Reference Type (Memory)

`Calculator c1 = new Calculator(); ?????`

참조변수 c1은 생성된 instance인 Calculator()를 저장하는데
실제로는 instance인 Calculator()가 위치한 메모리 주소값을 저장한다.

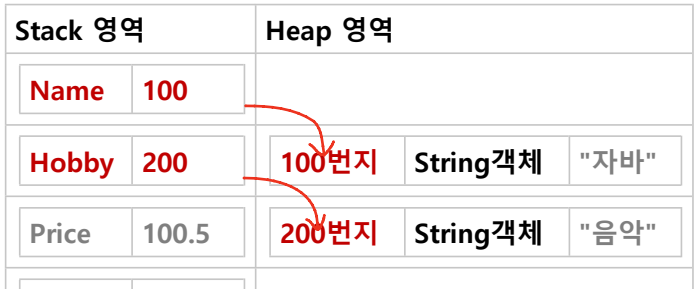
Data Type

- 기본타입(Primitive Type) : byte, char, short, int, long, float, double, boolean
  - 실제 값을 저장

- 참조타입(Reference Type) : 배열, 열거, 클래스, 인터페이스
  - 메모리의 주소번지 값 저장, 주소번지를 통해 객체를 참조(읽기) 한다.

Reference Type

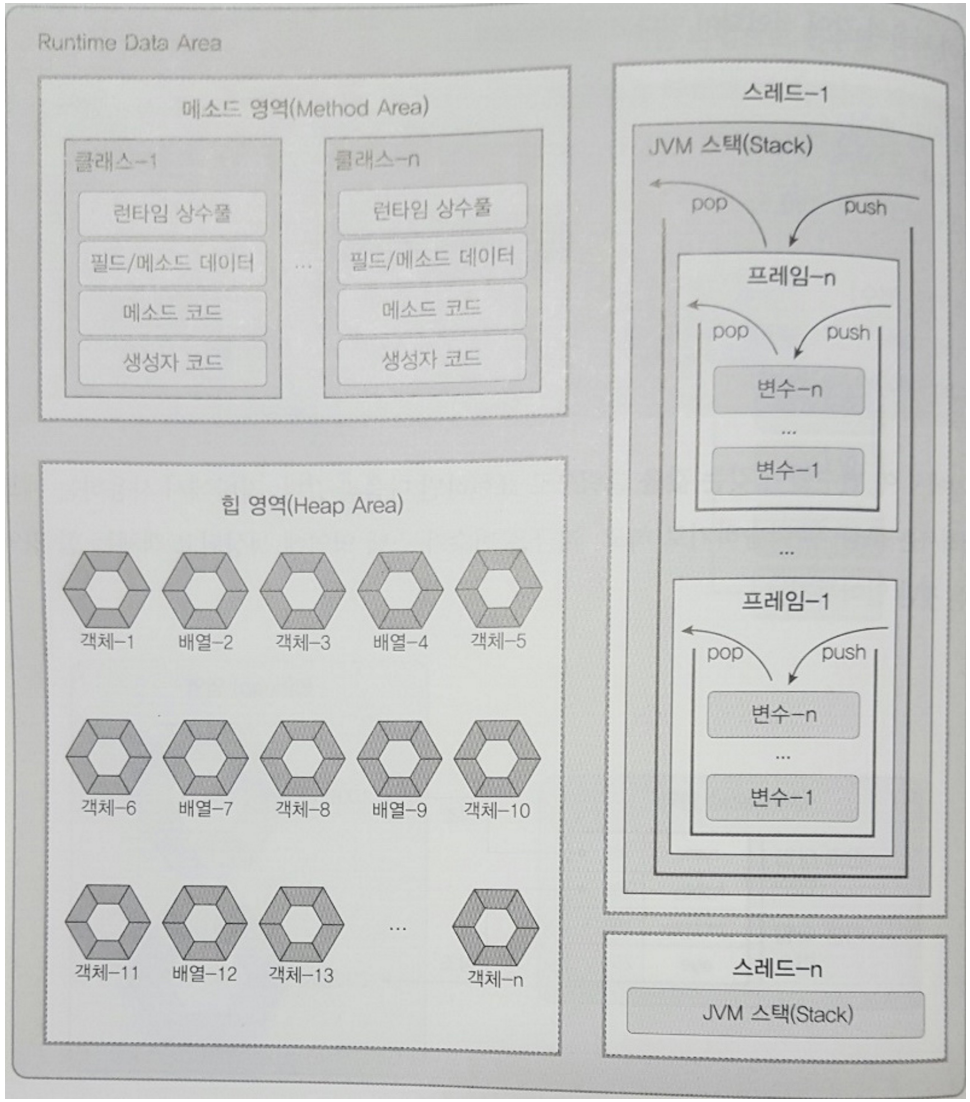
```
public static void main(String[] args) {
     // Primitive Type
     int age = 25;
     duble price = 100.5;

     // Reference Type
     String name = "자바";
     String hobby = "음악";
}
```



Age	25

- 변수는 메모리의 Stack 영역에 생성
- 객체는 메모리의 Heap 영역에 생성

Memory - Java Runtime Data Area



- **Method Area**

  코드에 사용되는 클래스(~.class)들을 클래스 로더로 읽어서 클래스 별로 코드를 분류해서 저장한다.
  JVM이 시작할 때 생성되고, 모든 스레드가 공유하는 영역

- **Heap Area**
  - **객체와 배열이 생성되는 영역**
  - 이들은 JVM 스택영역의 변수나 다른객체의 필드에서 참조
  - 참조하는 변수나 필드가 없으면 객체는 Garbage Collector에 의해 자동제거

- **JVM Stack Area**
  - 각 스레드마다 하나씩 존재, main 스레드가 기본.
  - 메소드 호출할 때 마다 Frame 추가(Push)
  - 메소드 종료시 Frame 제거(Pop)

Reference Type - ==, != Operator

- 참조변수가 동일한 객체를 참조하는지 알아볼 수 있다.(주소값 비교)

```
Calculator c1 = new Calculator();
Calculator c2 = c1;

System.out.println(c1 == c2); // true
System.out.println(c1 != c2); // false
```

Null, NullPointerException

- **참조변수는 Heap 영역의 객체를 참조하지 않겠다라는 뜻으로 null값을 가질 수 있다.**
- **참조변수는 null로 초기화 하지 않으면 단독으로 선언조차 불가능하다.**
- 참조변수는 null값으로 초기화 가능, 추후에 인스턴스나 참조변수등을 통해 주소값을 받을 수 있다.

- **NullPointerException**

  null로 초기화한 참조변수는 참조할 객체가 없기 때문에 사용할 수 없다.
  그래서 NullPointerException (Exception : 오류,예외) 발생

객체 생성시의 참조변수

Calculator c1 = new Calculator();

1. Calculator()라는 instance를 Heap영역에 생성한다.
2. 하지만 생성된 instance의 메모리주소를 모르면 사용할 수 없다.
3. 그러므로 생성된 instance의 주소를 저장할 수 있는 참조변수 c1을 Stack영역에 선언
4. 그런데 참조변수 c1은 Calculator()라는 객체를 참조해야 하므로 c1의 자료형은 Calculator가 된다.

---

new 연산자

객체생성만이 아니라 생성된 객체의 메모리주소 값을 반환해준다.

참조변수의 활용

```
class Calculator {
     int left, right;

     public void setOprands(int left, int right) {
          this.left = left;
          this.right = right;
     }

     public void sum() {
          System.out.println(this.left + this.right);
     }
}

public class CalculatorDemo4 {
     public static void main(String[] args) {
          Calculator c1 = new Calculator();
          c1.setOprands(1, 5);
          c1.sum();
```

Calculator Instance	100번지

```
          ContolInstanceC1(c1, 10, 20);
     }
```

```
        ContolInstanceC1(c1, 10, 20);
    }

    public static void ContolInstanceC1(Calculator cal, int left, int right)
        cal.setOprands(left, right);
        System.out.println((cal.left + cal.right) / 2);
    }
}
```

i. New를 통해 Calculator() 객체를 만들고 객체의 주소값을 참조변수 c1에 참조시킨다.

ii. Method ControlInstanceC1은 c1이 참조하고 있는 Instance의 멤버들을 사용하고 싶다.

iii. 그래서 ControlInstanceC1을 정의할 때, 참조변수를 매개변수로 만들고, c1을 인자로 받는다.

iv. 매개변수 cal은 c1이 가리키고 있는 Calculator()의 주소값을 참조해야 하기 때문에 **자료형이 Calculator**가 되어야 한다.

v. ContolInstanceC1(**c1**, 10, 20)을 통해 **참조변수 c1을 인자로 전달하면 매개변수 cal이 c1이 가리키는 주소값을 받는다.**

vi. **결과적으로 c1과 cal은 동일한 객체를 가리키게 된다.**

vii. ContolInstanceC1 메소드 안에서는 **cal. 접근을 통해 c1이 가리키고 있는 객체에 접근**하면된다.

의존성

- **한 객체 안에서 다른 객체가 생성(new) 되면 두 객체 사이에는 의존성이 생긴다!**
- **다만 메소드에 매개변수로 전달되는 인스턴스같은 경우에는 그 인스턴스를 참조한다고 한다!**
  (나의 영역이 아닌 다른영역에 생긴 인스턴스를 가져올 때에는 참조한다고 한다!)
