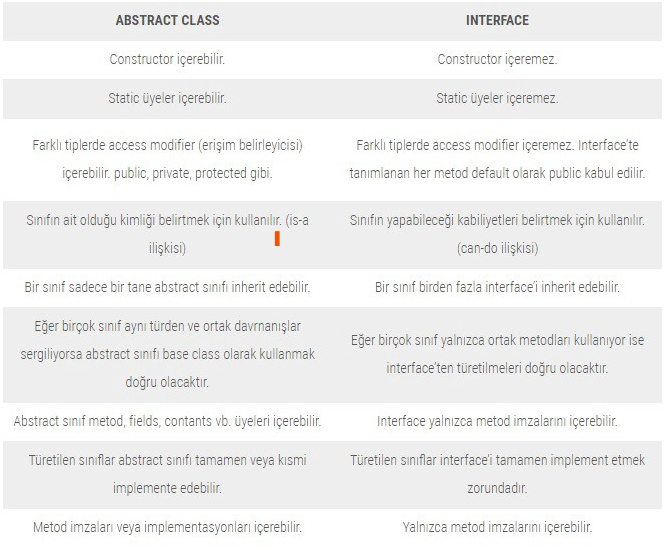
JAVA SIK SORULAN SORULAR

# Neden OOP ye ihtiyaç duyuyoruz? OOP dillerinden örnek veriniz?

OOP 1960’lı yıllarda yazılım dünyasının yaşadığı boyutlandırma sorunu sonrasında ortaya çıkmıştır. OOP ile nesne oluşturma bir sınıf içerisinde oluşturulur ve böylelikle tüm projelerde kullanılabilmektedir. Bu sayede sınıflar 1 kez oluşturulur ve uzun kodların tekrardan yazılmasına gerek olmaz. Uzun kodlar tekrardan yazılmadığı içinde yazılımların oluşturulması daha da kısalır.

OOP ile oluşturulan nesneler birinden bağımsız olduğu için daha güvenli olmaktadır. Sınıf oluşturma sistemi sayesinde de tüm yazılımda değişiklik yapmak yerine sadece sınıfta değişiklik yapmak yeterli olmaktadır. Popüler sınıf tabanlı OOP dilleri arasında Java, Python, C# ve C++ bulunur.

# Interface vs Abstract class ?



# Why we need equals and hashcode ? When to override ?

Öncelikle **equals** metodunu inceleyelim; İlgili metot, iki nesnenin **içeriğinin aynı olup olmadığını** kontrol eder. Override edilmeden == gibi çalışır.

**(==)** operatörü **adreslerin aynı olup olmadığını** kontrol eder.

**equals()** ve **hashcode()** Java programlarında nesne karşılaştırmalar etkin ve kolay hale getirmek için birleştirir. Basitçe söylemek gerekirse, bu yöntemler iki nesnenin aynı değerlere sahip olup olmadığını doğrulamak için birlikte çalışır.

Object sınıfından gelen **hashCode** metodunu gerektiği yerde geçersiz kılmamak (override) birçok hatanın kaynağını oluşturur. **equals** metodunu geçersiz kıldığınız her sınıfta **hashCode** metodunu da geçersiz kılmanız gerekir. Bunu yapmamak **Object.hashCode** metodunun sözleşmesini ihlal etmek demektir, bu da sınıfınızın hash tabanlı HashMap, HashSet, HashTable gibi veri yapılarıyla birlikte kullandığında yanlış çalışmasına yol açar. Bunun sebebi ise Object sınıfından gelen **hashCode** metodunun hash kodunu hesaplarken nesnenin o anda bulunduğu bellek adresini kullanmasıdır. Her nesnenin bellek adresi farklı olacağı için hesaplanan hash kodu da farklı olacaktır.

# Diamond problem in Java ? How to fix it?

Bir sınıfın birden fazla sınıftan **extend** etmeye çalışıldığında java buna izin vermeyecektir. Buna **diamond problem** denmektedir. Çümkü java ilk hangi sınıftan gelen methodu çağıracağını bilemeyecektir bu nedenle bir belirsizlik meydana gelecektir.

Bunu önlemek için interface’lerin birden fazla interface’i implemente edebilmeleriyle çözülmektedir.

# Why we need Garbagge Collector ? How does it run ?

Yazılımınız içinde gereksiz oluşturduğunuz nesneler bellekte gereksiz yer kaplayabilir belli süre sonra da memory leak problemleri ile karşılaşabilirsiniz. Java’da bellek yönetimi arka planda **JVM** ve onun içinde yer alan **Garbage Collector(Çöp Toplayıcısı)** ile yapılır.

**GC,** 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.

# Java ‘static’ keyword usage ?

**Static** metodlar direk çağırılabilirler. ClassIsmı.mthodIsmi( parametreler)

Math sınıfının tüm elemanları static’tir.

Static metodlar bulundukları sınıfın static olmayan method ve değişkenlerine erişemez.

Statik değişkenler oluşturulduğunda bellekte bir yer ayrılır ve o değişken kullanılsa da kullanılmasa da program sonlanana kadar o bellekte olur. Oluşturulan her bir nesne için tekrardan bellekte yer kaplamazlar. Yani new lenirlerse heap ta yeni bir yer açılmaz. Çünkü statik bir değişken ve nesnelerden bağımsız olarak sahip olduğu veriyi kendine özel olan bellekte tutmakta.

# Immutability means ? Where, How and Why to use it ?

Javada **immutable** yani değişmez yapılar vardır. String, Boolean, Integer gibi bütün wrapper classlar immutable’dır. Immutable classlar bir kere oluşturulur ve değişmezler.

Immutable nesneler ilk değerlerini constructur üzerinden alırlar ve bu değerde sabit kalırlar ve senkronize çalişmak zorunda kalmazlar. Bu da onların **thread-safe** olmasını sağlar.

Immutable nesnelerin değerleri değişmeyeceği için üzerinde ne kadar thread çalışırsa çalışsın hep aynı değerler üzerinden işlem yapılacaktır. Bu noktada ortaya çıkan bir dezavantajlık söz konusuda var tabii. Immutable nesnelerin dezavantajı ise, değerin değişmesini istiyorsak yeni bir nesne oluşturmamız gerekiyor. Bu da zaman, bellek ve garbage collector maliyetleri oluşturmaktadır.

**Immutable class** oluşturmak için gerekli olan özellikler şu şekilde sıralanabilir:

* Class içerisinde tanımlanan değişkenler **private** olmalı. Çünkü dışarıdan herhangi bir müdahalede bulunulmamalı.
* Class içerisindeki değişkenlerin **setter** methodları **olmamalı**. Bu da dışarıdan gelecek müdahalelere karşı sıkı bir önlem.
* Değişkenler **final** olarak işaretlenmeli. Bilindiği üzere final işaretlenen değişkenler ilk değeri aldıktan sonra değiştirilemezler.
* Final işaretlenen bu alanların ilk değerleri **constructor** ile verilmeli.
* Ve son olarak **classımızı da final** işaretlemeliyiz. Extend edilmemesi gerekiyor.

# Composition and Aggregation means and differences ?

**Aggregation** : Bu tür ilişki de nesnelerin yaşam döngüleri birbirlerinden ayrıdır. Bir nesne diğerinden bağımsız olarak da yaşamını sürdürebilir. Yani aralarında bir sahiplik ilişkisi (has-a) vardır. Örneğin Dizüstü Bilgisayarınız ile onun çantası arasında böyle bir ilişki vardır. Çantayı ayrı olarak ya da laptop’u ayrı olarak düşünebiliriz. Yaşam döngüleri ortak değildir. UML diagramında gösterimi ve koda dökülmüş bir örneği aşağıdaki gibidir.



**Composition** : Bu tür ilişki de nesnelerin yaşam döngüleri birbirleriyle bağlantılıdır. Bir nesne diğerinden bağımsız olarak kullanılamaz. Aralarındaki ilişki parçası olma(is-a-part-of ) ilişkisidir. Az önceki örneğimizden gidersek dizüstü bilgisayarımız ile ekranı arasında bu tarz bir ilişki vardır. UML diagramında gösterimi ve koda dökülmüş bir örneği aşağıdaki gibidir.



**Aggregation** ve **Composition** arasındaki mantıksal fark şudur: Aggregation sahip olunan nesnenin sahip olan nesneden bağımsız bir şekilde var olabilmesine denir. Composition ise sahip olunan nesnenin sahip olan nesneden bağımsız bir şekilde var olamamasına denir.

# Cohesion and Coupling means and differences ?

**Coupling**, modüller arasındaki karşılıklı bağımlılık derecesinin ölçüsüdür. Yani modüller arası bir kavramdır ve iki alt kavramı vardır. Bunlar; Gevşek Bağlılık ( Loosely Coupled) ve Sıkı Bağlılık (Tightly Coupled)’dır. Tightly Coupled’da ilişki içerisindeki modüller birbirlerine sıkı sıkıya bağımlıdır. Bu bağımlılık, modüllerin birinde yapılacak değişikliğin diğer modülleri de etkilemesine sebep olur ve bu nedenle projenin bakım, genişletilebilirliği ve maliyeti artar. Bunun önüne geçmek için **yazılımın olabildiğince Loosely Coupled** olması istenir. Yani bir modül üzerinde yapılan değişikliğin diğer modüllere sıçramaması gerekir.

**Cohesion** ise modülün elemanlarının işlevsel olarak ilişkili olma derecesinin bir ölçüsüdür. Yani bu da modül içi bir kavramdır. Tek bir görevi gerçekleştirmeye yönelik tüm öğeler tek bir bileşen altında toplanmalıdır. Aynı zamanda SOLID prensiplerinin S harfini temsil eden **(Single Responsibility Principle)** bu tanım cohesion’u açıklar. Eğer bir modül içerisinde birbirinden bağımsız öğeler(metodlar, veriler) varsa bunlar olabildiğince ayrıştırılmalı ve modülün Yapışkan (High Cohesion) olması sağlanmalıdır.

Özetle; yazılımın **Loosely Coupled (Düşük Bağımlı)** ve **High Cohesion (Yüksek İlişkili — Yapışkan)** olması gereklidir.

# Heap and Stack means and differences ?

**Stack** ve **Heap** bellekte (RAM’de) bulunan mantıksal yapılardır. **Değer tip (value type)** dediğimiz int, short, byte, long, decimal, double, float gibi tipler **stackte** tutulur. Stackte veriler üst üste (LIFO– Last in First out) mantığında dizilir ve sırası gelmeden aradaki bir değer ile işlem yapılamaz. **Class type (Sınıf tipi)** değişkenler **referans tiplerdir** referans ettikleri model (referans) stack’te değerleri ise heap’de saklanır.

## Stack ve Heap Arasındaki Farklar

* Heap ve Stack arasında ki en önemli farklardan birisi heapde veriler karışık bir şekilde saklanırken stackte artan ya da azalan adres mantığında (big and little endian) çalışır. Buna bağlı olarak heapde yer alan bir veriye erişmek stackte yer alan bir veriye erişmeye göre daha maliyetli bir işlemdir.
* Stackteki veri hemen silinirken heapdeki veri Garbage Collector algoritmasına bağlıdır.
* Stack bellekten statik olarak yer tahsisi için kullanılırken, Heap dinamik olarak yer tahsisi etmeyi sağlar. Her ikisi de Ram bölgesinde bulunur. Stack’te yer alan veriler direk bellek içine yerleştirilir dolayısıyla erişimi çok hızlıdır.
* Kullanacağınız yerin boyutunu tam olarak biliyorsanız Stack, ihtiyacınız olan boyutu tam olarak bilmiyorsanız Heap kullanımı daha mantıklı bir tercih olacaktır.

# Exception means ? Type of Exceptions ?

Exception, normal kod akışını bozan programların yürütülmesi sırasında meydana gelen olaylardır. Örnek olarak sıfıra bölme, index dışı dizi erişimi vb.

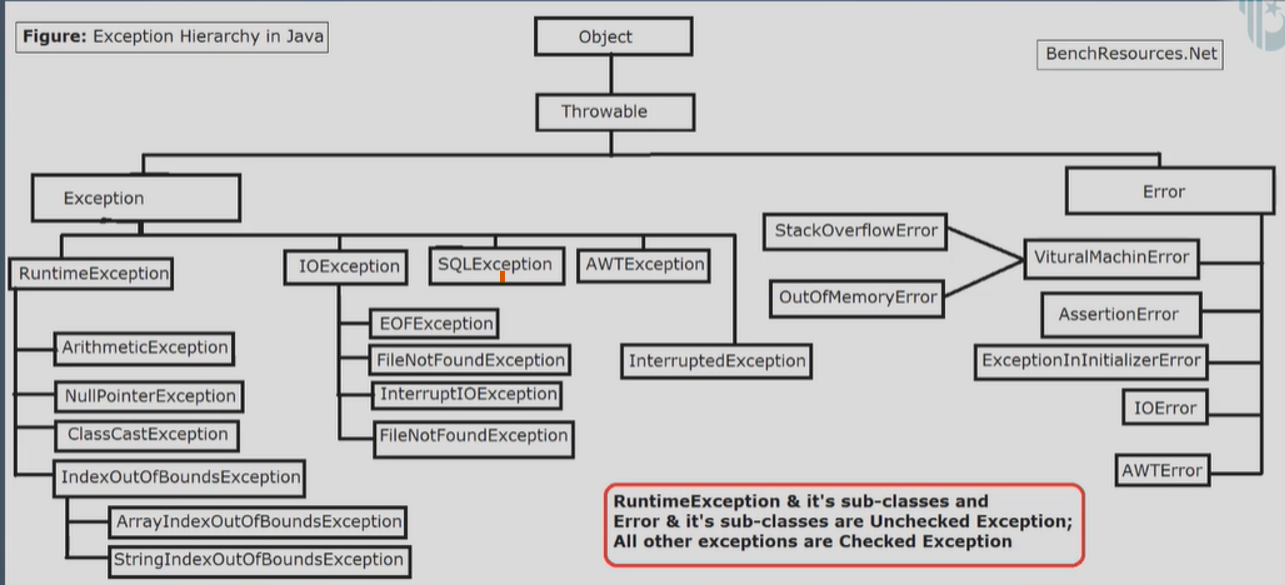
Exception sınıfları diğer Java sınıfları gibi Object sınıfından türer. Object sınıfının bir alt sınıfı Throwable sınıfıdır. Bu sınıftanda Exception sınıfı türer. Exception iki kısma ayrılır:

1. Check Exception 2- Uncheck Exception

## Check Exception

IDE’nin yazılımcıyı uyardığı exception türüdür. Checked Exceptionlar hepsi derleme zamanında işlenir. Örnek olarak birk kod yazıyoruz ve burada hata çıkması muhtemel ise IDE uyarı verir. Bu hataya Check Exception denir.

## Uncheck Exception

RuntimeException ve tüm alt sınıfları, Eror ve tüm alt sınıfları uncheck exception dur. IDE’nin yazılımcıya uyarmadığı exception türüdür

# How to summarize ‘clean code’ as short as possible ?

* Açıklayıcı adlar kullanın
* Okunabilir bir kod oluşturmak için boş satırlar kullanın
* Bir fonksiyona üçten fazla parametre göndermeyin
* Fonksiyonların tek bir şey yapması gerektiğini unutmayın
* Bir satırdaki karakter sayısını azaltın
* Yaptığın commit’e tanımlayıcı mesaj yaz
* Birim Testini kullanın ve Teste Dayalı Geliştirmeyi uygulayın
* Design Patterns öğren
* The DRY Principle: Don’t Repeat Yourself.

# What is the method of hiding in Java ?

Super class ve sub class parametreleri dahil olmak üzere aynı methodları içeriyorsa ve static olduklarında Super class’taki method subclass tarafından saklanacaktır. Örneğin aşağıdaki koda bakalım

class Demo{

public static void demoMethod() {

System.out.println("method of super class");

}

}

public class Sample extends Demo {

public static void demoMethod() {

System.out.println("method of sub class");

}

public static void main(String args[] ) {

Sample.demoMethod();

}

}

OUTPUT:

method of sub class

# What is the difference between abstraction and polymorphism in Java ?

**Abstraction**, bir şeyin belirli bir ayrıntısını ifade etmez ve **Polimorfizm**, aynı nesneye sahip, ancak farklı görevler yapan farklı nesnelerin yöntemlerini ifade eder.