2주차 과제 : 자바 데이터 타입, 변수 그리고 배열

프리미티브 타입 종류와 값의 범위 그리고 기본 값

	타입	기본값	값의 범위	값의 크기
정수형	byte	0	-128 ~ 127	1byte
	short	0	-32,768 ~ 32,767	2byte
	int	0	-2,147,483,648 ~ 2,147,483,647	4byte
	long	0L	9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807	8byte
실수형	float	0.0f	(3.4 X 10-38) ~ (3.4 X 1038) 의 근사값	4byte
	double	0.0	(1.7 X 10-308) ~ (1.7 X 10308) 의 근사값	8byte
문자형	char	'₩u0000'	0 ~ 65,535	2byte
논리형	boolean	FALSE	false, ture	1byte

프리미티브 타입과 레퍼런스 타입

- **Primitive Type**
  - 실제 데이터 값을 저장하는 타입
  - 기본값이 있기 때문에 null 존재 X
  - 값이 할당되면서, JVM 의 Runtime Data Area 의 `Stack 영역` 에 값이 저장됨
  - 값의 범위 벗어날 경우 컴파일 에러 발생
- **Reference Type**
  - 원시 타입을 제외한 모든 타입
    - Class, Interface, Enum, Array, String Type

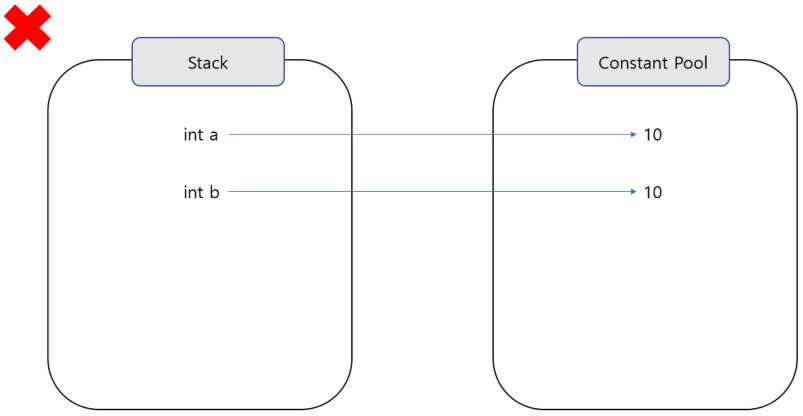
- 빈 객체를 의미하는 null 존재
- 값이 저장되어 있는 곳의 주소값을 저장하는 공간으로 `Heap 영역`에 저장

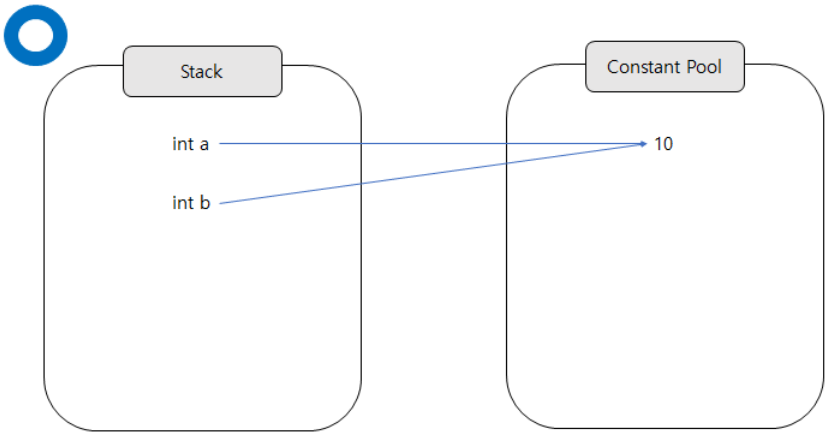
리터럴

- 종류 : 원시 타입, String
- 변하지 않는 데이터 그 자체, 쉽게 상수라고 생각하면 됨
- 원시 타입 예시

```
int a = 10;
int b = 10;
```

  - a, b 는 10이라는 동일한 값을 가지고 있고, 10 이라는 값을 Heap 메모리 내의 Constant Pool 이라는 메모리 영역에 할당하고
  - 똑같은 값을 또 호출했을 때 전에 할당했던 메모리 주소를 넘겨줌
  - a == b 가 true 인 이유 또한 a,b 가 동일한 메모리 주소를 가리키기 때문 (== 은 메모리 주소를 비교)





- String 타입 예시
  - 원시 타입과 마찬가지로, 같은 값 호출 시 새로운 메모리 할당하지 않고 String Constant Pool 영역에 이미 할당되어 있는 값을 가리킴

```
public class Main {
  public static void main(String[] args) {
    String a = "str";
    String b = "str";

    System.out.println(System.identityHashCode(str)); //result:2008362258
    System.out.println(System.identityHashCode(str2));  //result:2008362258
  }
}
```

변수 선언 및 초기화하는 방법

- 변수 선언
- 변수 초기화
  - 클래스에 생성한 **멤버 변수 (필드, 전역변수)** 의 경우, 선언 하자마자 **변수 자료형의 기본값으로 초기화** 이루어짐

```
public class Test {
  int result1; // 0
  double result2; // 0.0

  void test() {
    int testResult1;
    double testResult2;
```

```
    }
}
```

- 메소드 내에서 지역적으로 선언한 **지역 변수**의 경우, **초기화 되어있지 않음**
- 초기화 방법 3가지
  1. **명시적 초기화** : 변수를 선언과 동시에 초기화 / 복잡한 초기화 작업 필요 시 '초기화 블록' 또는 생성자 사용해야 함
  2. **생성자**
  3. **초기화 블록**

```
class InitBlock {
    {...} // 인스턴스 초기화 블록
    static {...} // 클래스 초기화 블록
}
```

  - 인스턴스 초기화 블록
    - **생성자보다 인스턴스 초기화 블록이 먼저** 수행됨
    - 인스턴스가 생성될 때마다 각 인스턴스별로 초기화 이루어짐
  - 클래스 초기화 블록
    - 클래스가 메모리에 처음 로딩될 때 한번만 수행됨
    - JVM 이 클래스 로더로 로딩하는 시점에 단 한번 초기화 이루어짐

- 변수 초기화 순서

  [클래스 변수 초기화]

  1. Method area 에 변수 생성 후 기본값으로 초기화
  2. 명시적 초기화
  3. 클래스 초기화 블록

  [인스턴스 변수 초기화]

  1. Heap 영역에 변수 생성 후 기본값으로 초기화 (이 때, static 변수는 이미 초기화 되어 있는 상황)
  2. 명시적 초기화
  3. 인스턴스 초기화 블록

4. 생성자

변수의 스코프와 라이프타임

- 변수의 **스코프** : 변수를 사용할 수 있는 `유효 범위`
- 변수의 **라이프타임** : 변수가 `유효한 시간`

```
class Example {
  int x, y; // 인스턴스 변수 - 클래스 내부 & (모든 메소드 및 블록 외부) 에 선언된 변수
  static int z; // 클래스 변수 - 클래스 내부 & (모든 블록 외부) 에 선언되고 static 변수

  void add(int a, int b){ // 인스턴스 및 클래스 변수가 아닌 모든 변수 (a, b)
    x = a;
    y = b;
  }
}
```

  - 클래스 변수의 스코프는 `클래스 전체` / 클래스가 메모리에 로드된 후 프로그램이 끝날 때까지
  - 인스턴스 변수의 스코프는 `static 메소드, static 블록을 제외한 클래스 전체` / 객체가 메모리에 남아있을 때까지 유효
  - 지역 변수의 스코프는 `선언된 블록 내` / 변수 선언 후 선언된 블록을 떠날 때까지 유효

타입 변환, 캐스팅 그리고 타입 프로모션

- 타입 변환 : 데이터 타입을 변환하는 것
  - boolean 을 제외한 나머지 primitive 타입 간의 타입 변환이 가능
- `캐스팅` (명시적 형변환) : 큰 데이터 타입 → 작은 데이터 타입

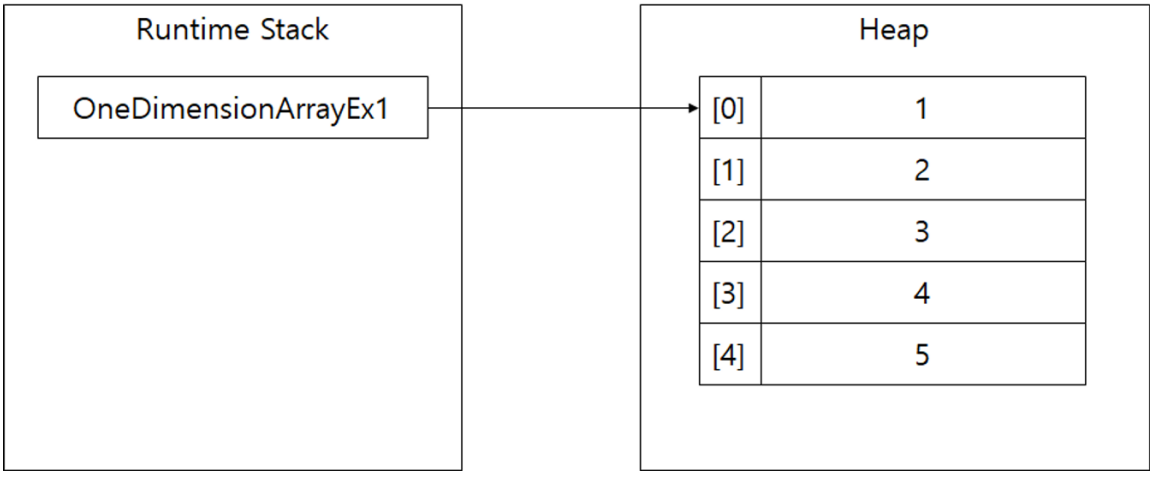
```
char a = 'a';
int num = 100;
a = (char) num;
```

- `타입 프로모션` (묵시적 [자동] 형변환) : 작은 데이터 타입 → 큰 데이터 타입
  - 데이터 손실이 발생하지 않거나, 손실이 최소화되는 방향으로 타입 변환 진행
  - 메모리 크기가 아니라 표현 가능한 `값의 범위` 가 큰 것이 큰 데이터 타입임

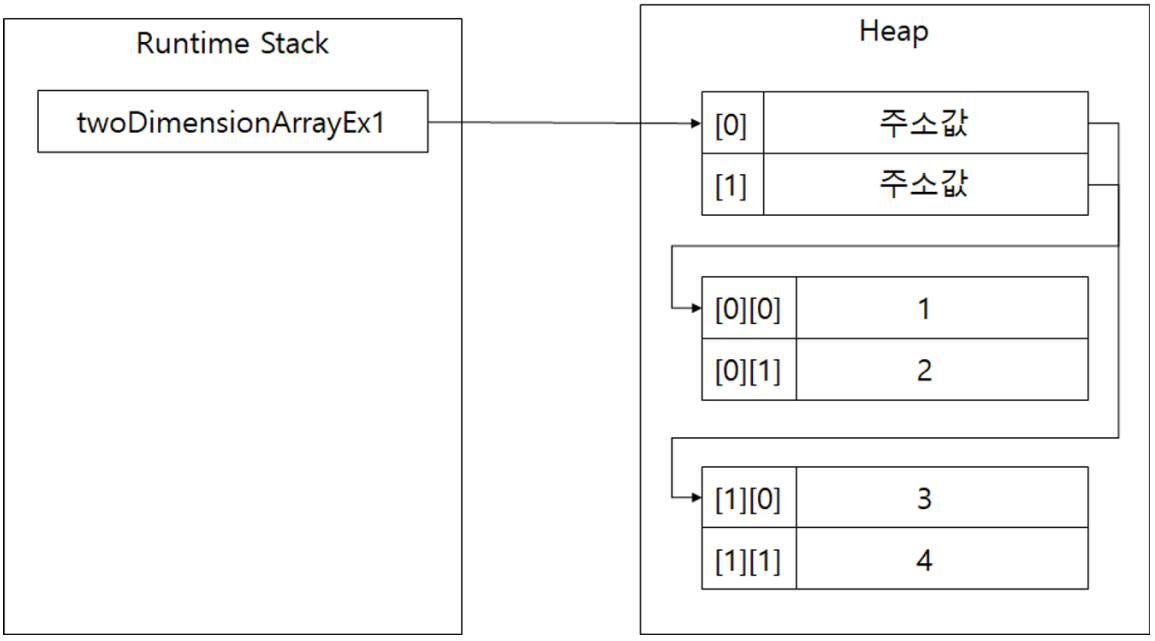
- long 타입 : 8byte, float 타입 : 4byte 인데, float a = (long 타입 변수) 로 자동 형변환이 가능한 이유

1차 및 2차 배열 선언하기

- 1차 배열 선언하기
  - int [] array = new int [10];



- 2차 배열 선언하기
  - int [][] array = new int [10][10];



타입 추론, var

- 타입 추론 : 정적 타이핑을 지원하는 언어에서, 타입이 정해지지 않은 변수에 대해 컴파일 시점에, `컴파일러`가 초기화된 `리터럴` 값으로 **변수 타입을 추론**하는 기능
  - 제네릭에서도 사용되고 있었음

```
HashMap<String, Integer> hashMap = new HashMap<>();
// 데이터 타입이 HashMap<String, Integer> 인 것을 바탕으로 추론하여
// new HashMap<>()에 따로 제네릭을 표시 하지 않아도 된다.
```

  - 정적 타이핑 언어란? 자료형을 컴파일 시에 결정하는 것
  - 타입 명시 안해도 됨
  - 코드량 줄이고, 가독성 높일 수 있음
  - Java 10 부터 var 키워드 생겼고, Java 11 부터는 이를 통한 람다 타입도 지원됨
  - var 는 컴파일러가 타입을 유추할 수 있도록 반드시 데이터 초기화 필요
- var
  - 지역변수에만 선언 가능

```
class Main {
    public static void main(String[] args) {
        var text = "Hello";
    }
}
```

  - 컴파일러가 타입을 유추할 수 있도록 반드시 데이터 초기화 필요 / null 로 초기화할 수는 없음
  - 키워드가 아니다. 즉, 어떤 타입도 아니고 클래스에서 사용하는 예약어가 아니기 때문에 var 라는 변수명을 만드는 것도 가능하다
  - var 는 런타임 오버헤드가 없다는 장점이 있다
    - 컴파일 시점에만 타입 추론이 일어나서, 바이트 코드에 변환된 타입으로 적힌다.
      var 로 선언된 변수는 중간에 타입 변경이 불가능 (컴파일에러)
      따라서, 타입추론 변수를 읽을 때마다 **타입을 알아내기 위한 연산을 하지 않아도 되기** 때문에, 런타임 오버헤드가 없다고 말한다
  - 람다 타입에서도 사용 가능

- 람다 표현식에서는 명시적인 타입을 지정해주어야 함
- **키워드 앞에만 사용할 수 있는 어노테이션을 사용할 수 있다**는 장점이 있음

```
Consumer<String> test = (var s) -> System.out.println("s = " + s);
Consumer<String> test = (@Nonnull var s) -> System.out.println("s = " + s);
```

- 배열 선언 시에는 타입을 명시해 주어야 한다

```
var arr = {1,2,3}; // 컴파일 에러
var arr = new Integer[] {1,2,3};
```

- var 와 <> 또는 제네릭을 함께 사용할 때 주의해야 한다

```
PriorityQueue<Item> itemQueue = new PriorityQueue<Item>();

PriorityQueue<Item> itemQueue = new PriorityQueue<>(); // 제네릭 표시 X
var itemQueue = new PriorityQueue<Item>(); // 제네릭 표시 O, var O
var itemQueue = new PriorityQueue<>(); // 제네릭 표시 X, var O
```

- 마지막 예시와 같이 <> 와 타입이 모두 제공되지 않으면, Object 와 같이 적용 가능한 타입으로 대체되어 PriorityQueue<Object> 로 추론됨

- 초기화 값이 숫자, 특히 정수 리터럴인 경우에는 주의가 필요함
  - 이미 타입이 명확한 경우에는, var 를 썼을 때 오히려 코드를 이해하기 어려울 수 있어 명시적 타입을 사용하는 것이 좋음

```
boolean ready = true      ->    var ready = true
char ch = '\ufffd'        ->    var ch = '\ufffd'
long sum = 0L             ->    var sum = 0L
String label = "wombat"   ->    var label = "wombat"

// 모두 int 로 추론됨
byte flags = 0            ->    var flags = 0
short mask = 0x7fff;      ->    var mask = 0x7fff
long base = 17            ->    var base = 17;
```