[💡](https://hoons-dev.tistory.com/96?category=1091955" \l "%F-%-F%--%A-%--Java%EC%-D%--%--%EC%-E%A-%EB%-B%A-%EC%A-%--%EC%--%--%--%EB%-C%--%ED%--%B-%--%EC%--%A-%EB%AA%--%ED%--%B-%EC%A-%BC%EC%--%B-%EC%-A%---) **[Java의 장단점에 대해 설명해주세요.](https://hoons-dev.tistory.com/96?category=1091955" \l "%F-%-F%--%A-%--Java%EC%-D%--%--%EC%-E%A-%EB%-B%A-%EC%A-%--%EC%--%--%--%EB%-C%--%ED%--%B-%--%EC%--%A-%EB%AA%--%ED%--%B-%EC%A-%BC%EC%--%B-%EC%-A%---)**

* JVM 위에서 실행되기 때문에 OS에 종속적이지 않고, 독립적으로 실행될 수 있습니다.
* 객체지향 언어로, 객체지향 프로그래밍이 가능합니다.
* 클래스로더에 의해 동적 로딩을 지원합니다. 실행 시 모든 객체를 생성하는 것이 아니라, 필요한 시점에 클래스를 생성합니다.
* 바이트코드로 변환하고, JVM에 의해 기계어로 번역되므로 한 번의 컴파일링을 거치는 언어에 비해서 조금 느립니다.

### [💡 JVM이 무엇인지 설명해주세요.](https://hoons-dev.tistory.com/96?category=1091955#%F-%-F%--%A-%--JVM%EC%-D%B-%--%EB%AC%B-%EC%--%--%EC%-D%B-%EC%A-%--%--%EC%--%A-%EB%AA%--%ED%--%B-%EC%A-%BC%EC%--%B-%EC%-A%---)

* JVM은 Java Virtual Machine의 줄임말로, OS와 Java Application 사이를 중재해주는 가상 머신입니다.
* JVM은 크게 실행에 필요한 클래스를 Runtime Data Area로 링킹 하고 로딩을 해주는 ClassLoader, 실제 실행과 번역을 담당하는 Execution Engine, 별도의 쓰레드로써 힙 메모리의 참조되지 않는 객체를 삭제하는 Garbage Collector, 각 메모리 영역 및 쓰레드가 담겨있는 Runtime Data Area로 구성이 되어있습니다.

**💡 Java Application이 JVM에서 실행되는 과정을 설명해주세요.**

* 1) JVM은 OS로부터 적당한 메모리(Runtime Data Area)를 할당 받음.
* 2) 자바 소스 코드(.java)를 자바 컴파일러(javac)에 의해 바이트코드(.class)파일로 변환
* 3) Class Loader를 통해 .class 파일을 Runtime Data Area로 로딩
* 4) 로딩된 class 파일을 Execution Engine을 통해 해석 및 실행

**💡 JVM의 메모리(Runtime Data Area)구조에 대해 설명해주세요.**

* 메서드(static) 영역
  + 클래스가 사용되면 해당 클래스의 파일(.class)을 읽어들여, 클래스에 대한 정보(바이트 코드)를 메서드 영역에 저장
  + 클래스와 인터페이스, 메서드, 필드, static 변수, final 변수 등이 저장되는 영역입니다.
* JVM 스택 영역
  + 스레드마다 존재하여 스레드가 시작할 때마다 할당
  + 지역변수, 매개변수, 연산 중 발생하는 임시 데이터 저장
  + 메서드 호출 시마다 개별적 스택 생성
* JVM 힙 영역
  + 런타임 시 동적으로 할당하여 사용하는 영역
  + new 연산자로 생성된 객체와 배열 저장
  + 참조가 없는 객체는 GC(가비지 컬렉터)의 대상
* pc register
  + 쓰레드가 현재 실행할 스택 프레임의 주소를 저장
* Native Method Stack
  + C/C++ 등의 Low level 코드를 실행하는 스택

**💡 Garbage Collector가 무엇인지, 어떻게 동작하는지 설명해주세요.**

* 동적으로 할당한 메모리 영역 중, 사용하지 않는 영역을 탐지해 해제하는 역할을 합니다.
* Java에서 동적 변수를 할당하는 영역은 Heap 영역이므로, JVM의 GC의 작동 대상은 Heap 메모리입니다.
* GC를 사용하면 다음과 같은 장단점이 있습니다.
  + 메모리 누수 방지
  + 해제된 메모리에 대해 접근하는 것을 방지
  + 해제한 메모리를 다시 해제하는 것을 방지
  + 개발자가 일일히 동적변수의 제거 선언을 하지 않아도 됨
  + 그러나, 개발자가 GC가 언제 수행되는지 정확하게 알 수 없음
  + 실행 중인 작업을 중단하고, 리소스를 GC 작업에 내줘야 하므로 오버헤드 발생

**💡 Java에서, ==와 equals의 차이는 무엇인지 알고 계시나요?**

* '==' 연산은 참조 비교로, 두 객체가 같은 메모리 공간을 가리키는 지를 확인하는 연산입니다.
* 'equals' 연산은, 두 객체의 내부 값이 같은 지 내용을 비교합니다. 기본 타입에 대해서는 사용할 수 없고, 객체 비교시 override해서 원하는 방식으로 수정이 가능합니다.
* == : 본질까지 같음. (같은 메모리 내의 데이터) / equals : 값만 같음.

**💡 Java의 접근 제한자에 대해 설명해주세요.**

* public : 접근에 제한이 없음
* private : 자기 자신 클래스 내에서만 접근 가능
* default : 동일한 패키지 내에서만 접근 가능
* protected : 동일한 패키지 내에서만 접근 가능 + 상속을 이용한 접근 가능

**💡 Java에서의 데이터 타입은 무엇이 있나요?**

* 원시자료형 타입(Primitive Type)
  + 기본 타입의 크기가 작고 고정적이기 때문에 메모리 Stack 영역 에 저장됩니다.
  + 정수형 : byte, short, int, long
  + 실수형 : float, double
  + 논리형 : boolean
  + 문자형 : char
* 참조 타입(Reference Type)
  + 원시자료형을 제외하고는 모두 참조형이다.
  + String과 박싱 타입인 Integer 등이 있습니다.
  + 참조 타입은 데이터의 크기가 가변적이고, 동적이므로 Heap 영역 에서 관리됩니다.
  + 데이터는 Heap 영역에서 관리되지만 메모리의 주소값은 Stack 영역에 담깁니다.
  + new 키워드를 이용해 객체를 생성하여 데이터가 생성된 주소를 참조하는 타입
  + String과 배열은 일반적인 참조 타입과 달리 new 없이 생성 가능하지만 참조타입입니다.
  + 더 이상 참조하는 변수가 없을 때 GC에 의해 삭제됩니다.

**💡 Java의 hashcode()에 대해 설명해주세요.**

두 객체가 동일한 객체인지 비교할 때 사용하고, Heap 영역에 저장된 객체의 메모리 주소를 반환합니다.

**💡 Java의 Wrapper Class에 대해 설명해주세요.**

원시자료형 타입의 데이터를 객체로 취급해야 할 경우 사용하는 클래스로, 기본 타입의 데이터를 객체로 변환해주는 클래스입니다.

산술 연산을 위해 정의된 클래스가 아니므로, 인스턴스 내의 값을 변경할 수는 없습니다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

**💡 Java의 static 변수에 대해 설명해주세요.**

클래스당 하나만 생성되고, 동일한 클래스의 모든 객체들에 의해 공유됩니다. 다른 객체들이 생기기 전에 이미 생성되고, 프로그램 종료시에 사라집니다.

**💡 Java의 main 메소드가 static인 이유에 대해 알고 계시나요? (public static void main(){String args[]})**

* static 멤버는 프로그램 시작 시, 클래스 로더에 의해 메모리에 로드되어 인스턴스를 생성하지 않아도 호출이 가능하기 때문입니다.
* Runtime Data Area에서 Method Area(=Class Area, Static Area)라고 불리는 영역에 static 키워드를 가진 변수, 메소드가 생성됩니다. 따라서 JVM은 별도로 인스턴스를 생성하지 않아도, Method Area에 로드된 main()을 실행하게 됩니다.

**💡 try-catch-finally의 단점과, 이로 인해 나온 구문에 대해 알고 계신다면 설명해주세요.**

try-catch-finally 구문에서 리소스를 생성하게 되면, 생성은 try에서 하고 반납은 finally에서 하다보니 실수의 발생 여지가 있었습니다. 그래서 나온 구문으로 try-with-resources가 있습니다. try 옆에 괄호로 리소스를 생성해주면, 따로 반납 코드를 작성하지 않아도 자동으로 리소스를 반납합니다.

**💡 Java의 제네릭에 대해 설명해주세요.**

제네릭(Generic)은 클래스 내부에서 타입을 지정하는 것이 아닌, 외부에서 사용자에 의해 지정되는 것을 의미합니다.

특정 타입(Specific)을 미리 지정하는 것이 아니라, 필요에 의한 지정을 그때 그때 할 수 있는 일반 타입(Generic)인 것 입니다.

잘못된 타입이 들어오는 것을 컴파일 단계에서 방지할 수 있습니다.

클래스 외부에서 타입을 지정하기 때문에 따로 타입을 체크하고 변환할 필요가 없어 관리하기 용이합니다.

비슷한 기능을 지원하는 경우 코드의 재사용성이 높다.

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

**💡 Java의 직렬화와 역직렬화에 대해 설명해주세요.**

* 직렬화
  + 자바 시스템 내부에서 사용되는 객체 또는 데이터를 외부의 자바 시스템에서도 사용할 수 있도록 바이트 형태로 데이터를 변환하는 기술
  + 조건
    - * 자바 기본 타입
      * Serializable 인터페이스 상속받은 객체
  + ObjectOutputStream 객체를 이용
* 역직렬화
  + 바이트로 변환된 데이터를 다시 객체로 변환하는 기술
  + ObjectInputStream 객체를 이용

**💡 Error와 Exception의 차이에 대해 설명해주세요.**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**💡 추상 클래스와 인터페이스에 대해 설명해**

**주세요.**

* 추상클래스는, abstract 키워드를 사용해 선언한 클래스로, 추상 메소드를 최소 한 개 이상 포함한 클래스
* 추상메소드는, abstract 키워드를 사용해 원형만 선언 되고 내부 코드는 작성하지 않은 메서드를 뜻합니다.
* 추상클래스 내부에 추상메소드 외의 다른 것들도 추가가 가능하다는 것이 특징이고, 추상클래스의 사용 주 목적은 관련성이 높은 클래스 간의 코드를 공유하고 확장하고 싶은 목적입니다.
* 인터페이스는, interface 키워드를 사용해 선언하며 default와 static을 제외하고는 추상 메소드와 상수만을 포함합니다.
* interface 내부의 모든 메소드는 추상 메소드로, abstract public이 생략되어 있는 상태입니다.
* 상수 필드는 public static final이 생략되어 있습니다.
* 인터페이스는 다중 상속이 가능하며 관련성이 없는 클래스들의 논리적으로 같은 기능을 자신에 맞게 구현을 강제 하는데에 목적을 갖습니다.

**💡 String과 StringBuilder, StringBuffer의 차이에 대해 설명해주세요.**

1) String

새로운 값을 할당할 때마다 새로운 동적 변수(인스턴스)가 생성됩니다. String에 저장된 문자열은, 내부적으로 private final char[] 형태이므로 변경이 불가능합니다. String에 대한 연결 연산으로 '+'를 사용하면, 두 String을 연결한 새로운 객체가 생성됩니다. 가비지 컬렉터가 호출되기 전까지 생성된 String 객체들은 Heap에 계속해서 머물기 때문에 메모리 관리 측면에서 치명적입니다.

2) StringBuilder

String과 다르게 가변성을 가지는 클래스입니다. append(), delete()등의 method를 사용해 동일 객체 내에서 문자열을 변경하는 것이 가능합니다. 동기화를 지원하지 않기 때문에, 멀티 쓰레드 환경에서 사용하는 것은 적합하지 않지만, 동기화를 고려하지 않는 만큼 단일 쓰레드에서의 성능은 StringBuffer보다 뛰어납니다.

3) StringBuffer

String과 다르게 가변성을 가지는 클래스입니다. append(), delete()등의 method를 사용해 동일 객체 내에서 문자열을 변경하는 것이 가능합니다. 동기화를 지원하기 때문에 멀티 쓰레드 환경에서 thread-safe합니다.

**💡 new String("")와 ""의 차이에 대해 메모리의 관점에서 설명해주세요.**

""으로 문자열을 초기화 하게 되면, Java Heap 메모리의 String Pool에 저장이 됩니다.

반면에, new String("")은 조금 다릅니다.

new 키워드를 사용해 새로이 생성했기 때문에, String Pool이 아닌, 그냥 Heap 영역에 각각 생성됩니다. 따라서 msg3와 msg4는 서로 다른 주소를 가리키고 있는 상태입니다. msg3.intern() 메소드를 활용하면 string pool에 등록할 수 있습니다.

굳이 객체로 만들어 GC 대상이 되는 것보다, String Pool로 만들어 활용하는 것이 메모리 효율 측면에서 더욱 유리할 것으로 보입니다.

**💡 Java 리플렉션에 대해 설명해주세요.**

컴파일 시간이 아닌, 런타임 시간에 메모리에 올라간 클래스나 메소드의 정의를 동적으로 찾아서 조작할 수 있는 기술을 의미합니다.

생성자 찾기(getDeclaredConstructor()), 메소드 찾기(getDeclaredMethods()), 필드 찾고 필드 변경(getDeclaredFields, set) 등 런타임 시간에 동적으로 해당 명령을 수행할 수 있도록 도와줍니다.

* 런타임 시점에 다른 클래스를 동적으로 로딩하여 접근할 때 사용
* 클래스와 멤버 필드, 메서드 등에 관한 정보를 얻어야 할 때 사용

**💡 Java의 Stream API에 대해 알고 계시나요? 안다면 아는대로 설명해주세요.**

Collections Type의 데이터를 Stream 메소드로 내부 반복을 통해 정렬 혹은 필터링을 지원해주는 API 입니다.

* parallel 메서드 제공을 통해 병렬처리 가 가능
* 각 스레드가 개별 큐를 가지고 있으며, 놀고 있는 스레드가 있으면 일하는 스레드의 작업을 가져와 수행
* 데이터를 변경하지 않는다(Immutable)
* 원본데이터로부터 데이터를 읽기만 할 뿐, 원본데이터 자체를 변경하지 않는다.
* 작업을 내부 반복으로 처리하므로 불필요한 코딩을 줄일 수 있다.
* 최종 연산 후 stream이 닫히므로 일회용 이다.

연산의 구조는 Stream 생성 → 중간연산 → 최종연산 순서로 이어집니다.

중간연산은 데이터를 가공하는 과정으로 필터링, 변환, 제한, 정렬, 연산 중간 결과 확인 등이 있습니다.

최종 연산은 Stream 안의 데이터를 모아서 반환하는 역할로써, 출력, 소모, 검색, 검사, 통계, 연산, 수집 등을 지원합니다.

중간 연산 작업은 바로 실행되는 것이 아니라, 종결처리의 실행이 필요할 때 비로소 중간 처리가 실행됩니다. (Lazy Evaluation)

**💡 Java의 Fork-Join Pool이 무엇인지 설명해주세요.**

Task(업무)를 Fork(분할)을 통해 작은 업무로 나눠 배분해서 일을 한 후에 다시 Join(취합)하는 형태의 쓰레드 풀입니다.

도표, 라인, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

A와 B는 각자 업무를 관리하는 queue가 있으며, 자신의 큐에 업무가 없으면 상대방의 큐에서 업무를 가져와서 (steal) 처리합니다.

최대한 놀고 있는 Thread를 줄이는 방법으로 설계되었습 니다.

**💡Java의 Optional이 무엇인지 설명해주세요.**

NullPointerException(NPE)를 피하기 위해 null 여부 검사를 하게 되는데, 검사를 해야 하는 변수가 많을 수록 코드가 번거롭습니다. 따라서 Optional<T> 클래스를 사용해 NPE 발생을 방지하도록 도와줍니다. null이 올 수 있는 값을 감싸는 Wrapper 클래스로, 참조하더라도 NPE가 발생하지 않도록 도와줍니다.

**💡 Java에서의 MultiThread 프로그래밍에 대해 설명해주세요.**

스레드 생성 방법

* 방법 1: Thread 클래스를 상속받아서 run을 오버라이드해서 정의한다.
* 방법 2: Runnable 인터페이스를 구현하여 Thread 생성자에 인자로 넘긴다.
* 방법 3: Callable 인터페이스를 구현하여 FutureTask에 한번 감싸서 Thread의 인자로 넘긴다.
* Runnable은 Exception이 없고 리턴값도 없으나 Callable은 리턴값이 있고 Exception을 발생시킬 수 있다.

스레드 실행 방법

* 보통 start() 메서드를 사용해서 호출하는데 start 한다고 해서 바로 실행되는 것은 아니고 실행 대기열에 저장된 후 차례가 오면 실행된다.
* 정확하게 말하자면, start는 새로운 스레드가 작업을 실행하는데 필요한 새로운 호출 스택을 생성해서 그곳에 run 메서드를 올려둔다. 이후 그곳에서 run 메서드를 호출하고 스레드가 별개의 작업을 수행하게 된다.

운영 시에는 ExecutorService와 Executors를 이용해 스레드풀을 생성 하여 병렬처리합니다.

**💡 동기화가 무엇인지, Java에서 동기화 문제를 해결하는 방법에 대해 설명해주세요.**

자바에서 동시성 문제를 해결하는데 3가지 방법이 있습니다.

* synchronized : 안전하게 동시성을 보장할 수 있습니다. 하지만 비용이 가장 큽니다.
* volatile : 키워드가 붙은 자원은 하나의 thread만이 write하고 나머지는 스레드는 read만 한다는 전제하에만 동시성을 보장합니다.
  + volatile 키워드를 붙인 자원은 read, write 작업이 CPU Cache Memory가 아닌 Main Memory에서 이뤄집니다.
  + 즉, 자원을 저장하는 메모리는 하나가 되기 때문에 같은 공유자원에 대해 각각 메모리별로 다른 값을 가지는 경우가 없습니다.
  + 하지만 여러 스레드에서 Main Memory에 있는 공유자원에 동시에 접근할 수 있으므로 여러 스레드에서 수정하게 되면, 계산값이 덮어씌워지게 되므로 동시 접근 문제를 해결할 수 없습니다.
  + 정리하면, 가시성 문제는 해결할 수 있지만, 동시 접근 문제는 해결할 수 없습니다.
* Atomic 클래스는 CAS(compare-and-swap)를 이용하여 동시성을 하므로 여러 쓰레드에서 데이터를 write해도 문제가 없습니다. synchronized 보다 적은 비용으로 동시성을 보장할 수 있습니다.
  + CAS 알고리즘이란 현재 스레드가 존재하는 CPU의 CacheMemory와 MainMemory에 저장된 값을 비교하여, 일치하는 경우 새로운 값으로 교체하고, 일치하지 않을 경우 기존 교체가 실패되고, 이에 대해 계속 재시도하는 방식입니다.
  + CPU가 MainMemory의 자원을 CPU Cache Memory로 가져와 연산을 수행하는 동안 다른 스레드에서 연산이 수행되어 MainMemory의 자원 값이 바뀌었을 경우 기존 연산을 실패처리하고, 새로 바뀐 MainMemory 값으로 재수행하는 방식입니다.

**💡 자바 8 버전과 11버전의 차이?**

**자바 8 특징**

1. 람다 표현식

함수형 프로그래밍을 자바에서 사용 가능하게 해줍니다.

별도로 함수를 구현하지 않고 화살표를 사용해서 간단히 한 줄로 함수를 만들 수 있습니다.

2. Stream

반복문을 처리하는 방법 중 하나로, 병렬처리가 가능합니다.

예로, 스레드 풀에 가용할 수 있는 스레드들을 가져와서 사용함으로 반복문 처리 속도가 빠릅니다.

3. interface의 default라는 메소드를 구현할 수 있음

기존에는 공통적인 함수라도 인터페이스의 구현체들은 똑같은 것을 각각 다 따로따로 구현을 해줘야 했는데, default가 생김으로써 인터페이스안에 공통적으로 사용하는 함수를 미리 정의할 수 있게 되었습니다.

4. Optional

Null이 될 수 있는 객체를 감싸는 래퍼클래스입니다.

객체에 null이 대입되어 발생되는 오류들이 많아서, 이를 좀 더 안전하게 처리하기 위해 만들어졌습니다.

**자바 11 특징**

1. String, File 클래스에 몇 가지 메소드 추가

2. 람다표현식에 var 키워드 사용 가능하도록 기능 추가

3. Http Client 추가

기존에는 아파치 라이브러리를 사용했는데, 11부터는 자바 표준 Http Client API가 생겼고, 성능이 이전 것보다 개선되었습니다.

**💡 자바 제네릭?**

자바에서 제네릭은 데이터의 타입을 일반화하는 것을 의미한다. 클래스나 메소드에서 사용할 데이터의 타입을 컴파일 시에 미리 지정하는 방법이다.

제네릭(Generic)이라는 단어의 의미에도 ‘일반적인’ 이라는 뜻이 있다.

우리가 자바로 프로그래밍을 할 떄 많이 사용하는 컬렉션 타입들이 제네릭을 구현되어있다.

제네릭은 데이터의 타입을 클래스 내부에서 지정하는 것이 아닌 외부에서 사용자에 의해 지정되는 것을 의미한다.

!! 제네릭을 사용할 수 없는 경우

* 제네릭으로 배열을 생성할 수 없다.
* Ststic 변수를 제네릭으로 사용할 수 없다.

**💡 subList()와 subset()에 대해?**

* subList()는 List를 잘라주는 메서드이다. 인수로 int fromIndex, int toIndex를 갖는다.
* subSet은 TreeSet과 NavigableSet에 정의되어 있다. 따라서 new TreeSet 후에 Set으로 받지 못한다.
* subList 처럼 index로 나누는 것이 아닌 값으로 나누게 된다.

**💡 arrayList의 addAll,removeAll,retainAll ?**

* addAll : 합집합
  + 두 개의 ArrayList의 값을 모두 합친다.
* removeAll : 차집합
  + 두 개의 ArrayList에서 공통되는 값을 뺀 나머지를 저장한다.
* retainAll : 교집합
  + 두 개의 ArrayList에서 공통으로 중복되는 값만 저장한다.

**💡 DAO, DTO, VO 개념과 차이**

1. DAO(Data Access Object)

데이터베이스의 data에 접근하기 위한 객체이며 데이터베이스 접근을 하기 위한 로직과 비즈니스 로직을 분리하기 위해 사용한다.

2. DTO ( Data Transfer Object)

VO라고도 표현하며 계층 간 데이터 교환을 위한 자바 빈즈(JAVA Beams)이다.

데이터베이스 레코드의 데이터를 매핑하기 위한 데이터 객체를 말한다.

DTO는 보통 로직을 가지고 있지 않고 data와 그 data에 접근을 위한 getter, setter만 가지고 있다.

정리하면 DTO는 Database에서 Data를 얻어 Service나 Controller등으로 보낼 때 사용하는 객체를 말한다.

3. VO(Value Object)

VO는 DTO와 혼용해서 쓰이긴 하지만 미묘한 차이가 있다.

VO는 값 오브젝트로써 값을 위해 쓰인다.

DTO와 VO의 공통점은 넣어진 데이터를 getter를 통해 사용하므로 주 목적은 같으나 DAO는 가변적인 성격을 가진 클래스이며(setter 활용) 그에 비해 VO는 불변의 성격을 가졌기에 차이점이 있다.

**💡 synchronized 키워드란?**

synchronized는 lock을 이용해 동기화를 수행하며 4가지의 사용 방법이 존재한다.

1. synchronized method
2. static synchronized method
3. synchronized block
4. static synchronized block

즉, synchronized 메소드는 인스턴스 단위로 lock을 걸지만, synchronized 키워드가 붙은 메소드들에 대해서만 lock을 공유한다. 쉽게 말해서 한 스레드가 synchronized 메소드를 호출하는 순간, 모든 synchronized 메소드에 lock이 걸리므로 다른 스레드가 어떠한 synchronized 메소드를 호출할 수 없다. (단, 일반 메소드는 호출 가능)

**💡 Junit 이란?**

유닛 테스트

* 프로그래밍에서 모든 함수와 메서드에 대한 테스트 케이스(Test case)를 작성하여 의도된 대로 잘 동작하는지 검증하는 절차
* 프로그램을 작은 단위로 쪼개어 각 단위가 정확하게 동작하는지 검사함으로써 프로그램의 안정성을 높임
* System.out.println()으로 하는 번거로운 디버깅이 필요 없으며, 개발기간 중 대부분을 차지하는 디버깅 시간을 단축

JUnit

* 자바 프로그래밍 언어용 유닛 테스트 프레임워크
* 테스트 결과는 Test클래스로 개발자에게 테스트 방법 및 클래스의 History를 공유 가능
* 단정(assert) 메서드로 테스트 케이스의 수행 결과를 판별
* 어노테이션으로 간결하게 지원(JUnit4부터)

기본 어노테이션

@Test : 테스트를 만드는 모듈 역할

@DisplayName : 테스트 클래스 또는 테스트 메서드의 사용자 정의 표시 이름을 정의

@ExtendWith : 사용자 정의 확장명을 등록하는데 사용

@BeforeEach : 각 테스트 메서드 전에 실행됨을 나타냄

@AfterEach : 각 테스트 메서드 후에 실행됨을 나타냄

@BeforeAll : 현재 클래스의 모든 테스트 메서드 전에 실행됨을 나타냄

@AfterAll : 현재 클래스의 모든 테스트 메서드 후에 실행됨을 나타냄

@Disable : 테스트 클래스 또는 메서드를 비활성화

**단위 테스트란?**

단위 테스트(Unit Test)는 프로그램의 기본 단위인 모듈(Module)을 테스트하는 것이다. 구현 단계에서 각 모듈의 개발을 완료한 후 명세서의 내용대로 정확히 구현되었는지를 테스트하는 것이다. 테스트가 가능한 최소 단위로 나눠서 테스트를 수행하며 개발 수명주기(Development LifeCycle)의 정황과 시스템에 의존적이면서도 시스템의 다른 부분에서 격리하여 독립적으로 수행해야 하는 테스트이다. 단위테스트를 하기 위해서는 가짜 프로그램, 객체(Mock Object)를 만들어서 활용할 수 있으며, 정교하게 테스트 하기 위해서는 테스트 케이스(Test Case) 작성은 필수라 할 수 있다.

종류 : JUnit(Java), DBUnit(DB), CppUnit(C++), NUnit(.net), PyUnit(Python)

**단위테스트의 장점**

* 개발단계 초기에 문제를 발결할 수 있게 도와줌
* 리팩토링 또는 라이브러리 업그레이드 등에서 기능 확인을 도와 줌(회귀 테스트)
* 기능에 대한 불확실성 감소
* 시스템에 대한 실제 문서 또는 예제로써 사용가능
* 빠른 피드백과 기능을 안전하게 보호 가능

💡 래퍼(wrapper) 클래스란?

자바의 자료형은 기본 타입(primitive type)과 참조 타입(reference type) 으로 나누어진다.

기본 타입은 정수, 실수, 문자, 논리 리터럴을 저장하는 타입이다.

참조 타입은 배열, 열거, 클래스, 인터페이스 등 객체의 위치를 참조하는 타입니다.

* 기본 타입 : byte, short, char, int, long, float, double, boolean
* 참조 타입 : class, interface, ...

래퍼 클래스란 8개의 기본 타입에 해당하는 데이터를 객체로 표현하기 위해 포장해주는 클래스라고 한다.

각각의 타입에 해당하는 데이터를 인수로 전달받아 해당 값을 가지는 객체로 만들어준다.

래퍼 클래스는 모두 java, lang 패키지에 포함되어 제공된다.

래퍼 클래스 특징

1. 기본 타입은 값을 갖는 객체인 포장 객체를 생성할 수 있다.
2. 래퍼 클래스는 각 타입에 해당하는 데이터를 파라미터로 전달받아 해당 값을 가지는 객체로 만들어준다.
3. 래퍼 클래스로 감싸고 있는 기본 타입 값은 외부에서 변경할 수 없다.
4. 변경하기 위해서는 새로운 포장 객체를 만들어야 한다.
5. 간단하게 말해 래퍼 클래스란 기본 타입의 객체화를 말한다

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* int와 char를 제외한 클래스들은 기본형의 첫 글자를 소문자에서 대문자로만 바꾸면 된다.

**💡 오토 박싱 & 언박싱(autoboxing & unboxing)**

래퍼 클래스는 산술 연산을 위해 정의된 클래스가 아니다.

따라서, 인스턴스에 저장된 값을 변경할 수 없다.

값을 참조하기 위해 새로운 인스턴스를 생성하고 생성된 인스턴스의 값 만을 참조할 수 있다.

1. 박싱: 기본타입의 데이터 -> 래퍼 클래스의 인스턴스로 변환하는 과정

2. 언박싱: 래퍼 클래스의 인스턴스에 저장된 값 -> 기본 타입의 데이터로 꺼내는 과정

**💡 JAVA 다형성 이란?**

다형성(polymorphism)이란 같은 자료형에 여러가지 타입의 데이터를 대입하여 다양한 결과를 얻어낼 수 있는 성질을 의미

다형성(polymorphism)이란 부모-자식 상속 관계에 있는 클래스에서 상위 클래스가 동일한 메시지로 하위 클래스들을 서로 다르게 동작시키는 객체 지향 원리입니다.

다형성을 활용하면 부모 클래스가 자식 클래스의 동작 방식을 알 수 없어도 오버라이딩을 통해 자식 클래스를 접근할 수 있습니다.

이유는 동적 바인딩 때문입니다. 동적바인딩이란, 메서드가 실행 시점에서 성격이 결정되는 바인딩인데요.

프로그램의 컴파일 시점에 부모 클래스는 자신의 멤버 함수밖에 접근할 수 없으나,실행 시점에 동적 바인딩이 일어나 부모클래스가 자식 클래스의 멤버함수를 접근하여 실행할 수 있습니다.

**다형성 장점**

1) 유지보수가 쉽다

개발자가 여러 객체를 하나의 타입으로 관리가 가능하기 때문에 코드 관리가 편리해 유지보수가 용이합니다.

2) 재사용성 증가

다형성을 활용하면 객체를 재사용하기 쉬워지기 때문에 개발자의 코드 재사용성이 높아집니다.

3) 느슨한 결합

다형성을 활용하면 클래스간 의존성이 줄어들며 확장성이 높고 결합도가 낮아져 안전성이 높아집니다.

**다형성 필수 조건**

1) 상속 관계

다형성을 활용하기 위해서는 필수로 부모-자식 간 클래스 상속이 이루어져야 합니다.

2) 오버라이딩 필수 (자식 클래스에서 메소드 재정의)

다형성이 보장되기 위해서는 하위 클래스 메소드가 반드시 재정의되어 있어야 합니다.

3) 업캐스팅 (자식 클래스의 객체가 부모 클래스 타입으로 형 변환 되는 것)

부모 타입으로 자식클래스를 업 캐스팅하여 객체를 생성해야 합니다.

다형성 구현 방법

1) 상속 클래스 구현 (부모-자식 클래스 구현)

2) 메소드 오버라이딩

3) 업 캐스팅하여 객체 선언

4) 부모 클래스 객체로 자식 메소드 호출

💡 **변수의 기본형 & 참조형 타입**

변수(variable)란 데이터(data)를 저장하기 위해 프로그램에 의해 이름을 할당 받은 메모리 공간을 의미한다. 그리고 자바에서 말하는 데이터 타입(자료형)이란, 변수에 적재할 데이터가 메모리에 어떻게 저장되고 프로그램에서 어떻게 처리되어야 하는지를 명시적으로 알려주는 키워드

이 데이터 타입(자료형)은 크게 기본형 타입 과 참조형 타입으로 구분 된다.

* 기본형(primitive type) : 계산을 위해 실제 값을 저장한다.

* 참조형(reference type) : 객체의 주소를 저장한다. null 또는 객체의 주소(4byte, 0x0 ~ 0xffffffff)를 갖는다.

기본형 타입 (Primitive Type)

기본형 타입에는 크게 논리형 (boolean), 문자형 (char), 정수형 (byte, short, int, long) 실수형 (float, double) 으로 나뉜다.

본형 타입에 대한 특징으로는 다음과 같다.

* 모두 소문자로 시작된다
* 비객체 타입이므로 null 값을 가질 수 없다. (기본값이 정해져 있음)
* 변수의 선언과 동시에 메모리 생성
* 모든 값 타입은 메모리의 스택(stack)에 저장됨
* 저장공간에 실제 자료 값을 가진다

참조형 타입 (Reference Type)

참조형 타입은 간단히 말하자면, 위의 8가지 자료형(primitive type)을 제외한 나머지를 말한다고 보면 된다. 기본적으로 제공하는 클래스, 프로그래머가 스스로 만든 클래스, 배열, 열거 타입 등을 모두 참조형이라고 한다.

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

참조형 타입 특징으로는 다음과 같다.

* 기본형 과는 달리 실제 값이 저장되지 않고, 자료가 저장된 공간의 주소를 저장한다.
* 즉, 실제 값은 다른 곳에 있으며 값이 있는 주소를 가지고 있어서 나중에 그 주소를 참조해서 값을 가져온다.
* 메모리의 힙(heap)에 실제 값을 저장하고, 그 참조값(주소값)을 갖는 변수는 스택에 저장
* 참조형 변수는 null로 초기화 시킬 수 있다

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

**💡 객체지향 프로그래밍이란?**

객체 지향 프로그래밍(Object-Oriented Programming, OOP)이란 컴퓨터 프로그램을 어떤 데이터를 입력받아 순서대로 처리하고 결과를 도출하는 명령어들의 목록으로 보는 시각에서 벗어나 여러 독립적인 부품들의 조합, 즉 객체들의 유기적인 협력과 결합으로 파악하고자 하는 컴퓨터 프로그래밍의 패러다임을 의미

* 객체 지향 프로그래밍의 장점

1. 프로그램을 보다 유연하고 변경이 용이하게 만들 수 있다는 점

2. 코드의 변경을 최소화하고 유지보수를 하는 데 유리

3. 코드의 재사용을 통해 반복적인 코드를 최소화하고, 코드를 최대한 간결하게 표현

4. 인간 친화적이고 직관적인 코드를 작성하기에 용이

**💡 객체지향 설계 원칙 (SOLID)?**

객체지향 설계 5대 원칙이 있다. SRP(단일 책임 원칙), OCP(개방-폐쇄 원칙), LSP(리스코프 치환 원칙), ISP(인터페이스 분리 원칙), DIP(의존 역전 원칙) 뜻하며 줄여서 SOILD 원칙이라고 부른다.

**1. SRP 단일 책임 원칙 (Single responsibility principle)**

* 한 클래스는 하나의 책임만 가져야 한다.

**2. OCP 개방-폐쇄 원칙 (Open/closed principle)**

* “소프트웨어 요소는 확장에는 열려 있으나 변경에는 닫혀 있어야 한다.”

**3. LSP 리스코프 치환 원칙 (Liskov substitution principle)**

* “프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다.”

**4. ISP 인터페이스 분리 원칙 (Interface segregation principle)**

* “특정 클라이언트를 위한 인터페이스 여러 개가 범용 인터페이스 하나보다 낫다.”

**5. DIP 의존관계 역전 원칙 (Dependency inversion principle)**

* 프로그래머는 “추상화에 의존해야지, 구체화에 의존하면 안된다.”

**💡 자바 JDK와 이클립스란?**

1. 자바

자바로 기술되 프로그램 개발 및 실횅을 할 수 있는 소프트웨어 모임을 말합니다.

프로그래밍 언어중 하나이고, 자바 프로그램은 어떤 운영체제에서도 작동할 수 있으며, 자바 프로그램을 실행하기 위해서는 자바 가상머신(JVM)과 개발에 필요한 표준라이브러리 세트와 컴파일러의 환경이 필요합니다.

2. JDK (Java Development Kit)

오라클 사이트에서 설치가 가능

JDK는 이클립스 설치전에 먼저 설치해야한다.

3. 이클립스 (Eclipse)

이클립스 개발 툴 통합개발환경

자바 개발을 편리하게 해주는 도구라고 할 수 있습니다.

무료 설치가 가능하고 개발자들에게 많이 쓰이고 있습니다.

**💡 StringBuffer와 StringBuilder?**

- StringBuffer는 각 메서드별로 Synchronized Keyword가 존재하여, 멀티스레드 환경에서도 동기화를 지원.

- 반면, StringBuilder는 동기화를 보장하지 않음.

그렇기때문에 멀티스레드 환경이라면 값 동기화 보장을 위해 StringBuffer를 사용하고,단일스레드 환경이라면 StringBuilder를 사용하는 것이 좋습니다.

단일 스레드환경에서 StringBuffer를 사용한다고 문제가 되는 것은 아니지만, 동기화 관련 처리로 인해 StringBuilder에 비해 성능이 좋지 않습니다.

String은 짧은 문자열을 더할 경우 사용합니다.

StringBuffer는 스레드에 안전한 프로그램이 필요할 때나, 개발 중인 시스템의 부분이 스레드에 안전한지 모를 경우 사용하면 좋습니다.

StringBuilder는 스레드에 안전한지 여부가 전혀 관계 없는 프로그램을 개발할 때 사용하면 좋습니다.

JDK 1.5 이후 버전에서는 String 클래스를 활용해도 StringBuilder와 성능상으로 차이가 없어졌습니다.

하지만 반복 루프를 사용해서 문자열을 더할 때에는 객체를 계속 추가한다는 사실에는 변함이 없습니다.

그러므로 String 클래스를 쓰는 대신, 스레드와 관련이 있으면 StringBuffer를, 스레드 안전 여부와 상관이 없으면 StringBuilder를 사용하는 것을 권장

단순히 성능만 놓고 본다면 연산이 많은 경우, StringBuilder > StringBuffer >>> String 입니다.

**💡 데브옵스(DevOps)란?**

테크 산업에서 떠오르는

ABCD (AI, Block-chain, Cloud, Data)

ABCD(인공지능:A.I, 블록체인:Blockchain, 클라우드:Cloud, 데이터:Data)

* 개발과 운영을 합친 것
* 용어부터 개발을 뜻하는 Development에서 DEV를, 운영을 뜻하는 Operations에서 OPS를 가져왔다.
* 데브옵스 엔지니어가 하는 일의 공통점은 “현대의 모든 서비스(APP)가 겪을 수밖에 없는 문제를 해결하기 위한 시도”라는 것

**💡 컬렉션(Collection) 프레임워크 – 주요 인터페이스**

자바에서 컬렉션 프레임워크는 다수의 데이터를 쉽고 효과적으로 처리할 수 있는 표준화된 방법을 제공하는 클래스의 집합을 의미한다.

즉, 데이터를 저장하는 자료 구조와 데이터를 처리하는 알고리즘을 구조화하여 클래스로 구조화하여 클래스로 구현해 놓은 것

주요 인터페이스

컬렉션 프레임워크는 컬렉션 데이터 그룹이 크게 3가지 타입이 존재한다고 인식하고 컬렉션을 다루는데 필요한 기능을 가진 3개의 인터페이스를 정의하였다.

1. List 인터페이스

2. Set 인터페이스

3. Map 인터페이스

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 포스트잇 노트, 폰트이(가) 표시된 사진

자동 생성된 설명

**💡 컬렉션 프레임워크(Collections Framework) - Stack, Queue, Iterator**

Stack, Queue

Stack은 마지막에 저장한 데이터를 가장 먼저 꺼내는 LIFO(Last in First Out)의 구조

Queue 처음 저장한 데이터를 가장 먼저 꺼내게 되는 FIFO(First in First Out)의 구조

Queue인터페이스 구현체

- PriorityQueue

저장한 순서에 관계없이 우선순위(priority)가 높은 것부터 꺼내게 된다는 특징이 있고, null을 저장하면 NullPointerException이 발생한다.

저장공간으로 배열을 사용하고, 각 요소를 힙(heap)이라는 자료구조 형태로 저장하여, 큰값과 작은값을 빠르게 찾을 수 있다.

- Deque(Double-Ended Queue)

한쪽 끝으로만 추가/삭제하는 Queue와 달리, 양쪽 끝에서 추가/삭제가 가능하다. Queue를 구현하며, 구현체로는 ArrayDeque와 LinkedList 등이 있다.

Iterator

Collection의 개체를 순회하는 인터페이스이며, 컬렉션에 저장되어 있는 요소들을 읽어온다.

개별마다 데이터를 읽는 방법이 다를 수가 있기 때문에 컬렉션의 접근이 어려워질 수 있기 때문에 Iterator를 통해 동일하게 접근할 수 있는 방법이 제공되는 것이다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

**💡 JDBC 프로그래밍**

JDBC는 자바 어플리케이션에서 표준화된 방법으로 다양한 데이터베이스에 접속할 수 있게 설계된 인터페이스를 말하며, 다양한 DBMS 회사들은 각각 JDBC 드라이버를 제공

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**💡 오버라이딩(overriding)**

상속받은 메소드의 내용을 변경하는 것을 오버라이딩 이라고 한다.

오버라이딩 조건

메소드의 내용만을 새로 작성하는 것이기 때문에 선언했던 부분은 동일해야 한다. 다시말해서 부모클래스와 자식클래스의 메소드는 이름, 매개변수, 반환타입이 같아야 한다.

1. 접근제어자는 부모 클래스의 메소드보다 좁은 범위로 변경할 수가 없다.

* 부모 클래스의 메소드 접근 제어자가 protected라면, 오버라이딩하려는 자식 클래스의 메소드는 접근제어자가 protected나 public이어야 한다.

2. 부모 클래스의 메소드보다 많은 수의 예외를 선언할 수 없다.

* 단순히 선언된 예외의 개수가 아니고, Exception은 상위개념이다.

3. 인스턴스메소드를 static메소드로 또는 static메소드를 인스턴스 메소드로 바꿀 수는 없다.

오버로딩과 오버라이딩

* 오버로딩(overloading) : 기존에 없는 새로운 메소드를 정의하는 것(new)
* 오버라이딩(overriding) : 상속받은 메소드의 내용을 변경하는 것(change, modify)

super

자식 클래스에서 부모 클래스로부터 상속받은 멤버를 참조하는데 사용되는 참조 변수이다.

상속받은 멤버와 자신의 이름이 같은 경우 super를 붙여서 구별한다.

super()

* this()는 같은 클래스의 다른 생성자를 호출하는데 사용되지만, super()는 조상 클래스의 생성자를 호출하는데 사용한다.
* Object클래스를 제외한 모든 클래스의 생성자는 첫 줄에 반드시 자신의 다른 생성자(this()) 또는 조상의 생성자(super())를 호출해야 한다.
* 그렇지 않으면 컴파일러가 생성자의 첫 줄에 'super();'를 자동으로 추가한다.
* 부모클래스의 멤버변수는 부모의 생성자에 의해 초기화되도록 해야한다.

**💡 오버로딩(overloading)**

오버로딩이란?

한 클래스 내에 이미 사용하려는 이름과 같은 이름을 가진 메소드가 있어도 매개변수의 개수 또는 타입이 다르면, 같은 이름을 사용할 수 있는데 , 한 클래스 내에서 같은 이름의 메소드를 여러 개 정의하는 것을 '메소드 오버로딩' 또는 '오버로딩'이라고 한다.

오버로딩의 조건

* 메소드 이름이 같아야 하고, 매개변수의 개수 또는 타입이 달라야 한다.
* 해당 조건을 만족하지 못하면 메소드의 중복으로 처리된다. 그리고 오버로딩된 메소드들은 매개변수에 의해서만 구별될 수 있어서, 반환타입은 오버로딩에 영향을 미치지 않는다.

오버로딩의 장점

같은 기능을 하는 메소드들이지만, 서로 다른 이름을 가져야 한다면, 이름을 짓는 것도 어렵고 구분하여 기억해야하는 것도 어려운 일이다. 그러나 오버로딩을 통해 여러 메소드들이 하나의 이름으로 정의될 수 있다면 기억하기도 쉽고 오류의 가능성을 줄일 수 있다. 메소드의 이름만 보고도 이 메소드들은 이름이 같기 때문에 같은 기능을 할 것이라고 추측할 수 있게 된다. 또한 이름을 여러가지로 사용하지 않아도 되므로, 수고를 줄일 수 있다.

**💡 메소드(method) 정의와 4가지 유형**

메소드란?

어떠한 문제를 처리하기 위한 방법을 소스 코드로 묶어 놓고 필요(호출)에 의해 동작하는 기능으로, 메소드는 클래스내에 변수와 같이 쓰면 메소드라고 부른다.

메소드를 사용하는 이유

* 높은 재사용성(reusability) : 한번 만들어 놓으면 여러번 사용이 가능
* 중복된 코드의 제거 : 반복되는 문장들 대신 메소드를 호출하여 한 문장을 대체가 가능,
* 프로그램의 구조화 : 프로그램을 개발하면, 작업단위로 나누어 여러 개의 메소드로 프로그램 구조를 단순화 시켜야 문제가 발생해도 쉽게 해결이 가능해진다.

메소드의 4가지 유형

* main 메소드

main 메소드는 프로그램의 시작점 역할을 하는 약속된 내용이다. main 메소드가 없는 프로그램은 별도로 동작할 수 가 없고, 컴퓨터가 소스 코드를 읽을 때 main메소드를 실행하고 거기에 정의된 로직에 따라 프로그램이 동작하게 된다.

* 반환(return) 타입

메소드의 리턴 타입은 자료형 또는 void가 올 수 있다. 자료형이 리턴 타입으로 오면 반환(return)해 주는 값(메소드 자료형과 동일한 값)이 반드시 있어야 하고 void는 반환 값이 없다. 반환하고자 하는 데이터가 있다면 그 데이터의 자료형을 메소드의 자료형으로 적어줘야 한다.

참조형 반환타입은 메소드가 객체의 주소를 반환한다는 것을 의미한다.

* 리턴(return)문

메소드 내에서 동작하고 얻은 결과(필요한 데이터)를 반환해주는 역할을 한다. 반환타입이 void가 아닌 경우 반드시 포함되어야 한다. return문은 단 하나의 값만 반환할 수 있는데, 메소드로의 입력(매개변수)은 여러 개일 수 있지만, 출력(반환값)은 최대 하나만 허용한다. return문 뒤에는 어떠한 구문도 올 수 없다.

return문을 사용하면 값을 반환해 주고 해당 메소드는 실행이 종료된다.

* 매개변수(parameter)

'매개변수(parameter)란 외부로부터 입력 값을 받기 위해 메소드의 괄호 안에 선언하는 변수'이다.

main 메소드에서 다른 메소드를 호출할 때 값을 전달하여 전달된 값에 따라 처리될 수 있도록 하는 기능이다.

필요한 값의 개수만큼 변수를 선언하며 각 변수 간의 구분은 쉼표,를 사용한다. 또한, 두 변수의 타입이 같아도 변수의 타입을 생략할 수 없다.

베소드를 호출할 대 매개변수로 지정한 값을 메소드의 매개변수에 복사해서 넘겨준다. 매개변수의 타입이 기본형일때는 기본형 값이 복사되지만 참조형이면 인스턴스의 주소가 복사된다. 참조형 매개변수로 선언하면 저장된 곳의 주소를 알 수 있으므로, 변수의 값을 읽고 변경도 할 수 있다.

클래스 메소드(static메소드)와 인스턴스 메소드

* 객체를 생성하지 않고도 '클래스이름.메소드이름(매개변수)'와 같이 호출이 가능하고, 인스턴스 메소드는 반드시 객체를 생성해야 호출할 수 있다.
* 인스턴스 메소드는 인스턴스 변수와 관련된 작업을 하는, 즉 메소드 작업을 수행하는데 인스턴스 변수를 필요로 하는 메소드이다. 인스턴스와 관계없는 메소드는 클래스 메소드로(static 메소드) 정의한다.

클래스 멤버와 인스턴스 멤버간의 참조와 호출

* 같은 클래스에 속한 멤버들 간에는 별도의 인스턴스를 생성하지 않고도 서로 참조 또는 호출이 가능하다. 그러나 클래스멤버가 인스턴스 멤버를 참조 또는 호출하고자 하는 경우에는 인스턴스를 생성해야 한다.
* 인스턴스 멤버가 존재하는 시점에 클래스 멤버는 항상 존재하지만, 클래스 멤버가 존재하는 시점에 인스턴스 멤버가 존재하지 않을 수 있기 때문이다.

**💡 POJO (Plain Old java Object)란?**

POJO (Plain Old java Object) 를 해석하면 평범 자바 오브젝트라고 한다.

POJO프레임워크는 POJO를 이용한 애플리케이션 개발이 가진 특징과 장점을 그대로 살리면서 EJB에서 제공하는 엔터프라이즈 서비스와 기술을 그대로 사용할 수 있도록 도와주느 프레임워크이다. 나아가 이는 기존의 EJB에서보다 훨씬 더 세련되고 나은 방법이다.

대표적으로 프레임워크 스프링 하이버네이트.

POJO프로그램의 진정한 가치는 자바의 객체지향적인 특징을 살려 비지니스 로직에 충실한 개발이 가능 하도록 하는것이기도 하다.

**💡 EJB(Enterprise Java Bean)**

EJB(Enterprise Java Bean), 기업환경의 시스템을 구현하기 위한 서버 측 컴포넌트 모델이다. 일반적으로 업무 로직을 가지고 있는 서버 어플리케이션을 EJB라고 한다.

Enterprise JavaBeans(EJB)는 독립한 부품이 아닌, 미국 Sun Microsystems사가 제창한 규약이다. EJB는 서버 어플리케이션의 개발을 용이하게해 다중다양한 Platform과 제품간의 이동성을 실현하기 위하여 비지니스로직과 시스템 서비스를 이용하는 로직을 분산해 그 사이의 규약을 규정하고 있다.

비지니스 로직을 탑제한 부품을 "Enterprise Bean"이라고 불린다. Database처리, Transaction처리등의 시스템 서비스를 이용한 로직을 감추고 있는 부품을 "컨테이너"라고 불린다.

**💡 J2EE (Java 2 Enterprise Edition)**

J2EE는 자바 기술로 기업환경의 어플리케이션을 만드는데 필요한 스펙들을 모아둔 스펙 집합.

독점적인 기술이라기 보다는 Java진영으로 불리는 여러 개발자들이 같이 만들어가고 공유하는 기술이라고 볼 수 있다.

\*엔터프라이즈 시스템: 서버에서 동작하며 기업과 조직의 업무를 처리해주는 시스템.

많은 사용자들의 요청을 동시 처리 해야하므로 서버지원을 효율적으로 공유하고 분배해서 사용할 수 있어야 한다. 또한 중요한 기업 핵심정보 다루기때문에 보안,안정성,확장성 이 요구된다.

웹을 통한 유저인터페이스 뿐만 아니라 타 시스템과의 자동화된 연계과 웹 이외의 클라와의 접속 위한 리모팅 기술도 요구된다.

대표 구성 요소

J2EE는 매우 방대한 범위를 다루는 스펙

Servlet

* 클라이언트가 보내는 HTTP 요청을 처리하는 서버측 자바 프로그램.

JSP(Java Server Pages)

* HTML이나 Java 코드를 써서 사용자에게 정보를 출력. JSP가 처음 실행될 때 Servlet 엔진이 이것을 Servlet 으로 컴파일시켜서 내부적으로는 Servlet으로 동작한다.

EJB(Enterprise Java Beans)

* Java에서 제공하는 분산 컴포넌트 기술로 비즈니스 로직이나 데이터, 메시지를 처리할 수 있다.

Java Database Connector(JDBC)

* 여러 종류의 데이터베이스 시스템에 접근하는 단일한 인터페이스를 제공.
* 각각의 데이터베이스에 맞는 JDBC 드라이버가 있어야 한다.

Remote Method Invocation(RMI)

* 프록시를 써서 원격에 있는 Java 객체의 메소드를 실행시키기 위한기술.

Java Naming DirectoryInterface (JNDI)

* 자바 기술로 만들어진 객체에 이름을 붙여 찾을 수 있도록 단일한인터페이스를 제공.

Java Connector Architecture(JCA)

* 이기종 플랫폼을 통합할 수 있도록 플랫폼 독립적인 인터페이스를 제공.

Java Message Service (JMS)

* 여러 가지 메시징 시스템에 대한 플랫폼 독립적인 인터페이스를 제공.

**💡 Java EE 와 SE 의 개념과 차이?**

자바 프로그래밍 언어 플랫폼

1. 표준 에디션의 자바 플랫폼 (Java SE)
2. 엔터프라이즈 에디션의 자바 플랫폼 (Java EE)
3. 마이크로 에디션의 자바 플랫폼 (Java ME)
4. JavaFX

Java SE (Standard Edition)

* 가장 대중적인 자바 플랫폼
* 흔히 자바 언어라고하는 대부분의 패키지가 포함된 에디션
  + java.lang.\*, java.util.\*, java.awt.\*, javax.rmi.\*, javax.net.\* 등
* Java SE의 API는 자바 프로그래밍 언어의 핵심기능들을 제공
  + 기초적인 타입
  + 네트워킹
  + 보안
  + 데이터베이스 처리
  + 그래픽 사용자 인터페이스 개발
  + XML 파싱
* 가상머신, 개발도구, 배포기술, 부가 클래스 라이브러리, 툴킷 등 제공

Java EE (Enterprise Edition)

* Java EE 플랫폼은 Java SE 플랫폼을 기반으로 그 위에 탑재된다.
* 웹 프로그래밍에 필요한 기능을 다수 포함
  + JSP, Servlet, JDBC, JNDI, JTA, EJB 등
* 대규모, 다계층, 확장성, 신뢰성, 보안 네트워킹 API, 환경 등을 제공

Java ME (Micro Edition)

* Java ME는 모바일 폰과 같은 자바 프로그래밍 언어 기반의 어플리케이션이 보다 조그만 가상 머신으로 동작시킬 수 있는 기능과 API 제공
* Java EE 처럼 Java SE를 기반으로 함
* 작은 작치에서 동작하는 전용 클래스 라이브러리들 제공
* Java EE 서비스의 클라이언트 역할을 하기도 함

Java FX

* 경량 사용자 인터페이스 API를 사용하여 리치 인터넷 어플리케이션을 만들 때 사용됨
* 하드웨어 수준에서 가속 기능을 사용할 수 있는 그래픽과 미디어 엔진을 갖추고 있어 클라이언트의 성능에 신경을 써야하는 분야에서 사용됨
* Java EE 처럼 클라이언트 역할을 하기도 함

요약

Java EE는 Java SE에서 API(lib 디렉토리에 포함되어 있는 JAR 파일들)가 추가된 것

**💡 Javadoc이란?**

Javadoc은 JDK와 함께 패키지로 제공되는 도구 입니다. JDK가 설치 되어있다면 Javadoc을 사용할 수 있으며, Java 소스 코드의 코드 문서를 생성하는데 도움을 주는 도구 입니다.

**💡 enum(열거형)이란?**

enum은 "Enumeration"의 약자다. Enumeration은 "열거, 목록, 일람표"라는 뜻도 갖고 있다.

열거형, 이넘(enum)은 요소, 멤버라 불리는 명명된 값의 집합을 이루는 자료형이다. 열거자 이름들은 일반적으로 해당 언어의 상수 역할을 하는 식별자다.

enum은 상수 그룹(final 변수처럼 바꿀 수 없는 변수)을 나타내는 특수한 "클래스"다. 열거형을 만들려면 class, interface 대신 enum 키워드를 사용하고 상수를 쉼표로 구분한다. 또한 대문자여야 한다.

장점

1. 코드의 양이 줄었다

2. 인스턴스 생성과 상속 시도 시 컴파일 에러

3. enum이란 키워드로 열거형의 의도를 명확히 드러냄

**💡 리팩토링이란?**

리팩토링(Refactoring)이란 "'결과의 변경 없이 코드의 구조를 재조장함'을 뜻한다.

주로 가독성을 높이고 유지보수를 편하게 하기 위해 사용되며, 버그를 없애거나 새로운 기능을 추가하는 행위는 아니다.

사용자가 보는 외부 화면은 그대로 두면서 내부 로직적인 부분의 구조를 변경하거나 개선하는 유지보수 행위를 뜻한다."

리팩토링 효과 (Refactoring Effect) / 리펙토링을 해야 하는 이유

* 소프트웨어의 유지보수를 하는 과정이 간단해진다.
* 협업을 하면서 소스를 공유하는 과정에서 어려움이 줄어든다.
* 코드 중복을 제거하고, 수정 용이성을 높이므로 작업시간이 줄어든다.
* 직관성 있는 네이밍으로 분석이 쉽다.

**💡 Runtime Data Area 는 어떻게 생겼을까?**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1) Method Area

* JVM이 실행되면서 생기는 공간이다.
* Class 정보, 전역변수 정보, Static 변수 정보가 저장되는 공간이다.
* Runtime Constant Pool 에는 말 그대로 '상수' 정보가 저장되는 공간이다.
* 모든 스레드에서 정보가 공유된다.

2) Heap

* new 연산자로 생성된 객체, Array와 같은 동적으로 생성된 데이터가 저장되는 공간
* Heap에 저장된 데이터는 Garbage Collector 가 처리하지 않는한 소멸되지 않는다.
* Reference Type 의 데이터가 저장되는 공간
* 모든 스레드에서 정보가 공유된다.

3) Stack

* 지역변수, 메소드의 매개변수와 같이 잠시 사용되고 필요가 없어지는 데이터가 저장되는 공간
* Last In First Out, 나중에 들어온 데이터가 먼저 나간다
* 만약, 지역변수 이지만 Reference Type일 경우에는 Heap 에 저장된 데이터의 주소값을 Stack 에 저장해서 사용하게 된다.
* 스레드마다 하나씩 존재한다.

4) PC Register

* 스레드가 생성되면서 생기는 공간
* 스레드가 어느 명령어를 처리하고 있는지 그 주소를 등록한다.
* JVM이 실행하고 있는 현재 위치를 저장하는 역할

5) Native Method Stack

* Java 가 아닌 다른 언어 (C, C++) 로 구성된 메소드를 실행이 필요할 때 사용되는 공간

**💡 원시타입, 참조타입(Primitive Type, Reference Type)**

* 자바에서는 데이터 타입에 크게 두 가지 원시 타입(Primitive Type)과 참조타입(Reference Type)이 있다.
* 원시 타입은 쉽게 말해, 정수, 실수, 문자, 논리 리터럴등의 실제 데이터 값을 저장하는 타입이고,
* 참조 타입은 객체(Object)의 번지를 참조(주소를 저장)하는 타입으로 메모리 번지 값을 통해 객체를 참조하는 타입이다.

**원시 타입(Primitive Type)**

**텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명**

* 실수형 데이터타입에서는 double형 데이터타입이 기본 데이터타입이다.
* 기본형 데이터타입이 double형 이기 때문에 float형 데이터타입의 실수형 데이터를 사용하고자 하는 경우에도long형과 마찬가지로 실수 데이터 맨 뒤 쪽에 접미사 'f'나 'F'를 붙여줘야 한다.
* float형 데이터타입보다 double형 데이터타입이 두배정도 더 정밀한 데이터를 표현할 수 있다.

**참조 타입(Reference Type)**

* 참조 타입은 원시 타입을 제외한 타입들(문자열, 배열, 열거, 클래스, 인터페이스)을 말한다.
* Java에서 실제 객체는 힙 영역에 저장되며 참조 타입 변수는 스택 영역에 실제 객체들의 주소를 저장하여, 객체를 사용할때 마다 참조 변수에 저장된 객체의 주소를 불러와 사용하는 방식이다.

**정적 메모리 스택(Stack) 영역**

* 스택 영역에는 기본타입 변수가 할당되고 변수의 실제 값들이 저장된다.
* 참조 타입의 변수들은 이 스택 영역에서 힙 영역에 생성된 객체들의 주소 값을 저장하고 있는다.
* 객체 안의 메소드의 작업이 종료되면 할당되었던 메모리 공간은 반환되어 비워진다.

**동적 메모리 힙(Heap) 영역**

* 힙 영역에는 객체와 배열이 생성된다.
* 그리고 참조타입(배열, 클래스, 인터페이스 등)들이 이 객체들의 주소를 스택 영역에 저장한다.
* 기본타입 변수들과는 다르게 크기가 정해져 있지 않다.
* 프로그램 실행시 메모리에 동적으로 할당된다.
* 참조하는 변수가 없으면 자바의 가비지 컬렉터가 제거한다.
* 가비지 컬렉터(Garbage collector) : 메모리의 힙 영역에 할당된 더 이상 사용되지 않는 객체를 제거 하는 역할
* 이렇게 객체를 제거하며 메모리가 관리된다.

**💡 Boxing, Unboxing**

* Boxing(박싱)은 원시 타입을 참조 타입으로 변환 시키는 것을 말하고, Unboxing(언박싱)은 참조 타입을 원시 타입으로 변환 시키는 것을 말한다.
* 자바 1.5 이전에는 일일히 변환 과정을 거쳐주어야 했지만, 자바 1.5부터 추가된 Auto Boxing / Unboxing 기능으로 아래의 예시와 같이 명시적으로 원시타입을 참조타입으로 감싸주지 않아도 자동으로 Boxing / Unboxing 해준다.
* 이러한 Auto Boxing / Unboxing 기능은 메모리 누수의 원인이 될 수도 있다.

**💡 원시 타입과 참조 타입 차이**

**1. Null 포함 가능 여부**

원시타입은 null을 담을 수 없지만, 참조 타입은 가능하다.

**2. 제너릭 타입에서 사용 가능 여부**

원시타입은 제너릭 타입에서 사용할 수 없지만, 참조 타입은 가능하다.

**💡 원시 타입의 장점**

**1. 접근속도**

원시타입은 '스택' 메모리에 값이 존재한다.

반면에 참조타입은 하나의 인스턴스이기 때문에 '스택' 메모리에는 참조값만 있고, 실제 값은 힙 메모리에 존재한다.

그리고 값을 필요로 할 때마다 언박싱 과정을 거쳐야 하니 원시타입과 비교해서 접근 속도가 느려지게 된다.

예외적으로 엄청 큰 숫자를 복사해야 한다면, 참조값만 넘길 수 있는 참조타입이 좋을 수 도 있다.

**2. 메모리 양**

원시 타입보다 참조 타입이 사용하는 메모리양이 압도적으로 높다.

따라서 메모리 사용적으로도 원시 타입이 참조 타입보다 효율적으로 사용할 수 있다.

**💡** **인터페이스 개념과 역할**

극단적으로 동일한 목적 하에 동일한 기능을 수행하게끔 강제하는 것이 바로 인터페이스의 역할이자 개념이다.

자바의 다형성을 극대화하여 개발코드 수정을 줄이고 프로그램 유지보수성을 높이기 위해 인터페이스를 사용한다.

**💡** **인터페이스 문법과 다형성 이해**

인터페이스는 interface 키워드를 통해 선언할 수 있으며 implements 키워드를 통해 일반 클래스에서 인터페이스를 구현할 수 있다.

또한, JAVA8 이전까지는 상수, 추상메소드만 선언이 가능하지만,(상수, 추상메소드만 가능케했다는 것을 통해 그만큼 강제성이 강했다는 것을 유추할 수 있다.)

JAVA8부터 디폴트메소드, 정적메소드가 추가되었다.

**상수 : 인터페이스에서 값을 정해줄테니 함부로 바꾸지 말고 제공해주는 값만 참조해라 (절대적)**

//상수

타입 상수명 = 값;

**추상메소드 : 가이드만 줄테니 추상메소드를 오버라이팅해서 재구현해라. (강제적)**

//추상 메소드

타입 메소드명(매개변수, ... );

**디폴트메소드 : 인터페이스에서 기본적으로 제공해주지만, 맘에 안들면 각자 구현해서 써라. (선택적)**

//디폴트 메소드

default 타입 메소드명(매개변수, ... ){

//구현부

}

**정적메소드 : 인터페이스에서 제공해주는 것으로 무조건 사용 (절대적)**

//정적 메소드

static 타입 메소드명(매개변수) {

//구현부

}

**💡** **많이 쓰는 14가지 핵심 GoF 디자인 패턴의 종류**

디자인 패턴은 프로그램을 개발하는 과정에서 빈번하게 발생하는 디자인 문제를 정리해서 상황에 따라 간편하게 적용할 수 있게 정리한 것입니다. 잘 활용할 수만 있다면 적지 않은 시간과 노력, 시행착오를 줄일 수 있습니다.

**생성 패턴(Creational Pattern)**

객체 인스턴스를 생성하는 패턴으로, 클라이언트와 그 클라이언트가 생성해야 하는 객체 인스턴스 사이의 연결을 끊어 주는 패턴입니다.

**1. 싱글턴 패턴(Singleton Pattern)**

특정 클래스에 객체 인스턴스가 하나만 만들어지도록 해주는 패턴입니다. 싱글턴 패턴을 사용하면 전역 변수를 사용할 때와 마찬가지로 객체 인스턴스를 어디서든지 액세스 할 수 있게 만들 수 있습니다. 클래스 인스턴스를 하나만 만들고 그 인스턴스로의 전역 접근을 제공합니다.

2. 추상 팩토리 패턴(Abstract Factory Pattern)

구상 클래스에 의존하지 않고도 서로 연관되거나 의존적인 객체로 이루어진 제품군을 생산하는 인터페이스를 제공합니다. 구상 클래스는 서브 클래스에서 만듭니다.

3. 팩토리 메소드 패턴(Factory method pattern)

객체를 생성할 때 필요한 인터페이스를 만듭니다. 어떤 클래스의 인스턴스를 만들지는 서브클래스에서 결정합니다. 팩토리 메소드 패턴을 사용하면 클래스 인스턴스 만드는 일을 서브클래스에게 맡기게 됩니다.

**행동 패턴(Behavioral Pattern)**

클래스와 객체들이 상호작용하는 방법과 역할을 분담하는 방법을 다루는 패턴입니다.

1. 템플릿 메소드 패턴(Template Method Pattern)

알고리즘의 골격을 정의합니다. 템플릿 메소드를 사용하면 알고리즘 일부 단계를 서브클래스에서 구현할 수 있으며, 알고리즘의 구조는 그대로 유지하면서 알고리즘의 특정 단계를 서브클래스에서 재정의할 수도 있습니다.

2. 상태 패턴(State Pattern)

상태 패턴을 사용하면 객체의 내부 상태가 바뀜에 따라서 객체의 행동을 바꿀 수 있습니다. 마치 객체의 클래스가 바뀌는 것과 같은 결과를 얻을 수 있습니다.

3. 반복자 패턴(iterator pattern)

컬렉션의 구현 방법을 노출하지 않으면서 집합체 내의 모든 항목에 접근하는 방법을 제공합니다.

**4. 전략 패턴(Strategy Pattern)**

알고리즘군을 정의하고 캡슐화해서 각각의 알고리즘군을 수정해서 쓸 수 있게 해 줍니다. 전략 패턴을 사용하면 클라이언트로부터 알고리즘을 분리해서 독립적으로 변경할 수 있습니다.

런타임 중에 알고리즘을 선택할 수 있게 하는 디자인 패턴

**5. 옵저버 패턴(observer pattern)**

한 객체의 상태가 바뀌면 그 객체에 의존하는 다른 객체에게 연락이 가고 자동으로 내용이 갱신되는 방식으로 일대다 의존성을 정의합니다.

많은 객체 간의 일대다 관계인 경우의 기초

**구조 패턴(Structural Pattern)**

클래스와 객체를 더 큰 구조로 만들 수 있게 구상을 사용하는 패턴입니다.

1. 데코레이터 패턴(Decorator Pattern)

데코레이터 패턴으로 객체에 추가 요소를 동적으로 더할 수 있습니다. 데코레이터를 사용하면 서브클래스를 만들 때보다 훨씬 유연하게 기능을 확장할 수 있습니다.

2. 프록시 패턴(Proxy Pattern)

특정 객체로의 접근을 제어하는 대리인(특정 객체를 대변하는 객체)을 제공합니다.

3. 컴포지트 패턴(Composite Pattern)

컴포지트 패턴으로 객체를 트리구조로 구성해서 부분-전체 계층구조를 구현합니다. 컴포지트 패턴을 사용하면 클라이언트에서 개별 객체와 복합 객체를 똑같은 방법으로 다룰 수 있습니다.

4. 어댑터 패턴(Adapter pattern)

특정 틀래스 인터페이스를 클라이언트에서 요구하는 다른 인터페이스로 변환합니다. 인터페이스가 호환되지 않아 같이 쓸 수 없었던 클래스를 사용할 수 있게 도와줍니다.

5. 퍼사드 패턴(facade pattern)

서브시스템에 있는 일련의 인터페이스를 통합 인터페이스로 묶어줍니다. 또한 고수준 인터페이스도 정의하므로 서브시스템을 더 편리하게 사용할 수 있습니다.