Pass By Value & Pass By Reference

Programlama dillerinde bir method cagrildiginda original data'nin nasil kullanilacagi iki sekilde belirlenebilir.

Pass By Value : Primitive bir data'yi parametre olarak bir method'a gonderdigimizde Java original variable yerine ayni degere sahip kopya bir variable olusturur ve method icerisinde kopya variable uzerinden islem yapilir.

Pass By Reference : de ise method cagrildiginda, data'nin original degeri ile islem yapilir. Eger method icerisinde data degistirilirse original deger de degismis olur.

Bu iki alternative gozonune alindiginda Java Pass By Value ozelligini kullanmaktadir.

https://www.youtube.com/watch?v=wWh4U4Np05w

Pass By Value & Pass By Reference

Soru : Verilen bir fiyat icin %10 indirim yapan bir method olusturun.

- Method'da indirim uygulanan fiyati yazdirin

- Method Call sonrasi original fiyati yazdirin ve method'da yapilan degisikligin orginal degeri degistirip degistirmedigini kontrol edin.

```
public class StaticBlock {

    public static void main(String[] args) {

        int fiyat = 100;

        System.out.println("Method'da hesaplanan fiyat : " + indirim(fiyat));
        System.out.println("Method call sonrasi fiyat : " + fiyat);
    }

    private static int indirim(int fiyat) {
        fiyat *= 0.90;
        return fiyat;
    }
```

Pass By Value & Pass By Reference

Soru2 : Verilen bir fiyat icin %10 , %20, %25 indirim yapan uc method olusturun.

- Method'da indirim uygulayip fiyati main method'da yazdirin

- Method'lari arka arkaya cagirip dogru calistiklarini kontrol edin

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

    int fiyat = 100;

    System.out.println("Method10'da hesaplanan fiyat : " + indirim10(fiyat));
    System.out.println("Method20'da hesaplanan fiyat : " + indirim20(fiyat));
    System.out.println("Method25'da hesaplanan fiyat : " + indirim25(fiyat));
    System.out.println("Method call sonrasi fiyat : " + fiyat);
}

public static int indirim10(int fiyat) {
    fiyat *= 0.90;
    return fiyat;
}

public static int indirim20(int fiyat) {
    fiyat *= 0.80;
    return fiyat;
}
public static int indirim25(int fiyat) {
    fiyat *= 0.75;
    return fiyat;
}
```

Pass By Value & Pass By Reference

Soru3 : Bir list olusturalim. Eleman olarak 10,11,12 ekleyelim. Iki method olusturup list elemanlarini artirmayi deneyelim

- 1. Method'da elemanlari for each loop kullanarak artirin
- 2. Method'da elemanlari set method'u kullanarak artirin
- Method'lari arka arkaya cagirip artislarin kalici olup olmadiklarini kontrol edelim.

NOT : Java'da list veya array'in elemanlarini update ettiginizde elemanlar kalici olarak degisir.

List veya array'in kendisi degismez ama elemanlari kalici olarak degisir

```
public static void main(String[] args) {
 List<Integer> list =new ArrayList<Integer>();
 list.add(10);
 list.add(11);
 list.add(12);

 degistir(list);
 System.out.println(list);
 degistir2(list);
 System.out.println(list);
}
public static void degistir(List<Integer> list) {
    for (Integer each : list) {
        each=each+3;
        System.out.print(each + " ");
    }
    System.out.println();
    System.out.println(list);
}
public static void degistir2(List<Integer> list) {

    for (int i = 0; i < list.size(); i++) {
        list.set(i, list.get(i)+3);
        System.out.print(list.get(i)+" ");
    }
    System.out.println();
    System.out.println(list);
}
```

Pass By Value & Pass By Reference

Soru4 : Bir list ve bir array olusturalim ve eleman olarak 10,11,12 ekleyelim. Iki method olusturup list ve array'i degistirmeyi deneyelim

- 1. Method'da array'e yeni bir array assign edelim ve yeni halini yazdiralim
- 2. Method'da list'e yeni bir list assign edelim ve yeni halini yazdiralim
- Method call'dan sonra main method'da yeniden yazdirip degisip degismediklerini kontrol edelim.

```
public static void main(String[] args) {
    int num[]= {10,11,12};
    degistirArray(num);
    System.out.println("method'dan sonra main method'un icinde array: " + Arrays.toString(num));//[10, 11, 12]
    List<Integer> list =new ArrayList<>();
    list.add(10);
    list.add(11);
    list.add(12);
    degistirList(list);
    System.out.println("method'dan sonra main method'un icinde list : "+ list); //[10, 11, 12]
}
public static void degistirList(List<Integer> list) {
    list=new ArrayList<>();
    list.add(40);
    System.out.println("list methodda : " + list); // [40]

}
public static void degistirArray(int num[]) {

    num=new int[5];

    System.out.println("array methodda : " + Arrays.toString(num)); // [0, 0, 0, 0, 0]
}
```

Mutable & Immutable Classes

Immutable (değişmez) class'lar, objeleri bir kez oluşturulduktan sonra değiştiremedigimiz class'lardir.

Mutable (değişebilir) class'lar ise tam tersi olarak, olusturdugumuz objeleri değiştirebildigimiz class'lardir.

NOT : Immutable class'dan olusturulan objeler de immutable olurlar.
Bu tur bir object'i oluşturabiliriz, fakat onları değiştiremeyiz.
Immutable bir objeyi değiştirmek istersek, Java o objeyi klonlar ve yapilan degisiklikleri klonlanmis yeni obje uzerinde gerceklestirir.

Peki böyle bir duruma niçin ihtiyacımız olur diye bir soru sorarsak,

cevabı thread safe (guvenli es zamanli calisma) konusudur.

Immutable nesnelerin değerleri değişmeyeceği için üzerinde kaç tane thread çalışırsa çalışsın hep aynı değerler üzerinden işlem yapılacaktır.

Mutable & Immutable Classes

Java'da yaygın olanlarından örnek verecek olursak

String ve tüm Wrapper (Integer, Long, Double, Byte….) class'lar immutable sınıflardır.

Date, StringBuilder,StringBuffer, Arrays ve ArrayList mutable Class'lardandır.

Asagidaki ornekte String methodu kullanildiginda str'in degeri degismezken method kullanilan list'in degeri degismektedir.

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

    String str= "Mehmet";
    str.toUpperCase();
    System.out.println(str); // Mehmet


    List <String> list= new ArrayList<>();
    list.add("Mehmet");
    list.add("Ayse");
    System.out.println(list); // [Mehmet, Ayse]

}
```

Mutable & Immutable Classes

String'de yaptigimiz degisikligin kalici olmasi icin assignment yapabiliriz.

Bu durumda da String immutable oldugu icin Java atadigimiz yeni deger icin klon bir variable olusturur ve yeni degeri yeni klonlanmis String'e assign eder, referansin isaret ettigi object de yeni klon object olur.

Ornek : Yandaki ornekte str String'i olusturulduktan sonra loop icerisinde 100 tane klon str olusturulur ve yazdirilir, loop sonuna gelip alt satira indigimizde Java son olusturulan klon'u refere eder.

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

    String str = "";
    for (int i = 0; i < 100; i++) {
        str = i +". deger";
        System.out.println(str);
    }

    System.out.println("son deger : " + str);
}
```

Mutable & Immutable Classes

Java'da bir String iki sekilde olusturulabilir.

1- String str = "Mehmet"; seklinde (her zaman yaptigimiz sekilde) oluşturulduğunda,

Java Virtual Machine öncelikle o değişkeni String Havuzunda arar ve bir karşılaşma bulursa, aynı referansı verir yeni String'e.

Bu yüzden asagidaki soruda str3==str4 true dondurur,

2- String str = new String("Mehmet"); seklinde yeni bir obje olarak oluşturulduğunda,

Java once yeni bir Object olusturur ve sonra istenen degeri assign eder.

Bu yuzden yandaki ornekte

str1==str2 false dondurur

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


    String str1=new String("mehmet");
    String str2=new String("mehmet");

    System.out.println("new  == " + (str1 == str2));
    System.out.println("new equals  " + (str1.equals(str2)));


    String str3="mehmet";
    String str4="mehmet";

    System.out.println("klasik  == " + (str3 == str4));
    System.out.println("klasik equals  " + (str3.equals(str4)));

}
```

Mutable & Immutable Classes

What is the result of the following code? (Choose all that apply)

```
13: String a = "";
14: a += 2;
15: a += 'c';
16: a += false;
17: if ( a == "2cfalse") System.out.println("==");
18: if ( a.equals("2cfalse")) System.out.println("equals");
```

A. Compile error on line 14.

B. Compile error on line 15.

C. Compile error on line 16.

D. Compile error on another line.

E. ==

F. equals

G. An exception is thrown.

Mutable & Immutable Classes

What is the result of the following statements?

```
6:  List<String> list = new ArrayList<String>();
7:  list.add("one");
8:  list.add("two");
9:  list.add(7);
10: for(String s : list)  System.out.print(s);
```

A. onetwo

B. onetwo7

C. onetwo followed by an exception

D. Compiler error on line 9.

E. Compiler error on line 10.

Mutable & Immutable Classes

What is the result of the following statements?

```
3: ArrayList<Integer> values = new ArrayList<>();
4: values.add(4);
5: values.add(5);
6: values.set(1, 6);
7: values.remove(0);
8: for (Integer v : values) System.out.print(v);
```

A. 4

B. 5

C. 6

D. 46

E. 45

F. An exception is thrown.

G. The code does not compile.