[💡](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 설명해주세요.**

* 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 설계되었습니다.

**💡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 필요없으며, 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 도와줌(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, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* int와 char를 제외한 클래스들은 기본형의 첫글자를 소문자에서 대문자로만 바꾸면 된다.

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

**💡 오토박싱 & 언박싱(autoboxing & unboxing)**

래퍼 클래스는 산술 연살을 위해 정의된 클래스가 아니다.

따라서, 인스턴스에 저장된 값을 변경할 수 없다.

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

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

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