HOMEWORK 1

**1. Pass by value, pass by reference kavramları nedir ? Java’ ile ilişkili olarak açıklayınız.**

**Pass by value** metotlara değişken geçirme yöntemidir. Bu yöntemde; metotlara geçilen değişkenin kendisi değil kopyası gönderilir. Bütün işlem değişkenin kopyası üzerinden döner. Orijinal değişkenin adres alanındaki değerine dokunulmaz. Bu şekilde metot içindeki değişkenin değeri sadece metot içinde geçerlidir. Metot dışına çıkıldığı an değişkenin orijinal adresindeki değer geçerlidir. Java dilinde de pass by value yaklaşımı uygulanır. Java dili üzerinde örnek verelim.

public class Test {

protected static void changeValue(int n) {

n = n \* 10;

System.out.println("addValue method -->" + " number:" + n);

}

public static void main(String[] args) {

Test T = new Test();

int number = 10;

System.out.println("Before method call -->"+ " number:" + number);

changeValue(number);

System.out.println("After method call --> " + "number:" + number);

}

}

----------------------------------------------------------------------------------------------------------------------------------------Output:

Before method call --> number:10

addvalue method --> number:100

After method call --> number:10

Örnekte de görüldüğü üzere number değişkenine ilk olarak 10 değeri atanmıştır. changeValue metodu ile değişkenin değeri 100 olacak şekilde bir işlem yapılmıştır. Metot içerisinde değişkenin değerini yazdırdığımızda 100 olarak görürüz. Değişkenin değerini metot dışında tekrardan yazdırmak istersek 10 değeri ile karşılaşırız. Çünkü metot içerisinde değişkenin orjjinal adres alanındaki değer üzerinde değil kopyası üzerinde değişiklik yapılmıştır. Bu pass by value yaklaşımıdır. Özet olarak metot içine geçirilen değişken metot içinde kalır. Dışarının haberi olmaz.

**Pass by reference**

Türkçesi referans ile geçirmedir. Bu yaklaşımda metota değişkenin kopyası değil direk adres alanı geçirilir. Metot içinde, değişkenin adres alanındaki değer üzerinde değişiklik yapabildiğimiz için değişken üzerindeki herhangi bir değişiklik o değişkeni kullanan bütün metotları, yapıları etkiler. Özetlersek metot içinde değişken üzerindeki değişiklik metot dışına da yansır. C, C++ dilleri pass by reference yaklaşımını uygular.

**2. Immutability nedir, neden önemlidir ? Bir Java sınıfı nasıl immutable yapılır ?**

**Immutability nedir?**

Immutability değişmezlik demektir. Java’da immutable sınıflar, nesneleri oluşturulduktan sonra üzerinde oynama yapılamayan, içeriği değiştirilemeyen sınıflardır. Nesneler oluşturulurken ilk değer atanır ve bir daha o değere dokunulmaz. Bunun tersi de mutable (değişebilir) sınıflardır. Burda nesneler immutable yapılmadığı sürece üzerlerinde istenilen değişiklikler yapılabilinir. Immutability bazı ihtiyaçlardan dolayı doğmuştur. Örnek vermek gerekirse çok iş parçacıklı ortamlarda (multithreaded) immutable şekilde oluşturulan nesnelerin değerleri ne kadar thread çalışırsa çalışsın değişmeyeceği için o nesneyi güvenilir yapar. Başka bir threadin nesneyi değiştiren bir iş parçacığının neden olduğu hataları önler. Immutability; çok iş parçacıklı ortamlarda, streamlerde nesneyi değişmez yapma, güvenilir yapma özelliğinden dolayı önemlidir ve tercih sebebidir. Immutable nesne veya sınıf kullanımın da bazı dezavantajları vardır. Eğer o nesneyi değiştirmek istersek kopyasını oluşturmalı ve onun üzerinden işlem yapmalıyız. Bu da bellek, zaman konusunda ek maliyetlere neden olur. Java’da immutable classlara örnek verecek olursak String ve tüm ilkel sarmalayıcı sınıflar (wrapper class)Byte, Short, Integer, Long, Float, Double, Char, Boolean, Byte**, BigDecimal, BigInteger** immutable sınıflardır.

**Bir Java sınıfı nasıl immutable yapılır ?**

* Classın başına final anahtarı getirilir.Bu şekilde extend olayı önlenmiş olur.
* Classın tüm değişkenleri private yapılır. Doğrudan erişime bu şekilde izin verilmez.
* Tüm değiştirilebilir alanlara final anahtarı eklenir.
* Final işaretlenen alanların ilk değerleri constructor ile atanır.
* Değişkenlere setter metotlar oluşturulmaz, varsa kaldırılır. Tekrardan değer ataması önlenir

public final class ImmutableClassExample {

private final int age;

private final String name;

     private final String surname;

    public ImmutableClassExample (final int age, final String name, final String surname) {

         this.age = age;

         this.name = name;

         this.surname = surname;

    }

    public int getAge() {

        return age;

    }

    public String getName() {

        return name;

    }

    Public String getSurname() {

        return surname;

}}

**3. Framework ve library arasındaki fark nedir ?**

Framework ve Library anlam olarak birbirine benzeyen teknik kavramlardır. Benzerliklerini ve amaçlarını da anlatırsak daha anlaşılır olacaktır. İkisi de başkaları tarafında oluşturulmuş daha basit ve daha az kod yazmamızı sağlayan hazır kod yapılarıdır. Aynı amaca hizmet eder. Daha basit ve daha az kod yazmak, sonuç olarak ikisi de işleri kolaylaştırır. Farklılıkları teknik kısımda ortaya çıkar. Library kullanırken, library bize bazı özellikleri verir ve o özellikleri kodda nerede ve ne şekilde kullanacağımıza karışmaz. Library’nin vermiş olduğu özellikleri kodun istediğimiz yerinde kullanabiliriz. Library kullanırken programcı kodu kontrol eder. Framework de ise durum tersidir. Framework kodu kontrol eder.

Framework kodu nerede ve ne zaman yazacağımız söyler. Genellikle kod yazarken frameworkün belirlediği kurallar dizinin dışına çıkamayız. Eğer frameworkün istemediği şekilde kod yazarsak bu sefer kod çalışmaz. İkinci fark ise şudur. Library görece daha basit ve bir konuya yönelik işleri gerçekleştirmek için kullanılır. Örnek vermek gerekirse sadece görüntü işleme için OpenCV, email işlemleri için JODD email gibi kütüphaneler oluşturulmuştur. Framework ise daha karmaşık işler için kullanılır. Mesela bir projede hem veri tabanı bağlantısı hem loglama gibi işler yapmak istiyorsak bunları içeren frameworkler kullanmamız gerekir. Frameworkleri kütüphaneler topluluğu olarak düşünebiliriz. Örnek; oyun için Unity, uygulama için Spring frameworkü geliştirilmiştir.

**4. Java’da Garbage Collector’ un görevi nedir ?**

Belleği otomatik olarak yönetmek için kurulmuş bir mekanizmadır. Garbage Collector bizim için memory allocation ve deallocation işlemlerini arka planda yapar. Gereksiz oluşturduğunuz nesneleri yok eder, böylece OutOfMemory, memory leak, gibi problemlerin önüne geçer.

**5. Memory leak nedir ? Java’da memory leak oluşması mümkün müdür ?**

Kullanılmayan nesnelerin erişilebilir nesneler tarafından bir şekilde başvurulduğu durumlarda, Garbage collection bu nesneleri yararlı nesne olarak işaretler, Bu nesneler bellekte kalmaya devam eder ve program işi bittikten sonra kullandığı hafıza bloğunu serbest bırakamaz. Memory leak budur. Memory leake sahip bir program zaman içinde giderek daha fazla hafıza kullanır. Bu da sistem kaynaklarının fazlaca tüketilmesine en son uygulamanın hata verip kapanmasına sebep olur. Java’da memory leak oluşabilir. Kaydı sonlandırılmamış dinleyiciler, kapatılmamış açık akışlar ve bağlantılar, anonim sınıflar, içi içe sınıf referansları Java’da memory leak oluşturabilir.

**6. Yeni Java sürümleri ne sıklıkla çıkmaktadır ?**

Oracle; Long Term Support (LTS) yani uzun süre destekli Java sürümlerini her 3 yılda bir, onun dışında non-LTS yani kısa süre destekli sürümleri 6 ayda bir çıkaracağını duyurdu.

* Java 7 🡪 2011
* Java 8 🡪 2014
* Java 9 🡪 2017
* Java 10  🡪 2018

**7. Stack ve Heap nedir ? Java’da hangi yapılar stack ile, hangi yapılar heap ile ilişkilidir ?**

Stack ve heap ikisi de ramin mantıksal bölümleridir*.* Stack; LIFO (Last in first out) yani son gelen ilk çıkar mantığına göre çalışır. Stack; Heap’e göre daha hızlıdır ve veriler art arda sıralanmıştır. Stack’de veri hemen silinebilir. Java’da primitive type yani ilkel veri tipleri Stack’de tutulur. Bunlarint, short, byte, long, double, float vs.

Heap de verilerin tutulduğu bir alandır ve veriler karışık bir şekilde tutulur. Heap’de silme işlemi garbage collector yapısına bağlıdır. Java’da class interface, array gibi referans veri tipleri heap kullanır.

**8. OpenJDK ve OracleJDK arasındaki farklar nelerdir ?**

* OpenJDK ve OracleJDK arasında yayın takvimi açısından farklar vardır.
* OracleJDK 3 yılda bir yeni sürüm çıkarırıken OpenJDK 6 ayda bir sürüm çıkarır. Bu yüzden Oracle ‘ın sürümleri daha tutarlı ve problemsizdir.
* OracleJDK uzun süreli destek sağlarken OpenJDK uzun süreli destek sağlamaz.
* Oracle aynı zamanda daha fazla Garbage Collector seçeneğine sahiptir ve performans açısından daha iyidir.

**9. @FunctionalInterface anotasyonu nerelerde kullanılabilir, neleri sağlar ?**

Functional interface; içinde sadece bir tane abstract method olan interfacelerdir. Functional interface’i belirtmek için **@FunctionalInterface** anotasyonu kullanılması zorunlu değildir bu sadece validasyon sağlar. Fuctional intefaceler lambda expressionların kullanılabilmesini sağlarlar. Böylece kod yazarken daha az kod yazarız.

@FunctionalInterface

public interface Runnable{

public abstract void run();

}}

//Lambda olmadan kullanımı

new Thred(new Runnable(){

@Override

public void run(){

System.out.println(“Thread çalışıyor.”);

}}.start();

//Lambda ile kullanımı

new Thread(() -> System.out.println(“Thread çalışıyor.”).start();

**10. Java’da hangi functional interface’ler yer almaktadır ? Yaptığınız araştırmada en popüler/göze çarpanlar hangileridir ?**

**java.util.function p**aketindeki **yer alan** functional interfaceler. **\*\*\***

**Consumer**: Bir argüman alır ve geri bir şey döndürmez. **Predicate**: Bir argüman alır ve bir boolean değer döndürür. **Function**: Bir argüman alır ve geri bir değer döndürür. **Supplier**: Argüman almaz ve geri bir değer döndürür. **BiConsumer**: İki argüman alır ve geri bir şey döndürmez. **BiPredicate**: İki argüman alır ve geriye bir boolean değer döndürür. **BiFunction**: İki argüman alır ve tek bir değer döndürür. **UnaryOperator**: Argüman alır ve yine aynı argümanı geri döndürür. Function'dan türer. **BinaryOperator**: İki argüman alır ve yine aynı tipte bir argüman döner. BiFunction'dan türer.

\*\*\*

java.util.function paketi altında 4 tane göze çarpan önemli functional interface vardır. Bunlar Predicate, Function, Consumer ve Supplier interfaceleridir.

\*\*\* Kaynak 🡪 <https://www.fibiler.com/Functional-Interface_Kategori_1276>

Tunahan Burak Dirlik