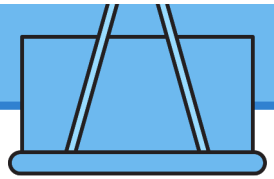


7 월 8 일

오늘의 수업 주제

객체지향 프로그래밍

&

자바

오늘의 학습목표

1. 객체지향 프로그래밍이란?
2. 자바의 컴파일 방식
3. 자바의 GC

개념과 정의

객체지향이란?

절차지향과의 차이점은?

객체지향의 4원칙

객체란?

- **객체란 우리 주변에 있는 모든 것이 될 수 있다.**
  - ex) TV, 컴퓨터, 책, 자동차 등 전부 객체가 될 수 있음

- **자신만의 고유한 특성과 행동을 가지며 다른 객체에게 행동을 요청하거나 정보를 주고 받는 등 상호작용 하며 존재**

- **객체는 크게 속성(필드, field)과 동작(메소드, method)로 구성됨**
  - **ex) 자동차의 속성 : 차종, 연식, 가격 등**
  - **ex) 자동차의 동작 : 가속, 정지, 기어변경 등**

객체지향 VS 절차지향

객체지향

- 문제를 여러개의 객체로 나누어 처리
  - 코드 재사용이 용이
  - 유지보수가 쉬움
  - 처리속도가 상대적으로 느림
  - 객체가 많아질수록 용량이 커짐
  - 설계에 많은 시간과 노력이 필요
- JAVA, Python, C# 등등

절차지향

- 문제를 여러개의 함수로 나누어 처리
  - 처리속도, 실행속도가 빠름
  - 유지보수가 어려움
  - 대규모 프로젝트에 불리
- C언어

객체지향의 4대원칙

캡슐화

- 관련된 필드와 메소드를 하나로 묶고 실제 구현내용을 외부로부터 감추는 기법
- 외부에서는 공개된 메소드를 통해 접근 가능
- ex) 알약(캡슐약)
  - 캡슐에 들어있는 약은 어떤 성분인지, 어떤 색인지 등등을 알 수 없으며, 외부의 접근으로부터 안전한 상태

객체지향의 4대원칙

상속성

- 상속이란 상위클래스의 모든 것을 하위클래스가 이어 받는 것
- 이미 작성된 클래스(상위 클래스) 의 특성을 그대로 이어받아
  새로운 클래스(하위 클래스)를 생성하는 기법
- 코드의 중복을 방지하기 위함임
  => 코드의 중복이 많아지면 개발단계와 유지보수에서 많은 비용이 소요됨

객체지향의 4대원칙

추상화

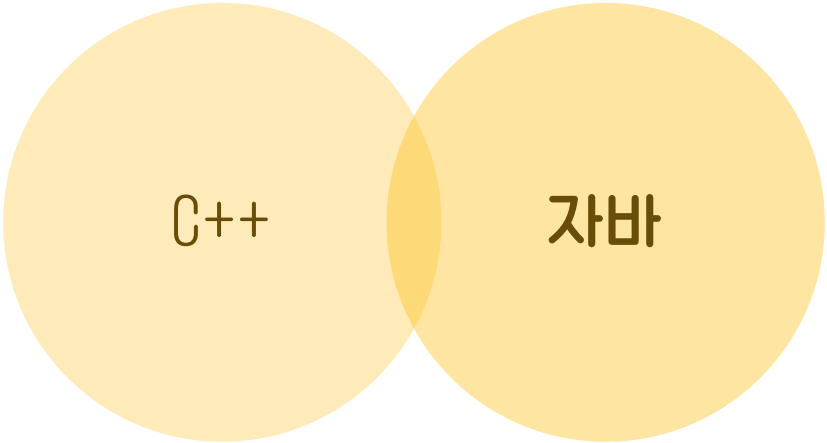
- 객체에서 공통된 속성과 행위를 추출
- 객체들의 공통적인 특성을 파악하고 불필요한 특성을 제거하는 과정

객체지향의 4대원칙

다형성

- 다양한 형태로 나타날 수 있는 능력을 말함
- 같은 이름의 메소드를 호출하더라도 객체에 따라 다르게 동작하는 것을 의미
- 상위클래스의 동작을 하위클래스에서 재정의 하는 overriding 또한 다형성
- 이름은 같지만 서로 다르게 동작하는 overloading 또한 다형성

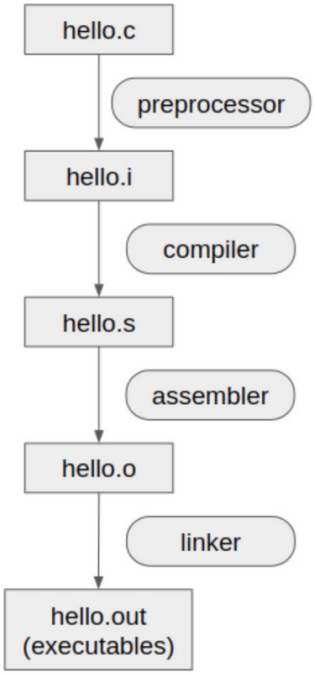
컴파일 방식



컴파일 방식

C++

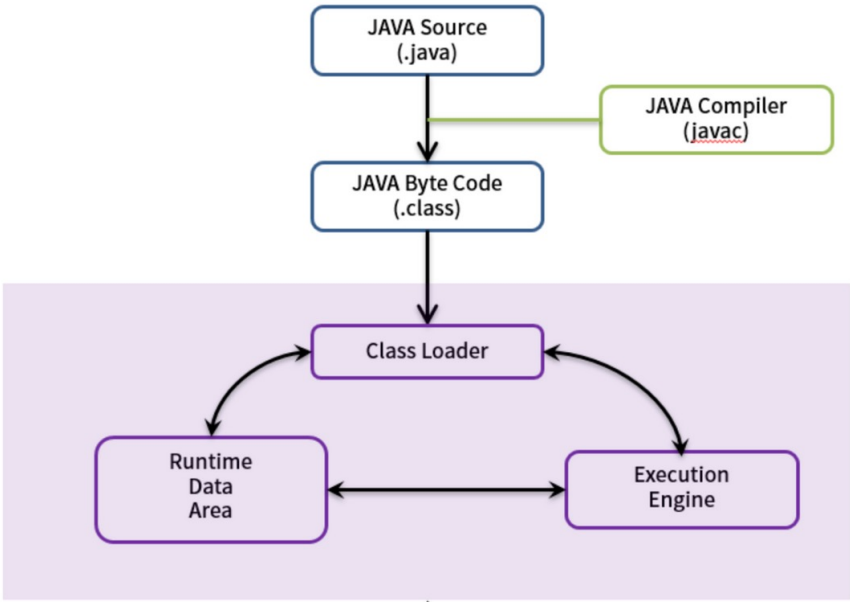
- 전처리기 (preprocessor) : 주석 제거, define 치환 등등
- 컴파일러 (compiler) : 어셈블리 파일로 변환
  - 어셈블리어 : CPU 명령어들의 조합 -> CPU에 의존적
- 어셈블러 (assembler) : object 코드 파일로 변환, 아직 주소 정보 매핑 x
  - object code : 0과 1로 이루어진 이진 코드
- 링커 (linker) : 오브젝트 파일들을 모아 실행코드로 변환, 운영체제가 로딩할 수 있도록 주소 정보를 할당한 파일을 만듬 -> 운영체제에 의존적



컴파일 방식

JAVA

- 소스코드를 JAVA Compiler가 byte code로 컴파일함 -> .class 파일 생성
- 생성된 .class 파일을 Class Loader가 Runtime Data Area (JVM의 메모리) 에 올림
- 실행엔진(Excution Engine) 은 메모리에 올라온 바이트 코드들을 명령어 단위로 하나씩 가져와서 실행함
  - 이때 실행 방식은 interpreter, JIT compiler 두가지

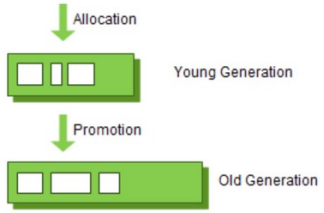


Garbage Collection

기본 개념

- 개발을 하다보면 유효하지 않은 메모리인 Garbage가 발생하게 된다.
- C언어는 free()라는 함수를 통해 직접 메모리 해제를 해줘야 한다.
- JAVA에서는 직접 해제할 필요가 없다.
  => JAVA에서는 JVM의 GC가 불필요한 메모리를 알아서 정리해준다.

Garbage Collection



Minor GC & Major GC

- JVM의 Heap은 처음 설계될 때 다음 2가지를 전제로 설계됨
  - 대부분의 객체는 금방 접근 불가능한 상태가 된다.
  - 오래된 객체에서 새로운 객체로의 참조는 아주 적게 일어난다.
- Young 영역 (Minor GC)
  - 새로 생성된 객체가 할당되는곳
  - 대부분의 객체가 금방 unreachable 되기 때문에 많은 객체가 여기서 생성되고 사라짐
- Old 영역 (Major GC)
  - young 영역에서 reachable 상태를 유지하며 살아남은 객체가 복사되는 영역
  - young 영역보다 크게 할당되며, 크기가 큰 만큼 garbage가 적게 발생

심화학습 주제

- 객체지향 개발 5원리 : SOLID
- interpreter VS JIT compiler
- GC (Garbage Collection)

7 월 15 일

다음의 수업 주제

1. 네트워크, DB 기초

2. 스프링 프레임워크란?