03-11 [Java]

소유자	종수 김
태그	

[Java 메모리 구조]

크게 3가지 영역으로 구분

- 클래스 정보를 보관하는 메서드 영역
- 실제 프로그램이 실행되는 영역 메서드를 실행할 때마다 하나씩 쌓임. 스택 영역
- 객체(인스턴스)가 생성되는 영역 new 명령어를 사용하면 이 영역을 사용, 실 생성된 인스턴스가 존재하는 공간. 힙 영역

[메서드 영역]
프로그램을 실행하는데 필요한 공통 데이터를 관리 / 이 영역은 프로그램의 모든 영역에서 공유

- 클래스 정보
  - 클래스의 실행 코드(바이트 코드), 필드, 메서드와 생성자 코드등 모든 실행 코드 존재
- static 영역
  - static 변수들을 보관
- 런타임 상수
  - 프로그램을 실행하는데 필요한 공통 리터럴 상수 보관
  - Ex) StringConstantPool
- 메서드는 메서드 영역에,
  = new를 통해 생성되는 객체의 경우 인스턴스마다 다른 메모리 주소를 갖고있고 힙 영역에서 관리되지만,
  메서드는 메서드가 여러 개 있다고 각각 다른 주소를 할당 할 필요는 없기에 효율적인 메모리 관리를 위해 메서드 영역에서 관리된다.
  각각 다른 인스턴스에서 내부 메서드를 호출하더라도 메서드 영역에 있는 메서드 주소를 공통적으로 사용한다.

[스택 영역]
자바 실행 시, 하나의 실행 스택이 생성 각 스택 프레임은 지역 변수, 중간 연산 결과, 메서드 호출 정보 등을 포함

- 스택 프레임
  - 스택 영역에 쌓이는 네모 박스가 하나의 스택 프레임,
    메서드를 호출할 때 마다 하나의 스택 프레임이 쌓이며 메서드가 종료되면 스택 프레임이 제거
- 스택 영역은 쓰레드 별로 하나의 실행 스택이 생성
  즉, 쓰레드 수 만큼 스택 영역이 생성. 톰캣과 같이 멀티 쓰레드로 동작한다면 스택 영역도 쓰레드 갯수만큼 할당

[힙 영역]
객체(인스턴스)와 배열이 생성되는 영역. 가비지 컬렉션이 이루어지는 주요 영역이며, 더 이상 참조되지 않는 객체는 GC에 의해 제거

[스택과 큐]

데이터를 보관하고 관리하는 구조

[스택]

- 후입 선출(Last In First Out)
  - 마지막에 넣은게 먼저 나오는 구조.
  - 입구가 하나인 좁은 블록

[큐]

- 선입 선출(First In First Out)
  - 처음 넣은게 먼저 나오는 구조
  - 미끄럼틀처럼 먼저 들어간 사람이 먼저 나오는 구조

[스택 영역]

```
package memory;

public class JavaMemoryMain1 {
```

```
    public static void main(String[] args) {
        System.out.println("main start");
        method1(10);
        System.out.println("main end");

    }

    static void method1(int m1) {
        System.out.println("method1 start");
        int cal = m1 * 2;
        method2(cal);
        System.out.println("method1 end");
    }
    static void method2(int m2) {
        System.out.println("method2 start");
        System.out.println("method2 end");
    }
}

==>
main start
method1 start
method2 start
method2 end
method1 end
main end

Process finished with exit code 0
```

모든 자바 파일은 메인 메서드가 시작지점이며, 호출되는 메서드들은 스택 구조를 이용하여 본인을 호출한 메서드 위에 스택프레임을 계속 쌓아간다.

모든 프로그램이 종료되면(더 이상 진행할 스택 프레임이 존재하지 않는다면), 프로그램을 종료한다.

[스택 영역과 힙 영역]

```
package memory;

public class JavaMemoryMain2 {
    public static void main(String[] args) {
        System.out.println("main start");
        method1(10);
        System.out.println("main end");

    }

    static void method1(int m1) {
        System.out.println("method1 start");
        Data data = new Data(1);
        method2(data);
        System.out.println("method1 end");
    }
    static void method2(Data data) {
        System.out.println("method2 start");
        System.out.println("data.getValue() = " + data.getVal
        System.out.println("method2 end");
    }
}
```

method1 , method2는 같은 data객체를 바라보고 있음.

method2가 종료되며 data를 참조하고 있는 곳이 한 곳 사라지고,

method1가 종료되면 data를 참조하고 있는 곳은 더 이상 존재하지 않는다.

- 이 때 GC는 참조가 모두 사라진 인스턴스의 메모리를 제거한다.
- 힙 영역 외부가 아닌, 힙 영역 안에서만 인스턴스끼리 서로 참조하는 경우에도 GC의 대상

지역 변수는 스택 영역에서 관리,

객체 인스턴스는 힙 영역에서 관리,

메서드 영역에서 관리하는 변수는 ?

[static 변수]

static 키워드는 주로 변수와 메서드에 사용

인스턴스에 구애받지 않는 공통된 변수를 사용하고 싶을 때, 공용 변수로 클래스명 + .(dot)으로 접근.

- static이 붙은 멤버 변수는 메서드 영역에서 관리.
    - static이 붙은 멤버 변수는 인스턴스 영역에서 생성되지 않는다.
      대신에 메서드 영역에서 이 변수를 관리한다.

```
public class Data3 {
 private String name;
 private static int count;
}
```

1. 예제 코드에서 name, count 모두 멤버 변수(필드)이다.
2. 멤버 변수는 인스턴스 변수, 클래스 변수로 나뉘어진다.
    a. 인스턴스 변수
        i. static이 붙지 않은 멤버 변수
        ii. static이 붙지 않은 멤버 변수는 인스턴스를 생성해야 사용할 수 있고, 인스턴스에 소속.
        iii. 인스턴스를 새로 생성할 때 마다 새로 만들어짐.
    b. 클래스 변수
        i. 클래스 변수, 정적 변수, static 변수 등으로 부름.
        ii. static이 붙은 멤버 변수는 인스턴스와 무관하게 클래스에 바로 접근해서 사용할 수 있고, 클래스 자체에 소속. ⇒ 인스턴스를 통해서도 접근이 가능하긴 하나, 인스턴스에 있는 변수로 사용하는 것이 아닌, 다시 클래스 영역에 static영역에 가서 static 변수로 사용
        iii. 클래스 변수는 자바 프로그램을 시작할 때 딱 '한 개'만 만들어지며 인스턴스와 다르게 여러곳에서 공유하는 목적을 가짐.

[변수와 생명주기]

1. 지역 변수 : 스택 영역에 있는 스택 프레임 안에 보관되며,
   메서드가 종료되면 스택 프레임이 제거되고 이 때 지역변수도 함께 제거.

2. 인스턴스 변수 : 인스턴스에 있는 멤버 변수이며 힙 영역에 보관
   GC가 발생하기 전까지 생존.
3. 클래스 변수 : 클래스 변수는 메서드 영역의 static영엑에 보관되는 변수.
   메서드 영역은 프로그램 전체에서 사용하는 공용 공간.
   클래스 변수는 해당 클래스가 JVM에 로딩되는 순간 생성되며, JVM이 종료될 때 까지 생명주기가 이어짐.

- 평균적으로 생명주기는
  클래스 변수 > 인스턴스 변수 > 지역 변수

[static 메서드]

멤버 변수도 없이 단순히 기능을 제공하는 경우, 해당 메서드를 사용하기 위해 객체를 생성할 필요가 없는 경우.
* 인스턴스가 필요한 이유는 멤버 변수(인스턴스 변수)등을 사용하는 목적이 큼.

[클래스 메서드]

static이 붙은 메서드, (= 정적 메서드, 클래스 메서드, static 메서드)

[인스턴스 메서드]

static이 붙지 않은 메서드

- 객체 생성 없이, 클래스에 있는 메서드를 바로 호출 가능
- 언제나 사용할 수 있는 것은 아님.

[정적 메서드 사용방법]

static 메서드는, static만 사용할 수 있다.

  - 클래스 내부의 기능을 사용할 때, 정적 메서드는 static이 붙은 정적 메서드나, 정적 변수만 사용할 수 있다.
  - 반대로 모든 곳에서 static을 호출할 수 있다.
    - 정적 메서드는 공용 기능이므로, 접근 제어자만 허락한다면 클래스 모든곳에서 static을 호출할 수 있음.

- static은 JVM이 로딩될 때 만들어지며, 인스턴스는 힙 영역에 필요한 순간 동적으로 할당됨.
  즉, 스태틱 메서드 및 변수는 인스턴스가 어느 주소를 갖게될 지 모르는 것. 불가능.
- 인스턴스는 이미 static 변수 및 메서드가 JVM이 기동될 때 메서드 영역에 올라가있으므로 주소를 알고있고, 이에따라 사용 가능.

```
package memory.staticmethod;

public class DecoData {
    private int instanceValue;
    private static int staticValue;

    public static void staticCall() {
        staticValue++; // 정적 변수 접근 : 접근 가능.
        staticValue(); // 정적 메서드 : 접근 가능.

//        instanceValue++; //인스턴스 변수 접근, 컴파일 에러
//        instanceMethod(); //인스턴스 메서드 접근, 컴파일 에러
    }

    public void instanceCall() {
        instanceValue++; // 본인의 인스턴스 밸류에 접근 : 접근 가능.
        instanceMethod(); // 본인의 인스턴스 메서드에 접근 : 접근 가

        staticValue++; // 정적 변수는 이미 JVM이 로딩될 때 메서드 영
        staticValue(); // 정적 메서드 또한 JVM이 로딩될 때 메서드 영
    }

    private void instanceMethod() {
        System.out.println("instanceValue = " + instanceValue
    }
    private static void staticValue() {
        System.out.println("staticValue = " + staticValue);
    }
}
```

[static import]

static 메서드에서 Class명.static메서드를 호출할 때, Class명을 생략할 수 있게 도와주는 기능

- 정적 변수에도 사용 가능

[Main Method]

인스턴스 생성 없이 실행하는 가장 대표적인 메서드
public
static void main(String[] args)

main()메서드는 프로그램을 시작하는 시작점.
객체 생성 없이 main메서드가 동작.

- 정적 메서드는 정적 메서드만 호출할 수 있음.
- 따라서 정적 메서드인 main이 호출하는 메서드에는 정적 메서드를 사용해야 함.
  * 정확히는, 정적 메서드는 같은 클래스 내부에서 정적 메서드만 호출 가능.