## 1 Pass by value, pass by reference kavramları nedir ?

## a)Pass By Value

Bu yöntem metotlara parametreler geçilmeden önce parametrenin değeri belirlenir.Bu belirlenen değer bellekte bir alana kopyalanır ve daha sonra parametre aktarımı yapılır.Parametre aktarımı yapılırken ,bu bellek alanının adresinin kendisi değil kopyası metota gönderilir.Yani aslında pass by value=pass-by-copy-of-the-bits-in-the-variable ( değişkendeki bitlerin kopyası) demektir.

Yani çağırdığımız metot içerisinde bu değişkenin değerini değiştirdik,o zaman metot içerisinde bu değişkenin yeni değeri geçerli olur.Fakat metot içerisinden çıktığımızda,değişken,değişmeden önceki değerine geri dönecektir.Çünkü biz metoda değişken adresinin kendisini değil,kopyasını verdik ve o kopyanın değeri değişti.

# b) Pass By Reference

Belirli bir sınıfın içinde tanımlanmış olan değişkenin (instance variable) RAM’de bulunduğu lokasyonu gösteren (refer eden) object referer yapısının metoda parametre olarak geçirilmesidir. Dolayısıyla, metodun içerisindeki işlemler (process) değişkenin orjinal değeriyle yapılır ve bunun sonucunda orjinal verinin (attribute) değeri değişebilir.

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

Immutable (değişmez), nesneler bir kez oluşturulduktan sonra içeriği değiştirilemeyen sınıflardır. Tam tersi olarak, değiştirilebilen sınıflar da Mutable (değişebilir) sınıflardır. Immutable nesneler değişmeyen nesnelerdir. Onları oluşturursun, fakat onları değiştiremezsin. Bunun yerine, değişmez bir nesneyi değiştirmek isterseniz, onu klonlamanız ve oluştururken klonu değiştirmeniz gerekir.

Immutable nesneler, çok iş parçacıklı(multi-threaded) ortamlarda ve streamlerde kullanışlıdır. Değişmeyen nesnelere güvenmek harikadır. Başka bir thread’in nesnesini değiştiren bir iş parçacığının neden olduğu hatalar olabilir. Immutable nesneler, bu sorunların tümünü çözmüş olacaktır.

**Java’da Immutable sınıf yapmak için aşağıdaki adımları uygulamanız gerekir.**

* Sınıfı, final anahtarı ile işaretlemek gerekir, böylece genişletilemez (extend edilemez).
* Sınıfın tüm alanlarını private yapın, böylece doğrudan erişime izin verilmez.
* Değişkenler için setter yöntemleri sağlamayın.
* Tüm değiştirilebilen alanları final yapın, böylece yalnızca bir kez atanabilir.
* Tüm alanların, contructor aracılığıyla ilk değerlerini atamasını sağla.
* Değiştirebilir olan tüm alanların dışarıya nesnelerin klonlanarak dönmesini gerçekleştirin.

private final String name;

public ImmutablePerson(String name) {

this.name = name;

} public String getName() {

return name;

} }

**Effective Java’da, Joshua Bloch bu etkileyici tavsiyeyi yapar:**

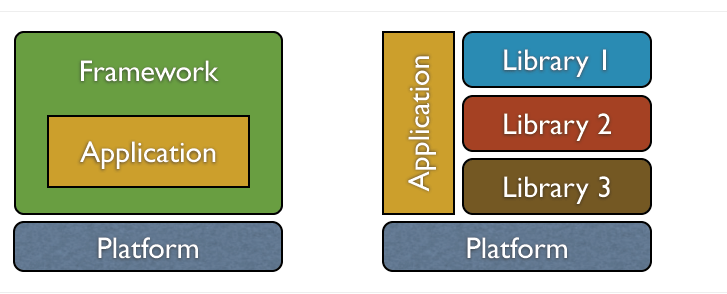
“Classes should be immutable unless there’s a very good reason to make them mutable….If a class cannot be made immutable, limit its mutability as much as possible.”

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

Framwork uygulama çatısıdır . Yani biz uygulamamızı bu çatıya göre geliştiririz örneğin ASPNET MVC bir framworktür. Biz her şeyi buna göre geliştiririz. Jquery ise bir library (kütüphanedir) MVC de query kullanabiliriz. Kütüphanelerin ortak özellikleri işleri kolaylaştırmaktır. Yani bir javascriptkullanarak Jquery ile yapılan her şeyi yapabiliriz zaten jquery saf java script kütüphanesi kullanılarak geliştirilmiştir. Library aslında bir metotlar topluluğudur. Bu metotlar ile bizlerin iş yükünü kolaylaştırmak amaçlanmaktadır.

Library i basitleştirilmiş kodlar, metotlar ,fonksiyonlar topluluğudur.

Framwork ise uygulama çatısıdır , uygulama framwork üzerine inşa edilir. MVC , Angular bir framworktür.

****

İki arasındaki temel teknik fark, kodun nasıl çağrıldığıdır. Library kullanırken, library size bazı özellikler vererek kullanmanızı sağlar, bu şekilde almış olduğunuzu kodu kendi sisteminize uygularken size kodu nerede ve ne zaman kullanacağınıza karışmaz. Framework ise Library'in tersine kullanacağız özelliğe göre kodu nerede ve ne zaman kullanacağınız söyler, kullanacağız bu işlev, eğer Framework dokümanın belirtildiği gibi kullanılmaz ise kullanım dışı kalır.

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

Garbage Collection, otomatik bellek yönetimi mekanizmasıdır. Bu işlem heap belleğe bakıp, kullanılan objelerin tespit edilmesi ve referans edilmeyenlerin silinmesi üzerine kuruludur. Kullanılmayan/referans edilmeyen nesnelerin kapladığı alan bellekte boşa çıkarılır ve bellekte boş yer açılmış olur. Bu işlemi yapan mekanizmaya da Garbage Collector denir.

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

“Memory leak”, programın kullandığı hafıza ile işi bittiği halde ilgili hafıza bloğunu serbest bırakmaması durumudur. Bu durum;Gerçekte kullanılmayan hafıza bloklarının referans edilmeye devam edilmesinden veya Native resource’ların düzgün bir şekilde serbest bırakılmamasından kaynaklanmaktadır.

Garbage collector hafızada referans edilmeyen, kullanımı sona ermiş olan hafıza bloklarını bulup, serbest bırakmaktadır. Fakat hâlen referans olarak gösterilen ve kullanılmasa da kullanımda görünen hafıza bloklarına ve unmanaged (native) kaynaklara dokunmamaktadır. Bu yüzden garbage collector’un memory leak’lere engel olduğuna dair bir kanıya sahip olmak oldukça yanlıştır. Çünkü memory leak durumunda kullanımı sona ermiş olan hafıza bloğu halen referans edilmeye devam edildiğinden veya unmanaged (native) kaynak kullanılmaya devam edildiğinden garbage collector tarafından bu hafıza bloğu serbest bırakılmayacaktır.

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

Her yılın mart ve eylül aylarında, yeni bir Java versiyonu yayınlanıyor.

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

Stack ve Heap kavramları, ram’in mantıksal bölümleridir diyebiliriz. Stack’de değer tipleri, pointer ve adresler saklanırken, Heap’de ise referans değerleri saklanmaktadır.

Stack’e erişim Heap’den daha hızlıdır ve Stack, LIFO (Last-In-First-Out) mantığında çalışmaktadır. Yani son gelen ilk olarak çıkar. Bu sebep ile aradan herhangi bir eleman çıkartamazsınız, birbirleri ile ilişki içerisindedirler.

Struct tipindeki değişkenler değer tipleridir ve Stack içerisinde saklanmaktadır. Class tipindeki değişkenler ise referans tipleridir ve referansları Stack’de kendisi ise Heap’de saklanır.

Stack & Heap’in genel özelliklerine kısaca bir bacak olursak;

* Stack ve Heap ram’in mantıksal bölümleridir.
* Stack LIFO mantığında çalışır. Yani son gelen ilk olarak çıkar.
* Stack’de değer tipleri, pointer ve adresler saklanırken Heap’de ise referans değerleri saklanır.
* Stack daha hızlıdır. Ulaşılmak istenen veriler ard arda sıralanmış olur.
* Heap ortak olarak kullanılır ve uygulama başlatıldığında başlar.

NOT: ref anahtar kelimesi ile bir method’a değer tipi dahi yollasak, (yukarıda bahsettik değer tipleri Stack üzerinde tutulmaktadır) söz konusu değişkenin içeriğini Stack bölgesinden Heap bölgeye aktarır ve adresini ise Stack bölgesindeki alanına yazar.

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

Oracle her üç yılda bir yeni sürüm sunarken, OpenJDK altı ayda bir yeni sürümünü piyasaya sürecektir. Oracle, sürümleri için uzun vadeli destek sağlar. Öte yandan, OpenJDK, yalnızca bir sonraki sürüm yayınlanana kadar bir sürümdeki değişiklikleri destekler.

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

Functional interface, Java 8 ile gelmiş özelliklerden biridir.

Function interface, içerisinde sadece bir tane abstract metodu olan interface’dir. Eğer ilgili interface’in türetildiği interface’de abstract metot varsa bu durumda da functional interface olur.

Functional interface’ler, lambda expression’ların kullanılabilmesi için tanımlanırlar.

@FunctionalInterface

interface Matematics {

void operation (int a, int b);

}public class TestCustom {

public static void main(String[] args) {

Matematics sum = (a, b) -> System.out.println("Sum: " + (a + b));

sum.operation(10, 6);

Matematics minus = (int a, int b) -> System.out.println("Minus: " + (a - b));

minus.operation(10, 6);

Matematics multiply = (int a, int b) -> System.out.println("Multiply: " + (a \* b));

multiply.operation(10, 6);

}

}

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

java.util.function paketi altında 4 tane önemli functional interface vardır. Predicate, Function, Consumer ve Supplier interfaceleridir. Geri kalan functional interfaceler bunların primitive tip versiyonları, binary versiyonları ya da bunlardan türeyen interfacelerdir.

Predicate Interface

Consumer Interface

Supplier Interface

Function Interface

Diğer funcitonal interface’lerde generic olarak alınan tek bir tip vardı. Onu parametre olarak ya alır ya da geri döndürürdü. Ancak benim parametre olarak aldığım tip ile geri döndüreceğim tip farklı ise ne kullanmalıyım sorusunun cevabı: Function Interface. Bu da diğerleri gibi generic’tir. İçerisinde apply() adında bir tane abstract metod vardır. 2 tane de default metodu vardır.

**R apply(T t):** İçerisine T tipinde değer alır R tipinde değeri geri döndürür.

**andThen(Function<? super R, ? extends V> after):** Funciton interface’lerini zincirleme olarak çağırır.

**compose(Function<? super V, ? extends T> before):** İçerisine aldığı function nesnesinin geri döndürdüğü değeri, kendisini çağıran function nesnesine parametre olarak geçer.

Person person = new Person(1, "John","Doe");

Function<Person, String> personFunction = Person::getName;

String result = personFunction.apply(person);

System.out.println("Person name is: " + result);