En İyi 50+ Temel Java Mülakat Soruları ve Cevapları

top 50 core java interview questions

En Sık Sorulan Java Mülakat Soruları ve Cevapları örneklerle:

Bu eğiticide, yeni başlayanlar ve deneyimli adaylar için neredeyse 50'den fazla önemli temel Java mülakat sorusunu ele aldık.

JAVA Mülakat Soruları hakkındaki bu gönderi, mülakat amaçlı Java programlamanın temel kavramlarını anlamanıza yardımcı olmak için hazırlanmıştır. Tüm önemli JAVA kavramları, kolay anlaşılmanız için örneklerle burada açıklanmıştır.

Bu eğitim, temel Java tanımları, OOP kavramları, Erişim belirleyicileri, Koleksiyonlar, İstisnalar, İş Parçacıkları, Serileştirme vb. Gibi JAVA konularını mükemmel bir şekilde hazırlamanızı sağlayacak örneklerle kapsar herhangi bir JAVA röportajıyla güvenle yüzleşmek.

En Popüler Java Mülakat Soruları ve Cevapları

Aşağıda, en önemli ve en sık sorulan temel ve gelişmiş Java programlama mülakat sorularının ayrıntılı cevaplarla birlikte kapsamlı bir listesi verilmiştir.

S # 1) JAVA nedir?

Cevap: Java, üst düzey bir programlama dilidir ve platformdan bağımsızdır.

Java, nesnelerin bir koleksiyonudur. Sun Microsystems tarafından geliştirilmiştir. Java kullanılarak geliştirilen birçok uygulama, web sitesi ve oyun var.

S # 2) JAVA'nın özellikleri nelerdir?

Cevap: Java'nın özellikleri aşağıdaki gibidir:

OOP kavramları

- Nesne odaklı
- Miras
- Kapsülleme
- Polimorfizm
- Soyutlama

- **Platform bağımsız:** Tek bir program, herhangi bir değişiklik yapmadan farklı platformlarda çalışır.
- **Yüksek performans:** JIT (Just In Time derleyici) Java'da yüksek performans sağlar. JIT, bayt kodunu makine diline dönüştürür ve ardından JVM, yürütmeyi başlatır.
- **Çok iş parçacıklı:** Yürütme akışı, İş Parçacığı olarak bilinir. JVM, ana iş parçacığı olarak adlandırılan bir iş parçacığı oluşturur. Kullanıcı, iş parçacığı sınıfını genişleterek veya Runnable arabirimini uygulayarak birden çok iş parçacığı oluşturabilir.

S # 3) Java yüksek performansı nasıl sağlar?

Cevap: Java, yüksek performans sağlamak için Tam Zamanında derleyiciyi kullanır. Talimatları bayt kodlarına dönüştürmek için kullanılır.

S # 4) Java IDE'leri adlandırılsın mı?

Cevap: Eclipse ve NetBeans, JAVA'nın IDE'leridir.

S # 5) Yapıcı derken neyi kastediyorsunuz?

Cevap: Oluşturucu listelenen noktalarla ayrıntılı olarak açıklanabilir:

- Bir programda yeni bir nesne oluşturulduğunda, sınıfa karşılık gelen bir kurucu çağrılır.
- Yapıcı, sınıf adıyla aynı ada sahip bir yöntemdir.
- Bir kullanıcı örtük olarak bir yapıcı oluşturmazsa, varsayılan bir kurucu oluşturulur.
- Yapıcı aşırı yüklenebilir.
- Kullanıcı bir parametre ile bir kurucu oluşturduysa, o zaman bir parametre olmadan açıkça başka bir kurucu oluşturmalıdır.

S # 6) Yerel değişken ve Örnek değişkeni ne anlama geliyor?

Cevap:

Yerel değişkenler yöntemin kendisinde bulunan değişkenlerin yöntemi ve kapsamında tanımlanır.

Örnek değişkeni sınıfın içinde ve yöntemin dışında tanımlanır ve değişkenlerin kapsamı sınıf boyunca mevcuttur.

S # 7) Sınıf nedir?

Cevap: Tüm Java kodları bir Sınıfta tanımlanmıştır. Değişkenleri ve yöntemleri vardır.

Değişkenler bir sınıfın durumunu tanımlayan niteliklerdir.

Yöntemler tam iş mantığının yapılması gereken yerdir. Belirli gereksinimi karşılamak için bir dizi ifade (veya) talimat içerir.

Misal:

```
public class Addition{ //Class name declaration int a = 5; //Variable declaration int b= 5; public void add(){ //Method declaration int c = a+b; } }
```

S # 8) Nesne nedir?

Cevap: Bir sınıfın bir örneğine nesne denir. Nesnenin durumu ve davranışı vardır.

JVM 'new ()' anahtar kelimesini her okuduğunda, o sınıfın bir örneğini oluşturacaktır.

Misal:

```
public class Addition{ public static void main(String[] args){ Addion add = new Addition();//Object creation } }
```

Yukarıdaki kod, Addition sınıfı için nesneyi oluşturur.

S # 9) OOP kavramları nelerdir?

Cevap: OOP kavramları şunları içerir:

- Miras
- Kapsülleme
- Polimorfizm
- Soyutlama
- Arayüz

Önerilen Okuma* = >> *En Popüler OOP Mülakat Soruları

S # 10) Kalıtım nedir?

Cevap: Miras, bir sınıfın başka bir sınıfa genişletilebileceği anlamına gelir. Böylece kodlar bir sınıftan diğerine yeniden kullanılabilir. Mevcut sınıf Süper sınıf olarak bilinirken, türetilmiş sınıf bir alt sınıf olarak bilinir.

Misal:

```
Super class: public class Manupulation(){ } Sub class: public class Addition extends Manipulation(){ }
```

Miras, yalnızca kamu ve korunan üyeler için geçerlidir. Özel üyeler devralınamaz.

S # 11) Kapsülleme nedir?

Cevap: Kapsülleme Amacı:

- Kodu başkalarından korur.
- Kod sürdürülebilirliği.

Misal:

Bir tamsayı değişkeni olarak 'a' yı ilan ediyoruz ve bu negatif olmamalı.

```
public class Addition(){ int a=5; }
```

Birisi tam değişkeni ' ***a = -5 '*** o zaman kötü.

Sorunun üstesinden gelmek için aşağıdaki adımları izlememiz gerekiyor:

- Değişkeni özel veya korumalı yapabiliriz.
- Set ve get gibi genel erişimci yöntemlerini kullanın.

Böylece yukarıdaki kod şu şekilde değiştirilebilir:

```
public class Addition(){ private int a = 5; //Here the variable is marked as private }
```

Aşağıdaki kod alıcı ve ayarlayıcıyı gösterir.

Değişken ayarlanırken koşullar sağlanabilir.

```
get A(){ } set A(int a){ if(a>0){// Here condition is applied ..... } }
```

Kapsülleme için, tüm örnek değişkenlerini özel yapmalı ve bu değişkenler için ayarlayıcı ve alıcı oluşturmalıyız. Bu da diğerlerini verilere doğrudan erişmek yerine ayarlayıcıları aramaya zorlar.

S # 12) Polimorfizm nedir?

Cevap: Çok biçimlilik, birçok biçim anlamına gelir.

Tek bir nesne, polimorfizm adı verilen referans türüne bağlı olarak süper sınıfa veya alt sınıfa başvurabilir.

Misal:

```
Public class Manipulation(){ //Super class public void add(){ } } public class Addition extends Manipulation(){ // Sub class public void add(){ } public static void main(String args[]){ Manipulation addition =
```

```
new Addition();//Manipulation is reference type and Addition is
reference type addition.add(); } }
```
Manipulation referans türünü kullanarak, Addition sınıfı 'add ()' yöntemini çağırabiliriz. Bu yetenek, Polimorfizm olarak bilinir. Polimorfizm için geçerlidir **ağır basan** ve için değil **aşırı yükleme** .

S # 13) Yöntemi Geçersiz Kılma ile kastedilen nedir?

Cevap: Yöntemin geçersiz kılınması, alt sınıf yönteminin Süper sınıf yöntemiyle aşağıdaki koşulları sağlaması durumunda gerçekleşir:

- Yöntem adı aynı olmalıdır
- Argüman aynı olmalı
- Dönüş türü de aynı olmalıdır

Geçersiz kılmanın en önemli yararı, Alt-sınıfın, bu alt-sınıf türü hakkında süper-sınıfa göre bazı özel bilgiler sağlayabilmesidir.

Misal:

```
public class Manipulation{ //Super class public void add(){ …… } }
Public class Addition extends Manipulation(){ Public void add(){ …….. }
Public static void main(String args[]){ Manipulation addition = new
Addition(); //Polimorphism is applied addition.add(); // It calls the
Sub class add() method } }
```
add.add () yöntem, üst sınıfta değil, Alt sınıfta add () yöntemini çağırır. Bu nedenle, Süper sınıf yöntemini geçersiz kılar ve Yöntemi Geçersiz Kılma olarak bilinir.

S # 14) Aşırı Yükleme ile ne kastedilmektedir?

Cevap: Yöntem aşırı yükleme, farklı sınıflar için veya aynı sınıf içinde gerçekleşir.

Metot aşırı yükleme için, alt sınıf metodu, aynı sınıftaki Süper sınıf metodu (veya) metotları ile aşağıdaki koşulları karşılamalıdır:

- Aynı yöntem adı
- Farklı argüman türleri
- Farklı iade türleri olabilir

Misal:

```
public class Manipulation{ //Super class public void add(String name){
//String parameter …… } } Public class Addition extends
Manipulation(){ Public void add(){//No Parameter …….. } Public void
add(int a){ //integer parameter } Public static void main(String args[])
{ Addition addition = new Addition(); addition.add(); } }
```
Burada add () yöntemi, Addition sınıfında farklı parametrelere sahiptir, süper sınıfla aynı sınıfta aşırı yüklenir.

Not: Çok biçimlilik, yöntem aşırı yüklemesi için geçerli değildir.

S # 15) Arayüz ile ne kastedilmektedir?

Cevap: Java'da birden çok miras elde edilemez. Bu sorunun üstesinden gelmek için Arayüz kavramı tanıtıldı.

Arayüz, yalnızca yöntem bildirimlerine sahip olan ve yöntem uygulamasına sahip olmayan bir şablondur.

Misal:

```
Public abstract interface IManupulation{ //Interface declaration Public abstract void add();//method declaration public abstract void subtract(); }
```

- Arayüzdeki tüm yöntemler dahili olarak **genel soyut boşluk** .
- Arayüzdeki tüm değişkenler dahili olarak **genel statik final** bu sabitler.
- Sınıflar arayüzü uygulayabilir ve genişletilemez.
- Arabirimi uygulayan sınıf, arabirimde bildirilen tüm yöntemler için bir uygulama sağlamalıdır.

```
public class Manupulation implements IManupulation{ //Manupulation class uses the interface Public void add(){ …… } Public void subtract(){ ……. } }
```

S # 16) Soyut sınıf ile kastedilen nedir?

Cevap: Abstract sınıfını, sınıf isminden önce 'Abstract' anahtar sözcüğünü kullanarak oluşturabiliriz. Soyut bir sınıf, somut bir sınıf olan hem “Soyut” hem de “Soyut olmayan” yöntemlere sahip olabilir.

Soyut yöntem:

Yalnızca bildirime sahip olan ve uygulamaya sahip olmayan yönteme soyut yöntem denir ve 'soyut' olarak adlandırılan anahtar kelimeye sahiptir. Bildirimler noktalı virgülle biter.

Misal:

```
public abstract class Manupulation{ public abstract void add();//Abstract method declaration Public void subtract(){ } }
```

- Soyut bir sınıfın soyut olmayan bir yöntemi de olabilir.
- Abstract sınıfını genişleten somut Alt Sınıf, soyut yöntemlerin uygulanmasını sağlamalıdır.

S # 17) Dizi ve Dizi Listesi arasındaki fark.

Cevap: Dizi ve Dizi Listesi arasındaki fark aşağıdaki tablodan anlaşılabilir:

Dizi	Dizi Listesi

Dizi bildirimi sırasında boyut verilmelidir. Dize [] ad = yeni Dize [2]	Boyut gerekli olmayabilir. Boyutu dinamik olarak değiştirir. ArrayList name = new ArrayList
Bir nesneyi diziye yerleştirmek için dizini belirtmemiz gerekir. name [1] = 'kitap'	Endeks gerekmez. name.add ('kitap')
Dizi parametreleştirilmiş tür değil	Java 5.0'daki ArrayList parametreleştirilmiştir. Örneğin: Bu açılı ayraç, String listesi anlamına gelen bir tür parametresidir.

S # 18) String, String Builder ve String Buffer arasındaki fark.

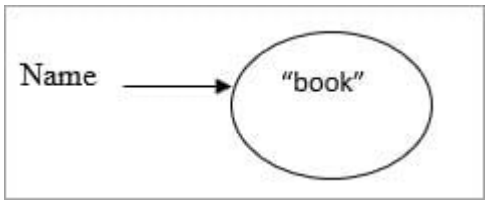
Cevap:

Dize: Dize değişkenleri 'sabit dizgi havuzunda' saklanır. Dize referansı, “sabit dizgi havuzunda” bulunan eski değeri değiştirdiğinde silinemez.

Misal:

Dize adı = 'kitap';

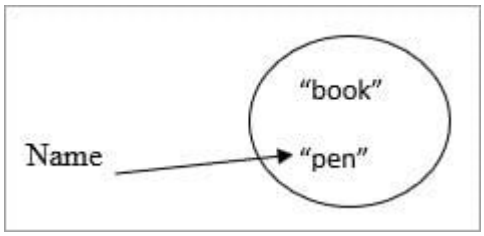
Sabit dize havuzu



.

Ad-değeri 'kitap' dan 'kalem' e değişmişse.

Sabit dize havuzu



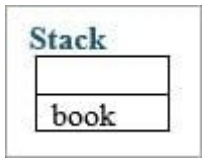
Daha sonra eski değer sabit dizgi havuzunda kalır.

Dize Arabelleği:

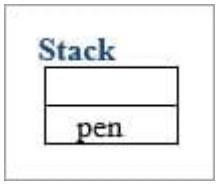
- Burada dize değerleri bir yığın halinde saklanır. Değerler değiştirilirse, yeni değer eski değerin yerini alır.
- Dize tamponu senkronize edilir ve iş parçacığı için güvenlidir.
- Performans, String Builder'dan daha yavaştır.

Misal:

Dize Arabelleği adı = ”kitap”;



Ad değeri 'kalem' olarak değiştirildiğinde, yığındaki 'kitap' silinir.



Dize Oluşturucu:

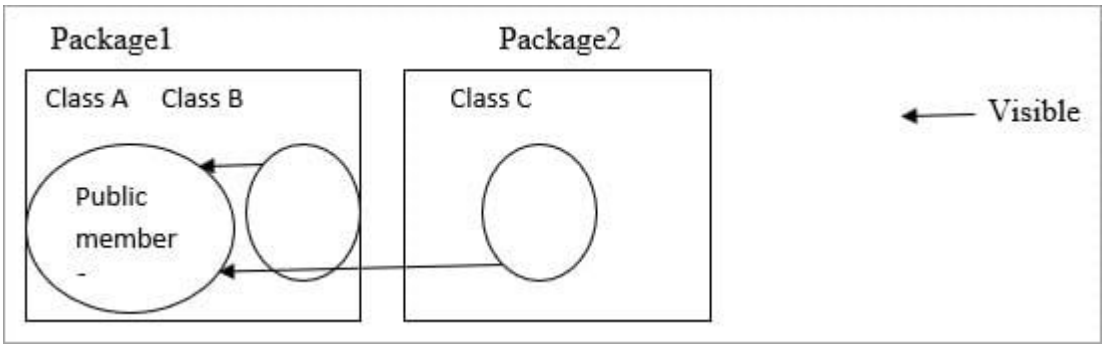
Bu, senkronize edilmeyen ve güvenli bir şekilde iş parçacığı oluşturmayan Dize Oluşturucu haricinde Dize Arabelleği ile aynıdır. Yani açıkçası performans hızlı.

S # 19) Genel ve Özel erişim belirleyicileri açıklayın.

Cevap: Yöntemler ve örnek değişkenleri üyeler olarak bilinir.

Halka açık:

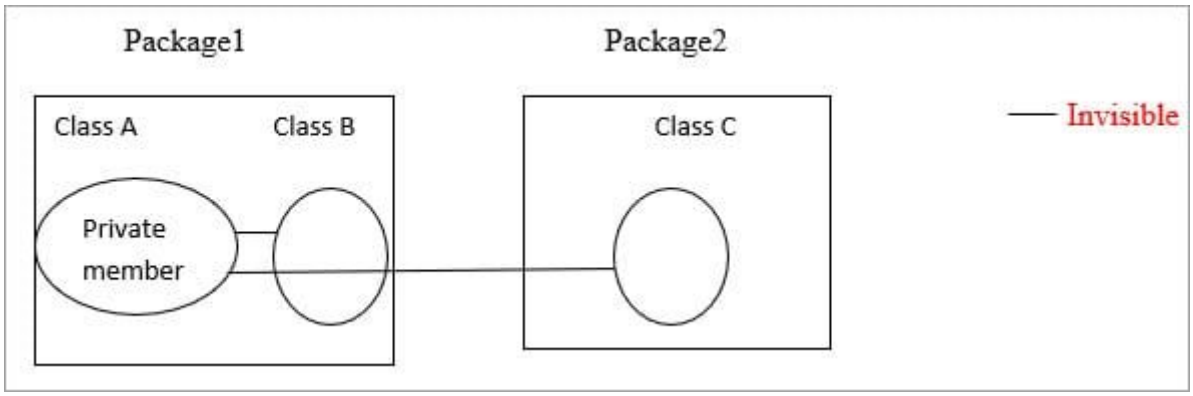
Genel üyeler aynı pakette ve diğer paketler için olan dış pakette görülebilir.



A Sınıfının genel üyeleri, Sınıf B (aynı paket) ve Sınıf C (farklı paketler) tarafından görülebilir.

Özel:

Özel üyeler yalnızca aynı sınıfta görülebilir ve aynı paketteki diğer sınıflar ve dış paketlerdeki sınıflar için görünmez.

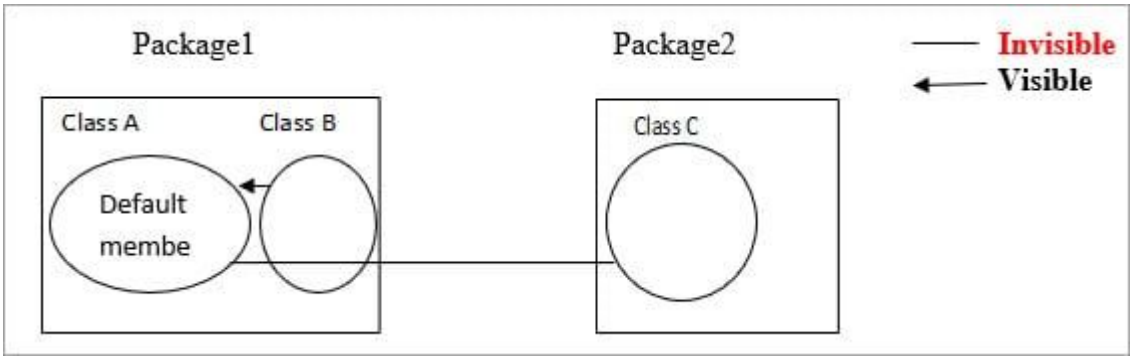


A sınıfındaki özel üyeler yalnızca o sınıfta görülebilir. Hem B sınıfı hem de C sınıfı için görünmezdir.

S # 20) Varsayılan ve Korumalı erişim belirleyicileri arasındaki fark.

Cevap:

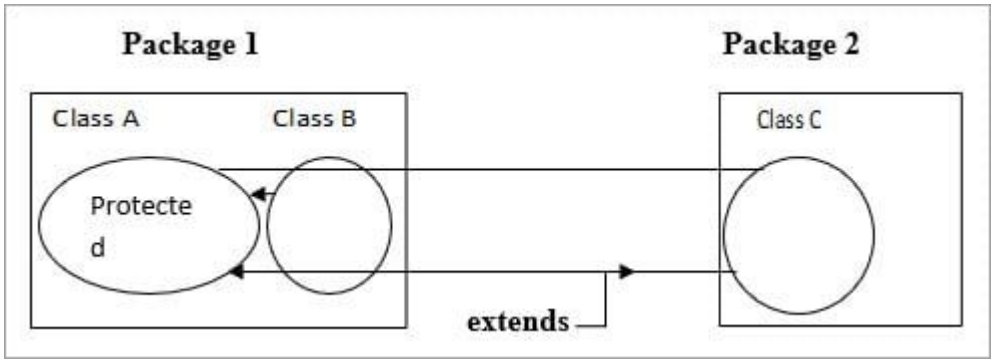
Varsayılan: Herhangi bir erişim belirticisi olmadan bir sınıfta bildirilen yöntemler ve değişkenler varsayılan olarak adlandırılır.



A Sınıfındaki varsayılan üyeler, paketin içindeki ve paketin dışındaki sınıflar tarafından görünmeyen diğer sınıflar tarafından görülebilir.

Yani A Sınıfı üyeler B Sınıfı tarafından görülebilir ve C Sınıfı tarafından görünmez.

Korumalı:



.

Korumalı, Varsayılan ile aynıdır, ancak bir sınıf genişlerse, paketin dışında olsa bile görünür.

A Sınıfı üyeleri, paketin içinde olduğu için B Sınıfı tarafından görülebilir. C Sınıfı için görünmezdir, ancak C Sınıfı A Sınıfını genişletirse, üyeler paketin dışında olsa bile C Sınıfı tarafından görülebilir.

S # 21) HashMap ve HashTable arasındaki fark.

Cevap: HashMap ve HashTable arasındaki fark aşağıda görülebilir:

HashMap	HashTable
Yöntemler senkronize değil	Anahtar yöntemler senkronize edilir
İplik güvenliği değil	İplik güvenliği
Yineleyici, değerleri yinelemek için kullanılır	Numaralandırıcı, değerleri yinelemek için kullanılır
Bir boş anahtara ve birden çok boş değere izin verir	Boş olan hiçbir şeye izin vermez
Performans HashTable'dan yüksektir	Performans yavaş

S # 22) HashSet ve TreeSet arasındaki fark.

Cevap: HashSet ve TreeSet arasındaki fark aşağıda görülebilir:

HashSet	Ağaç Kümesi
Eklenen öğeler rastgele sıradadır	Öğeleri sıralı düzende tutar
Boş nesneleri depolayabilir	Boş nesneler saklanamadı
Performans hızlıdır	Performans yavaş

S # 23) Soyut sınıf ve Arayüz arasındaki fark.

Cevap: Soyut Sınıf ile Arayüz arasındaki farklar aşağıdaki gibidir:

Soyut Sınıf:

- Abstract sınıflarının varsayılan bir kurucusu vardır ve somut alt sınıfın somutlaştırıldığı her seferinde çağrılır.
- Soyut yöntemler ile Soyut Olmayan yöntemler içerir.
- Abstract sınıfını genişleten sınıf, tüm yöntemlerin uygulanmasını gerektirmemelidir, sadece soyut yöntemlerin somut alt sınıfta uygulanması gerekir.
- Soyut sınıf, örnek değişkenleri içerir.

Arayüz:

- Herhangi bir kurucusu yoktur ve somutlaştırılamaz.
- Tek başına soyut yöntem beyan edilmelidir.
- Arabirimi uygulayan sınıflar, tüm yöntemler için uygulama sağlamalıdır.
- Arayüz yalnızca sabitleri içerir.

Q # 24) Java'da Koleksiyonların anlamı nedir?

Cevap: Koleksiyon, nesneleri depolamak ve nesneleri depolamak için tasarımı manipüle etmek için tasarlanmış bir çerçevedir.

Koleksiyonlar, aşağıdaki işlemleri gerçekleştirmek için kullanılır:

- Aranıyor
- Sıralama
- Manipülasyon
- Yerleştirme
- Silme

Bir grup nesne koleksiyon olarak bilinir. Toplama için tüm sınıflar ve arayüzler Java kullanım paketinde mevcuttur.

S # 25) Koleksiyonlarda bulunan tüm Sınıflar ve Arayüzler nelerdir?

Cevap: Aşağıda, Koleksiyonlarda bulunan Sınıflar ve Arayüzler verilmiştir:

Arayüzler:

- Toplamak
- Liste
- Ayarlamak
- Harita
- Sıralanmış Küme
- Sıralanmış Harita
- Kuyruk

Sınıflar:

- Listeler:
- Dizi Listesi
- Vektör
- Bağlantılı liste

Setler:

- Karma kümesi
- Bağlantılı Karma Kümesi
- Ağaç Seti

Haritalar:

- Hash Haritası
- Hash Tablosu
- Ağaç Haritası
- Bağlantılı Hashed Harita

Sıra:

Öncelik Kuyruğu

S # 26) Koleksiyonlarda Sıralı ve Sıralı ifadesi ne anlama geliyor?

Cevap:

Sipariş verildi: Bir koleksiyonda depolanan değerlerin, koleksiyona eklenen değerlere dayandığı anlamına gelir. Böylece koleksiyondaki değerleri belirli bir sırayla yineleyebiliriz.

Sıralandı: Sıralama mekanizmaları dahili veya harici olarak uygulanabilir, böylece belirli bir koleksiyonda sıralanan nesne grubu nesnelerin özelliklerine dayanır.

S # 27) Koleksiyonda bulunan farklı listeleri açıklayın.

Cevap: Listeye eklenen değerler dizin konumuna bağlıdır ve dizin konumuna göre sıralanır. Kopyalara izin verilir.

Liste türleri şunlardır:

a) Dizi Listesi:

- Hızlı yineleme ve hızlı Rastgele Erişim.
- Sıralı bir koleksiyondur (dizine göre) ve sıralanmamıştır.
- Rastgele Erişim Arayüzünü uygular.

Misal:

```
public class Fruits{ public static void main (String [ ] args){
ArrayList names=new ArrayList (); names.add (“apple”); names.add
(“cherry”); names.add (“kiwi”); names.add (“banana”); names.add
(“cherry”); System.out.println (names); } }
```

Çıktı:

[Elma, kiraz, kivi, muz, kiraz]

Çıkıştan, Dizi Listesi ekleme sırasını korur ve kopyaları kabul eder. Ama sıralanmamış.

b) Vektör:

Dizi Listesi ile aynıdır.

- Vektör yöntemleri senkronize edilir.

- İplik güvenliği.
- Ayrıca Rastgele Erişimi de uygular.
- İş parçacığı güvenliği genellikle performansın düşmesine neden olur.

Misal:

```
public class Fruit { public static void main (String [ ] args){ Vector names = new Vector ( ); names.add (“cherry”); names.add (“apple”); names.add (“banana”); names.add (“kiwi”); names.add (“apple”); System.out.println (“names”); } }
```

Çıktı:

[kiraz, elma, muz, kivi, elma]

Vector ayrıca ekleme sırasını korur ve kopyaları kabul eder.

c) Bağlantılı Liste:

- Öğeler birbirine iki kez bağlıdır.
- Performans, Dizi listesinden daha yavaştır.
- Eklemek ve silmek için iyi bir seçim.
- Java 5.0'da, peek (), Pool (), Offer () vb. Yaygın kuyruk yöntemlerini destekler.

Misal:

```
public class Fruit { public static void main (String [ ] args){ Linkedlist names = new linkedlist ( ) ; names.add(“banana”); names.add(“cherry”); names.add(“apple”); names.add(“kiwi”); names.add(“banana”); System.out.println (names); } }
```

Çıktı:

[muz, kiraz, elma, kivi, muz]

Ekleme sırasını korur ve kopyaları kabul eder.

S # 28) Bir koleksiyondaki Set ve türlerini açıklayın.

Cevap: Set benzersizliğe önem verir. Yinelemelere izin vermez. Burada, iki nesnenin aynı olup olmadığını belirlemek için 'equals ()' yöntemi kullanılır.

a) Karma Küme:

- Sırasız ve sıralanmamış.
- Değerleri eklemek için nesnenin hash kodunu kullanır.
- Gereksinim 'yineleme yoksa ve siparişi önemsemiyorum' olduğunda bunu kullanın.

Misal:

```
public class Fruit { public static void main (String[ ] args){ HashSet names = new HashSet ( ) ; names.add(“banana”); names.add(“cherry”);
```

```
names.add(“apple”); names.add(“kiwi”); names.add(“banana”);
System.out.println (names); } }
```

Çıktı:

[muz, kiraz, kivi, elma]

Herhangi bir kampanya siparişini takip etmiyor. Kopyalara izin verilmez.

b) Bağlantılı Karma set:

- Karma kümenin sıralı bir sürümü Bağlantılı Karma Kümesi olarak bilinir.
- Tüm öğelerin çift bağlantılı bir listesini tutar.
- Yineleme sırası gerektiğinde bunu kullanın.

Misal:

```
public class Fruit { public static void main (String[ ] args){
LinkedHashSet; names = new LinkedHashSet ( ) ; names.add(“banana”);
names.add(“cherry”); names.add(“apple”); names.add(“kiwi”);
names.add(“banana”); System.out.println (names); } }
```

Çıktı:

[muz, kiraz, elma, kivi]

Sete eklendikleri ekleme sırasını korur. Kopyalara izin verilmez.

c) Ağaç Seti:

- Sıralanmış iki koleksiyondan biridir.
- 'Siyah Oku' ağaç yapısını kullanır ve elemanların artan sırada olacağını garanti eder.
- Karşılaştırılabilir (veya) bir karşılaştırıcı kullanarak yapıcıyla bir ağaç kümesi oluşturabiliriz.

Misal:

```
public class Fruits{ public static void main (String[ ]args) { Treeset
names= new TreeSet( ) ; names.add(“cherry”); names.add(“banana”);
names.add(“apple”); names.add(“kiwi”); names.add(“cherry”);
System.out.println(names); } }
```

Çıktı:

[elma, muz, kiraz, kivi]

TreeSet, öğeleri artan sırada sıralar. Ve kopyalara izin verilmez.

S # 29) Harita ve türleri hakkında açıklama yapın.

Cevap: Harita benzersiz tanımlayıcıyı önemsiyor. Benzersiz bir anahtarı belirli bir değerle eşleyebiliriz. Bu bir anahtar / değer çiftidir. Anahtar temelinde bir değer arayabiliriz. Küme gibi, harita da iki anahtarın aynı mı yoksa farklı mı olduğunu belirlemek için "equals ()" yöntemini kullanır.

Harita aşağıdaki türlerdendir:

a) Karma Harita:

- Sırasız ve sıralanmamış harita.
- Hashmap, siparişi önemsemediğimizde iyi bir seçimdir.
- Bir boş anahtara ve birden çok boş değere izin verir.

Misal:

```
Public class Fruit{ Public static void main(String[ ] args){ HashMap names =new HashMap( ); names.put("key1","cherry"); names.put ("key2","banana"); names.put ("key3","apple"); names.put ("key4","kiwi"); names.put ("key1","cherry"); System.out.println(names); } }
```

Çıktı:

{anahtar2 = muz, anahtar1 = kiraz, anahtar4 = kivi, anahtar3 = elma}

Harita'da yinelenen anahtarlara izin verilmez.

Herhangi bir kampanya siparişi tutmaz ve sıralanmamıştır.

b) Karma Tablosu:

- Vektör anahtarı gibi, sınıfın yöntemleri de senkronize edilir.
- İplik güvenliği ve dolayısıyla performansı yavaşlatır.
- Boş olan hiçbir şeye izin vermez.

Misal:

```
public class Fruit{ public static void main(String[ ]args){ Hashtable names =new Hashtable( ); names.put("key1","cherry"); names.put("key2","apple"); names.put("key3","banana"); names.put("key4","kiwi"); names.put("key2","orange"); System.out.println(names); } }
```

Çıktı:

{anahtar2 = elma, anahtar1 = kiraz, anahtar4 = kivi, anahtar3 = muz}

Yinelenen anahtarlara izin verilmez.

c) Bağlantılı Karma Haritası:

- Kampanya siparişini korur.

- Hash haritasından daha yavaş.
- Daha hızlı bir yineleme bekleyebilirim.

Misal:

```
public class Fruit{ public static void main(String[ ] args){
LinkedHashMap names =new LinkedHashMap( ); names.put(“key1”,“cherry”);
names.put(“key2”,“apple”); names.put(“key3”,“banana”);
names.put(“key4”,“kiwi”); names.put(“key2”,“orange”);
System.out.println(names); } }
```

Çıktı:

{anahtar2 = elma, anahtar1 = kiraz, anahtar4 = kivi, anahtar3 = muz}

Yinelenen anahtarlara izin verilmez.

d) Ağaç Haritası:

- Sıralanmış Harita.
- Ağaç kümesi gibi, kurucu ile bir sıralama düzeni oluşturabiliriz.

Misal:

```
public class Fruit{ public static void main(String[ ]args){ TreeMap
names =new TreeMap( ); names.put(“key1”,“cherry”);
names.put(“key2”,“banana”); names.put(“key3”,“apple”);
names.put(“key4”,“kiwi”); names.put(“key2”,“orange”);
System.out.println(names); } }
```

Çıktı:

{key1 = kiraz, anahtar2 = muz, anahtar3 = elma, anahtar4 = kivi}

Anahtara göre artan sırada sıralanır. Yinelenen anahtarlara izin verilmez.

S # 30) Öncelik Kuyruğunu açıklayın.

Cevap: Sıra Arayüzü

Öncelik Sırası: Bağlantılı liste sınıfı, kuyruk arayüzünü uygulamak için geliştirildi. Kuyruklar bağlantılı bir liste ile ele alınabilir. Bir kuyruğun amacı “Öncelikli, Öncelikli” dir.

Bu nedenle, elemanlar doğal olarak veya karşılaştırıcıya göre sıralanır. Öğelerin sıralaması, göreceli önceliklerini temsil eder.

S # 31) İstisna ile kastedilen nedir?

Cevap: İstisna, normal yürütme akışı sırasında ortaya çıkabilecek bir sorundur. Bir yöntem, çalışma zamanında bir şey ağladığında bir istisna atabilir. Bu istisna ele alınamazsa, görevi tamamlamadan yürütme sona erer.

İstisnayı ele alırsak, normal akış devam eder. İstisnalar, java.lang.Exception'ın bir alt sınıfıdır.

İstisnayı ele alma örneği:

```
try{ //Risky codes are surrounded by this block }catch(Exception e){
//Exceptions are caught in catch block }
```

S # 32) İstisna türleri nelerdir?

Cevap: İki tür İstisna vardır. Aşağıda ayrıntılı olarak açıklanmıştır.

a) Kontrol Edilmiş İstisna:

Bu istisnalar, derleme sırasında derleyici tarafından kontrol edilir. Çalışma Zamanı istisnası ve Hata dışında Throwable sınıfını genişleten sınıflar, kontrol edilen İstisna olarak adlandırılır.

Kontrol Edilmiş İstisnalar, uygun try / catch ile çevrili throws anahtar kelimesini (veya) kullanarak istisnayı bildirmelidir.

Örneğin, ClassNotFound İstisnası

b) Kontrol Edilmemiş İstisna:

Bu istisnalar derleme sırasında derleyici tarafından kontrol edilmez. Derleyici bu istisnaları işlemeye zorlamaz. **O içerir:**

- Aritmetik İstisna
- ArrayIndexOutOfBounds İstisnası

S # 33) İstisnaları ele almanın farklı yolları nelerdir?

Cevap: İstisnaları ele almanın iki farklı yolu aşağıda açıklanmıştır:

a) Try / catch kullanımı:

Riskli kod, try bloğu ile çevrilidir. Bir istisna oluşursa, catch bloğu tarafından yakalanır ve ardından try bloğu gelir.

Misal:

```
class Manipulation{ public static void main(String[] args){ add(); }
Public void add(){ try{ addition(); }catch(Exception e){
e.printStacktrace(); } } }
```

b) throws anahtar kelimesini bildirerek:

Yöntemin sonunda, throws anahtar sözcüğünü kullanarak istisnayı bildirebiliriz.

Misal:

```
class Manipulation{ public static void main(String[] args){ add(); }
public void add() throws Exception{ addition(); } }
```

S # 34) İstisna işlemenin avantajları nelerdir?

Cevap: Avantajlar aşağıdaki gibidir:

- Bir istisna işlenirse yürütmenin normal akışı sona erdirilmez
- Sorunu catch bildirimi kullanarak belirleyebiliriz

S # 35) Java'da istisna işleme anahtar sözcükleri nelerdir?

Cevap: Aşağıda listelenen iki İstisna İşleme Anahtar Kelimesi verilmiştir:

bir deneme:

Riskli bir kod bir try bloğu ile çevrildiğinde. Try bloğunda meydana gelen bir istisna, bir catch bloğu tarafından yakalanır. Try, ya catch (ya da) nihayet (ya da) her ikisi ile takip edilebilir. Ancak bloklardan herhangi biri zorunludur.

b) yakalamak:

Bunu bir deneme bloğu izler. İstisnalar burada yakalanmıştır.

c) nihayet:

> mysql vs sql sunucusu vs oracle

Bunu ya try bloğu (ya da) catch bloğu takip eder. Bu blok, istisnadan bağımsız olarak yürütülür. Bu nedenle, genel olarak temizleme kodları burada verilmiştir.

S # 36) İstisna Yayılımını açıklayın.

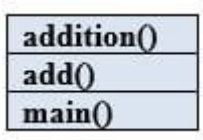
Cevap: İstisna ilk olarak yığının tepesindeki yöntemden atılır. Yakalanmazsa, o zaman yöntemi açar ve önceki yönteme geçer ve elde edilene kadar bu şekilde devam eder.

Buna İstisna yayılımı denir.

Misal:

```
public class Manipulation{ public static void main(String[] args){
add(); } public void add(){ addition(); }
```

Yukarıdaki örnekten, yığın aşağıda gösterildiği gibi görünür:



Bir istisna meydana gelirse **ilave()** yöntem yakalanmaz, sonra yönteme geçer **Ekle()** . Daha sonra **ana()** yöntemi ve ardından yürütme akışını durduracaktır. İstisna Yayılımı olarak adlandırılır.

S # 37) Java'daki son anahtar kelime nedir?

Cevap:

Değişken son: Bir değişken nihai olarak bildirildiğinde, değişkenin değeri değiştirilemez. Sabit gibidir.

Misal:

final int = 12;

Son yöntem: Bir yöntemdeki son bir anahtar kelime geçersiz kılınamaz. Bir yöntem son olarak işaretlenmişse, alt sınıf tarafından geçersiz kılınamaz.

Son sınıf: Bir sınıf nihai olarak bildirilirse, o zaman sınıf alt sınıflara alınamaz. Son sınıfı hiçbir sınıf uzatamaz.

S # 38) Konu nedir?

Cevap: Java'da, yürütme akışına Thread adı verilir. Her java programında ana iş parçacığı adı verilen en az bir iş parçacığı vardır, ana iş parçacığı JVM tarafından oluşturulur. Kullanıcı, Runnable arabirimini uygulayarak Thread sınıfını (veya) genişleterek kendi iş parçacıklarını tanımlayabilir. Konu eşzamanlı olarak yürütülür.

Misal:

```
public static void main(String[] args){//main thread starts here }
```

S # 39) Java'da nasıl iş parçacığı oluşturursunuz?

Cevap: İplik oluşturmanın iki yolu vardır.

a) Thread sınıfını genişletin: Bir Thread sınıfını genişletmek ve run yöntemini geçersiz kılmak. Konu java.lang.thread'de mevcuttur.

Misal:

```
Public class Addition extends Thread { public void run () { } }
```

Bir iş parçacığı sınıfı kullanmanın dezavantajı, evre sınıfını zaten genişlettiğimiz için başka sınıfları genişletememizdir. Sınıfımızdaki run () yöntemini aşırı yükleyebiliriz.

b) Çalıştırılabilir arayüzü uygulayın: Başka bir yol da çalıştırılabilir arayüzün uygulanmasıdır. Bunun için arayüzde tanımlanan run () metodunun uygulamasını sağlamalıyız.

Misal:

```
Public class Addition implements Runnable { public void run () { } }
```

S # 40) join () yöntemini açıklayın.

Cevap: Join () yöntemi, şu anda çalışan iş parçacığının sonuyla bir iş parçacığını birleştirmek için kullanılır.

Misal:

```
public static void main (String[] args){ Thread t = new Thread (); t.start (); t.join (); }
```

Yukarıdaki koda göre, ana iş parçacığı yürütmeyi başlattı. Koda ulaştığında ***t.start ()*** daha sonra 'thread t' yürütme için kendi yığını başlatır. JVM, ana iş parçacığı ve 'iş parçacığı t' arasında geçiş yapar.

Koda ulaştığında ***t.join ()*** daha sonra tek başına 'thread t' çalıştırılır ve görevini tamamlar, sonra sadece ana iş parçacığı yürütmeye başlar.

Statik olmayan bir yöntemdir. Join () yönteminin aşırı yüklenmiş bir sürümü var. Dolayısıyla join () yönteminde zaman süresinden ".s" de bahsedebiliriz.

S # 41) Thread sınıfının getiri yöntemi ne işe yarar?

Cevap: Bir verim () yöntemi, o anda çalışan iş parçacığını çalıştırılabilir bir duruma taşır ve diğer iş parçacıklarının yürütülmesine izin verir. Böylece eşit önceliğe sahip iş parçacıklarının çalışma şansı olur. Statik bir yöntemdir. Herhangi bir kilidi açmaz.

Yield () yöntemi, iş parçacığını yalnızca Runnable durumuna geri taşır, iş parçacığını sleep (), wait () (veya) bloğunu değil.

Misal:

```
public static void main (String[] args){ Thread t = new Thread (); t.start (); } public void run(){ Thread.yield(); } }
```

S # 42) wait () yöntemini açıklayın.

Cevap: bekle () yöntemi, iş parçacığının bekleme havuzunda beklemesini sağlamak için kullanılır. Bir iş parçacığı yürütme sırasında wait () yöntemi yürütüldüğünde, iş parçacığı nesnedeki kilidi hemen bırakır ve bekleme havuzuna gider. Wait () yöntemi, iş parçacığına belirli bir süre beklemesini söyler.

Daha sonra iş parçacığı, notify () (veya) all () yöntemi çağrıldıktan sonra uyanacaktır.

Wait () ve yukarıda belirtilen diğer yöntemler, o anda yürütülmekte olan iş parçacığı eşitlenmiş kodu tamamlayana kadar nesneye hemen kilit vermez. Çoğunlukla senkronizasyonda kullanılır.

Misal:

```
public static void main (String[] args){ Thread t = new Thread (); t.start (); Synchronized (t) { Wait(); } }
```

S # 43) Java'da notify () yöntemi ile notifyAll () yöntemi arasındaki fark.

Yanıt: notify () yöntemi ile notifyAll () yöntemi arasındaki farklar aşağıda listelenmiştir:

bildir ()	notifyAll ()
Bu yöntem, bekleme havuzundaki tek bir iş parçacığını uyandırmak için bir sinyal göndermek için kullanılır.	Bu yöntem, bekleyen bir makaradaki tüm iş parçacıkları uyandırmak için sinyal gönderir.

S # 44) Java'da bir iş parçacığı nasıl durdurulur? Bir ileti dizisindeki sleep () yöntemini açıklayın?

Cevap: Aşağıdaki iş parçacığı yöntemlerini kullanarak bir iş parçacığını durdurabiliriz:

- Uyuyor
- Bekliyorum
- Engellendi

Uyku: Sleep () yöntemi, o anda yürütülen iş parçacığını belirli bir süre boyunca uyutmak için kullanılır. İplik uyandığında, çalıştırılabilir duruma geçebilir. Dolayısıyla, yürütmeyi bir süre geciktirmek için sleep () yöntemi kullanılır.

Statik bir yöntemdir.

Misal:

Konu. Uyku (2000)

Böylece iş parçacığının 2 milisaniye uyumasını geciktirir. Sleep () yöntemi kesintisiz bir istisna atar, bu nedenle bloğu try / catch ile çevrelememiz gerekir.

```
public class ExampleThread implements Runnable{ public static void main (String[] args){ Thread t = new Thread (); t.start (); } public void run(){ try{ Thread.sleep(2000); }catch(InterruptedException e){ } }
```

S # 45) Java'da Runnable arayüzü Vs Thread sınıfı ne zaman kullanılır?

Cevap: Sınıfımızın iş parçacığı dışında başka sınıfları genişletmesine ihtiyacımız varsa, çalıştırılabilir arayüzle gidebiliriz çünkü java'da yalnızca bir sınıfı genişletebiliriz.

Herhangi bir sınıfı genişletmeyeceksek, thread sınıfını genişletebiliriz.

S # 46) İş parçacığı sınıfının start () ve run () yöntemi arasındaki fark.

Cevap: Start () yöntemi yeni bir iş parçacığı oluşturur ve run () yönteminin içindeki kod yeni iş parçacığında yürütülür. Doğrudan run () yöntemini çağırırsak, yeni bir iş parçacığı oluşturulmaz ve şu anda çalıştırılan iş parçacığı run () yöntemini yürütmeye devam eder.

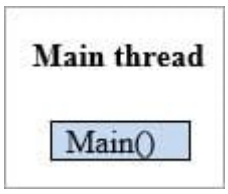
S # 47) Çoklu iş parçacığı nedir?

Cevap: Aynı anda birden fazla iş parçacığı yürütülür. Her iş parçacığı, iş parçacıklarının akış (veya) önceliğine göre kendi yığınını başlatır.

Örnek Program:

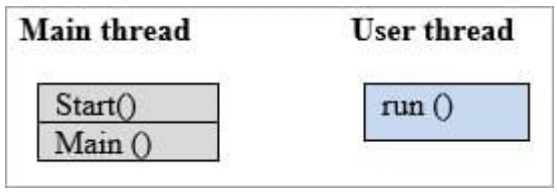
```
public class MultipleThreads implements Runnable { public static void main (String[] args){//Main thread starts here Runnable r = new runnable (); Thread t=new thread (); t.start ();//User thread starts here Addition add=new addition (); } public void run(){ go(); }//User thread ends here }
```

1. satır yürütmede, JVM ana yöntemi çağırır ve ana iş parçacığı yığını aşağıda gösterildiği gibi görünür.

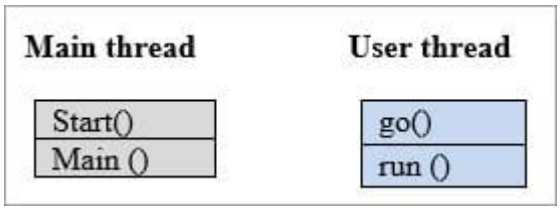


İnfaz ulaştığında, **t.start ()** satır sonra yeni bir iş parçacığı oluşturulur ve iş parçacığı için yeni yığın da oluşturulur. Şimdi JVM yeni iş parçacığına geçer ve ana iş parçacığı çalıştırılabilir duruma geri döner.

İki yığın aşağıda gösterildiği gibi görünür.



Şimdi, kullanıcı iş parçacığı run () yöntemi içindeki kodu çalıştırdı.



Run () yöntemi tamamlandıktan sonra, JVM ana iş parçacığına geri döner ve kullanıcı iş parçacığı görevi tamamladı ve yığın kayboldu.

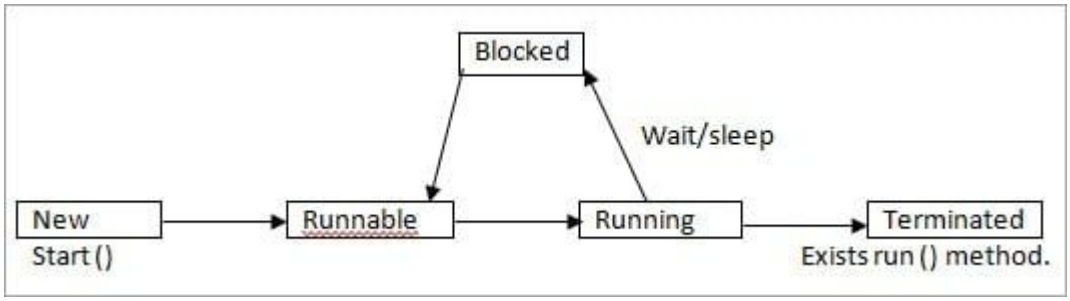
JVM, her iki iş parçacığı tamamlanana kadar her iş parçacığı arasında geçiş yapar. Bu, Çoklu iş parçacığı olarak adlandırılır.

S # 48) Java'daki iş parçacığı yaşam döngüsünü açıklayın.

Cevap: İş parçacığı aşağıdaki durumlara sahiptir:

- Yeni
- Runnable

- Koşu
- Çalıştırılamaz (Engellendi)
- Sonlandırılmış



- **Yeni:** Yeni durumunda, bir Thread örneği oluşturuldu, ancak start () yöntemi henüz çağrılmadı. Şimdi ipin canlı olduğu kabul edilmiyor.
- **Runnable** : İş parçacığı, start () yönteminin çağrılmasından sonra, ancak run () yöntemi çağrılmadan önce çalıştırılabilir durumdadır. Ancak bir iş parçacığı bekleme / uyku durumundan da çalıştırılabilir duruma dönebilir. Bu durumda, iplik canlı kabul edilir.
- **Koşu** : İş parçacığı, run () yöntemini çağırdıktan sonra çalışır durumda. Şimdi iş parçacığı yürütmeye başlar.
- **Çalıştırılamaz** (Engellendi): İleti dizisi canlı ancak çalıştırılmaya uygun değil. Çalıştırılabilir durumda değil, aynı zamanda bir süre sonra çalıştırılabilir duruma geri dönecektir. **Misal:** bekle, uyu, engelle.
- **Sonlandırılmış** : Çalıştırma yöntemi tamamlandıktan sonra sonlandırılır. Şimdi konu canlı değil.

S # 49) Senkronizasyon nedir?

Cevap: Senkronizasyon, bir seferde bir kod bloğuna erişmek için yalnızca bir iş parçacığı oluşturur. Birden fazla iş parçacığı kod bloğuna erişirse, sonunda yanlış sonuçların olma ihtimali vardır. Bu sorunu önlemek için, hassas kod bloğu için senkronizasyon sağlayabiliriz.

Senkronize anahtar kelime, bir iş parçacığının senkronize edilmiş koda erişmek için bir anahtara ihtiyaç duyduğu anlamına gelir.

Kilitler nesne başınadır. Her Java nesnesinin bir kilidi vardır. Bir kilidin yalnızca bir anahtarı vardır. Bir iş parçacığı senkronize edilmiş bir yönteme ancak iş parçacığı kilitlenecek nesnelerin anahtarını alabiliyorsa erişebilir.

Bunun için 'Senkronize' anahtar kelimesini kullanıyoruz.

Misal:

```
public class ExampleThread implements Runnable{ public static void main (String[] args){ Thread t = new Thread (); t.start (); } public void run(){ synchronized(object){ { } }
```

S # 50) Senkronizasyonun dezavantajı nedir?

Yıl: Tüm yöntemlerin uygulanması için senkronizasyon önerilmez. Çünkü eğer bir iş parçacığı eşitlenmiş koda erişirse, sonraki iş parçacığı beklemek zorundadır. Yani diğer tarafta yavaş bir performans sergiliyor.

S # 51) Serileştirme ile kastedilen nedir?

Cevap: Bir dosyayı bayt akışına dönüştürmek Serileştirme olarak bilinir. Dosyadaki nesneler, güvenlik amacıyla bayta dönüştürülür. Bunun için bir java.io.Serializable arayüz uygulamamız gerekiyor. Tanımlayacak bir yöntemi yoktur.

Geçici olarak işaretlenen değişkenler serileştirmenin bir parçası olmayacaktır. Böylece, geçici bir anahtar kelime kullanarak dosyadaki değişkenler için serileştirmeyi atlayabiliriz.

Daha Fazla Bilgi Edinin = >> Serileştirilebilir ve Klonlanabilir

S # 52) Bir geçici değişkenin amacı nedir?

Cevap: Geçici değişkenler, serileştirme sürecinin bir parçası değildir. Seriyi kaldırma sırasında, geçici değişkenlerin değerleri varsayılan değere ayarlanır. Statik değişkenlerle kullanılmaz.

Misal:

geçici int sayıları;

S # 53) Serileştirme ve Seriyi Kaldırma işlemi sırasında hangi yöntemler kullanılır?

Cevap: ObjectOutputStream ve ObjectInputStream sınıfları daha yüksek seviyeli java.io. paketi. Bunları FileOutputStream ve FileInputStream alt seviye sınıfları ile kullanacağız.

ObjectOutputStream.writeObject **—->** Nesneyi serileştirin ve serileştirilmiş nesneyi bir dosyaya yazın.

ObjectInputStream.readObject **->** Dosyayı okur ve nesnenin serisini kaldırır.

Bir nesnenin serileştirilmesi için serileştirilebilir arabirimi uygulaması gerekir. Üst sınıf Serializable'ı uygularsa, alt sınıf otomatik olarak serileştirilebilir olacaktır.

S # 54) Uçucu Değişkenin amacı nedir?

Cevap: Uçucu değişken değerleri her zaman ana bellekten okunur, iş parçacığının önbelleğinden değil. Bu, esas olarak senkronizasyon sırasında kullanılır. Yalnızca değişkenler için geçerlidir.

Misal:

uçucu int sayısı;

S # 55) Java'da Serileştirme ve Serileştirme Arasındaki Fark.

Cevap: Java'da serileştirme ve seriyi kaldırma arasındaki farklar şunlardır:

Serileştirme	Seriyi kaldırma
Serileştirme, nesneleri bayt akışına dönüştürmek için kullanılan işlemdir.	Seriyi kaldırma, nesneleri bayt akışından geri alabildiğimiz serileştirmenin tam tersidir.
Bir nesne, bir ObjectOutputStream yazılarak serileştirilir.	Bir nesne, bir ObjectInputStream'den okunarak seri durumdan çıkarılır.

S # 56) SerialVersionUID nedir?

Cevap: Bir nesne Serileştirildiğinde, nesne, nesne sınıfı için bir sürüm kimlik numarasıyla damgalanır. Bu kimliğe SerialVersionUID adı verilir. Bu, gönderen ve alıcının Serileştirme ile uyumlu olduğunu doğrulamak için seriyi kaldırma sırasında kullanılır.

Sonuç

Bunlar, programlama için hem temel hem de gelişmiş Java kavramlarını ve geliştirici görüşmelerini kapsayan temel JAVA mülakat sorularından bazılarıdır ve bunlar, JAVA uzmanlarımız tarafından cevaplanan sorulardır.

Umarım bu eğitim, size JAVA çekirdek kodlama kavramları hakkında ayrıntılı olarak harika bir fikir verir. Yukarıda verilen açıklamalar bilginizi gerçekten zenginleştirecek ve JAVA programlama anlayışınızı artıracaktır.

Kendinizden emin bir şekilde bir JAVA röportajı yapmaya hazır olun.

Önerilen Kaynaklar