섹션0 - Java 개발 시작하기

- 기업에서 백엔드 개발 시 가장 많이 사용하는 언어

- 기본적으로 객체지향 언어

++ 버전이 증가하면서 다양한 장점들 흡수

(람다, stream API, 병렬 프로그래밍 등)

Hello.java로 java program 생성

-> Hello class안에 Hello를 출력하는 main함수 작성

=> 이후 javac Hello.java 명령어로 컴파일 -> Hello.class 생성됨

-> java Hello를 명령어로 입력하면 JVM이 실행되면서 Hello.class를 실행

(자바프로그램을 실행하는 것은 .class 파일을 실행하는 것)

- 자바는 운영체제 dependent하지 않게 모든 운영체제에서 동일한 프로그램이 돌아가기를 원했음

-> 소스코드를 기계어로 변환할 때, 바로 변환하지 말고 그 중간에 바이트 코드를 생성하자

=> .class file들이 byte코드 <-> C언어는 컴파일하면 바로 기계어 파일이 생성

=> java는 JVM을 다양한 운영체제와 CPU에 맞게 제공(intel 용, MAX용 등 -> 각 개발자는 자신의 CPU에 알맞은 JVM을 설치해야 함)

-> 해당 PC에 맞는 JVM이 설치되어 있다면 .class 바이트 코드가 어느 운영체제에서 작성되었든 상관없이 실행이 가능

=> JVM은 바이트코드를 한 줄씩 읽어들이면서 인터프리터 방식으로 기계어로 변환하면서 실행

결론: 자바에서 컴파일은 기계어를 생성하는 것이 아니라 바이트 코드를 생성하게 된다 -> 이후 이러한 바이트 코드는 JDK안에 JVM이 한 줄씩 읽으면서 실행

섹션1 - 자바 기본 문법

- 기본형 type: int, short, long, char, Boolean 등

- 참조형 type: 기본형이 아닌 type

=> class나 interface가 해당됨(대문자로 시작하는 것이 좋음)

기본형은 메모리를 직접 할당받고 값을 저장하고 변수는 해당 메모리를 가르키는 것

참조형은 값을 직접 갖는 것이 아니라 참조하는 형식

++ i = 1과 같이 선언하면 1은 정수 리터럴이라고 함(리터럴: 변수에 입력되는 값)

섹션2 - 객체지향 문법 1/3

class는 구조(filed와 method를 가짐)

class로 만드는 객체들이 object(=instance)

생성한 인스턴스를 사용하려면 참조를 해야함

book이라는 class가 있다고 가정하자

book b = new book();이라고 선언할 때,

- new: 할당, 인스턴스 생성

- b: book(): book class의 생성자로 생성

- b: 참조 변수, reference variable

=> 결과적으로 Heap 메모리 영역에 book class의 instance가 생성되고 영역을 차지

-> 이때 이 메모리 영역을 참조하는 변수가 b인 것

++그냥 new book()이;이라고 하면, Heap 영역에 book instance가 할당되어 생성됨 but 이 메모리를 참조하는 변수가 없는 것 -> garbage가 생성되는 것과 동일함

++ 이때 참조하는 과정에 대해서는 알 필요없음, C의 pointer 개념이 자바에는 없기에 그냥 자바가 알아서 참조해준다고 생각하면 됨

인스턴스를 만드는 3가지 방법

1. new연산자와 생성자를 이용

2. 클래스 로더를 이용

3. 메모리에 있는 인스턴스를 복제하여 생성

java에서 new keyword를 사용하면 heap이라는 메모리 영역에 해당 인스턴스가 동적할당됨

VendingMachine vm1 = new VendingMachine();

VendingMachine vm2 = new VendingMachine();

와 같이 선언하면 VencdingMachine class의 인스턴스 2개가 heap에 할당됨

++

intellij가 아닌 터미널에서 직접 컴파일하려면

javac VendingMachine.java <- class 컴파일

javac VendingMachineMain.java <- class 컴파일

java VendingMachineMain <- main method가 있는 class파일 실행

과 같은 명령어 3개로 가능

여기서 java VendingMachineMain을 입력하면, JVM은 VendingMachineMain.class파일을 찾아서 실행시켜줌 -> 이걸 어디서 어떻게 찾아서 실행?

=> 현재 폴더? 아님

=> CLASSPATH 경로에서 클래스를 찾아서 실행하는 것

=> CLASSPATH=. 으로 설정되어 있어서 현재 경로에서 class파일을 찾아 실행시키는 것

-> CLASSPATH로 설정되어 있는 위치에서 class파일을 탐색해서 실행시켜줌

++ intellij에서 현재 예제를 기준으로 CLASSPATH = out/production/HappyJava로 설정되어 있어서 이 경로에서 class파일을 찾아 실행하는 것

텍스트, 친필, 아동 미술, 그림이(가) 표시된 사진

자동 생성된 설명

=> JVM은 찾은 class 정보를 PERM이라는 메모리 영역에 저장(인스턴스가 아니라 class 정보 그 자체가 올라가는 것)

-> 메모리에 올라갔으니 JVM은 해당 class가 어떤 static method를 가지는 지, 어떤 인스턴스 method를 가지는 지 알 수 있음

=> JVM은 main method를 찾고, 실행시킴

-> main을 실행하면서 JAVA Stack 메모리에 올려서 실행시킴

-> 이후 다른 method가 실행되면 그 때마다 JAVA stack에 점점 쌓이는 것

=> 각 method의 지역변수들은 각각 method의 스택 엔트리에 저장됨

예를 들어 main의 args, vm1, vm2는 main method의 스택엔트리에 저장되어 있음 -> 각각은 동적할당 되어 있는 참조변수 이기에 heap영역에 저장되어 있는 메모리를 참조하는 형태로 존재

- JAVA math API

field로 E와 PI를 가짐 <- 둘 다 static

=> 인스턴스를 안 만들어도 사용 가능하고, Math.E, Math.PI로 사용하는 것이 적합함

또한 method들이 모두 static

=> math는 인스턴스를 생성해서 사용하는 것이 아니라 Math.abs(-1)와 같이 사용해야할 것