💡Java의 특징

- 객체지향 프로그래밍 언어

- 기본 자료형을 제외한 모든 요소들이 객체로 표현되고, 객체 지향 개념의 특징인

캡슐화, 상속, 다형성이 잘 적용된 언어

- 장점

\* JVM(자바가상머신) 위에서 동작하기 때문에 운영체제에 독립적이다.

\* GabageCollector를 통한 자동적인 메모리 관리가 가능하다.

- 단점

\* JVM 위에서 동작하기 때문에 실행 속도가 상대적으로 느리다.

\* 다중 상속이나 타입에 엄격하며, 제약이 많다.

💡JVM의 역할

- JVM은 스택 기반으로 동작, Java Byte Code를 OS에 맞게 해석 해주는 역할을 하고 가비지컬렉션을 통해 자동적인 메모리 관리를 해준다.

💡Java의 컴파일 과정

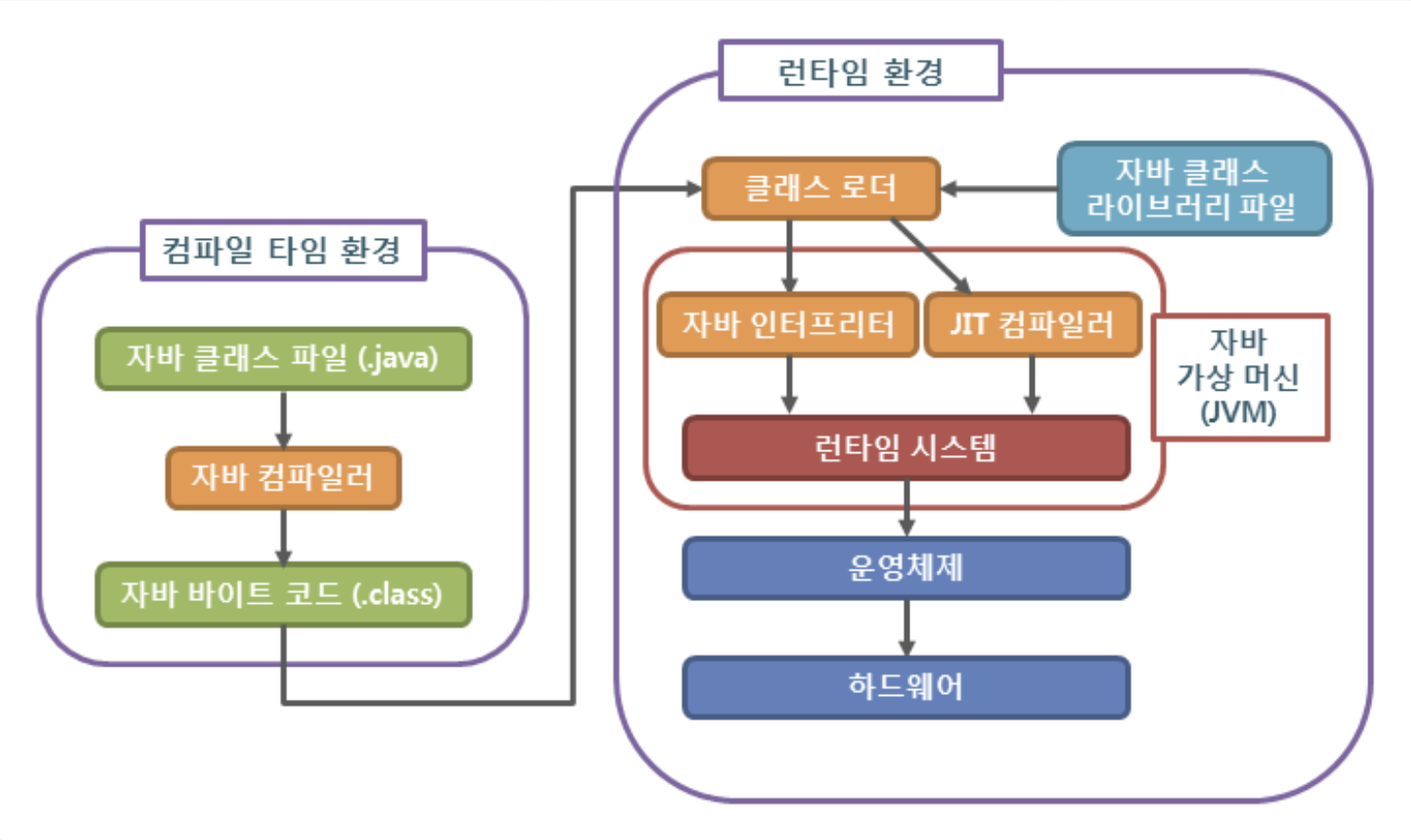
1. 개발자가 .java 파일을 생성한다.

2. build(빌드)를 한다.

3. java compiler의 javac의 명령어를 통해 바이트코드(.class)를 생성한다.

4. Class Loader를 통해 JVM 메모리 내로 로드한다.

5. 실행엔진을 통해 컴퓨터가 읽을 수 있는 기계어로 해석한다.(각 운영체제에 맞는 기계어)



💡Java에서 제공하는 원시 타입들에 무엇이 있고, 각각 몇 바이트를 차지하는지

- 정수형 byte, short, int, long / 실수형 float, double / 문자형 char / 논리형 Boolean

- 정수형 1, 2, 4, 8 / 실수형 4, 8 / 문자형 2 / 논리형 1 바이트

💡오버라이딩(Overriding)과 오버로딩(Overloading)

- 오버라이딩 : 상위 클래스에 있는 메소드를 하위 클래스에서 재정의 하는 것

- 오버로딩 : 매개변수의 개수나 타입을 다르게 하여 같은 이름의 메소드를 여러 개 정의 하는 것

💡객체지향 프로그래밍(OOP)

- 우리가 실생활에서 쓰는 모든 것을 객체라 하며, 객체 지향 프로그래밍은 프로그램 구현에 필요한 객체를 파악하고 상태와 행위를 가진 객체를 만들고 각각의 객체들의 역할이 무엇인지 정의하여 객체들 간의 상호작용을 통해 프로그램을 만드는 것

즉, 기능이 아닌 객체가 중심이며 “누가 어떤 일을 할 것인가?”가 핵심

특징 : 캡슐화, 상속, 다형성, 추상화 등이 있고. 모듈 재사용으로 확장 및 유지보수가 용이

💡try-with-resources

- try-with-resources는 try-catch-finally의 문제점을 보완하기 위해 나온 개념

- try(…) 안에 자원 객체를 전달하면, try블록이 끝나고 자동으로 자원 해제 해주는 기능을 말한다.

- 따로 finally 구문이나 모든 catch 구문에 종료 처리를 하지 않아도 되는 장점이 있음

💡불변 객체? 대표적인 Java의 예시

- 불변 객체는 객체 생성 이후 내부의 상태가 변하지 않는 객체

- Java에서는 필드가 원시 타입인 경우 final 키워드를 사용해 불변 객체를 만들 수 있고,

- 참조 타입일 경우엔 추가적인 작업이 필요하다

📙참조 타입 일 경우 추가적인 작업은 어떤게 있는지 설명해주세요.

- 참조 타입은 대표적으로 1. 객체를 참조할 수도 있고, 2. 배열이나 3.List 등을 참조할 수 있다.

1. 참조 변수가 일반 객체인 경우 객체를 사용하는 필드의 참조 변수도 불변 객체로 변경해야 한다.

2. 배열일 경우 배열을 받아 copy해서 저장하고, getter를 clone으로 반환하도록 한다.

(배열을 그대로 참조하거나, 반환할 경우 외부에서 내부 값을 변경할 수 있음. 때문에 clone을 반환해 외부에서 값 변경하지 못하게 함)

3. 리스트인 경우에도 배열과 마찬가지로 생성시 새로운 List를 만들어 값을 복사하도록 해야 한다. 배열과 리스트는 내부 복사하여 전달하는 데, 이를 방어적 복사(defensive-copy)라고 한다.

📘불변 객체나 final을 굳이 사용해야 하는 이유가 있을까요?

1. Thread-Safe하여 병렬 프로그래밍에 유용하며, 동기화를 고려하지 않아도 괸다.

(공유 자원이 불변이기 때문에 항상 동일한 값을 반환하기 때문)

2. 실패 원자적인 메소드를 만들 수 있다.

(어떠한 예외가 발생되더라도 메소드 호출 전의 상태를 유지할 수 있어 예외 발생 전과 똑같은 상태로 다음 로직 처리 가능)

3. 부수효과를 피해 오류를 최소화 할 수 있다.

※부수효과 : 변수의 값이 바뀌거나 객체의 필드 값을 설정하거나 예외나 오류가 발생하여 실행이 중단되는 현상

4. 메소드 호출 시 파라미터 값이 변하지 않는다는 것을 보장할 수 있다.

5. 가비지 컬렉션 성능을 높일 수 있다.

(가비지 컬렉터가 스캔하는 객체의 수가 줄기 때문에 Gc 수행 시 지연시간도 줄어든다.)

💡추상 클래스와 인터페이스

- 추상 클래스 : 클래스 내 추상 메소드가 하나 이상 포함되거나 abstract로 정의된 경우

- 인터페이스 : 모든 메소드가 추상 메소드로만 이루어져 있는 것

- 공통점

○ new 연산자로 인스턴스 생성 붉가능

○ 사용하기 위해서는 하위 클래스에서 확장/구현 해야 한다.

- 차이점

○ 인터페이스는 그 인터페이스를 구현하는 모든 클래스에 대해 특정한 메소드가 반드시 존재하도록 강제함에 있고,

○ 추상클래스는 상속받는 클래스들의 공통적인 로직을 추상화 시키고, 기능 확장을 위해 사용한다.

○ 추상클래스는 다중상속이 불가능하지만, 인터페이스는 다중상속이 가능하다.

**💡**싱글톤 패턴

- 싱글톤 패턴은 단 하나의 인스턴스를 생성해 사용하는 디자인 패턴

- 인스턴스가 1개만 존재해야 한다는 것을 보장하고 싶은 경우와

- 동일한 인스턴스를 자주 생성해야 하는 경우에 주로 사용한다. (메모리 낭비 방지)

📙싱글톤 패턴의 대표적인 예시

- 싱글톤 패턴의 대표적인 예시는 Spring Bean이다.

스프링의 빈 등록 방식은 기본적으로 싱글톤 스코프이고, 스프링 컨테이너는 모든 빈들을 싱글톤으로 관리한다.

스프링은 요청할 때마다 새로운 객체를 생성해서 반환하는 기능도 제공한다.(프로토타입 빈, @Scope(“prototype”))