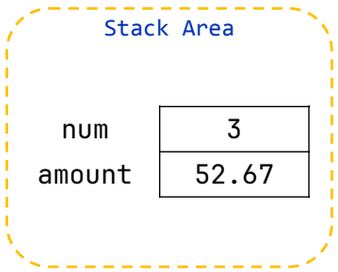
03_1 Reference Types

Object-Oriented Programming

본 강의에서는 객체지향기법을 본격적으로 다루기 전에
Java의 reference type에 대해 알아보도록 하겠습니다.

Primitive Types

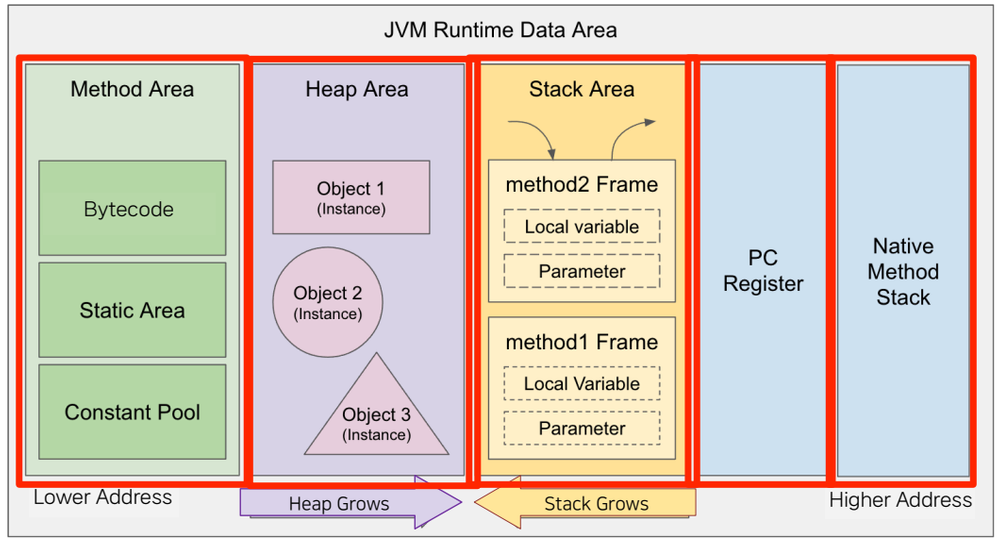
- Primitive Types
  - Integer Types
    - byte, char, short, int, long
  - Real Number Types
    - float, double
  - Caracter Type
    - char
  - Logical Types
    - boolean
- Ex)
  - `int num = 3;`
  - `float amount = 52.67;`



페이지 2

이전에 봤던 Java 언어의 data type에 대해 다시한번 review를 해 보겠습니다.
먼저 primitive type에는 integer를 나타내는 byte, char, short, int, long 등이 있고
real number를 표시하는 float, double과 character를 나타내는 char,
logical type인 boolean이 있습니다.
이 primitive type은 value만 하나를 가지는 단순한 type인데
그 value들은 memory의 stack area에 저장이 됩니다.

Memory Structure of JVM



페이지 3

Method Area에는 Java program에서 사용되는 class information과 함께
class variable, 즉, static variable이 저장됩니다.
이 area에는 Bytecode인 .class file을 읽어 들여서
class와 interface에 대한 runtime constant pool,
member variable (즉, field), class variable (즉, static variable),
constructor와 methods를 저장합니다.
이 area는 program이 시작하기 직전에 load되고 프로그램이 종료 될 때 소멸됩니다.
runtime constant pool에는 compile time에 알려진 숫자 리터럴(literal)부터
런타임에 확인되어야 하는 메소드 및 필드 참조에 이르기까지 여러 상수가 포함됩니다.
Method area는 memory의 가장 낮은 주소 영역에 존재하며
그 크기는 변하지 않습니다.

Heap Area는 Java 프로그램을 위한
모든 class object들의 instance들이 저장됩니다.
New operator를 실행하는 순간 object가 dynamic하게 allocate되고
이 object 데이터는 runtime에 JVM에 의해 관리됩니다.
Heap은 메모리의 가장 낮은 주소로 부터 높은 주소쪽으로 확장됩니다.

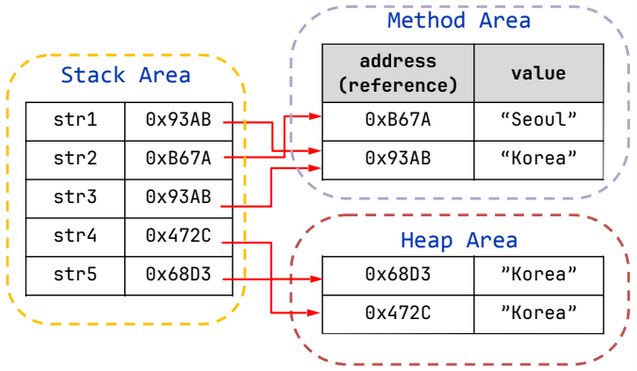
스택은 Java 프로그램에서 메서드가 호출될 때 메서드의 스택 프레임이 저장되는 영역입니다.
Java 프로그램에서 메서드가 호출되면 JVM은 메서드 호출과 관련된
로컬 변수 및 매개 변수를 스택 영역에 저장합니다.
이 스택 영역은 메서드 호출과 함께 할당되며 메서드 호출이 완료되면 소멸됩니다.
스택 영역에 저장된 메서드 호출 정보를 스택 프레임이라고 합니다.
스택 영역은 푸시를 통해 데이터를 입력하고 팝을 통해 데이터를 출력합니다.
이러한 스택은 마지막으로 저장된 데이터가 가장 먼저 팝되는
LIFO(Last in, First out) 방식으로 작동합니다.
스택 영역은 메모리에서 높은 주소에서 낮은 주소 방향으로 할당됩니다.

PC 레지스터는 현재 실행 중인 JVM 명령어의 주소를 포함합니다.
CPU의 PC(프로그램 카운터)에 해당하며,
프로그램 실행은 CPU에 명령을 내리는 방식으로 이루어집니다.
명령어를 실행하는 동안 CPU는 레지스터라고 하는 CPU의 메모리에 필요한 정보를 저장합니다.
CPU는 명령어를 실행하는 동안 필요한 정보를 레지스터에 저장하는데,
레지스터는 연산 결과를 메모리로 전달하기 전에 저장하는 CPU 내부 메모리 장치입니다.
따라서 Java의 철학을 실현하기 위해 CPU 레지스터의 이러한 역할은
JVM에서 논리적 메모리 영역을 말합니다.

네이티브 메서드 스택 영역은 Java 이외의 언어로 작성된
네이티브 코드를 위한 스택입니다.
즉, 자바 네이티브 인터페이스(JNI)를 통해 호출되는
C, C++ 등의 코드를 실행하기 위한 스택입니다.
네이티브 메서드의 매개변수, 로컬 변수 등을 바이트 코드로 저장합니다.

Reference Type

- Reference Types
  - Having reference (address) of objects and arrays
- Ex)
  - String str1 = "Korea";
  - String str2 = "Seoul";
  - String str3 = "Korea";
  - String str4 = new String("Korea");
  - String str5 = new String("Korea");



페이지 4

Reference type은 reference (주소) 를 저장함으로써
object나 array등으로 모여있는 데이터의 집합을 가리고 있습니다.
Stack area에 있는 reference variable들의 value는
method area나 heap area의 object의 주소로 볼 수 있습니다.
Java에서 모든 literal들은 class 정보를 가지고 있는 Method area에 위치합니다.
str1의 Korea라는 literal은 Method area에 위치하고 있으며
주소는 0x93AB 입니다.
str2는 literal "Seoul" 인데 역시 Method area에 있으며
주소는 0x867A 입니다.
str3에 literal "Korea"가 다시 나오는데 이 때 literal은 한번 출현한 것은
다시 만들지 않으며, 이미 있는 literal을 가리키게 합니다.
따라서 str3은 str1과 같은 주소를 가지게 됩니다.
반면에 new로 생성하는 object는 Heap area에
new를 call할때마다 생성됩니다.
먼저 str4의 "Korea" 는 new로 생성되었으므로
Heap area의 주소 0x472C를 가집니다.
그리고 str5도 new로 생성되었는데
str1, str3, str4와 같은 내용의 String이기는 하지만
new를 사용했기 때문에 완전히 다른 새 object로 생성됩니다.
따라서 str5는 주소 0x68D3를 가리키게 됩니다.

==, !=

- For Primitive Type Variables
  - Test whether the values of two variables are the same or not
  - ex)

```
int x = 3;
int y = 2;
if (x == y || x != y + 2) { ... }
```

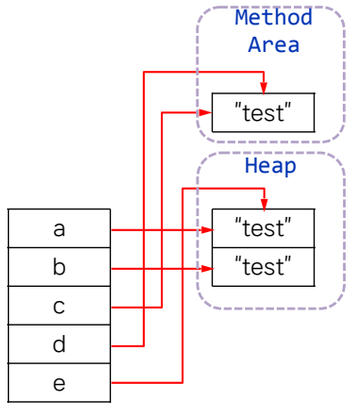
- For Reference Type Variables
  - Test whether the address (reference) of two variables are the same or not
  - That is, test whether the two variables are accessing the same object or not

페이지 5

primitive type variable들에서 == (이퀄)과 != (낫이퀄)의 의미는
두 variable들의 value가 같은지 아닌지를 테스트 하는 것입니다.
Reference type variable들에 대해서는
그 value들이 나타내는 reference (즉, 주소)가 같은지 아닌지를 나타내는 것입니다.
즉, 즉 두 reference variable이 같은 address를 참조하고 있는지 아닌지를 나타내는 것입니다.

==, !=

```java
public class TestTheSameReferences {
    public static void main(String[] args) {
        String a = new String("test");
        String b = new String("test");
        String c = "test";
        String d = "test";
        String e = a;
        System.out.println("a == b ? " + (a == b)); // false
        System.out.println("a == c ? " + (a == c)); // false
        System.out.println("a == d ? " + (a == d)); // false
        System.out.println("a == e ? " + (a == e)); // true
        System.out.println("b == c ? " + (b == c)); // false
        System.out.println("b == d ? " + (b == d)); // false
        System.out.println("b == e ? " + (b == e)); // false
        System.out.println("c == d ? " + (c == d)); // true
        System.out.println("c == e ? " + (c == e)); // false
    }
}
```



페이지 6

프로그램에 출현하는 String variable a, b, c, d, e 가
어느 곳의 무엇을 가리키게 되는지 주의깊게 살펴보도록 합시다.
new String("test") 로 create되는 a와 b는
정상적인 String object이기 때문에 heap 영역에 만들어지게 됩니다.
String의 내용이 a, b가 모두 "test" 이기는 하지만
이 들은 모두 별개로 생성된 object들 이므로
차지하고 있는 memory 영역이 다릅니다.
c와 d는 String literal을 assign 받는데, literal은 Method 영역에 만들어져서
같은 내용의 literal을 두번 만들지는 않습니다.
e는 a를 그대로 assign 했으므로 a와 같은 reference value를 가집니다.
이에 따라 아랫부분의 a, b, c, d, e 간의 equal 여부는
그 reference (주소)가 동일한 지의 여부에 따라 결정됩니다.

null

- Meaning: No object reference
  - ex)

```
String str = null;  // preferably initialized to null if there are no objects to reference
...
if (str == null)  { ... }  // if str doesn't reference any objects yet
else { ... }  // if str reference any valid object
```

- Dereferencing a 'null' reference causes a 'NullPointException'
  - ex)

```
String s = null;  // no object yet
System.out.println(s.length()); // Throws NullPointerException
```


페이지 7

null은 아무 object도 가리키지 않는 reference를 나타냅니다.
class variable을 declare하지만 그 object를 create할 수 없는 상황에서
일단 initial value를 null로 해 놓을 수 있습니다.
이렇게 해 놓으면 variable이 valid한 object를 가리키고 있는지 아닌지를
null과의 이퀄리티 테스트에 의해 쉽게 결정할 수 있습니다.
한편 null reference를 가진 class variable s가 있을 때
s가 valid한 object를 가리키고 있다고 착각하여
s.length() 처럼 method를 call하거나
s.x 처럼 instance variable을 access하려 하는 것은
NullPointException을 발생시킵니다.
exception은 error와는 종류가 다르며
exception이 있을 것을 대비한 code를 개발자가 미리 준비해 둘 수 있습니다.
exception handling에 대해서는 추후 더 자세히 알아볼 것입니다.

Example: Initialization as Null (1/2)

```
public class NullInitializationExample {
    public static void main(String[] args) {

        String str; // no initialization

        // COMPILE ERROR: "variable str might not have been initialized"

        if (str != null)
            System.out.println("Length of str: " + str.length());
        else
            System.out.println("str is null");

        str = null; // initialization

        if (str != null)
            System.out.println("Length of str: " + str.length());
        else
            System.out.println("str is null");
```

should be commented out

페이지 8

빨간 상자에 담겨 있는 코드는 NullPointerException을 발생시킵니다.
str이 아무런 initialization이 되어 있지 않기 때문에
str이 null과 같지 않냐는 테스트는 true로 통과하게 되어 있고
따라서 true part의 str.length() 를 시도하는 순간
exception이 발생하게 될 것입니다.
그래서 아랫부분에서 처럼
str을 일단 null로 초기화해 놓고
이 문제를 str이 null 이 아닌지를 테스트하는 것은 정확하기 때문에
else part에서 "str is null" 을 프린트하게 됩니다.

Example: Initialization as Null (2/2)

```
        str = "Hello"; // other case of initialization


        if (str != null)
            System.out.println("Length of str: " + str.length());
        else
            System.out.println("This line will not be reached");
    }
}
```

```
str is null
Length of str: 5
```


페이지 9

한편 str을 "Hello" 로 초기화 해 놓는다면
정상적으로 str 낫이쿠얼 null 이 true가 되고
"Length of str: " 과 함께 str.length() 가 프린트 됩니다.