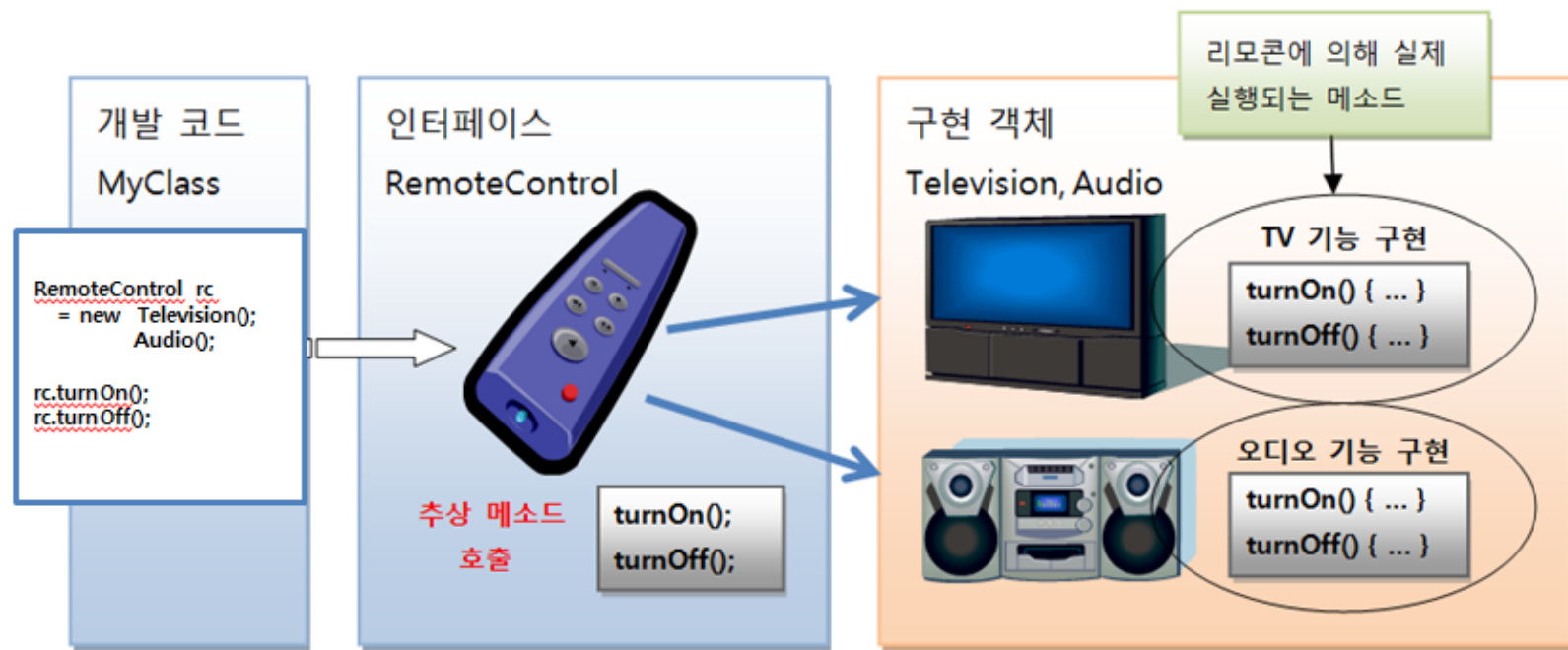
---

## \* 익명 구현 객체

- 명시적인 구현 클래스 작성 생략하고 바로 구현 객체를 얻는 방법
- 이름 없는 구현 클래스 선언과 동시에 객체 생성

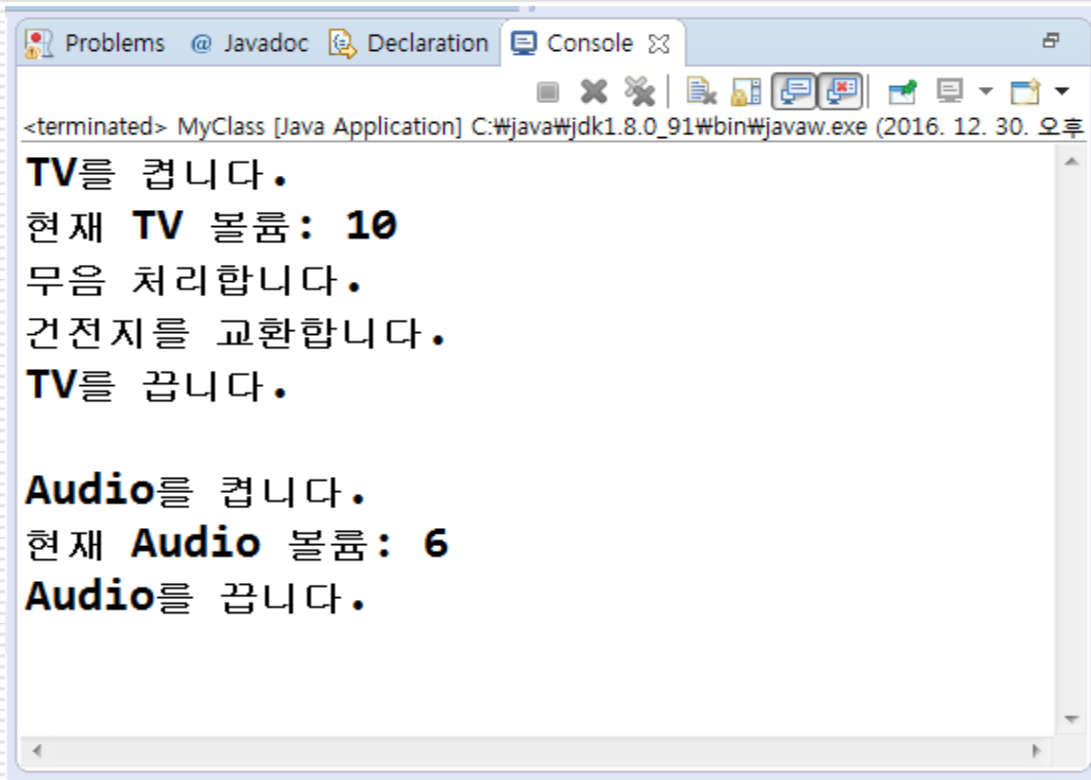
```
인터페이스 변수 = new 인터페이스() {  
    //인터페이스에 선언된 추상 메소드의 실제 메소드 선언  
};
```

# 인터페이스 응용



## 프로그램 실행결과

---



The screenshot shows a Java IDE's console window with the following text:

```
<terminated> MyClass [Java Application] C:\java\jdk1.8.0_91\bin\javaw.exe (2016. 12. 30. 오후  
TV를 켭니다.  
현재 TV 볼륨: 10  
무음 처리합니다.  
건전지를 교환합니다.  
TV를 끕니다.  
  
Audio를 켭니다.  
현재 Audio 볼륨: 6  
Audio를 끕니다.
```

# 람다식(Lambda Expressions)

---

함수적 프로그래밍 기법을 자바에 접목시키자!

$y = f(x)$

$() \rightarrow \{ \};$

$x \quad y$

$(\text{타입 매개변수}, \dots) \rightarrow \{ \text{실행문}; \dots \}$

# 표준 API의 함수적 인터페이스

---

- |              |     |                           |
|--------------|-----|---------------------------|
| 1. Consumer  | 소비자 | => 매개값은 있고, 리턴값은 없다.      |
| 2. Supplier  | 공급자 | => 매개값은 없고, 리턴값은 있다.      |
| 3. Function  |     | => 매개값은 있고, 리턴값도 있다.      |
| 4. Operator  |     | => 매개값은 있고, 리턴값도 있다.      |
| 5. Predicate | 조사  | => 매개값은 있고, 리턴타입이 boolean |

# 스트림(Stream)

---

자바 8부터 추가된 컬렉션(배열 포함)의 저장 요소를 하나씩 참조해서 랴다식(함수적스타일)으로 처리할 수 있도록 해주는 **반복자**이다.

## \* 스트림의 특징

1. 랴다식으로 요소 처리 코드를 제공한다.
2. 내부 반복자를 사용하므로 병렬처리가 쉽다.  
=> 내부 반복자를 사용해서 얻는 이점은  
컬렉션 내부에서 어떻게 요소(즉, 객체)를 반복시킬 것인가는  
컬렉션에게 맡겨두고, 개발자는 요소처리 코드에만 집중할 수 있다.
3. 스트림은 '**중간처리**'와 '**최종처리**'를 할 수 있다.

# 스트림을 얻어오는 방법

---

1. 컬렉션으로부터 스트림 얻기 => List 컬렉션의 참조변수를 통해서
2. 배열로부터 스트림 얻기       => `Arrays.stream(String[ ] 참조변수);`
3. 숫자범위로부터 스트림 얻기 => `IntStream.range(1, 100);`
4. Random 클래스 객체의 `ints()` 메소드로부터 스트림 얻기  
=> `IntStream intStream = new Random().ints(1, 46);`
5. 파일로부터 스트림 얻기       => Files 또는 `BufferedReader` 클래스의  
`lines()` 메소드를 통해서 스트림 얻어옴



# 스트림 중간처리, 최종 처리 메소드들

## '중간처리' 메소드

종류	메소드
스트림 필터링	filter(), distinct()
스트림 변환(매핑)	map(), mapToInt()
스트림 제한	limit(), skip()
스트림 정렬	sorted()
스트림 연산 결과 확인(루핑)	peek()

## '최종처리' 메소드

종류	메소드
요소의 출력	forEach()
요소의 검사(매칭)	anyMatch(), allMatch()
요소의 검색	findAny(), findFirst()
요소의 통계(집계)	count(), min(), max()
요소의 연산(집계)	sum(), average()
요소의 수집	collect()
요소의 소모	reduce()

- 중간처리 메소드와 최종처리 메소드를 쉽게 구분하는 방법은 리턴 타입에서 차이점이 있다.  
리턴타입이 **스트림**이면 중간처리 메소드이고, **기본 타입이거나 OptionalXXX**라면 최종처리 메소드이다.

# 스트림 메소드 (매개변수)의 람다식 표현

---

filter(Predicate 람다식 표현) 메소드

anyMatch(Predicate 람다식 표현) 메소드

mapToInt(ToIntFunction mapper) 메소드

(예) mapToInt(클래스이름::getter메서드이름)

forEach(Consumer 람다식 표현) 메소드

---

# 실무에서 웹 개발시 DB 이력 형상 관리(Configuration Management) 클래스 구현시 람다식과 스트림 적용 사례

## DBVersionManager 클래스 구현

```
private String[] creSeqSql = { // 시퀀스 생성 sql
    "create sequence MEMBER_SEQ increment by 1 start with 1 NOCYCLE", // 회원 시퀀스
    "create sequence MOVIE_SEQ increment by 1 start with 1 NOCYCLE", // 영화 시퀀스
    "create sequence FAQ_SEQ increment by 1 start with 1 NOCYCLE", // FAQ 시퀀스
    "create sequence NOTICE_SEQ increment by 1 start with 1 NOCYCLE" // 공지사항 시퀀스
};
```

```
Arrays.stream(creSeqSql).forEach(sql -> { try { stmt = conn.prepareStatement(sql);
    stmt.executeUpdate();
} catch (Exception e) {
    System.out.println("SQL 실행 안됨 : " + sql);
}
});
```

# 자바 스레드(Thread)

---

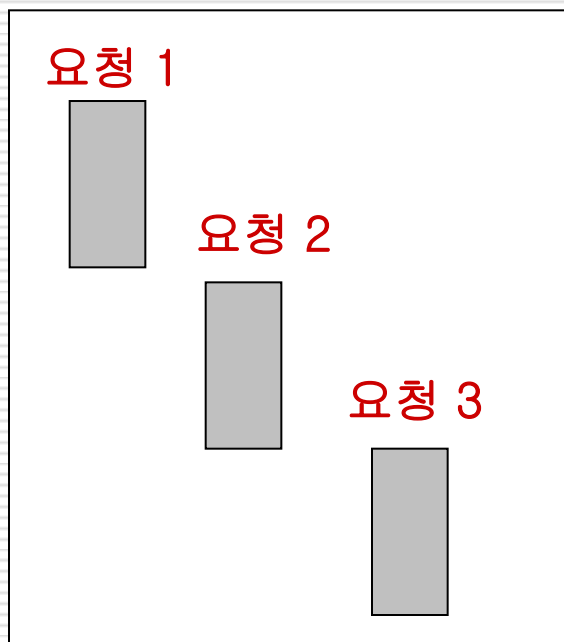
- 하나의 프로그램이 동시에 여러 개의 일을 수행할 수 있도록 해주는 기법
- 자바는 언어차원에서 스레드가 지원되기 때문에 안전성과 효율성이 보장됨
- Multi-Threading이 지원되면

- 하나의 프로그램내에서 여러 개의 일을 동시에 수행하는 것이 가능하고
- 또 스레드간에 데이터 공유가 가능하므로 멀티프로세스 시스템보다 더 효율적으로 프로그램을 작성 가능하다.

# 인터넷 서버 프로그램 작성시

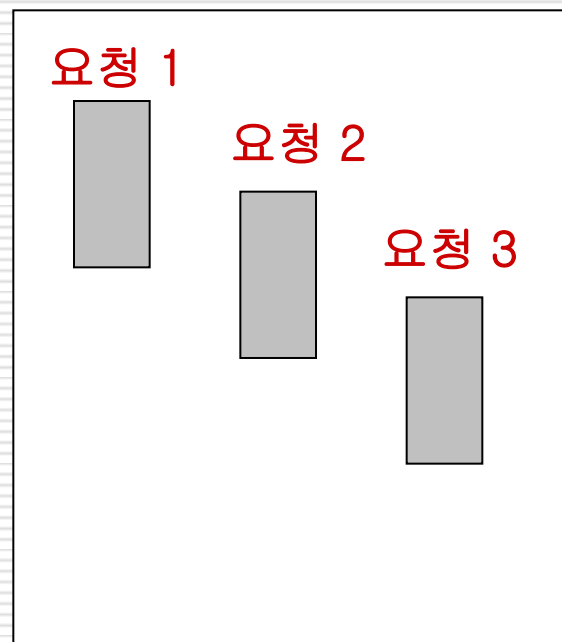
---

“순차적으로 요구 처리”



▲ 스레드를 사용하지 않는 구조

“각 요구를 처리하는 스레드를 만들어 각각을 처리해주면 된다.” – 응답속도 빠름



▲ 스레드를 사용하는 구조

---

# 스레드 우선 순위

---

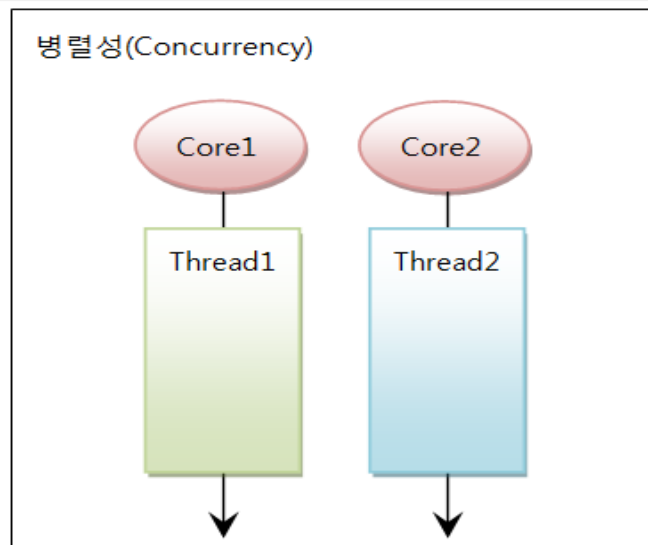
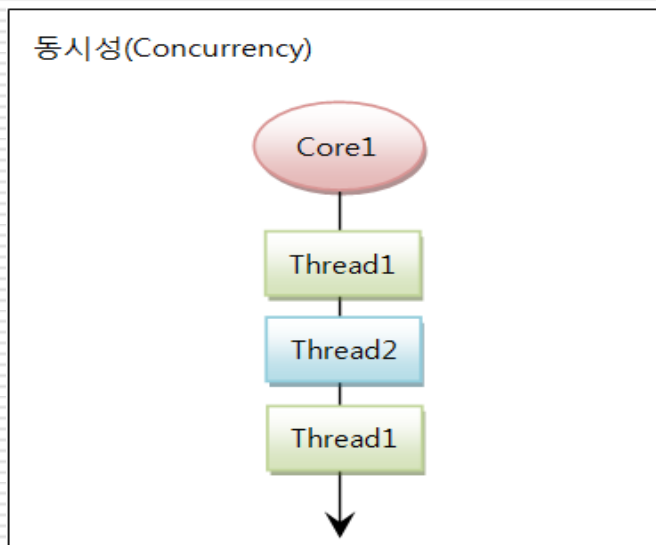
## □ 동시성과 병렬성

### ■ 동시성

멀티 작업 위해 하나의 코어에서 멀티 스레드가 번갈아 가며 실행하는 성질

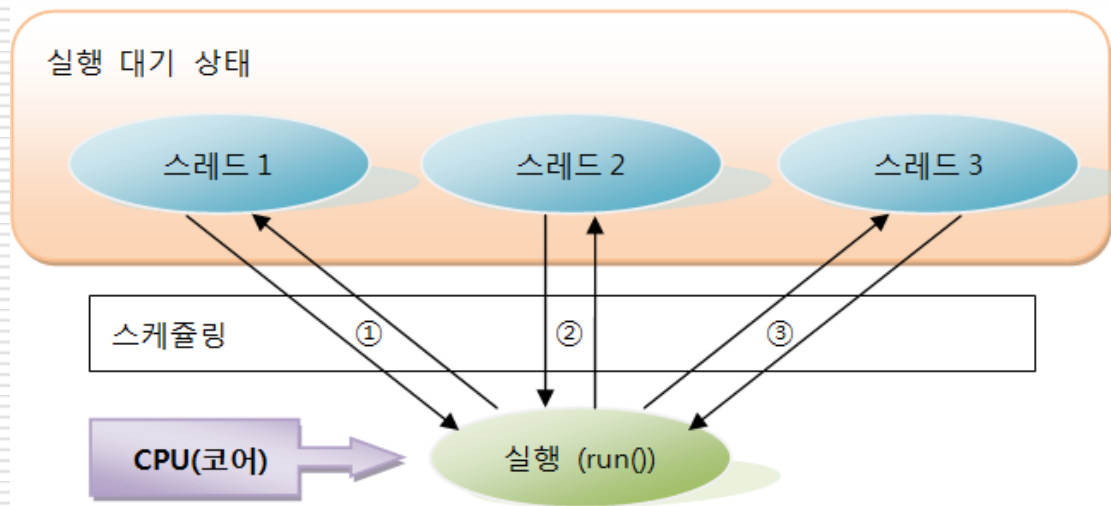
### ■ 병렬성

멀티 작업을 위해 멀티 코어에서 개별 스레드를 동시에 실행하는 성질



# 스레드 스케줄링

- 스레드의 개수가 코어의 수보다 많을 경우
  - 스레드를 어떤 순서로 동시성으로 실행할 것인가 결정 → 스레드 스케줄링
  - 스케줄링 의해 스레드들은 번갈아 가며 `run()` 메소드를 조금씩 실행



# Multi-Threading 정리

---

- 하나의 프로세스가 여러 가지의 일을 처리해야 할 때, 프로세스 내에서 각 일을 처리하는 **모듈들이 독립적으로 수행되는 것**을 의미한다.
  - 이 각각의 실행하는 독립적인 모듈을 스레드라고 한다.
  - 즉, 스레드란 프로세스내에서 일을 처리하는 '**세부 실행단위**'를 의미함
-



# 자바의 스레드 지원 방식

---

- (1) `java.lang.Thread` 클래스를 사용하는 방법
- (2) `java.lang.Runnable` 인터페이스를 사용하는 방법

## <차이점>

- (1) 방법: 스레드를 지원하고자 하는 클래스를 생성시  
즉 `Thread`를 상속받는 class를 `start()` 시킨다.
  - (2) 방법: `Runnable`를 구현한 class를 `new Thread()`의 인수로 주어서  
새로 만든 다른 클래스로부터 상속이 필요한 경우 `Thread` 객체를  
`start()` 시키는것
-

# Thread 클래스 사용법

---

- (1) Thread 클래스를 상속받은 뒤, 해당 스레드에서 지원하고 싶은 코드를 run() 메서드에서 오버라이딩 해준다.
- (2) 해당 스레드 객체를 생성한 후, 스레드 객체의 start() 메서드를 호출한다.

# start() 메서드와 run() 메서드의 관계

---

- 프로그래머는 스레드의 start() 메서드만을 호출할 수 있다.
- 그러면, JVM이 스레드를 준비(Ready) 상태로 바꾼다.
- 이런, 준비 상태의 스레드를 JVM내의 스레드 스케줄러가 실행(Run) 상태로 바꾸어주면, 이때 비로소 스레드의 run() 메서드가 실행된다.

# Runnable 인터페이스의 사용법

---

- (1) Runnable 인터페이스를 구현한 스레드 클래스를 정의하고, 해당 스레드에서 지원하고 싶은 코드를 `run()` 메서드에서 오버라이딩 해준다.
- (2) 해당 클래스 객체를 생성한 후, Thread 클래스 생성자의 인자로 넘겨서 Thread를 생성하고, 이 Thread 클래스 객체의 `start()` 메서드를 호출한다.

# 자바의 스레드: Runnable 인터페이스

---

- 자바에서는 여러 개의 클래스에서 동시에 상속 받는 클래스를 만들 수 없다.
- 따라서, 다른 상위 클래스로부터 이미 상속 받은 하위 클래스는 Thread 클래스를 상속받을 수 없다.
- 이런 경우를 위하여, 자바에서는 Runnable 인터페이스를 제공한다. 이 인터페이스의 run() 메서드를 구현하면, 다른 클래스로부터 상속을 받더라도 스레드의 이용이 가능하다.

(예1) `public class GraphicThread extends Frame implements Runnable`

(예2) `public class ChatClient extends Frame implements Runnable,  
WindowListener,  
ActionListener`

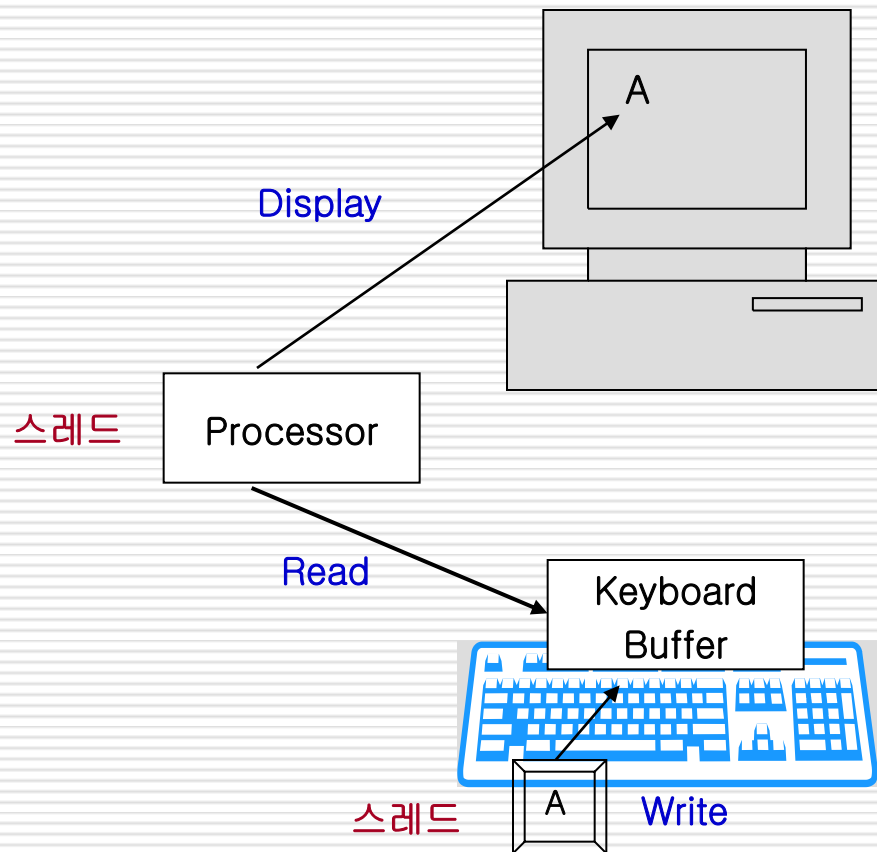
---

# 동기화

---

- 자바에서는 언어 차원에서 멀티스레딩을 지원한다
- 여러 개의 스레드가 같은 데이터나 메소드를 사용할 때 주의해야 할 점이 있는데, 바로 동기화를 맞추는 것이다

# 간단한 생산자/소비자 관계



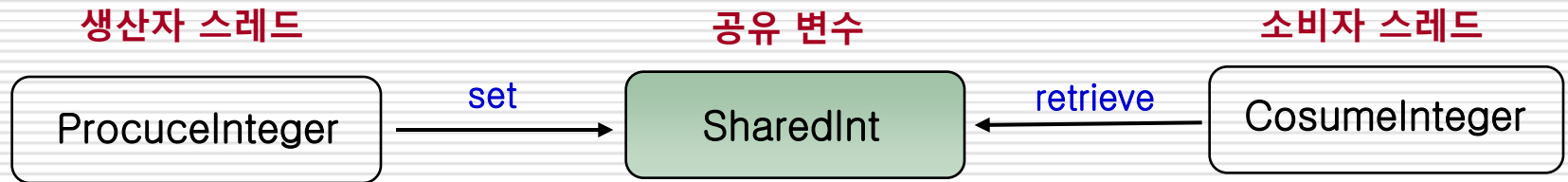
예를 들어 키보드의 입력 과정을 보면  
사용자의 키입력은 키보드 버퍼에  
쌓이게 되고, 프로세서는 키보드 버퍼  
에서 입력된 키의 값을 읽어와 모니터에  
디스플레이 한다

이 때 발생할 수 있는 문제는 프로세서가  
채 읽어가지 못해서 키보드 버퍼가 꽉  
차있는 상태에서 사용자가 입력한 키의  
값이 버퍼에 덮어 쓰이는 경우와  
키보드 버퍼가 비어 있는데 프로세서가  
키의 값을 읽어오려는 경우가 있다

이런 경우에 키보드 버퍼를 공유 데이터라  
볼 수 있고, 사용자 입력과 프로세서를  
각각의 스레드라 볼 수 있다

# 생산자/소비자 관계의 예

---



생산자 스레드는 공유 변수 SharedInt의 값을 1부터 10까지로 순서대로 세팅하고  
소비자 스레드는 공유 변수의 값을 읽어서 그 값을 출력한다

---



# 비동기적 생산자/소비자 관계

---

□ 동기화를 맞추지 않았을 때 문제 발생

## (1) 데이터 잃어버림(data lost)

- 소비자 스레드가 공유변수 값을 사용하기 전에 다시 세팅

## (2) 데이터 중복(data duplication)

- 생산자 스레드가 공유변수의 값을 바꾸기 전에 소비자 스레드가 데이터의 값을 다시 사용
-

# 동기적 생산자/소비자 관계

---

- 하나의 스레드가 '공유 메모리'를 액세스하고 있는 동안에 이 스레드는 임계영역 안에 있는 것이다
  - 자바 언어에서는 synchronized 키워드를 사용하여 임계영역을 선언할 수 있다
-

# 동기화 메소드(synchronized method)

---

- 동기화 블록은 블록 안의 코드만을 동기화하는 반면에 동기화 메소드는 메소드 전체를 동기화한다.
- 동기화 메소드의 형태는 다음과 같다.

```
public synchronized void 메소드이름() {  
    //임계영역; => 단 하나의 스레드만 실행  
    [코드]  
}
```

# 동기화 블록 (synchronized block)

---

- 동기화 블록은 어떤 객체의 모니터(monitor)를 가지고 있는 스레드가 실행 권한을 획득하게 하는 방법이다.

```
synchronized(someObject)
{
    [코드]
}
```

- ♣ 모니터(monitor)란 특별한 데이터 아이템(제어변수)과 데이터에 락(lock)을 거는 부분으로 이루어져 있다
-

# 스레드 구현 - 생산자, 소비자 스레드

---


```
생산자 생산 sharedInt : 1
소비자 가져가기 : 1
생산자 생산 sharedInt : 2
소비자 가져가기 : 2
생산자 생산 sharedInt : 3
소비자 가져가기 : 3
생산자 생산 sharedInt : 4
소비자 가져가기 : 4
생산자 생산 sharedInt : 5
소비자 가져가기 : 5
생산자 생산 sharedInt : 6
소비자 가져가기 : 6
생산자 생산 sharedInt : 7
소비자 가져가기 : 7
생산자 생산 sharedInt : 8
소비자 가져가기 : 8
생산자 생산 sharedInt : 9
소비자 가져가기 : 9
생산자 생산 sharedInt : 10
소비자 가져가기 : 10
```

<http://www.eclipse.org/downloads/>

# 고객정보 관리 시스템 INTRO

고객관리 시스템\_information X

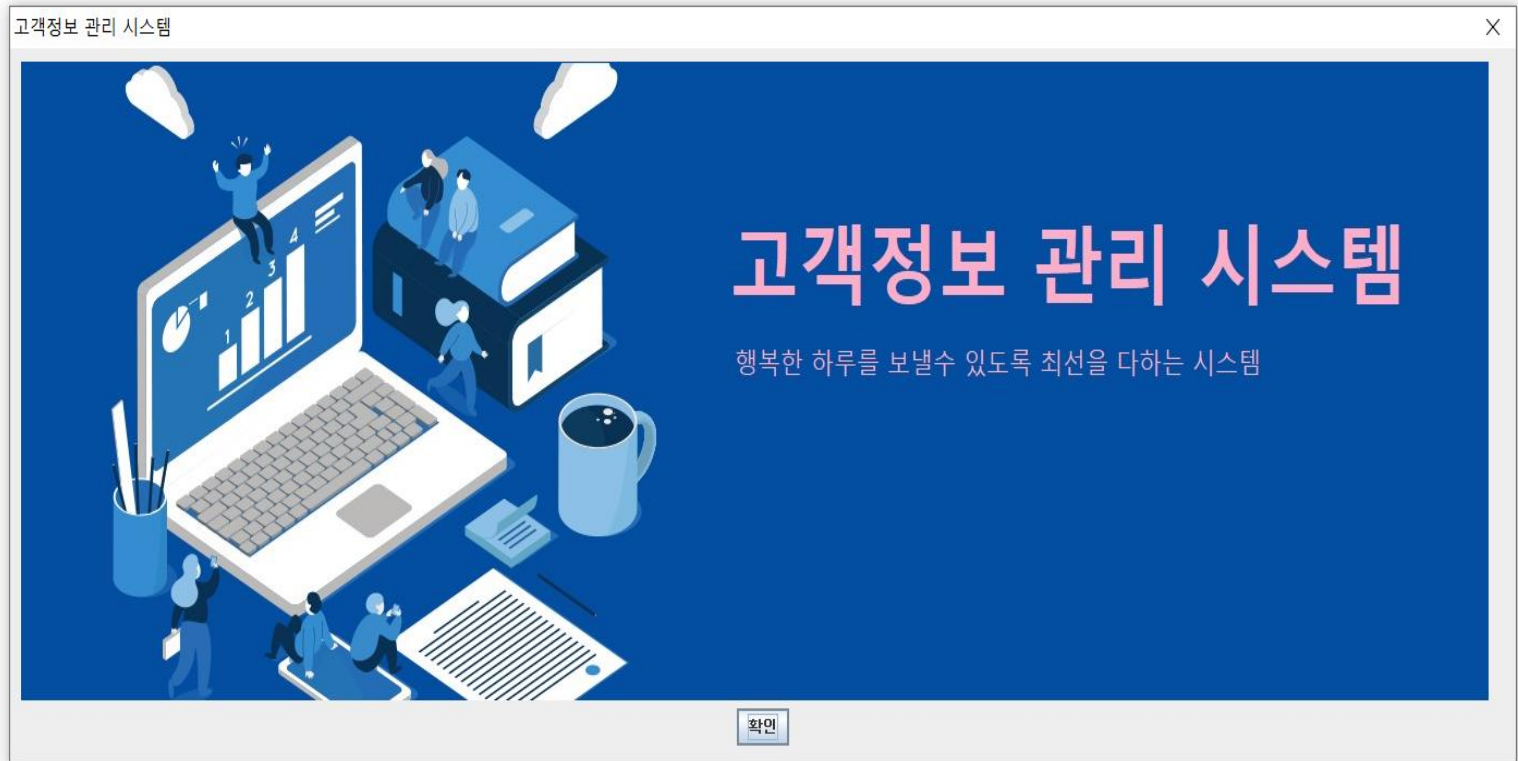
Introduction ○



고객정보 관리 시스템

확인

# 고객정보 관리 시스템 INTRO



# 고객정보 관리 시스템 INTRO

고객정보 관리 시스템

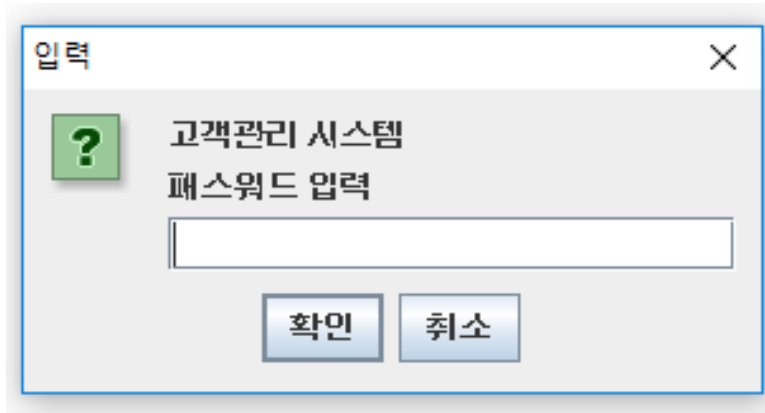


## 고객정보 관리 시스템

확인



## 고객정보 관리 시스템 Password 인증처리



입력

고객관리 시스템  
패스워드 입력

확인 취소

## ■ 고객정보 관리시스템(실행화면)

고객정보 관리시스템

파일

정렬

도움말

입력

번호

이름

핸드폰 번호

이메일

주민등록번호

직업

선택

선택정보

나 이 :

성 별 :

출생지역 :

생 일 :

번호	이름	핸드폰번호	E-Mail	주민등록번호	나이	성별	출생지역	생일	직업

추가

삭제

이전

다음

수정

검색

## ■ '검색' 버튼 클릭시 실행화면

고객정보 관리시스템

파일 정렬 도움말

입력

번호

이름

핸드폰 번호

이메일

주민등록번호

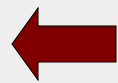
직업

정보검색

☐ 이름 ☐ 직업 ☐ 출생지역 ☐ 생일 월일

번호	이름	핸드폰번호	E-Mail	주민등록번호	나이	성별	출생지역	생일	직업
----	----	-------	--------	--------	----	----	------	----	----

추가 삭제 이전 다음 수정 **검색**



'검색' 버튼 클릭 시 CardLayout

# Contents



# CRM 이란? - 고객관계관리

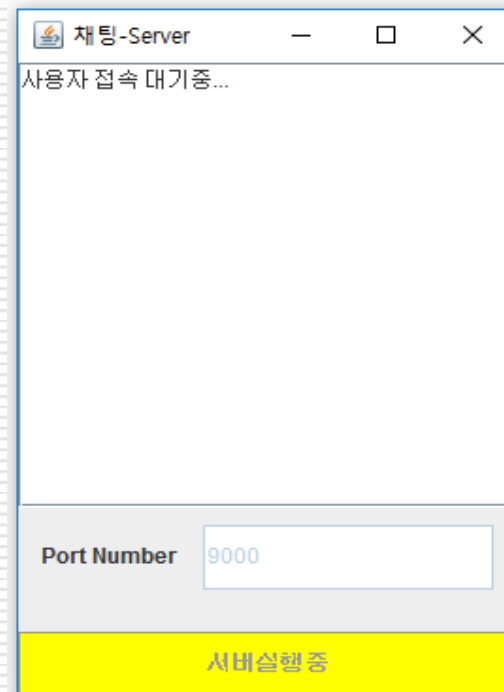
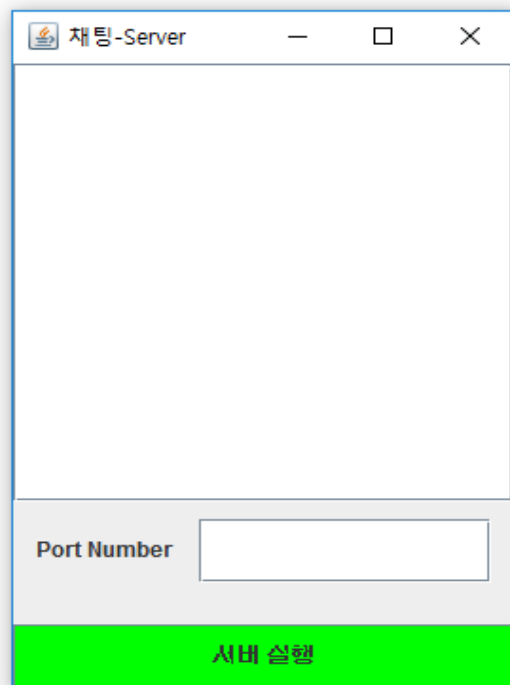
---

1. Customer (고객)
2. Relationship (관계)
3. Management (관리)

1. 지속적인 성장을 유지하기 위하여 가치있는 고객을 파악, 획득 및 유지하는 일련의 활동
  2. 마케팅, 고객 생일 이벤트, 판매, 서비스 등이 포함
  3. 전산 시스템의 변화가 아니라 경영전략, 조직 프로세스, 고객접점 채널상의 모든 변화를 의미
  4. 특정 직업 군 이나 출신도 별로 표적집단을 선정하여 마케팅 할 수 있도록 직업과 출신도 별 검색 가능한 시스템 구축.
-

# TCP 네트워킹 – 채팅 프로그램

---



# Client 연결요청, Server 연결수락

---

채팅-Client

ID

Server IP

Port

채팅-Client

매개 변수로 넘겨온 값 : park1234 localhost 9000  
정상 접속 되었습니다

채팅-Server

사용자 접속 대기중...  
사용자 접속!!  
접속자 ID park1234  
사용자 접속 대기중...

Port Number

# Client와 Server 통신

채팅-Client

매개 변수로 넘어온 값 : park1234 localhost 9000  
정상 접속 되었습니다

안녕하세요!!

전 송

채팅-Client

ID

Server IP

Port

접속

채팅-Server

사용자 접속 대기중...  
사용자 접속!!  
접속자 ID park123  
사용자 접속 대기중...  
사용자로부터 들어온 메세지 : 안녕하세요!!  
사용자 접속!!  
접속자 ID kim567  
사용자 접속 대기중...  
사용자로부터 들어온 메세지 : 하이!! 조은 날씨  
사용자로부터 들어온 메세지 : Hello Sever

Port Number

서버실행중



# 오라클 DB에 Data Insert 작업

---

DB 연동 테스트

추가

번호

이름

직책

부서번호

이메일

# JDBC

---

- JDBC(Java Database Connectivity)
  - JDBC는 자바로 작성된 프로그램을, 일반 데이터베이스에 연결하기 위한 응용프로그램 인터페이스 규격
  - 응용프로그램 인터페이스는 데이터베이스 관리시스템에 넘겨질 SQL 형태의 데이터베이스 접근요구 문장을, 각 시스템에 맞도록 바꾸어준다.
-

# JDBC의 출현 배경

---

- JDBC가 SQL을 자바로 객체모델링 한 것에 불과하기 때문에, 몇 가지 스텝만 거치면 누구라도 쉽게 사용 가능
  - JDBC가 나오기 이전에는 자바를 이용해서 DB에 접근하기 위해서는, DB 루틴을 C언어 등으로 네티브 메소드(Native Method)를 만들어 연결하는 수 밖에 없었다.
  - 그러나, 이런 과정은 자바 프로그램의 이식성에 큰 문제점으로 지적되었다. 또한, DB의 종류별로 네티브 메소드를 만드는 것도 쉽지 않은 일이었다.
-

# JDBC 드라이버 타입

---

- Type 1 : JDBC-ODBC Bridge
- Type 2 : Native-API/partly Java driver
- Type 3: Net-protocol/all-Java driver
- Type 4 : Native-protocol/all-Java driver

## [참고]

- (1) 드라이버 관리자는 데이터베이스에 맞는 드라이버를 찾고  
JDBC 초기화 작업을 수행한다  
Type 4는 JDBC 호출을 DBMS에 바로 전달한다.
- (2) 100% 자바 클래스로 되어 있으며, **오라클의 경우 Thin 드라이버**  
가 있다

# JDBC 드라이버

---

JDBC 드라이버	Type	연결URL	드라이버클래스
오라클 thin 드라이버	4	"jdbc:oracle:thin:@<server-IP>:1521:xe"	oracle.jdbc.driver.OracleDriver

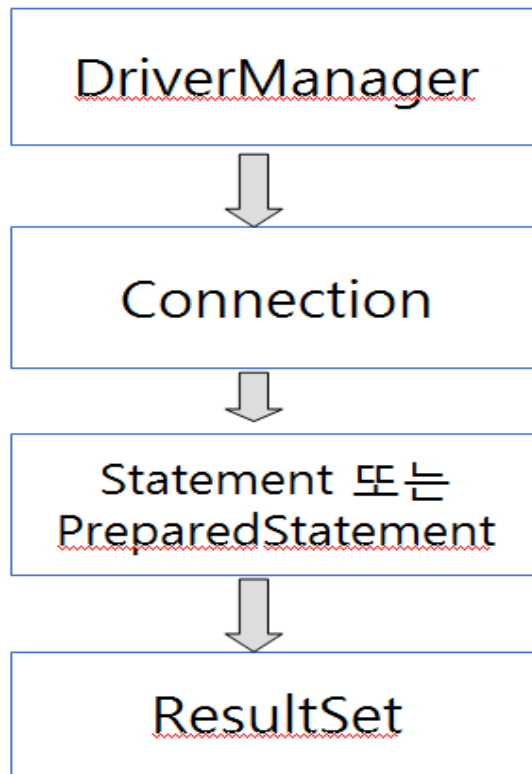
[참고]

오라클 11g XE(Express Edition) 설치시  
전역 데이터베이스 이름은 'xe'로 설정됨

---

# JDBC의 사용법

---



—————→ 1단계: JDBC 드라이버를 OPEN

—————→ 2단계: SQL DB와 연결

—————→ 3단계: SQL 쿼리 실행

—————→ 4단계: 결과를 얻어낸다.

4단계는 검색(select) 할때만 적용

# 오라클 SQL 언어의 종류

구 분	문 장	설 명
DQL(데이터 질의어; Data Query Language)	SELECT	검색시 사용
DML(데이터 조작어; Data Manipulation Language)	INSERT UPDATE DELETE	변경시 사용
DDL(데이터 정의어; Data Definition Language)	CREATE ALTER DROP	Object의 생성과 변경시
TCL(트랜잭션 제어어; Transaction Control Language)	COMMIT ROLLBACK SAVEPOINT	Transaction 종료 및 취소
DCL(데이터 제어어; Data Control Language)	GRANT REVOKE	권한 부여 및 취소

# 오라클 데이터베이스의 자료형

---

자료형	설명	크기
NUMBER(n)	숫자 데이터	최대 n byte 정수
CHAR(size)	고정길이 문자열	최대 255문자
VARCHAR2(size)	가변 길이 문자열 데이터	영문 4,000 문자 (한글 2,000 문자)
DATE	날짜형(형식은 지정 가능)	

---



# Class 클래스

---

- 이 클래스는 일반적으로 클래스의 객체에 대한 정보를 갖고 있는 메타 클래스이다.
  - 프로그램이 실행시에 객체를 생성하려면 JVM은 그 Type의 Class 객체가 메모리에 로드되었는지 먼저 확인하고, 로드되지 않았다면 class 파일을 찾아서 로드한다.
-

# 1. 드라이버의 로드

---

- ❑ `Class.forName("oracle.jdbc.driver.OracleDriver");`
  - ❑ `forName()` 메소드는 클래스이름을 나타내는 String 인자를 받으며, 그 클래스의 정보를 갖고 있는 Class 객체 참조를 리턴한다
  - ❑ static Class `forName(String className)` throws **ClassNotFoundException**
-

# OracleDriver.class 파일

---

[참고]

C:\Woraclexe\Wapp\Woracle\Wproduct\W11.2.0\server\Wjdbc\Wlib  
폴더에 **ojdbc6.jar** 파일이 존재한다.

압축을 풀면 oracle\Wjdbc\Wdriver 폴더에  
OracleDriver.class 파일 존재함.

---

## 2. Connection 생성하기

---

□ DriverManager 클래스는

- Connection 클래스 객체 생성 역할 => getConnection()  
메소드 이용

▶ Connection 클래스 객체는 데이터베이스와의 실제 연결을 객체 모델링한 것이기 때문에 매우 중요하다.

이 Connection 클래스 객체가 생성되어야, 비로서 DB에 접근 가능  
<사용 예>

```
Connection conn = null;
```

```
conn = DriverManager.getConnection(url,user,passwd);
```

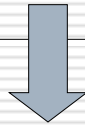
---

# JDBC URL

---

- JDBC URL은 데이터베이스에 대한 다양한 정보를 포함하고 있다.
- JDBC URL 구조

JDBC:<서브프로토콜>:<데이터원본식별자>



oracle:thin:@IP주소:포트번호

---

### 3. 실제 SQL 문의 실행

---

- Statement 클래스에 의해서 수행
- 먼저, Connection 클래스 객체를 이용해서 Statement 객체 생성  
=> createStatement() 메소드 사용

<사용 예>

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

---

## 4. 데이터의 검색(Select)

---

- ❑ 데이터를 검색하기 위해서는 Select문을 사용하여, Statement 객체에 요청해주어야 한다.
- ❑ Select문은 이미 존재하는 데이터를 검색하는 기능이기 때문에 executeQuery( ) 메소드를 실행해 주어야 한다.
- ❑ executeQuery( ) 메소드의 결과로 ResultSet 이라는 객체가 얻어진다.

<사용 예>

```
ResultSet rs = null;  
rs = stmt.executeQuery(query);
```

## 데이터의 검색(Select)

- 보통, next() 메소드를 실행하여 레코드간을 움직이고, 각 레코드의 필드에 대한 접근은 getXxx() 메소드를 이용하여 접근한다.

<사용 예>

```
while(rs.next()) {  
    System.out.println(rs.getInt("empno")+"Wt"+  
        rs.getString("ename") + "Wt" +  
        rs.getFloat("sal") + "Wt" +  
        rs.getInt("deptno"));  
}
```



# 실습 전 확인 작업

---

(1) 명령 프롬프트에서

> netstat -a

TCP      컴퓨터이름 : 1521      컴퓨터이름 : 0      LISTENING

(2) [시작] – 제어판- 시스템 및 보안 - 관리도구 – [서비스] 에서 확인

	상태	시작유형
	-----	-----
OracleXETNSListener	실행	자동
OracleServiceXE	실행	자동

---

# 자바에 Mybatis 프레임워크 적용하기

---

- FactoryService 클래스 작성하기 전에  
WebContent/WEB-INF/lib 폴더에 mybatis-3.5.5.jar 파일을 넣어 둔다.  
그래야 SqlSessionFactory 인터페이스가 import 된다.
  - Mybatis를 이용하여 DAO를 구현하려면 SqlSession 객체가 필요하다.  
그런데 이 SqlSession 객체를 얻으려면 SqlSessionFactory 객체가 필요하다.
  - 모든 Mybatis 애플리케이션은 SqlSessionFactory 인스턴스를 사용한다.  
SqlSessionFactory 인스턴스는 SqlSessionFactoryBuilder를 사용하여 만들 수 있다.
-