

프로그래밍언어응용

■ 개발편의성 측면에 따른 분류

저급언어 : 컴퓨터가 직접 이해할 수 있는 언어로 실행속도는 빠르나 기계마다 기계어가 상이하여 호환성이 없고 유지관리가 어렵다

( 기계어, 어셈블리어 )

고급언어 : 개발자가 이해할 수 있도록 소스코드를 작성할 수 있는 언어로 실행을 위해서는 번역과정이 필요하다

(C,C++, JAVA, PYTHON, .NET 등 대대수 언어)

■실행 및 구현방식에 따른 분류

－ 명령형 언어: 컴퓨터가 동작해야 할 알고리즘을 통해 프로그램의 상태를 변경시키는 구문에 중점을 둔 방식으로 FORTRAN , C

－ 함수형언어: 함수의 응용을 강조하면서 자료의 처리는 수학적인 함수의 연산으로 취급하고 상태와 가변 데이터는 멀리하는 방식으로 LISP, Scala

－ 논리형언어 : 논리문장을 이용하여 프로그램을 표현하고 조건이 만족되면 연관된 규칙이 실행되는 방식으로 PROLOG등이 속한다

－ 객체지향언어: 객체 간의 메시지 통신을 이용하여 동작하는 방식으로 JAVA, C++등이 속한다

메시지란? 객체간의 매서드 호출을 통해서 이루어진다

■ 빌드 방식에 따른 분류

프로그램의 소스코드가 실행 가능한 형태로 변하는 과정을 빌드라고 한다

빌드방식에 따른 분류

－ 컴파일언어 : 소스코드가 기계어 실행 파일로 빌드되는 방식이다 , C, C++

－ 인터프리터 언어 : 소스코드를 한 줄씩 번역하여실행하는 방식이다 phthon

－ 바이트코드언어 : 컴파일을 통해 가상머신이 번역할 수 있는 byte code 로 변환되며 가상머신은 다신 Native OS가 이 해할 수 있는 기계어로 번역하는 방식이다 , JAVA

■ 컴파일러와 인터프리터

－ 컴파일러 동작방식

컴파일러 : 소스코드를 목적코드 (기계가 이해할 수 있는 코드)로 변환하여 실행하는 방식

: 소스코드 전체 기계어로 미리 만들고 실행시키는 방법

인터프리터 : 문장단위로 읽어들이 해석을 하여 실행하는 방식 : 라인단위의 해석 및 실행

■ 컴파일러와 인터프리터 비교

구분	컴파일러	인터프리터
개발편의성	코드를 수정하고 실행이 필요한 경우 재컴파일 필요	코드수정 후 즉시 실행 가능
번역단위	전체소스코드	문장단위
실행 파일 및 속도	실행파일 생성 처리속도 빠름	실행파일 미생성 처리속도 느림
오류 확인 및 처리	전체코드에 대한 컴파일 수행시 발생한 오류 확인 가능	프로그램 실행 후 오류가 발생한 문장 이후의 코드는 실행하지 않음
파일 용량 및 보안	실행파일전체를 처리해야 하므로 용량이 크며 원시코드 유출가능성이 상대적으로 낮음	용량이 상대적으로 작고 원시 코드의 유출가능성이 높음
주요언어	c, c++ , java	phthon, javasript, ruby

## ■ 주요프로그래밍 언어

시기	
1960년 이전	ASSEMBLY , FORTRAN , LISP
1960년대	COBOL , PL/I, BASIC
1970년대	PASCAL, C, SMALLTALK, PROLOG
1980년대	ADA, C++ , PERL
1990년대	RUBY , JAVA, JAVASCRIPT, PHP, VISUAL BASIC,python
2000년 이후	C# , SCALA, GO, CLOJURE, CEYLON, KOTLIN , DART

java의 경우

인터프리터와 컴파일언어 두 부류로 분류할 때는 컴파일언어에 속함

## ■ 프로그램 언어별 설명 및 특성

**JAVA:** c++에 비해 단순하고 분산환경과 객체지향, 보안성을 지원하고 컴파일을 통해 class파일을 생성하며 가상머신에서 실행

**JAVASCRIPT:** html, css와 함께 웹을 구성하는 핵심 요소로 거의 모든 웹 브라우저에 스크립트 엔진이 존재, 웹페이지의 동작 구현이 가능하고 빠른 개발과 확장성이 뛰어나지만 다른 언어에 비해 상대적으로 보안이나 성능이 부족

**PYTHON:** 배우기 쉽고 이식성이 좋은 언어로 다양한 함수들도 많이 제공되며 최근 트렌드와 맞물려 스타트업과 글로벌 기업 등에서도 많이 사용하는 언어

**GOLANG:** Google에서 만든 언어로 짧게는 해라고도 부르며 내장라이브러리가 많이 지원

**KOTLIN:** JAVA보다 간결한 문법을 가지고 있는 JVM기반의 언어로 JAVA와의 상호운영이 100%지원되고 2019 구글이 안드로이드의 공식언어로 채택

**R:** 통계 및 그래프 작업을 위한 인터프리터프로그래밍 언어로 수많은 통계관련 패키지가 개발되어 빅데이터 분석 및 기계학습등에 유용

## ■ 절차형언어

: 절차(실행순서)가 중점이 됨

## ■ 객체지향언어

: 객체의 종류와 속성등에 더 중점을 둔 개발 패러다임

절차지향언어 : 절차(실행순서)가 중점이 됨( 절차에 데이터가 흘러들어 가는 모습) ( 절차가 중심)

=> `print(학생정보)`

객체지향언어 : 객체의 종류와 속성 등에 중점을 둔 개발 패러다임 ( 데이터: 속성들 중심)

=> `학생정보.print()`

## ■ 객체지향의 구성요소

- **객체** : 객체와 속성, 매서드로 구성된 클래스의 instance 의미
- **클래스** : 공통된 특성(속성, 연산)을 가지는 객체 집합으로 객체타입을 정의하고 생성하는 틀이다
- **메시지** : 객체 간의 상호작용은 메시지를 통해 이루어지며 메시지는 객체에서 객체로 전달된다  
(메시지는 객체의 매소드를 호출함으로써 이루어진다)

## ■ 객체지향의 특징

- **캡슐화** : 연관된 데이터와 데이터를 처리하는 함수를 함께 묶어 외부에는 필요한 인터페이스만을 노출함
- **정보은닉** : 다른 객체에서 자신의 필드 및 매소드 등을 은닉하고 자신의 연산만을 통하여 접근을 허용함 ( private, public으로 내부와 외부로 지정함)
- **추상화** : 불필요한 부분은 생략하고 주어진 문제나 시스템 중에서 중요한 부분에 집중하여 모델링한다
- **상속** : 하위 클래스에서 상위 클래스의 속성과 매서드를 물려 받는 기법으로 클래스와 객체의 재사용이 가능하다.
- **다형성** : 하나의 메시지에 대해 각 객체의 고유한 방법으로 응답한다.

## ■ 절차지향과 객체지향의 차이

구분	절차지향	객체지향
구성	함수	객체
구현방식	전체적인 기능 동작을 고려한 각 단계별 기능 구현	필요한 속성의 객체 모델링 후 상호작용기능 구현
접근제어	없음	가능(public, private)
상속/다형성	없음	가능
보안성	낮음	높음
장점	프로그램 흐름을 쉽게 추적 가능 복잡도가 단순하고 실행속도가 상대적으로 빠름	뛰어난 재사용 및 확장성, 유지보수의 용이성 보유 규모가 크고 협업이 필요한 대형 프로젝트 적합
단점	큰프로젝트의 경우 구조 복잡, 중복코드 발생, 유지성이 높아 신규 기능 추가 등이 어려움	상대적으로 속도가 느리고 메모리 사용율이 높음 설계과정에 많은 시간 소요
언어	C, Fortran, Pascal	java, C++ , Python

## ■ 프로그래밍 언어별로 사용 용도

웹(프런트엔드 ) : html, css, javascript , jsp, php, asp

모바일개발

-안드로이드 : java, kotlin

-iOS : Object C, Swift

데스크톱: C, C++, C#

게임 : C, C++, Unity

AI/빅데이터분석: Python, R등

## ■ 코드 인스펙션( Code Inspection)

:프로그램 소스코드를 실행하지 않고 코드상에서의 잠재적인 오류와 표준 미준수 등의 결함을 사전에 식별하여 개선하는 공식적인 리뷰(Review) 기법

### - 인스펙션 수행 절차

계획( Planning ) -> 개관( Overview ) -> 준비( Preparation ) -> 검토회의( Meeting ) -> 재작업( Rework ) -> 추적 ( Follow-up)

## ■ 코드리뷰 기법 비교

동료검토, 워크스루, 테크니컬리뷰, 인스펙션 등의 기법을 선택하여 적용할 수 있다

■ **동료검토**: 별도의 절차 없이 비공식적으로 계획 없이 임의적으로 실시되며, 개발자가 동료와 코드 및 산출물의 결함을 식별하는 기법

■ **워크스루**: 개발자가 리뷰의 주제와 시간을 정해서 발표를 하고 동료들로부터 의견이나 아이디어를 듣는 시간을 가질 수 있으며, 사례에 대한 정보 공유나 아이디어를 수집을 위해서 사용될 수 있다

■ **Inspection**: 역할과 절차, 체크리스트를 기준으로 결함을 식별하는 공식적인 리뷰 기법

## ■ 리팩토링( Refactoring)

: SW의 원래 기능은 유지하면서 소스코드의 내부 구조를 수정 및 보완하여 가독성, 성능 향상 및 로직을 개선하는 기법

리팩토리 대상	리팩토링 방법
중복된코드(Duplicated Code)	
긴 매소드(Long Method)	매서드의 내용 코드 줄이 긴 것을 의미함 매서드를 적정 수준의 크기로 나누어야 한다
긴 파라미터 리스트 ( Long Parameter List)	파라미터 개수를 줄여야 한다 , 파라미터가 너무 길면(많으면) 사용하기 불편하다
게이른 클래스 (Lazy Class)	자식클래스와 부모 클래스의 차이가 없으면 합친다
주석(Comment)	
메시지 체인(Message Chain)	특정 객체를 얻기 위한 다수 객체는 간소화한다
미들맨( Middel Man)	다른 클래스로 위임하는 역할만 담당하는 클래스는 제거를 검토한다.
불완전 라이브러리( Incomplete Library)	불완전 시 필요 부분 추가 구성한다
스위치 명령문(Switch Statments)	지나치게 많은 case를 사용하는 Switch문장은 코드 중복의 신호이다

## 디자인 패턴 (생성패턴, 구조패턴, 행위패턴 구분하기)

생성패턴: Factory Method, Abstract Factory, Builder, Prototype, Singleton

구조패턴: Adapter, Bridge, Composite, Decorator, Facade, Proxy

행위패턴: Interpreter, Template, Command, Iterator, Mediator, Observer, Strategy, Visitor

## ■ 프로그래밍 최적화 기법을 활용하여 최적화를 수행함

### 1.공통코드를 중복실행하는 경우 한번만 실행될 수 있도록 변경한다

최적화 수행 전	최적화 수행 후
a= speed* time + k;	tmpData = speed* time;
b= speed* time *m;	a= tmpData +k ;
c= speed* time /m;	b= tmpData *m ;
	c= tmpData /m;

2. 반복문 안에 변하지 않는 값이 반복 수행되지 않도록 변경한다.

최적화 수행 전	최적화 수행 후
<pre>int[] arr= {1,2,3,4}; int height=2, length=3, depth=4; for( int i=0; i&lt; arr.length; i++){     arr[i] *= height * length * depth; }</pre>	<pre>int[] arr= {1,2,3,4}; int height=2, length=3, depth=4; int tmp = height* length* depth; for( int i=0; i&lt; arr.length; i++){     arr[i] *= tmp; }</pre>

3. 수행작업에 대한 알고리즘 최적화를 진행한다

최적화 수행 전	최적화 수행 후
<pre>for( int i=1; i&lt; Cnt; i++){     SUM +=i; }</pre> <p>1~10까지의 합: 55</p>	<p>SUM= ( Cnt* ( Cnt+1) )/2;</p> <p>10 *(10+1) =&gt; 110 /2 =&gt; 55</p>

4. API를 이용한 최적화를 진행한다  
( 표준 라이브러리 )

최적화 수행 전	최적화 수행 후
<pre>int[] arr={ 5,7,10,14,9}; int max=0; for( inti=1; i&lt;arr.length; I++){     if( arr[i] &gt; imax) {         imax = arr[i] ;     } }</pre>	<pre>import java.util.Arrays;  int[] arr= { 5,7,10,14,9} ; Arrays.sort(arr); imax = arr[arr.length-1];</pre>

5. 문자열 연산시 String 대신 StringBuilder 이용하여 최적화를 수행한다

:String은 불변의 특성을 가지고 있어 문자열이 할당된 메모리 공간이 변하지 않아 + 연산자들을 이용하여 문자열을 추가하는경우 새로운 객체를 생성하여 시간과 자원을 사용하게 된다. 다만 동기화가 필요한 경우라면 StringBuffer를 사용한다

최적화 수행 전	최적화 수행 후
<pre>int Cnt=10000; String str=""; for( int i=1 ; i&lt;=Cnt; i++){     str+="string+ i; }</pre>	<pre>int Cnt=10000; StringBuilder builder = new StringBuilder(); for( int i=1; i&lt;=Cnt; i++){     builder.append("string").append(i); }</pre>

6. 최적화 수행 후 수행시간을 측정하여 최적화 여부를 확인한다

수행시간( 적용전)	수행시간 (적용 후)
<pre>int Cnt=1000000; long start =System.nanoTime(); String str=""; for(int i=1;i&lt;=Cnt; i++){     str+= "string" +i; } long end = System.nanoTime(); long duration = end- start; System.out.println( duration/ 1000000000.0);</pre>	<pre>int Cnt=1000000; long start =System.nanoTime(); StringBuilder builder = new StringBuilder();  for(int i=1;i&lt;=Cnt; i++){     builder.append("string").append(i); } long end = System.nanoTime(); long duration = end- start; System.out.println( duration/ 1000000000.0);</pre>
결과: 8.9792875	결과: 0.0076047

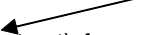
나노 초 (ns 또는 nsec으로 표기)는 십억 분의 1초

■ 프로그래밍 언어데 적용 가능한 패턴을 확인하여 적용한다

싱글톤패턴

파일	소스코드
Singleton.java	<pre>public class Singleton{     private static Singleton instance;     private Singleton(){ } // 생성자를 private으로 하여 생성자를 호출할 수 없게 한다     public static Singleton getInstance(){ // 객체생성없이 매서드 사용을 위해 static으로 선언함         if( instance == null) {             System.out.println("create instance");             instance = new Singleton();         }         return instance;     } }</pre>
SingletonTest.java	<pre>public class SingletonTest{     public static void main(String[] args) {         Singleton instance1 = Singleton.getInstance();         Singleton instance2 = Singleton.getInstance();         System.out.println( instance1 == instance2 );     } }</pre>
결과값	<pre>create instance true (같은 객체를 의미함)</pre>
적용장점	서로 다른 Class에서 instance를 호출하여도 최초1회만 실행되며 반환된 인스턴스의 값은 동일하다 따라서 동일한 데이터가 설정 파일이나 DB조회를 통해 빈번하게 참조되는 경우에는 싱글톤 패턴을 통해 1회만 수행한 후 동일 인스턴스를 사용함으로써 자원 사용율을 향상시킬 수 있다.

■ 스트래티지 패턴( Strategy Pattern) : 같은 전략( 인터페이스)을 따르면 다른 코드로 쉽게 바꿀 수 있다.  
 : 사용자가 자신에게 맞는 전략을 취사선택하여 로직을 수행할 수 있는 방법  
 ( 향후 확장성에 대한 고려가 필요한 경우 소스코드를 참조하여 스트래티지 패턴을 적용한다)

파일	소스코드
Animal.java	<pre> public interface Animal{     public void printName(); } </pre>
AnimalCall.java	<pre> public class AnimalCall{     private Animal animal;     public void setAnimal( Animal animal) {         this.animal = animal;     } } </pre> <div> <div>// Animal 자료형이면 ok</div>  </div>
Cat.java	<pre> public class Cat implements Animal{     public void printName(){         System.out.println( "Cat!!" );     } } </pre>
Dog.java	<pre> public class Dog implements Animal{     public void printName(){         System.out.println(" Dog!!");     } } </pre>
StrategyTest.java	<pre> public class StrategyTest{     public static void main(String[] args){         AnimalCall anicall = new AnimalCall();         anicall.setAnimal( new Dog());         anicall.callName();         anicall.setAnimal( new Cat());         anicall.setName();     } } </pre>
결과값	Dog!! Cat!!
적용장점	전략패턴의 가장 큰 장점은 확장성으로 Animal인터페이스를 상속해서 같은 기능(책임)=>인터페이스 만 수행할 수 있다면 코드 영향 범위를 최소화하면서 얼마든지 다른 동물도 추가할 수 있다.

gof 디자인패턴의 종류

구분	종류	설명
생성패턴	Factory Method	어떤 클래스의 인스턴스를 만들지 서브클래스에서 결정하도록 책임을 위임하는 패턴
	Abstract Factory	관련있는 객체들을 모아서 팩토리로 만들고 조건에 따라 팩토리중에서 선택하게 하는 패턴
	Builder	객체의 생성 단계를 캡슐화하여 구축 공정을 동일하게 이용하도록 하는 패턴
	Prototype	원형이 되는 객체를 복제하여 새로운 객체를 생성하는 패턴
	Singleton	인스턴스를 한번만 생성하여 메모리에 저장하여 사용함으로써 하나의 인스턴스를 보장하는 패턴
구조패턴	Adapter	서로다른 인터페이스를 가진 클래스를 함께 사용할 수 있도록 하는 패턴
	Bridge	기능과 구현을 분리하여 독립적으로 변형과 확장이 가능하도록 결합도를 낮춘패턴
	Composite	여러 개의 객체들로 구성된 복합 객체와 단일 객체를 클라이언트에서 동일하게 사용하는 패턴
	Decorator	객체에 추가적인 기능을 유연하게 확장하는 패턴
	Facade	복잡한 클래스들을 묶어 통합된 인터페이스를 제공하는 패턴
	Proxy	실제 객체가 아닌 가상 객체를 통해 기능을 처리하는 패턴
행위패턴	Interpreter	문법 규칙을 정의하고 해석하는 패턴
	Template	공통 매소드를 상위 클래스에서 정의, 세부사항은 하위클래스에서 구현하는 패턴
	Command	요청을 캡슐화하여 여러 기능을 실행할 수 있는 패턴
	Iterator	컬렉션의 구현은 노출시키지않고 요소에 접근할 수 있는 패턴
	Mediator	객체들 간의 상호작용 행위를 캡슐화하여 관리하는 패턴
	Memento	객체의 상태저보를 저장하고 원하는 시점의 이전 상태로 복원할 수 있는 패턴
	Observer	객체의 상태가 업데이트되면 객체에 의존하는 다른 객체에 알리고 자동으로 내용을 갱신하는 패턴
	Strategy	행위를 클래스로 캡슐화하여 필요에 따라 동적으로 대체가 가능하도록 한 패턴
	Visitor	데이터와 구조를 분리하여 구조를 수정하지 않고 새로운 기능을 추가할 수 있는 패턴



■ 프로그래밍 언어 특성에 따른 보안 취약점을 검토하고 최적화 한다

- 1) 입력 데이터 검증 및 표현 측면의 보안 약점을 확인하고 최적화 한다
  - SQL삽입공격에 대한 보안 취약점을 검토하고 제거한다 (preparedStatement사용)
  - 경로조작 및 자원삽입 공격에 대한 보안 취약점을 검토하고 제거한다
  - 파일 업로드에 대한 보안 취약점을 검토하고 제거한다
  - 정수 오버플로에 대한 보안 취약점을 검토하고 제거한다
- 2) 보안 기능 측면의 보안약점을 확인하고 최적화한다
  - 중요정보를 적절한 인증없이 조회하거나 변경할 수 있는 취약점이 존재하는지 확인하고 인증을 추가한다
  - 개인정보 및 비밀번호 등의 중요 정보를 전송하거나 저장 시 암호화하지 않아 정보가 노출가능한 취약점이 존재하는지 확인하고 암호화하여 처리한다
  - 소스코드에 중요정보( 암호화키, 비밀번호)를 직접 코딩하는 경우 소스코드 유출시 중요정보가 노출되고 주기적 변경이 어려운 취약점이 존재하는지 확인하고 중요정보는 별도의 설정파일에 관리하도록 개선한다
- 인증시도 수를 제한하지 않아 공격자가 반복적으로 임의 값을 입력하여 계정권한을 획득 가능한 취약점이 존재하는지 확인하고 인증시도를 제한하도록 개선한다
- 3) 예외처리, 코드 오류 등에서 발생할 수 있는 보안 약점을 확인하고 최적화 한다
- 4) 언어 특성에 따라 발생할 수 있는 성능과 가용성 측면의 문제를 확인하여 최적화를 수행한다.
- 5) 소스코드품질 측면에서의 최적화를 수행한다
  - 프로그램 명명 가이드를 기준으로 최적화를 수행한다
  - 프로그래밍 코딩 가이드를 기준으로 최적화를 수행한다
  - 불필요한 코드가 존재하는지 파악하여 최적화를 수행한다
  - 정적 분석 도구를 활용하여 최적화를 수행한다
- 정적분석도구 : PMD ( 교재 p.39 )
- 6) 최적화가 완료된 프로그램 소스에 대한 테스트를 수행
  - 단위테스트
  - 통합테스트
  - 회귀테스트( 최적화 이전에 정상적으로 동작했던 기능들이 수정된 후에도 정상적으로 동작하는지 확인한다)

시큐어코딩: 소프트웨어 개발함에 있는 개발자의 실수, 논리적 오류 등으로 인해 sw에 내포될 수 있는 보안 취약점을 배제하기 위한 코딩 기법으로 관련 가이드는 한국인터넷진흥원자료실에서 다운로드 받을 수있다

PMD의 경우 전자정부 표준 프레임워크에서는 Code Inspection을 위한 룰셋으로 논리오류/구문오류/참조오류 영역을 대상으로 하는 총 39개의 룰을 표준으로 선정하고 있다.

■ 라이브러리 활용하기

라이브러리:

라이브러리는 프로그램을 효율적으로 개발할 수 있도록 필요한 기능이 구현된 프로그램을 모은 집합체이며 일반적으로 설치파일과 도움말, 예시 코드등을 제공한다.

■ 라이브러리 유형

- 표준라이브러리: 프로그래밍 언어와 함께 제공되는 라이브러리로 표준 라이브러리를 이용하면 별도의 파일 설치 없이 다양한 기능을 이용할 수 있다
- 외부라이브러리:
  - 별도의 파일을 설치하여 사용할 수 있는 라이브러리를 의미한다
  - 누구나 개발하여 사용하거나 공유할 수있다 ( MyCalculator , MyMath) 관련 커뮤니티나 인터넷 검색등을 이용하여 공유된 라이브러리를 사용할 수 있다.

■ 코드결합방식에 따른 유형

**정적라이브러리** : 프로그램을 빌드할 때 라이브러리가 제공하는 코드를 실행 파일에 넣는 방식으로 컴파일러가 원시 소스를 컴파일할 때 참조되는 모듈이다.

**동적라이브러리** : 공통적으로 필요한 기능들을 프로그램과 분리하여 필요시에만 호출하여 사용할 수있도록 만든 라이브러리를 의미한다  
필요할 때만 해당 라이브러리를 메모리로 불러들일 수 있어 실행파일의 크기를 줄일 수 있고  
사용이 끝나면 메모리에서 제거되어 효율적인 메모리 사용이 가능하다

■ 라이브러리와 유사용어설명

Library : 코드 재사용 및 부품을화를 위하여 필요한 기능에서 호출하여 사용할 수 있도록 제공되는 모듈의 집합

( jQuery, Class, Jar )

Framework : 응용프로그램 표준 구조를 구현하기 위한 클래스와 라이브러리 모임( Spring, Django)

Architecture : 여러 가지 컴퓨터 구성요소들에 대한 전반적인 기계적 구조와 이를 설계하는 방법

Platform :여러가지 기능을 제공해 주는 프로그램 실행이 가능한 공통 실행환경으로 플랫폼 위에 다른 플랫폼이 존재할 수 있음  
( Window, Linux , JVM )

#### ■ 자바대표 외부라이브러리 종류 및 역할

Apache Commons	자바로 구현된 라이브러리를 모은 프로젝트로 codec, compress, collections , CSV ,IO, Lang. logging등의 컴포넌트 제공
Google Guava	Apache Commons과 유사한 모듈식이며 컬렉션, I/O, 캐싱, 해싱 등의 많은 독립라이브러리 제공
Jackson	java Object를 Json으로 변경하거나 JSON을 java object로 변환 xml/yaml/csv등 다양한 형식의 데이터를 지원
Log4j 2	대표적인 로깅 라이브러리로 비동기 로깅을 통한 성능향상 및 플러그인 아키텍처, 클라우드 지원
Mockito	동작제어가 가능한 가짜 객체를 지원하는 테스트 프레임워크
Hibernate	java 클래스를 데이터베이스 테이블/컬렉션과 매핑

#### ■파이썬 대표 외부라이브러리

종류	역할
pandas	데이터분석시 데이터의 수집, 조작, 분석을 위해 사용되는 라이브러리
numpy	벡터, 행렬 등 수치연산을 수행하는 계산 과학 분야 라이브러리
ploty	최신의 반응용, 브라우저 기반의 차트를 그릴 수 있는 시각화 라이브러리
matplotlib	데이터를 차트나 플롯으로 그려주는 라이브러리
scipy 싸이피	분석 및 과학 , 엔지니어링을 위한 여러 기본적인 작업을 위한 라이브러리
openCV	얼굴인식 등의 영상처리를 위한 실시간 이미지 프로세싱 오픈소스 라이브러리
fbprophet	입력받은 데이터를 기반으로 다음값을 예측
dlib	기계학습 및 이미지 프로세싱, 얼굴 인식 등을 위한 라이브러리

객체지향의 첫 단추 !!!!

캡슐화 => class로 표현함  
(데이터(변수) +데이터를 처리하는 기능 (함수)를 묶어서 제공함)  
접근제한자로 사용하여 외부인터페이스만을 이용할 수 있도록 함

라이브러리란?

코딩할때 유용하게 쓸 수 있는 기능들을 모아 놓은것 : 광일님  
만들어진 클래스를 외부에서도 사용할 수 있도록 하는 것 : 헤리님  
자바 자체에서 지원하는 기본 기능 : 선명님  
필요한 기능을 저장해둔 모듈모임: 현조님  
모듈집합: 민주님  
필요한 기능을 호출하여 사용할 수 있도록 하는 것 : 정현님  
  
유용한 기능을 사용할 수 있도록 만든 프로그램의 집합체 : 의수님  
프로그램을 효율적으로 개발할 수 있도록 편리한 기능이 구현된 프로그램을 모은 집합체:세희님,범준님

라이브러리란

- : 남이 만들어서 제공하는 코드 (기능) , 함수나 클래스 단위로제공됨  
컴파일하여 압축된형태로제공됨 ( .jar 파일형태)
- 표준라이브러리 언어자체에서 기본으로 제공되는 기능
- 외부라이브러리 : 특정기능을 수행하기 위해 사용함 , 다운로드 받아야 함

라이브러리 결합방식

:정적라이브러리  
:동적라이브러리