제목 : 7/6 일지

1. 개발환경 세팅 Connect 보기
2. 인턴 연수 계약서, 정보보안 서약서, 임직원 서약서
3. NSA 계정 발급 안됨
4. Spring, JAVA, Java Script

스프링 부트(Spring Boot)로 시작하는 프레임워크(Framework)

절차형 언어 -> 객체지향 언어. => 설계의 중요성 증가

:객체들이 서로 메세지를 주고 받고 서로를 사용하면서 작업을 수행하기 때문에 객체를 어떻게 설정하고 객체 간 관계를 어떻게 정할 것인지 이런 설계 작업이 핵심이다.

이러한 구현 방법을 잘 정리해 놓은 것이 디자인 패턴

Spring boot

네이버 코드 컨벤션(JAVA)

가이드의 취지

이 가이드는 프로젝트 구성원 간의 형식에 대한 합의 수준을 높히기 위한 목적

1. 기본적인 코드 형식에 대해서 프로젝트마다 초기에 반복 논의하는 시간을 줄인다.
2. 코드 리뷰 과정에서 핵심 로직에 집중할 수 있게 한다.

파일 공통 요건
-> 파일 인코딩은 UTF-8
[encoding-utf8]
모든 소스, 텍스트 문서 파일의 인코딩을 통일한다.

-> 새줄 문자는 LF

Unix 형식의 새줄 문자(newline)인 LF(Line Feed, 0x0A)을 사용한다.

-> 파일의 마지막에는 새줄

파일의 마지막은 새줄 문자 LF로 끝나야 한다.

이름(Naming)
-> 식별자에는 영문/숫자/언더스코어만 허용
변수명, 클래스명, 메서드명 등에는 영어와 숫자만들 사용한다. 상수에는 단어 사이의 구분을 위하여 언더스코어(_)를 사용한다.
정규표현식에 부합해야한다.

-> 한국어 발음대로의 표기 금지
한국어 고유명사는 예외이다.

-> 대문자로 표기할 약어 명시
HttpApiUrl 약어가 없는 경우 앞만 대문자

HttpAPIUrl API만 약어가 있다

HTTPAPIURL 전부다 약어

-> 패키지 이름은 소문자로 구성
단어별 구문을 위해 언더스코어나 대문자를 섞지 않는다.

-> 클래스/인터페이스 이름에 대문자 카멜표시법 적용
단어의 첫 글자를 대문자로 시작하는 방법이다. =:> 파스칼표기법이라고도 불림!

Ex) public class AcessToken

-> 클래스 이름의 명사 사용(명사절가능)

-> 인터페이스 이름에 명사/형용사 사용(명사절,형용사절 가능)

-> 테스트 클래스는 'Test'로 끝남
Ex) public class WatcherTest

-> 메서드 이름에 소문자 카멜표기법 적용
예시가 없으니 찾아보기!
첫 번째 단어를 소문자로 작성하고 이어지는 단어의 첫 글자를 대문자로 작성한다. 테스트 클래스의 메서드 이름에서는 언더스코어를 허용한다.
아마두? =:> doHomework()

-> 메서드 이름은 동사/전치사로 시작

기본적으로 동사를 사용한다!

다른 타입으로 전환하는 메서드나 빌더 패턴을 구현한 클래스의 메서드에는 전치사를 쓸 수 있다.

동사 renderHtml()

전환메서드 toString()

Builder 패턴 적용한 클래스의 메서드의 전치사 witrhUserId(String id)

-> 상수는 대문자와 언더스코어로 구성
Public final String POSTA_CODE = “111”;

-> 변수에 소문자 카멜표기법 적용
Private int accessToken

-> 임시 변수 외에는 1 글자 이름 사용 금지
메서드 블럭 범위 이상의 생명 주기를 가지는 변수에는 1글자로 된 이름을 쓰지 않는다. 반복문의 인덱스나 람다 표현식의 파라미터 등 짧은 범위의 임시 변수에는 관례적으로 1글자 변수명을 사용할 수 있다.

선언(Declarations)
클래스, 필드, 메서드, 변수값, import문 등의 소스 구성요소를 선언할 때 고려해야할 규칙이다.

->소스파일당 1개의 탑레벨 클래스를 담기
```
public class LogParser {
//굳이한파일안에선언해야한다면내부클래스로선언
        class        LogType {
        }
}
```

-> static import에만 와일드 카드 허용
와일드 카드 -> *
나쁜 예 import java.util.*;

좋은 예 import.java.util.ArrayList;
          import.java.util.List;

-> 제한자 선언의 순서

클래스/메서드/멤버변수의 제한자는 Java Language Specification에서 명시한 아래의 순서로 쓴다.
public protected private abstract static final transient volatile synchronized native strictfp -> 이게 순서임! 영어의 형용사 수식 순서 느낌
(Java Language Specification - Chapter 18. Syntax 참조)

-> 애너테이션(annotation) 선언 후 새줄 사용
클래스, 인터페이스, 메서드, 생성자에 붙는 애너테이션은 선언 후 새줄을 사용한다. 이 위치에서도 파라미터가 없는 애 너테이션1개는같은줄에선언할수있다.

예시
@RequestMapping("/guests")
public void findGuests() {}

@Override public void destroy() {}
파라미터가 없는 애너테이션

-> 한 줄에 한 문장
문장이 끝나는 ; 뒤에는 새줄을 삽입한다. 즉, 한 줄에 여러 문장을 쓰지 않는다.

-> 하나의 선언문에는 하나의 변수만
나쁜예 int base, weight;

-> 배열에서 대괄호는 타입 뒤에 선언
String[] names;

-> ‘long’형 값의 마지막에 ‘L’ 붙이기
long base = 54423234211L;
소문자를 안 쓰는 이유는 1이랑 비슷하게 생겨서

-> 특수 문자의 전용 선언 방식을 활용
\b, \f, \n,\r,\t, `\", \\ 와 같이 특별히 정의된 선언 방식이 있는 특수 문자가 있다. 이런 문자들은 숫자를 이용한 \008 이나 `\u0008`와 같은 숫자를 넣은 선언보다 전용 방식을 활용한다.

나쁜 예.
System.out.println("---\012---");

좋은 예.
System.out.println("---\n---");

들여쓰기(Indentation)
들여쓰기는 코드의 계층을 구분하가 위해 추가하는 문자이다.

-> 하드탭 사용
탭(tab) 문자를 사용하여 들여쓴다. 탭 대신 스페이스를 사용하지 않는다.
!!! 이를 잘 준수하기 위해서 스페이스와 탭을 구별해서 보여주도록 에디터를 설정한다.

-> 탭의 크기는 4개의 스페이스
탑 1번 == 스페이스 4번

-> 블럭 들여쓰기
클래스, 메서드, 제어문 등의 코드 블럭이 생길 때 마다 1단계를 더 들여쓴다.

중괄호(Braces)
중괄호({,}) 는 클래스, 메서드, 제어문의 블럭을 구분한다.

-> K&R 스타일로 중괄호 선언
클래스 선언, 메서드 선언, 조건/반복문 등의 코드 블럭을 감싸는 중괄호에 적용되는 규칙이다. 중괄호 선언은 K&R 스타 일(Kernighan and Ritchie style)을 따른다. 줄의 마지막에서 시작 중괄호`{`를 쓰고 열고 새줄을 삽입한다. 블럭을 마친 후에는 새줄 삽입 후 중괄호를 닫는다.

```
public class SearchConditionParser {
    public boolean isValidExpression(String exp) {
        if (exp == null) {
            return false;
        }
        for (char ch : exp.toCharArray()) { ....
        }
        return true;
    }
}
```

한 마디로 줄의 마지막에서 중괄호를 시작하라.

-> 닫는 중괄호와 같은 줄에 else, catch, finally, while 선언
아래의 키워드는 닫는 중괄호 } 와 같은 줄에 쓴다.
•else
•catch, finally
•do-while 문에서의 while

모르겠으면 네이버 코딩 컨벤션(JAVA) 13P를 참조한다.

-> 빈 블럭에 새줄 없이 중괄호 닫기 허용
public void close() {}

-> 조건/반복문에 중괄호 필수 사용
조건,반복문이 한 줄로 끝더라도중괄호를 활용한다.이 문서에 언급된 중괄호의 전후의 공백,제어문 앞 뒤의 새줄 규칙도 함께 고려한다.

```
if (exp == null) {
        return false;
}
for (char ch : exp.toCharArray()) {
        if(ch==0){
                return false;
        }
}
```

줄바꿈(Line-wrapping)

-> 최대 줄 너비는 120

최대 줄 사용 너비는 120자까지 가능하다.

-> packager, import 선언문은 한 줄로

-> 줄바꿈 후 추가 들여쓰기
120자가 넘어서 줄바꿈을 한 경우 다음 줄에서 적어도 1번의 들여쓰기를 한다.

IDE의 자동 포메팅 기능으로 이를 동일하게 맞추fu면 Appendix C의 각 IDE별 설정을 참고한다.

-> 줄바꿈 허용 위치
가독성을위해줄을바꾸는위치는다음중의하나로한다.
•extends 선언 후
•implements 선언 후
•throws 선언 후
•시작 소괄호(() 선언 후
•콤마(,) 후
•.전
•연산자전
•+,-,*,/,%
•==, ?=, >=, >,⇐, <, &&, || •&,|,^,>>>,>>,<<,?
•instanceof

```
public boolen isAbnormalAccess (
        User user, AccessLog log) {

        String message = user.getId() + "|" | log.getPrefix()
                + "|" + SUFFIX;
}
```

빈줄(Blank lines)
->빈 줄은 명령문 그룹의 영역을 표시하기 위하여 사용한다.

```
package com.naver.lucy.util;

import java.util.Date;
```

-> import 선언의 순서와 빈 줄 삽입

import 구절은 아래와 같은 순서로 선언한다.
A.static imports

B.JDK의 클래스
•java.
•javax.

C.외부 라이브러리의 클래스
•org.
•net.
•com.
•그 외의 클래스

D.네이버 내부에서 만든 클래스
•com.nhncorp.
•com.navercorp.
•com.naver.

A,B,C,D 그룹의 사이에는 빈 줄을 삽입한다. 그룹 내부에서 세부적인 구분을 위해 빈줄을 삽입하는 것도 허용한다.

좋은예

```
import java.util.Date;
 import java.util.List;

import javax.naming.NamingException;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.util.Assert;

import com.google.common.base.Function;

import com.naver.lucy.util.AnnotationUtils;
```

이 규칙은 대부분 IDE에서 자동으로 정리해주는 대로 쓰기 때문에 IDE 설정을 일치시키는데 신경을 써야 한다.

-> 메소드 사이에 빈 줄 삽입
메서드의 선언이 끝난 후 다음 메서드 선언이 시작되기 전에 빈줄을 삽입한다.

공백(Whitespace)

-> 공백으로 줄을 끝내지 않음
빈줄을 포함하여 모든 줄은 탭이나 공백으로 끝내지 않는다.

-> 대괄호 뒤에 공백 삽입
닫는 대괄호(]) 뒤에 `;`으로 문장이 끝나지 않고 다른 선언이 올 경우 공백을 삽입한다.

```
int[] masks = new int[] {0, 1, 1};
```

-> 중괄호의 시작 전, 종료 후에 공백 삽입
여는 중괄호({) 앞에는 공백을 삽입한다. 닫는 중괄호(}) 뒤에 else ,catch 등의 키워드가 있을 경우 중괄호와 키워드 사 이에 공백을 삽입한다.

```
public void printWarnMessage(String line) {
        if (line.startsWith(WARN_PREFIX)) {
                ...
        } else{
                ...
        }
}
```

->제어문 키워드와 여는 소괄호 사이에 공백 삽입
if, for, while, catch, synchronized, switch`와 같은 제어문 키워드의 뒤에 소괄호((,)`)를 선언하는 경우, 시작 소괄호 앞에 공백을 삽입한다.

```
if (maxLine > LIMITED) {
    return false;
}
```

-> 식별자와 여는 소괄호 사이에 공백 미삽입
식별자와 여는 소괄호(() 사이에는 공백을 삽입하지 않는다. 생성자와 메서드의 선언, 호출, 애너테이션 선언 뒤에 쓰이 는 소괄호가 그에 해당한다.

```
public StringProcessor() {} // 생성자
@Cached("local")
public String removeEndingDot(String original) {
        assertNotNull(original);
        ...
}
```

-> 타입 캐스팅에 쓰이는 소괄호 내부 공백 미삽입
타입캐스팅을 위해 선언한 소괄호의 내부에는 공백을 삽입하지 않는다.

```
String message = (String)rawLine;
```

-> 제네릭스 산괄호의 공백 규칙

제네릭스(Generics) 선언에 쓰이는 산괄호(<,>) 주위의 공백은 다음과 같이 처리한다.

•제네릭스메서드선언일때만<앞에공백을삽입한다.

•< 뒤에 공백을 삽입하지 않는다.

•> 앞에 공백을 삽입하지 않는다.

•아래의 경우를 제외하고는 `>`뒤에 공백을 삽입한다.

•메서드 레퍼런스가 바로 이어질 때

•여는 소괄호('(')가 바로 이어질 때

•메서드 이름이 바로 이어질 때

```
public static <A extends Annotation> A find(AnnotatedElement elem, Class<A>
type) { // 제네릭스 메서드 선언
        List<Integer> l1 = new ArrayList<>(); // '(' 가 바로 이어질때
        List<String> l2 = ImmutableList.Builder<String>::new; // 메서드 레퍼런스가 바
로 이어질 때
        int diff = Util.<Integer, String>compare(l1, l2); // 메서드 이름이 바로 이어질 때
}
```

-> 콤마/구분자 세미콜론의 뒤에만 공백 삽입
콤마(,)와 반복문(while, for)의 구분자로 쓰이는 세미콜론(;)에는 뒤에만 공백을 삽입한다.

```
for(inti=0; i<length; i++) {
        display(level, message, i)
}
```

-> 콜론의 앞 뒤에 공백 삽입
반복문과 삼항연산자에서 콜론(:)의 앞 뒤에는 공백을 삽입한다. 라벨 선언 뒤에는 아무런 문자열이 없으므로 앞에만 공 백을 삽입한다.
!! 라벨 선언이 먼지 모르겠음.

```
for (Customer customer : visitedCustomers) {
AccessPattern pattern = isAbnormal(accessLog) ? AccessPattern.ABUSE :
AccessPattern.NORMAL;
int grade = evaluate(customer, pattern);
    switch (grade) {
        case GOLD :
            sendSms(customer);
        case SILVER :
            sendEmail(customer);
        default :
          increasePoint(customer);
    }
}
```

-> 이항/ 삼항 연산자의 앞 뒤에 공백 삽입

== 이나 += 같은 것들

-> 단항 연산자와 연상 대상 사이에 공백을 미삽입

++, —, +, -, ~, !

전위/후위 연산자 단항인 경우에 동일하다

++index

-> 주석문 기호 전후의 공백 삽입

주석의 전후에는 아래와 같이 공백을 삽입한다.
 •명령문과같은줄에주석을붙일때 // 앞
•주석시작기호 // 뒤
•주석시작기호 /* 뒤
•블록주석을한줄로작성시종료기호*/앞

```
/*
 *공백후주석내용시작
 */
System.out.print(true); // 주석 기호 앞 뒤로 공백

/* 주석내용앞에공백, 뒤에도공백 */
```