1 – Why we need to use OOP ? Some major OOP languages ?

Doğal : Bu sayede daha anlaşılır program yazılır ve geliştirilir. Ayrıntılarda boğulmazsınız. Doğal dil ile yazılmış kod diğer geliştiricilerin de anlamasını sağlamalıdır.
Güvenilirlik : İyi tasarlanmış, özenle yazılmış nesne yönelimli programlar güvenilirdir. Modüler Nesnelerin doğası, programınızın bir bölümünü etkilemeden değişiklik yapmanıza izin verir.Güvenilirliği artırmanın bir yolu kapsamlı bir testtir. OO, testi kolayştırır. Bilgi ve sorumluluğu tek bir yerde izole etmenizi sağlar. Bu yalıtım, her bir bileşeni bağımsız olarak test etmenizi ve doğrulamanızı sağlar. Bir bileşeni doğruladıktan sonra, onu güvenle tekrar kullanabilirsiniz.
Yeniden kullanılabilirlik: Bir inşaatçı, her evi inşa ederken yeni bir tuğla keşfi yapıyor mu? elektrik mühendisi her devre tasarımı yaparken yeni bir direnç türünü keşfeder mi? Peki neden programcılar bir sorun çözüldükten sonra onu tekrar tekrar kullanmasın?
"Tekerleği yeniden icat mı edeyim?"
İyi hazırlanmış nesne yönelimli sınıfları kolayca yeniden kullanabilirsiniz.
İyi hazırlanmış sınıflar oluşturmak zor bir yetenektir.Soyutlamaya odaklanmayı ve dikkat etmeyi gerektirir.
Sürdürülebilirlik: Bir programın yaşam döngüsü teslim ettiğinizde bitmiyor. Aslında çalışmak için harcanan zamanın% 60 ila % 80'ini programın bakımıdır. Kalkınma kısmı denklemin sadece% 20'si!
İyi tasarlanmış nesne yönelimli kod bakımlıdır. Bir hata, basitçe sorunu tek bir yerde düzeltilir. Uygulamada bir değişiklik şeffaf olduğu için diğer nesneler otomatik olarak donanımdan fayda sağlayacaktır.
Genişletilebirlik: Tıpkı bir programı korumanız gerektiği gibi, kullanıcılarınız sizden yeni işlevsellik eklemenizi isteyebilir. OOP nesnelerin bir kütüphanesini oluştururken, aynı zamanda nesnelerinin işlevselliğini de ele alır. Yazılım statik değildir. Yazılımın büyümesini ve değişmesini de göz önünde bulundurmak gerekmektedir. OOP, programcıya kodu genişletmek için çeşitli özelliklerle sunar. Bu özellikler inheritance, polymorphism, overriding,delegation, ve design patterns gibi.
Zamandan Tasarruf: Modern yazılım projesinin yaşam döngüsü genellikle haftalarca sürebilir. OOP sayesinde bu süreç daha hızlı olmaktadır. Programınızı nesneler halinde oluşturmuşsanız bir problem olduğunda birden fazla kod parçası üzerinde ayrı ayrı çalışabilir hatta birden fazla geliştirici tarafından bağımsız olarak aynı anda problemi çözebilirsiniz. Bu size zamandan tasarruf etmenizi sağlayacaktır.

Java, C#, C++, Python popüler OOP dillerine örnek verilebilir

2. – Interface vs Abstract class ?

- Interface'ler çoklu kalıtımı sağlamaya yardımcı abstract classlar ise çoklu kalıtımı desteklemez.
- Interface' lerde metodların içerisini dolduramayız ama abstract classlarda doldurabiliriz . Böylece bütün alt sınıfların belli bir özelliğe sahip olmasını sağlayabiliriz.
- Interface ile yapabildiğimiz herşeyi hatta daha fazlasını abstract classlar ile de yapabiliriz.
- Eğer türeteceğimiz classlarda belli başlı varsayılan özellikleri tekrar tekrar kopyala-yapıştır yapmak istemiyorsak o zaman abstract class kullanmamız gerekir.
  Çünkü abstract classlarla bir metodu tüm alt classlarda varsayılan metod şeklinde tanımlayabiliriz ve alt classlarda bunları tekrar yazmamıza gerek kalmaz kalıtımla aktarılmış olur.
- Kalıtım sağlamak istiyorsak abstract classlar kullanmamız gerekir.
- Abstract classları kullanmak hız açısından avantaj sağlar.
- Interface de yeni bir metod yazdığımız zaman bu interfaceden implement ettiğimiz tüm classlarda bu metodun içini tek tek doldurmak gerekiyor ancak abstract classlarda durum farklıdır burada bir metod tanımlayıp içini doldurduğumuzda abstract sınıfımızdan türetilmiş bütün sınıflar bu özelliği kazanmış olur.

3. Why we need equals and hashcode ? When to override ?

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. Object sınıfı belirtiminde tarif edildiği üzere sözleşme genel olarak şu şekildedir:

• Equals karşılaştırmasında kullanılan alanlar sabit kaldığı sürece, hashCode metodu aynı uygulama içerisinde üst üste çağrıldığında her zaman aynı sonucu üretmelidir.

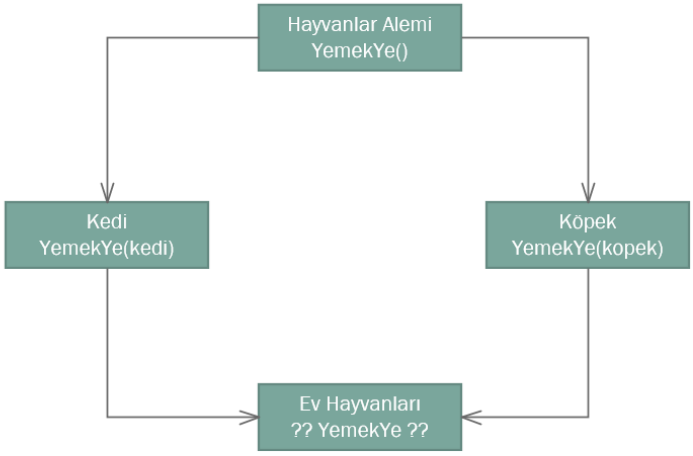
• Eğer iki nesne equals metoduna göre birbirine eşitse, bu iki nesnenin hashCode metotları da aynı integer değerini üretmelidir.

• Eğer iki nesne equals metoduna göre eşit değilse, hashCode metodu bu iki nesne için farklı integer sonuçları üretmek zorunda değildir. Ancak yazılımcı bilmelidir ki eşit olmayan nesneler için farklı hash kodları üretmek hash table performansını artırabilir.
equals ile birlikte hashCode metodunu geçersiz kılmadığınız zaman 2. koşulu ihlal etmiş olursunuz: eşit nesneler eşit hash kodu üretmelidir. Bu durumda birbirinden farklı iki nesne equals metoduna göre eşit olsa da, hashCode metoduna göre aralarında pek de bir benzerlik olmayacaktır. Bu yüzden de hashCode metodu sözleşmenin aksine bu iki eşit nesne için çok farklı değerler üretecektir.

4. Diamon problem in Java ? How to fix it?

C++ dilinde bir sınıfın birden fazla sınıftan kalıtım alabildiğini yukarılarda bahsetmiştik.Kalıtım hiyerarşisinde bulunan base sınıflarda virtual yapılarına sahip elemanlar varsa eğer ve bu base sınıflardan kalıtım alan tekil sınıflar virtual sınıfları eziyorsa(override) bir sıkıntı söz konusu değildir.Lakin bu derived class yapılarına sahip sınıflar C++ özelliği gereğince tek sınıfa çoklu kalıtım vererek o sınıfta da base classlardan gelen virtual metod override edilirse eğer işte burada C++ dilinin kafası karışıyor ve derleyici hata veriyor.Bunun sebebi, çoklu kalıtım alan sınıfın override ettiği elemanı hangi üst sınıftan aldığını mimarinin bilememesidir.

Son olarak akılda kalıcılığını arttırmak için bu konudaki çok duyduğunuz ama gerçek hayattan bir örnek ile konuyu bitirelim. Doğadaki bütün hayvanları temsil eden “Hayvanlar Alemi” isimli bir sınıf var. “Hayvanlar Alemi” sınıfının “Yemek ye” isimli metodu olsun. Bu sınıfın altında bulunan birçok hayvan sınıfı var. Örneğin bu hayvan sınıfları arasında bulunan “Kedi” ve “Köpek” isimli iki sınıfı inceleyelim. Bu iki sınıfta “Hayvanlar Alemi” sınıfından kalıtım yoluyla türetildiği için “Yemek Ye” metoduna sahip.
Ama “Kedi” ve “Köpek” sınıflarının kendine has yemek yeme özellikleri olduğu için “Yemek ye” metodunu kendilerine göre tekrar (Override) oluşturuyorlar. İlerleyen süreçte evde beslemek üzere olan türlerin “Kedi” ve “Köpek” türleri oluştuğu için “Ev Hayvanları” isimli bir sınıf oluşucaktır. Bu sınıf tabii ki de “Hayvanlar Alemi” sınıfından değil “Kedi” ve “Köpek” sınıfından kalıtım ile türeteceğiz. Çünkü her hayvan ev hayvanı değildir. Şimdi bizim “Kedi” ve “Köpek” sınıfından türetilen “Ev Hayvanları” sınıfımızın da “Yemek ye” metoduna sahip olması gerekiyor.
Ev hayvanımıza yemek ye komutunu verirsek nasıl yemek yiyecek? Bir kedi gibi mi yoksa köpek gibi mi? Buradaki durumun elmas şeklini andırmasından dolayı (<>) Diamond Problem ismini almaktadır. Son olarak bunun da diyagramını yapıp konumuzu bitirebiliriz.



C# ve Java gibi dillerde çoklu kalıtım yapabilmenin birinci yolu interface yapılarını kullanabilmekten geçer. Kısaca değinmek gerekirse, C# mimarisinde bir sınıf sadece ve

sadece bir sınıftan türeyebilir ama bir ve birden fazla interface yapısından da kalıtım alabilir. Örnek olarak aşağıdaki kod verilebilir.

```
interface MyInterface1{
   public static int num = 100;
   public default void display() {
      System.out.println("display method of MyInterface1");
   }
}
interface MyInterface2{
   public static int num = 1000;
   public default void display() {
      System.out.println("display method of MyInterface2");
   }
}
public class InterfaceExample implements MyInterface1, MyInterface2{
   public void display() {
      MyInterface1.super.display();
      //or,
      MyInterface2.super.display();
   }
   public static void main(String args[]) {
      InterfaceExample obj = new InterfaceExample();
      obj.display();
   }
}
```

5. Why we need Garbagge Collector ? How does it run ?

Garbage Collector, Çöp Toplayıcısı anlamına gelmektedir.

Java uygulamanızda kullanılan objeler için sistemin bellek yöneticisinden bellek alması gereklidir ve nesne ile işiniz bittiğinde bu ayrılan belleğin sistemin bellek yöneticisine geri verilmesi gerekmektedir.
Buradaki artık kullanılmayan obje belleklerinin bellek yöneticisine geri verme işlemini "garbage collection (çöp toplama)" gerçekleştirmektedir.

Eğer bu işlem yapılmaz ise bellek sızıntısı (memory leak) oluşur, yani bellekte boş yer kalmaz ve artık uygulama çalışamaz hale gelir. Bu bellek ayrımını ve geri verilmesi işlemlerini "garbage collector" (çöp toplayıcısı) Java'da bizim yerimize yapmaktadır ve bizi büyük yükten kurtarmaktadır. Eğer garbage collector olmasaydı, her nesnemi için önce bellek yöneticisine başvuracaktır ve ilgili nesnemiz için bellek talebinde bulanacaktır ve bu nesnemiz ile işimizin hangi noktada bitip bitmediğine karar verecektir ve bu nesne için belleği serbest bırakacaktık, ancak serbest bırakma işlemini gözden kaçırırsak fazla bellek tüketimi yapmış olacaktık ve memory leak sorunlarını çözmek için ayrıca bir zaman ayırmamız gerekecekti.

Heap Hafıza alanı içerisinde JVM ve GC'nin çalıştığı alanlar aşağıdaki gibidir.

<table>
<tr><td>Kalıcı Alan</td><td>Eski Nesil Alanı</td><td colspan="3">Yeni Nesil Alanı</td></tr>
<tr><td>X MB</td><td>Eski Nesil</td><td>BA #1</td><td>BA #2</td><td>Başlangıç Alanı</td></tr>
<tr><td>JVM</td><td colspan="4">Uygulama Alanı</td></tr>
<tr><td colspan="5">Toplam Heap Alanı</td></tr>
</table>

Kalıcı Alan: Bu bölgede JVM ait özel bilgiler bulunur. Örneğin: -XMaxPermSize=64MB komutuyula alanın büyüklüğü belirtilir.

Yeni Nesil Alanı: Bu bölüm toplam üç alandan oluşur. (BA#1, BA#2, Başlangıç Alanı) Oluşturulan nesneler ilk olarak Başlangıç alanında bulunurlar. Belirli bir olgunluğa eriştikten sonra BA#1 veya BA#2 alanlarından boş olana kopyalanır. BA#1 veya BA#2 alanlarından birisi bir sonraki kopyalama için her zaman boş tutulur.

Eski Nesil Alanı: Bu bölümde eski nesil nesneler tutulur. Uygulamanın kullandığı uzun ömürlü nesneler yeni alanından bu alana taşınırlar. Eski nesil alanından biriken nesneleri silmek için işaretle ve süpür algoritması kullanılır.

Garbage Collector işlemini 3 adımda tamamlar. Bunlar:

1. **İşaretle:** Kullanılan ve kullanılmayan nesneler işaretlenir.
2. **Silme:** Referansı olmayan nesneleri heap alanından temizler.
3. **Sıkıştırma ve Silme:** Silme işlemine ek olarak daha büyük nesnelere boş alan oluşturmak için kalan nesneleri bir araya getirir.

6. Java ‘static’ keyword usage ?

Bildiğiniz gibi sınıflar içinde bulunan alanlar (değişkenler), o sınıftan yaratılan her nesne için ayrı ayrı oluşturulurlar. Yani ad, soyad ve yaş gibi 3 bilgi içeren bir sınıfımız varsa, bu sınıftan oluşturduğumuz her nesne için farklı farklı ad, soyad ve yaş değişkenleri bellekte oluşturulur. Bu yüzden bu tür değişkenlere nesneye özgü oldukları için “nesne değişkeni” denir. Bir sınıftan oluşturduğumuz her nesne için tanımlı olan nesne değişkenlerinin değerleri de o nesneye ait olacaktır. Ancak Java’da sadece nesneye ait değil, sınıfa ait değişkenler tanımlamak da mümkün. İşte tam burada “static” anahtar kelimesi devreye giriyor.

Static anahtar kelimesi kullanılarak oluşturulan değişkenler nesne değişkeni değil “sınıf değişkeni” olarak adlandırılırlar. Bu değişkenler nesneye ait değil, sınıfa ait bilgileri taşırlar. Sınıf değişkenleri içinde tanımlandığı sınıftan hiçbir nesne oluşturulmamış olsa bile bellekte yer kaplarlar. Nesne değişkenleri ise ancak bir nesne tanımlandığında bellekte yer kaplarlar. Bu iki tür değişkenin ayrıldığı bir başka nokta da sınıf değişkenlerinin sadece bir örneğinin olmasıdır. Yani o sınıftan kaç tane nesne oluşturulursa oluşturulsun, bellekte tek bir tane sınıf değişkeni vardır ve ne şekilde erişirsek erişelim, aynı sınıf değişkenine erişiriz.

Static Metotlar

Şimdi ise static anahtar kelimesinin başka bir kullanım alanı olan static metotlara bakalım. Normal şartlarda bir sınıftaki bir metodu çalıştırmak istiyorsak, önce o sınıfı kullanarak bir nesne oluşturmalı, sonra bu nesne referansı üzerinden metodu çağırmalıyız. Ancak değişkenlerde olduğu gibi metotlar için de static kelimesini kullanarak nesnelerden bağımsız sınıf metotları yazabiliriz. Sınıf metotlarını çağırmak için o sınıftan bir nesne oluşturmamız gerekmez. Değişkenlere erişir gibi direk sınıf ismi ile bu metotları çağırmamız mümkündür. Static metotlarla ilgili bilinmesi gereken en önemli şey şudur: static metotlar içinden static olmayan bir sınıf öğesine erişemeyiz

Static Kod Blokları

Static kod blokları static değişkenlere ilişkin ilk değer atamalarını yapmak için kullanılan kod bloklarıdır. Bunlara literatürde “static initializer” denmektedir. Static kod blokları sınıf değişkenleri belleğe yüklendikten hemen sonra işletilirler. JVM, o sınıfa ait bir nesne oluşturulmadan önce static kod bloklarının işletimini garanti eder.

Static Import

Static metotların sınıf adı kullanılarak çağırıldığını biliyoruz. Örneğin Java’da “java.lang” paketinde bulunan “Math” sınıfındaki tüm metotlar static metotlardır ve bunları çağırmak için Math.metotAdi() şeklinde bir kod yazmak gerekmektedir. Math sınıfındaki static metotları kodumuza import edip sınıf adını kullanmadan sadece metot adı ile static metotları çağırmak istiyorsak aşağıdaki gibi static import kullanabiliriz. Buradaki tek fark normalde kullandığımız import kelimesinin yanına bir de static eklemektir.

7. Immutability means ? Where, How and Why to use it ?

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 değişmez 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.

8. Composition and Aggregation means and differences ?

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. Ne demek istiyorumu bir örnekle izah edeyim: Mesela araba ve tekerlek nesnelerini düşünelim. Bir araba tekerleğe "sahiptir". Tekerleksiz araba hayliyle düşünülemez. Bu yüzden bu nesneler arasında sıkı bir ilişki vardır. Böylece association'ı elemiş olduk. Fakat bir tekerlek arabasız olabilir. Yani sonuçta raflarda satılacak tekerlekler arabayla yer almaz. Dolayısıyla böyle de bir ilişkide esneklik vardır. İşte bu örnekte sahip olunan tekerlek nesnesi kendi başına ayrı olarak var olabildiği için bu iki nesne arasındaki ilişkiye aggregation'dır deriz. Yani sahip olunan nesnenin sahip olan nesneden bağımsız bir şekilde var olabilmesine aggregation denir.

Composition ise daha önce söylediğimiz gibi sahip olunan nesnenin tek başına var olamayacağı durumu ifade etmek için kullanılan bir ilişkidir. Mesela vücud ve kan hücresi nesnelerini ele alalım. Sizce normal şartlar altında vücudun olmadığı yerde kan hücresi olur mu? Olmaz. İşte bu daha sıkı ilişkiye composition adı verilir. Yani sahip olunan nesne olan kan hücresi tekerleğin aksine tek başına var olamayacağı için bu ilişkiye composition denir. Fakat aklınıza şu soru gelmiş olabilir: Hastanelerdeki şişelerde hastalardan alınan kanlar varken bir kan hücresi vücuddan ayrı bir şekilde var olamaz mı? Evet, olur. Fakat burada ilişkinin aggregation mı composition mı olduğunu iş şahası belirliyor. Eğer bir hastane için yazılım geliştiriyorsanız vücud ile kan hücresi aggregation ilişkisine tekabül eder. Çünkü kan hücresi vücuddan ayrıyetten var olabiliyor. Eğer bir fabrika için yazılım geliştiriyorsanız vücud ile kan hücresi composition ilişkisine tekabül eder. Çünkü bir kan hücresi personelin vücudundan ayrıyetten olamaz. Yani bu ilişki, iş sahasına göre görecelidir.

Sonuç olarak bir ilişkinin aggregation mı composition mı olduğunu anlamak için ilişkideki sahip olunan nesnenin tek başına var olup olamayacağını saptamak yeterlidir

9. Cohesion and Coupling means and differences

Sınıfın modülerliği, sınıf içerisinde yapılacak bir değişikliğin diğer sınıflara olabildiğinde sıçramaması gerektiğidir. Örneğin bir sınıf içerisindeki bir metoda yeni bir parametre eklendiğinde, bu durumunun diğer bir çok sınıfta problem çıkarmaması gerekir.
Sınıflar/Nesneler arası ilişki coupling(bağlama) kavramını yansıtır. İyi bir yazılımda coupling düşük(low) tutulmalıdır. Yani nesneler arasındaki ilişki sıkı sıkıya bağlı olmamalıdır.
Sınıfın sorumluluğunun tek olması ise, içerisinde bulunan metotların ve üyelerin, birbirinden alakasız olmaması, kullanılan verilerin ortak olmasını gerektirir.

Örneğin sınıf içerisinde bulunan bir verinin, yalnızca tek bir metot tarafından kullanılırken, öbürlerinin bunun farkında bile olmaması yanlış bir tutumdur. Öte yandan her bir metodun yaptığı iş aynı amaca hizmet etmelidir. Yapılan işlerin alakasız olması yine yanlış bir tutum olarak görülür. Sınıf içinde verilerin ortak olarak kullanılması ve metotların aynı amaca hizmet etmesi cohesion(yapışma) kavramını yansıtır. İyi bir yazılımda cohesion yüksek(high) tutulmalıdır. Yani sınıf içi metot ve veriler yapışık bir şekilde kullanılmalıdır.
Bu isterler ile sınıf tasarımlarının yapılması, sınıfların bakımını kolaylaştırdığı gibi karmaşıklığını azaltır. Bir yazılım; düşük bağlamlık ve yüksek yapışkanlık(low coupling and high cohesion) özelliklerine sahip olmalıdır.

Özetle Coupling(bağlama) kavramı sınıflar-arası düzen olarak değerlendirilirken, cohesion(yapışma) kavramı ise sınıf-içi düzen olarak değerlendirilir.

10. Heap and Stack means and differences ?

Stack, bir işlev tarafından oluşturulan geçici değişkenleri depolayan özel bir bilgisayar belleği alanıdır. Stackte, değişkenler çalışma zamanı sırasında bildirilir, saklanır ve başlatılır.Geçici bir depolama belleğidir. Hesaplama görevi tamamlandığında, değişkenin belleği otomatik olarak silinecektir. Stack bölümü çoğunlukla yöntemleri, yerel değişkeni ve referans değişkenlerini içerir.

Heap, global değişkenleri depolamak için programlama dilleri tarafından kullanılan bir bellektir. Varsayılan olarak, tüm global değişkenler yığın bellek alanında depolanır. Dinamik bellek ayırmayı destekler.Heap sizin için otomatik olarak yönetilmez ve CPU tarafından sıkı bir şekilde yönetilmez. Daha çok serbest yüzen bir bellek bölgesi gibidir.

Stack ile heap arasındaki farka gelecek olursak bunların birinci farkı bellekte farklı yerleri ifade ediyor olmalarıdır. İkinci farkı ise şudur ki heap, stack'ten farklı bir kullanım amacına sahiptir. O da bir programın çalıştığı sıralar bellekten yer talep edildiği durumlarda talebin heap'ten isteniyor olmasıdır. Yani heap'e aslında dinamik bellektir diyebiliriz. Anlamadığınızı varsayarak yine bir örnek ile açıklamaya çalışayım: C'de listelerle alakalı bir program yazdığınızı düşünün. Ve programın algoritmasını kullanıcının girdiği sayı oranında liste düğümü olacak şekilde ayarladığınızı varsayın. Bu durumda program çalışırken kullanıcının girdiği sayı kadar malloc() fonksiyonu çalıştırılacak ve o kadar bellek alanı talebinde bulunulacaktır. İşte bu talepler sonucu cevap olarak dönen alan tahsisinin yapıldığı yer heap'tir. Heap'in zaman zaman dinamik bellek olarak adlandırılmasının nedeni de budur. Stack de istif bellek olarak adlandırılabilir.

11. 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.

Hata (Error) : Ölümcül bir hatayı işaret eder ve telafisi çok zordur. Örneğin OutOfMemoryError(Yetersiz Bellek) hatası oluşmuş ise uygulamanın buna müdahele edip düzeltmesi olanaksınızdır.
Kontrolsüz İstisnalar(Unchecked Exceptions) : Bu istisna tiplerine Çalışma Anı İstisnaları da (Run-Time Exceptions) denilir. Çünkü çalışma anında meydana gelen istisnalardır. Eğer uygulama normal seyrinde giderse ortaya çıkmaması gerekir. Örneğin, ArrayIndexOutOfBoundsException istisna tipi, bir dizinin olmayan elemanına eriştiğimiz zaman ortaya çıkar. Yani kontrolsüz kodlamadan dolayı meydana gelen istisna tipleridir. Java bu tür istisnalar için önceden bir önlem alınmasını şart koşmaz; yine de önlem almakta özgürsünüzdür.
Kontrollü İstisnalar(Checked Exceptions) : Bu istisna tiplerine Derleme Anı İstisnaları da (Compile-Time Exceptions) denilir. Çünkü derleme anında ide'ler tarafından uyarılırız. Eğer derleyici derleme zamanında exceptionlar için try -catch bloğu göremezse hata verecektir ve kodumuz biz handle edene kadar derlenmeyecektir. Bu istisnalar çevresel koşullardan dolayı oluşabilirler. Örneğin erişilmek istenilen dosyanın yerinde olmaması (FileNotFoundException) veya ağ (Network) bağlantısının kopması sonucu ortaya çıkabilecek olan istisnalardır. Bu istisnalar için önceden önlem alınması gereklidir.

12 – How to summarize 'clean code' as short as possible ?

Yazmayan kişinin bile baktığında kendi yazmış gibi hissedeceği kod temiz koddur.

13 - What is the method of hiding in Java ?

Veri gizleme, dahili nesne ayrıntılarını, yani veri üyelerini gizleme tekniğidir. Nesne yönelimli bir programlama tekniğidir. Veri gizleme, sınıf üyelerine veri erişimini kısıtlamayı sağlar veya garantiler diyebiliriz. Veri bütünlüğünü korur. Veri gizleme, sınıfın dışından doğrudan erişimini önlemek için dahili verileri sınıf içinde gizlemek anlamına gelir.

Veri kapsülleme hakkında konuşursak, Veri kapsülleme özel yöntemleri ve sınıf veri parçalarını gizlerken, Veri gizleme yalnızca sınıf veri bileşenlerini gizler. Hem veri gizleme hem de veri kapsülleme, nesne yönelimli programlamanın temel kavramlarıdır. Kapsülleme , kullanıcıya daha basit bir görünüm sunmak için karmaşık verileri sararken, Veri gizleme , veri güvenliğini sağlamak için veri kullanımını kısıtlar.

Veri gizleme, yazılım bileşenleri arasındaki karşılıklı bağımlılıkları sınırlayarak sağlamlığı artırmak için sistem karmaşıklığının azaltılmasına da yardımcı olur. Veri gizleme, özel erişim belirteci kullanılarak gerçekleştirilir.

Private, Public ve Protected keywordleri ile veri gizleme gerçekleştirilir.

Private olarak bildirilen işlevlere ve değişkenlere yalnızca aynı sınıf içinde erişilebilir ve bildirildikleri sınıfın dışında erişilemezler

Public altında tanımlanan işlevler ve değişkenlere her yerden erişilebilir.

Protected olarak bildirilen işlevlere ve değişkenlere, bir alt sınıf dışında sınıfın dışından erişilemez. Bu belirteç genellikle kalıtımda kullanılır.

14. What is the difference between abstraction and polymorphism in Java ?

Abstraction, bir programcının belirli terimler yerine genel terimlerle düşünerek yazılımı daha iyi tasarlamasına olanak tanırken, Polymorphism bir programcının çalışma zamanında yürütmek istediğiniz kodu seçmeyi ertelemesine izin verir.

Polymorphism ve Abstraction arasındaki diğer bir fark, Abstraction’ın Java'da abstract class ve interface kullanılarak uygulanması, Polymorphism ise Java'da overloading ve overriding ile desteklenmesidir.

Overloading derleme zamanı Polymorphism olarak da bilinse de, overriding gerçek olandır çünkü bir kodun farklı çalışma zamanı koşullarında farklı davranmasına izin verir, bu da polimorfik davranış sergilemek olarak bilinir.