(디지털컨버전스) 스마트 콘텐츠와 웹 융합 응용 SW개발자 양성과정

-14일차 학습 및 질문 노트-

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – Kyeonghwan Lee(이경환)
airtrade7@naver.com

▣ 객체 전달과 값 전달의 차이점

1. 기본 타입의 값 전달(call by value)

- 값이 복사되어 전달 된다.
- 메소드의 매개변수가 변경되어도 호출한 실인자 값은 변경되지 않는다.

2. 객체 혹은 배열 전달(call by reference)

- 객체나 배열의 레퍼런스만 전달한다.
- 객체 혹은 배열이 통째로 복사되어 전달되는 것이 아니다.
- 메소드의 매개변수와 호출한 실인자 객체나 배열을 공유한다.
- 배열의 이름은 배열의 대표로 객체가 전달된다(원본 형태로 전달)
- 배열 전달 시 인덱스 지정하여 전달하면 값이 전달된다.(나머지는 전부 값)

▣객체 전달과 값 전달의 차이점

```java
class CloneMemory {
    int[] arr;
    int num;

    public CloneMemory () {
        arr = new int[3];
        num = 3;

        for (int i = 0; i < 3; i++) {
            arr[i] = (int)(Math.random() * 6 + 1);
        }
    }
    public void reRandArr () {
        for (int i = 0; i < 3; i++) {
            arr[i] = (int)(Math.random() * 6 + 1);
        }
    }
    public void reNum () {
        num = 7;
    }

    public int[] getCloneArr () {
        return arr;
    }

    public int getCloneVariable () {
        return num;
    }

    public String toString () {
        return "arr[0] = " + arr[0] +
                ", arr[1] = " + arr[1] +
                ", arr[2] = " + arr[2];
```

```java
public class MemoryCloneTest {
    public static void main(String[] args) {
        CloneMemory cm = new CloneMemory();

        System.out.println(cm);

        int[] save = cm.getCloneArr();

        System.out.printf("save[0] = %d, save[1] = %d, save[2] = %d\n",
                save[0], save[1], save[2]);

        cm.reRandArr();

        System.out.println("객체에 접근해 출력");
        System.out.println(cm);

        System.out.println("사전 저장 정보 출력");
        System.out.printf("save[0] = %d, save[1] = %d, save[2] = %d\n",
                save[0], save[1], save[2]);

        // 결론: 자바에서 객체에 대한 접근은 모두 메모리를 제어하는 방식이 된다.

        int num = cm.getCloneVariable();

        System.out.println("객체내 변수값 획득: " + num);

        cm.reNum();

        System.out.println("변경 후 사전 획득한 정보 재출력: " + num);

        // 결론: 앞서서도 확인했지만 값에 대해서는 복제가 이루어짐을 확인할 수 있다.
        System.out.println("변경 정보 파악: " + cm.getCloneVariable());
    }
```

```
arr[0] = 4, arr[1] = 3, arr[2] = 4
save[0] = 4, save[1] = 3, save[2] = 4
객체에 접근해 출력
arr[0] = 6, arr[1] = 3, arr[2] = 4
사전 저장 정보 출력
save[0] = 6, save[1] = 3, save[2] = 4
객체내 변수값 획득: 3
변경 후 사전 획득한 정보 재출력: 3
변경 정보 파악: 7

Process finished with exit code 0
```

결론: 자바에서 객체에 대한 접근은 모두 메모리를 제어하는 방식이 된다. 게터고 나발이고 다 떠나서 객체는 메모리 자체를 전달하며 값은 메모리가 아닌 값을 전달한다.

▣ArrayList

```java
public class ArrayListTest {
    public static void main(String[] args) {
        // 용도: 일종의 배열임
        //      배열의 사이즈를 지정하고 사용해야 하지만
        //      이 녀석은 넣고 싶은대로 아무때나 막 넣어도 된다.
        //      (참고로 이 녀석도 Heap을 이용한 동적할당을 수행함)

        // 사용법: ArrayList<내부에저장할데이터타입> 변수명 = new ArrayList<내부에저장할데이터타입>();

        // 일반 배열과의 차이점은 ?
        // 배열은 메모리가 연속적으로 배치된다.
        // 이 녀석은 불연속 배치다.
        // 어떻게 ?
        // | 데이터1 | 다음링크 | ---> | 데이터2 | 다음링크 | ---> | 데이터3 | 다음링크 | ---> ....
        // 배열은 ?
        // | 데이터1 | 데이터2 | 데이터3 | 데이터4 | 데이터5 | 데이터6 | 데이터7 | ...
        ArrayList<String> lists = new ArrayList<String>();

        lists.add("빵");
        lists.add("버터");
        lists.add("우유");
        lists.add("계란");
        lists.add("쥬스");
        lists.add("베이컨");
        lists.add("파스타");
        lists.add("비프샐러드");
        lists.add("피자");

        for (String list : lists) {
            System.out.println("현재 항목은 = " + list);
        }

        // 내가 몇 개를 쓸지 알고 있는 상황: 배열
        // 몇 개가 들어올지 모르겠네 ? ArrayList
        // ex) 회원 가입 몇명 ???
    }
}
```

```
현재 항목은 = 빵
현재 항목은 = 버터
현재 항목은 = 우유
현재 항목은 = 계란
현재 항목은 = 쥬스
현재 항목은 = 베이컨
현재 항목은 = 파스타
현재 항목은 = 비프샐러드
현재 항목은 = 피자

Process finished with exit code 0
```

ArrayList

- 크기가 가변적이다.(동적할당)
- 데이터 추가는 add, 삭제는 remove를 사용한다.
- 데이터 추가 삭제 시 메모리를 재할당 하기 때문에 속도가 배열보다 느리다.
- 불 연속적으로 배치된다.

사용법

- ArrayList<내부에 저장할 데이터 타입>변수명= new ArrayList<내부에 저장할 데이터 타입>();

Array

- 배열의 크기는 한번 정하면,크기를 변경할 수 없다.
- 배열 초기화 시 메모리에 할당 되어 ArrayList 보다 속도가 빠르다.
- 연속적으로 배치된다.

▣ArrayList 활용

```java
class Shop {
    ArrayList<String> lists;
    Scanner scan;

    public Shop () {
        lists = new ArrayList<String>();
        scan = new Scanner(System.in);
    }

    public void deliveryCome () {
        System.out.print("필요한 물품을 말씀하세요: ");
        lists.add(scan.nextLine());
    }

    public void cancelOrder () {
        System.out.print("취소할 물품을 말씀하세요: ");
        lists.remove(scan.nextLine());
    }

    // toString 으로 자동 완성 가능
    // 객체 정보 출력에 사용합니다.
    // 아직 인터페이스 배우지 않았으므로 설명은 향후 진행
    @Override
    public String toString() {
        return "Shop{" +
                "lists=" + lists +
                '}';
    }
}
```

```java
public class ShopTest {
    public static void main(String[] args) {
        Shop s = new Shop();

        for(int i = 0; i < 3; i++) {
            s.deliveryCome();
        }

        s.cancelOrder();

        // 아래와 같이 객체를 전달하면 toString이 호출됨
        System.out.println(s);
    }
}

// ArrayList는 Queue 혹은 Stack 역할을 할 수 있는데 기본이 Queue(큐) 역할
// 맨 처음 넣은 정보가 가장 앞에 배치되고
// 두 번째 넣은 정보가 두 번째에 배치되고 ...
```

```
필요한 물품을 말씀하세요: 김
필요한 물품을 말씀하세요: 밥
필요한 물품을 말씀하세요: 햄
취소할 물품을 말씀하세요: 단무지
Shop{lists=[김, 밥, 햄]}

Process finished with exit code 0
```

remove: 입력하지 않은 값을 삭제할 경우 에러가 나지않고 스킵하고 출력한다.