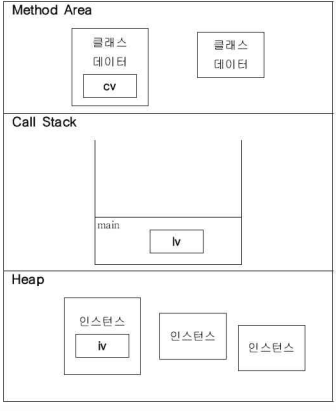
제8장
객체와 클래스 - 2

9. JVM의 메모리 구조

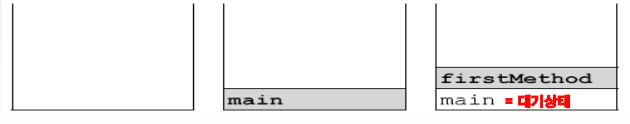


- 메서드영역(Method Area), 클래스 영역, static 영역
  - 클래스 정보와 클래스 변수가 저장되는 곳
  - static이 붙은 변수나 메서드가 저장됨.
- 호출스택(Call Stack) – 후입선출 개념
  - 메서드의 작업공간. 메서드가 호출되면 메서드 수행에 필요한 메모리공간을 할당받고 메서드가 종료되면 사용하던 메모리를 반환한다.
- 힙(Heap)
  - 인스턴스가 생성되는 공간. new연산자에 의해서 생성되는 배열과 객체는 모두 여기에 생성된다.

10. Call Stack(호출 스택)

▣ 호출 스택의 특징

- 프로그램 시작점인 main()가 시작되면서, 프로그램은 수많은 인스턴스도 생성하고 메서드도 수 없이 호출된다. 이 때, **생성자 호출이나 메서드 호출 시에 필요한 메모리 공간을 스택 공간에 할당**한다.
- 물론, 메서드의 수행이 끝이 나면 할당된 메모리 공간이 반환이 된다.
- **호출 스택에서 맨 위에 있는 메서드가 현재 실행 중**이며, **아래에 있는 메서드가 바로 위에 메서드를 호출한 메서드이며 대기 상태**로 있다.

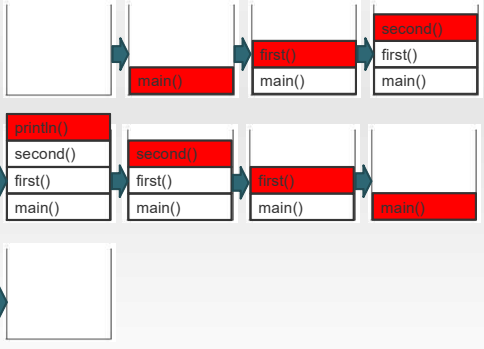


11. Call Stack(호출 스택)예제와 메모리 상태

```
public static void main(String[] args) {
    //static은 static만 부를수 있다.
    CallStackExample.first();
}

public static void first() {
    second();
}

public static void second() {
    System.out.println("second()");
}
```



12. 기본형 매개변수와 참조형 매개변수

▣ 기본형 매개변수 – read only

- 흔히, C언어에서 call by value라고 불리며, 메서드 호출 시에 매개변수로 넘겨주는 값은 메서드의 지역변수로 복사가 이루어지는 형태
  (수정을 해도 호출한 메서드의 값에는 전혀 영향을 미치지 않는다.)

▣ 참조형 매개변수 – read & write

- C언어에서 call by reference라고 불리며, 메서드 호출 시에 매개변수로 넘겨주는 값은 주소값을 넘겨주는 형태
  (호출된 메서드에서 수정을 하면, 호출한 메서드의 값에도 직접적으로 영향을 미친다.)

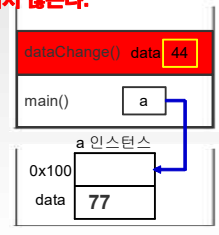
13. 기본형 매개변수를 넘길 때

```
class A {
    int data;
}
public class PrimitiveParam {

    public static void main(String[] args) {

        A a = new A();
        a.data = 77;
        System.out.println("main()메서드 내 data 값 : " + a.data);
        //아래 메서드 호출은 실제 값을 넘기고 있다.(Call by value)
        //즉, dataChange(int data)의 매개변수로 복사가 되어 지고 있다.
        PrimitiveParam.dataChange(a.data);
        System.out.println("dataChange()호출 후");
        System.out.println("main()메서드 내 data 값 : " + a.data);
    }
    //static은 static만 호출할 수 있다.
    public static void dataChange(int data) {
        data = 44;
        System.out.println("dataChange()메서드 내 data값 : " + data);
    }
}
```

call by value는 값에 의한 복사의 개념으로, dataChange(a.data)를 호출 시, 매개변수가 호출한 메서드의 매개변수로 복사가 되어진다. 하여, 메인메서드가 스택에 먼저 쌓이고, data Change(Int data)가 위에 쌓이는 형태가 되는 것이다. 결론은 값 복사는 호출한 곳에는 전혀 영향을 끼치지 않는다.



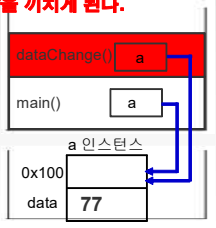
14. 참조형 매개변수를 넘길 때

```
class A {
    int data;
}
public class PrimitiveParam {

    public static void main(String[] args) {

        A a = new A();
        a.data = 77;
        System.out.println("main()메서드 내 data 값 : " + a.data);
        //아래 메서드 호출은 인스턴스의 주소를 넘기고 있다.(Call by reference)
        //즉, dataChange(a)의 매개변수로 주소값이 넘어가고 있다.
        PrimitiveParam.dataChange(a);
        System.out.println("dataChange()호출 후");
        System.out.println("main()메서드 내 data 값 : " + a.data);
    }
    //static은 static만 호출할 수 있다.
    public static void dataChange(A a) {
        a.data = 44;
        System.out.println("dataChange()메서드 내 data값 : " + a.data);
    }
}
```

call by reference는 주소에 의한 호출의 개념으로, dataChange(a)를 호출 시, 매개변수가 호출한 메서드의 매개변수로 주소가 넘어가게 되어 주소공유가 일어난다. 하여, 메인메서드가 스택에 먼저 쌓이고, dataChange(A a)가 위에 쌓이는 형태가 되는 것이다.
결론은 주소에 의한 호출은 호출한 곳에 직접적으로 영향을 끼치게 된다.



15. 재귀 호출(recursive call)

▣ 재귀 호출

- 메서드 내에서 자기자신을 반복적으로 호출하는 것을 말한다.
- 재귀 호출은 얼마든지 반복문으로 바꿀 수 있다.
- 또한, 반복문보다 성능이 떨어진다.(스택에 계속 쌓인다.)
- 그래도, 코드가 간결하고 가독성이 좋아 많이 사용한다.

▣ 재귀 호출의 예

- 팩토리얼, 제곱, 트리, 폴더목록 등

*팩토리얼(factorial)

4! = 4 * 3 * 2 * 1

f(n) = n * f(n-1) 단, f(1) = 1

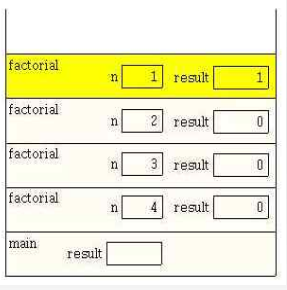
```
long factorial(int n) {
    long result = 0;
    if(n==1) {
        result = 1;
    } else {
        result = n * factorial(n-1);
    }
    return result;
}
```

16. 재귀 호출의 예(팩토리얼)

```
public class FactorialEx {

    public static void main(String[] args) {

        long result = factorial(4L);
        System.out.println("4!(팩토리얼)값 : " + result);
    }
    //자기 자신을 호출하는 재귀
    //for문으로 해도 되나, 코드를 보는것과 코드중복을 제거함으로써 좋다.
    public static long factorial(long n) {
        long result = 0L;
        //비로소 1일때, 재귀호출이 더 이상 안이루어진다. f(1) = 1 이니깐.
        if (n == 1) {
            result = 1;
        }
        else {
            System.out.println("result값 : " + result + " n값 : " + n);
            result = n * factorial(n-1); // 메서드 자신을 호출한다.
        }
        return result;
    }
}
```




```
Console
<terminated> FactorialEx [Java Applicatio
result값 : 0 n값 : 4
result값 : 0 n값 : 3
result값 : 0 n값 : 2
4!(팩토리얼)값 : 24
```

감사합니다.