Java Mülakatlara Hazırlık Soruları 4

1. Java'da kilit (lock) veya kilitlerin amacı nedir?

Kilit (lock), Java'da paylaşılan bir kaynağa birden fazla iş parçacığının erişmesini önlemek için kullanılır. Bir kilidin amacı, paylaşılan bir kaynağa erişmek isteyen bir iş parçacığının, öncelikle o kaynağın kilidini alması ve bu kaynağa erişmek için izin almasıdır. Eğer başka bir iş parçacığı tarafından zaten kilit alınmışsa, o zaman diğer iş parçacığı, bu kaynağa erişmek için beklemek zorunda kalır ve kilidin serbest bırakılmasını bekler.

Java'da kilit, synchronized anahtar kelimesi veya Lock arabirimini kullanarak elde edilir. Bir nesneyi kilitlemek için synchronized anahtar kelimesi kullanılır ve bu, ilgili kod bloğunun sadece bir iş parçacığı tarafından aynı anda yürütülmesine izin verir.

Kilitler, kodun belirli bölümlerini koruyarak, sadece bir iş parçacığının aynı anda yürütebileceği şekilde düzenlenir.
Örneğin:

```
public class SharedResource {
  private int count = 0;

  public synchronized void increment() {
    count++;
  }

  public synchronized int getCount() {
    return count;
  }
}
```

Yukarıdaki örnekte, increment ve getCount metodlarına synchronized anahtar kelimesi uygulanmıştır. Bu, bu metotların aynı anda yalnızca bir iş parçacığı tarafından erişilmesini sağlar ve böylece count değişkenine güvenli bir şekilde erişilir.

2. Java'da senkronizasyon (synchronization) yapmanın kaç yolu vardır?

Java'da senkronizasyon yapmanın iki temel yolu vardır:
1. Senkronize metotlar (Synchronized methods)
2. Senkronize bloklar (Synchronized blocks)

Senkronizasyon, aynı anda birden fazla iş parçacığının aynı kod bloğuna veya metoda erişmesini engellemek için kullanılır. Senkronizasyon, paylaşılan kaynaklara güvenli bir şekilde erişimi sağlar ve veri bütünlüğünü korur.

3. Java'da Senkronize metotlar (synchronized methods) nedir?

Senkronize metotlar, Java'da bir nesnenin yönteminin aynı anda yalnızca bir iş parçacığı tarafından erişilmesini sağlamak için kullanılır. Bu, paylaşılan kaynaklara güvenli ve koordineli erişim sağlamak için kullanılır. Senkronize bir metot tanımlamak için synchronized anahtar kelimesi kullanılır.

Senkronize bir metot tanımlamak için kullanılan imza şu şekildedir:

```
public synchronized void methodName() {
   // Metot içeriği
}
```

Senkronize bir metot çağrıldığında, ilgili nesnenin üzerinde otomatik olarak bir kilidin alınması gerekir. Başka bir deyişle, senkronize bir metot çağrıldığında, o metot üzerindeki kilidin otomatik olarak alınması ve metotun yalnızca tek bir iş parçacığı tarafından aynı anda yürütülmesi sağlanır.

Senkronize bir metot, yalnızca o metoda ilişkin kilidin serbest olduğu durumda çalışabilir. Eğer bir başka iş parçacığı tarafından o metoda ilişkin kilidin alındığı durumda, diğer iş parçacığı o metoda erişmek için beklemek zorunda kalır. Böylece, senkronize metotlar paylaşılan kaynaklara güvenli bir şekilde erişim sağlar ve veri bütünlüğünü korur.

4. Java'da Senkronize metotları ne zaman kullanırız?

Java'da senkronize metotlar, birden fazla iş parçacığının aynı nesnenin yöntemine erişmeye çalıştığı durumlarda kullanılır. Özellikle, bir yöntem nesnenin durumunu değiştirebiliyorsa ve bu durumun tutarlılığının korunması gerekiyorsa senkronizasyon kullanılır.

Örneğin, bir nesnenin belirli bir yöntemi, nesnenin iç durumunu değiştiriyorsa ve birden fazla iş parçacığı bu yönteme aynı anda erişmeye çalışıyorsa, bu durumda senkronize metotlar kullanılabilir. Senkronize bir metot, sadece bir iş parçacığının aynı anda o metoda erişmesine izin verir, böylece nesnenin durumu tutarlı kalır ve beklenmeyen sonuçlar oluşmaz.

Özetle, senkronize metotlar, bir nesnenin iç durumunu güvenli bir şekilde değiştirmek ve veri bütünlüğünü korumak için kullanılır. Eğer birden fazla iş parçacığı aynı anda bu yönteme erişmeye çalışıyorsa, senkronize bir metot kullanarak bu durumu güvenli bir şekilde ele alabiliriz.

5. Java'da bir thread senkronize yöntemleri yürütürken, diğerlerini yürütmek mümkün müdür?

Bir thread bir senkronize metodu çalıştırırken, o metot içindeyken diğer threadlerin aynı nesnenin başka senkronize metotlarını aynı anda çalıştırması mümkün değildir. Bu durumda, bir threadin senkronize bir metodu çalıştırması, o nesnenin üzerinde kilidi alır ve bu nesnenin başka senkronize metotlarına diğer threadlerin erişimini engeller.

Senkronize metotlar, bir nesnenin belirli bir kısmını tek bir thread tarafından aynı anda işlenmesini sağlamak için kullanılır. Dolayısıyla, bir threadin bir senkronize metodu çalıştırırken, o nesnenin diğer senkronize metotlarını başka threadlerin aynı anda çalıştırmasına izin verilmez.

Bu mekanizma, veri bütünlüğünü sağlamak ve senkronizasyonu yönetmek için kullanılır. Eğer birden fazla thread aynı nesneyi değiştirmeye çalışıyorsa, senkronizasyon olmadan beklenmedik sonuçlar oluşabilir. Bu nedenle, senkronize metotlar ve diğer senkronizasyon mekanizmaları, paylaşılan kaynaklara güvenli bir şekilde erişmek için önemlidir.

6. Java'da senkronize bloklar nelerdir?

Java'da senkronize bloklar, sadece belirli kod satırlarını veya kod bloklarını senkronize etmek için kullanılır. Bununla birlikte, tüm bir metodu senkronize etmek yerine, yalnızca belirli bir kod parçasını senkronize etmek istediğimiz durumlarda kullanılırlar.

Senkronize bir blok tanımlamak için synchronized anahtar kelimesi kullanılır. Senkronize blokun içine girildiğinde, belirtilen nesnenin kilidi alınır ve bu nesne üzerindeki diğer senkronize bloklara ve

senkronize metotlara erişim engellenir. Senkronize bloklar, aynı anda yalnızca bir iş parçacığının belirli kod bloğuna erişmesini sağlar.

Senkronize bir bloğun imzası şu şekildedir:

```
synchronized (objectReference) {
   // Senkronize edilecek kod bloğu
}
```

objectReference, senkronize bloğun kilidini alacak nesneyi belirtir. Bu nesne, senkronize edilen kod bloğuna aynı anda erişmeye çalışan tüm iş parçacıkları için bir referans noktası sağlar. Genellikle, bu nesne, paylaşılan bir kaynağı temsil eden bir nesnedir.

7. Senkronize blokları ne zaman kullanırız ve senkronize blok kullanmanın avantajları nelerdir?

Eğer çok az kod satırı senkronize edilmesi gerekiyorsa, o zaman senkronize blokların kullanılması önerilir. Senkronize blokların senkronize metotlara göre başlıca avantajı, bekleyen iş parçacıklarının bekleme süresini azaltması ve sistem performansını artırmasıdır.

Senkronize bloklar, yalnızca belirli bir kod bloğunu senkronize etmek için kullanılır. Bu, senkronizasyon gerektiren kodun sadece küçük bir kısmının senkronize edilmesine olanak tanır. Bu durum, senkronize blokların senkronize metotlara göre daha esnek olmasını sağlar.

Senkronize blokların avantajları şunlardır:

- Esneklik: Senkronize bloklar, sadece belirli kod parçalarını senkronize etmemize olanak tanır. Böylece, senkronizasyonu yalnızca ihtiyaç duyulan yerlerde uygulayabiliriz.

- Daha Az Bekleme Süresi: Senkronize bloklar, senkronize metotlara göre daha küçük kilitlenme alanlarına sahiptir. Bu, diğer iş parçacıklarının belirli bir bloğa erişimini beklerken daha az bekleme süresi geçirilmesini sağlar.

- Performans Artışı: Senkronize bloklar, senkronize edilmesi gereken kod bloğunun tamamını senkronize etmek yerine, yalnızca belirli bir bölümünü senkronize eder. Bu, iş parçacıklarının daha az kilitlenme durumu yaşamasını sağlar ve böylece genel performansı artırır.

Özet olarak, senkronize bloklar, senkronizasyon gereksinimlerini daha hassas bir şekilde kontrol etmek ve performansı artırmak için kullanılır. Bu nedenle, çok az kodun senkronize edilmesi gerektiğinde veya senkronizasyonun sadece belirli bir kod bloğunda uygulanması gerektiğinde senkronize bloklar tercih edilir.

8. Java'da class düzeyi kilit nedir?

Sınıf seviyesi kilidi (class level lock), bir sınıfın örneği yerine sınıfın kendisi üzerinde kilidi almayı ifade eder. Sınıf seviyesi kilidi, bir sınıfın tüm örneklerine uygulanır ve bu nedenle sınıfın tüm örneklerine erişen threadlerin senkronize edilmesini sağlar.

Sınıf seviyesi kilidi ile nesne seviyesi kilidi arasındaki fark şudur: Sınıf seviyesi kilidi, sınıfın kendisi üzerinde alınırken, nesne seviyesi kilidi, sınıfın belirli bir örneği üzerinde alınır.

Örneğin, bir sınıf seviyesi kilidi aşağıdaki gibi tanımlanabilir:

```
public class MyClass {
   public static synchronized void myStaticMethod() {
      // Class level lock, sınıfın tüm örneklerine uygulanır
   }
}
```
Burada, myStaticMethod metodu static olarak tanımlanmış ve synchronized anahtar kelimesiyle işaretlenmiştir. Bu nedenle, bu metoda erişen tüm iş parçacıkları, sınıf seviyesi kilidi üzerinde senkronize edilirler.

Sınıf seviyesi kilidi, genellikle paylaşılan kaynaklara erişim sağlamak ve veri bütünlüğünü korumak için kullanılır. Bununla birlikte, sınıf seviyesi kilidinin kullanımı, gereksinimlere ve uygulamanın doğasına bağlı olarak değişir.

9. Java'daki statik metodları senkronize edebilir miyiz?

Java'da static metotları senkronize edebiliriz. Her sınıfın bir kilidi (lock) vardır ve bir thread, statik bir senkronize yöntemi çalıştırmak istiyorsa önce sınıf seviyesindeki kilidi elde etmelidir. Bir thread, statik bir senkronize yöntemini yürütürken, diğer bir thread aynı sınıfın statik senkronize yöntemini çalıştıramaz çünkü kilidin sınıf üzerinde alınmış olması gereklidir.

Ancak, aynı anda aşağıdaki yöntemlerin çalıştırılmasına izin verilir:
Normal static methods
Normal instance methods
Senkronize instance methods
Bir sınıf seviyesi kilidi almak için kullanılan imza şu şekildedir:

```
synchronized(ClassAdı.class) {
   // Kod bloğu
}
```
Bu yapı, sınıfın kendisi üzerinde bir kilidin alınmasını sağlar. Bu, sınıfın tüm örneklerine aynı anda erişimi kontrol etmek için kullanılır ve sınıf seviyesinde senkronizasyon sağlar.

10. Primitive için senkronize bloğu kullanabilir miyiz?

Java'da senkronize bloklar yalnızca nesneler için geçerlidir. Senkronize bloklar, belirli bir nesne üzerinde senkronize edilir ve bu nesne, senkronizasyonun gerçekleştirildiği kilit mekanizmasını sağlar.

Eğer senkronize blokları ilkel veri türleri için kullanmaya çalışırsak, derleme zamanında hata alırız. Çünkü senkronize bloklar yalnızca nesneler üzerinde çalışır ve ilkel veri türleri nesne değildir.

Örneğin, aşağıdaki kod derleme zamanında hata verecektir:

```
int counter = 0;
synchronized(counter) {
   // Kod bloğu
}
```
Çünkü counter bir ilkel veri türüdür (int) ve senkronize bloklar yalnızca nesneler üzerinde çalışır. Dolayısıyla, senkronize blokları ilkel veri türleriyle kullanmaya çalıştığımızda derleme zamanında hata alırız.

11. Java'da threadlerin öncelikleri ve önceliklerinin önemi nelerdir?

Thread prioriteleri, bekleyen birçok thread arasında hangi threadin çalıştırılacağını belirler. Java programlama dilinde her threadin bir önceliği vardır. Bir thread, ebeveyn threadin önceliğini miras alır. Varsayılan olarak, bir thread normal bir önceliğe sahiptir.

Thread planlayıcısı (thread scheduler), her threadin ne zaman çalıştırılacağına karar vermek için threadin önceliklerini kullanır. Thread planlayıcısı, daha yüksek öncelikli threadi önce çalıştırır.

Thread prioriteleri, sistemdeki threadlerin çalışma sırasını belirlemekte ve öncelikli threadlerin daha hızlı yanıt vermesini sağlamakta önemlidir. Ancak, thread prioritelerinin mutlak davranışları, Java'nın çalıştığı platforma ve JVM'in yapılandırmasına bağlı olabilir. Bu nedenle, thread prioritelerini kullanırken dikkatli olunmalı ve platform bağımlılıkları göz önünde bulundurulmalıdır.

12. Farklı threadlerin öncelik türlerini açıklayın?

Java'da her threadin öncelikleri 1 ile 10 arasındadır. Varsayılan olarak, bir threadin önceliği 5'tir (Thread.NORM_PRIORITY olarak adlandırılır). Maksimum öncelik 10, minimum öncelik ise 1'dir. Thread sınıfı aşağıdaki sabitleri (static final değişkenler) özelliklerini tanımlamak için tanımlar:

Thread.MIN_PRIORITY = 1; En düşük öncelik değeri
Thread.NORM_PRIORITY = 5; Normal öncelik değeri (varsayılan)
Thread.MAX_PRIORITY = 10; En yüksek öncelik değeri
Öncelik değerleri, threadlerin çalışma sırasını belirler. Öncelik daha yüksek olan threadleri, öncelikleri daha düşük olanlardan önce çalıştırılır. Ancak, işletim sistemi ve JVM'nin uygulama davranışını etkileyebilecek faktörler olduğunu unutmamak önemlidir. Bu nedenle, thread önceliklerini ayarlamak ve kullanmak programcıların dikkat etmesi gereken bir konudur.

13. Threadin önceliği nasıl değiştirilir veya threadin önceliği nasıl ayarlanır?

Thread sınıfı, threadin önceliğini ayarlamak için bir set metodu ve threadin önceliğini almak için bir get metodu içerir.

İşte öncelik ayarlamak için kullanılan setPriority metodunun imzası:

final void setPriority(int value);
setPriority() metodu, threadin önceliğini ayarlamak için JVM'e bir istek gönderir. Ancak, JVM bu isteği kabul edip etmeme konusunda serbesttir.

Ayrıca, mevcut threadin önceliğini almak için Thread sınıfının getPriority() metodunu kullanabiliriz:

final int getPriority();
Bu metodun çağrılması, mevcut threadin öncelik değerini döndürür.

Özetlemek gerekirse, setPriority() metoduyla bir threadin önceliğini ayarlayabilir ve getPriority() metoduyla mevcut threadin önceliğini alabiliriz. Ancak, threadin önceliğini ayarlamak, işletim sistemi ve JVM tarafından belirli koşullara bağlı olduğu için kesin bir sonuç garanti edilmez.

14. İki threadin önceliği aynıysa hangi thread önce yürütülür?

Eğer eşit önceliğe sahip iki thread varsa, hangi threadin önce çalıştırılacağı konusunda garanti verilemez. Bu durum, thread planlayıcısına, hangi threadin çalıştırılacağına karar verme yetkisi verir. Thread planlayıcısı şu işlemleri yapabilir:

Havuzdaki herhangi bir threadi seçip çalıştırabilir ve tamamlanana kadar onu çalıştırabilir.
Zaman dilimlemesi (time slicing) yöntemiyle tüm iş parçacıklarına eşit fırsatlar verebilir.
Yani, eğer eşit önceliğe sahip iki thread varsa, hangisinin önce çalıştırılacağı işletim sistemine ve thread planlayıcısının belirlediği faktörlere bağlıdır. Bu nedenle, öncelikler eşit olduğunda hangi threadin öncelikli olduğunu kesin olarak söylemek mümkün değildir.

15. Threadin yürütülmesini önlemek, durdurmak için hangi yöntemler kullanılır?

Thread'lerin çalışmasını durduran üç yöntem vardır:

1. yield(): Bir threadin çalışmasını durdurarak, thread planlayıcısına diğer threadlere öncelik vermesi için bir işarettir. Ancak, bu sadece bir öneridir ve thread planlayıcısı, threadnın çalışmasını durdurup durdurmayacağına karar verir.

2. join(): Bir threadin diğer bir threadi tamamlanmasını beklemesini sağlar. Örneğin, bir threadin diğer bir threadin tamamlanmasını beklemesi gereken senaryolarda kullanılır.

3. sleep(): Belirli bir süre boyunca bir threadi uyutur. Bu işlem, belirli bir süre boyunca threadin çalışmasını durdurur ve sonra tekrar devam eder.

Bu yöntemler, threadlerin kontrolünü sağlamak ve senkronizasyonu düzenlemek için kullanılır.

16. Java’da thread sınıfında yield() metodunu açıklayınız?

yield() yöntemi, mevcut çalışan threadi bekleyen durumundaki diğer eşit önceliğe sahip threadlere fırsat vermek için çalışma durumundan Runnable durumuna geçirir. yield(), mevcut threadi belirli bir süre için uyutmak için kullanılmaz. Mevcut çalışan threadin çalışma durumunu Runnable durumuna geçirir, böylece thread planlayıcısı diğer eşit önceliğe sahip threadleri çalıştırmak için seçenek sahibi olur.

Ancak, yield() yöntemini çağırmak, threadin bir kilitleme durumunda olduğunda herhangi bir etkiye sahip değildir. Eğer bir thread daha önce bir kilidi almışsa, yield() yöntemi threadin kilidini kaybetmesine neden olmaz.

Yani, yield() yöntemi, mevcut threadi diğer eşit önceliğe sahip threadlere fırsat vermek için Runnable durumuna geçirir. Ancak, bu durumun işleyişi, thread planlayıcısına bağlıdır ve kesinlikle garantilenmez.

17. yield() yöntemi kullanılan threadin tekrar çalıştırılma şansını yakalaması mümkün mü?

yield() metodu, mevcut threadi belirli bir süre için uyutmak ve eşit önceliğe sahip diğer iş parçacıklarının çalışmasına fırsat vermek için kullanılır. Ancak, yield() metodunun çağrılmasından sonra uyuyan threade tekrar çalışma fırsatı verilip verilmeyeceği, thread planlayıcısına bağlıdır.

Yani, yield() yöntemi çağrıldıktan sonra uyuyan threadin tekrar çalışma fırsatı alıp almayacağı tamamen thread planlayıcısının insafına bağlıdır. Thread planlayıcısı, uygun olduğunu düşündüğü

zamanda uyuyan threadi tekrar çalıştırabilir veya başka bir threade öncelik verebilir. Dolayısıyla, yield() yöntemi çağrıldıktan sonra uyuyan threadin tekrar çalışma fırsatı alıp almayacağı, thread planlayıcısının takdirine bağlıdır.

18. Java'da thread sınıfında join() yönteminin önemini açıklayınız?

Thread sınıfındaki join() yönteminin önemi şudur:

Bir thread, diğer bir threadin tamamlanmasını beklemek için join() yöntemini çağırabilir. Örneğin, t1 ve t2 adında iki thread olduğunu varsayalım. Çalışan bir thread t1, join() yöntemini t2 thread üzerinde çağırırsa, t1 thread t2'nin tamamlanmasını bekler ve bu süre boyunca bekleyen durumunda kalır. t2 thread tamamladığında, t1 thread çalışmasını sürdürür.

join() yöntemi, bir threadin başka bir threadi tamamlanmasını beklemesini sağlar. Ancak, join() yöntemi, Interrupted Exception (Kesinti İstisnası) fırlatabilir. Dolayısıyla, join() yöntemini kullanırken Interrupted Exception'ı yönetmek için try-catch bloğu veya throws kullanarak istisnai durumları ele almak önemlidir.

join() yönteminin imzaları şunlardır:

public final void join() throws InterruptedException {}
public final synchronized void join(long millis) throws InterruptedException {}
public final synchronized void join(long millis, int nanos) throws InterruptedException {}

Bu yöntemler, threadin belirli bir süre veya süre ve nanosaniye cinsinden beklemesini sağlar.

19. Java'da thread sınıfında sleep() yönteminin amacını açıklayınız?

sleep() yöntemi, mevcut çalışan threadi belirli bir süre için uyutmak için kullanılır. sleep() yöntemi, mevcut threadin çalışmasını belirtilen süre boyunca duraklatır. Ancak, sleep() yöntemi, mevcut threadin tam olarak belirtilen süre boyunca uyumasını garanti etmez. Belirtilen sürenin alt sınırıdır ve thread uyandırıldığında, belirtilen süreden önce uyanabilir.

sleep() yönteminin imzaları şunlardır:

public static native void sleep(long millis) throws InterruptedException {}
public static void sleep(long millis, int nanos) throws InterruptedException {}

İlk imzada, sadece millisaniye cinsinden bir süre belirtilir ve thread bu süre boyunca uyur. İkinci imzada, millisaniye ve nanosaniye cinsinden bir süre belirtilir. Bu yöntemler, mevcut threadi belirtilen süre boyunca uyutur ve belirli bir süre boyunca çalışmasını duraklatır.

20. sleep() yöntemi başka bir threadin uyumasına neden olabilir mi?

Hayır, sleep() yöntemi sadece mevcut threadi uyutmak için kullanılır, başka bir threadi uyutmaz. sleep() yöntemi, sadece çağrıldığı threadi belirli bir süre boyunca uyutmak için kullanılır. Diğer threadler üzerinde herhangi bir etkisi yoktur.
Bir işlem (process), bir programın yürütülen halidir. Yani, bir uygulama başlatıldığında, bu uygulama bir işlem olarak adlandırılır. Her işlem, kendi bellek alanına (adres alanına) sahiptir, bu nedenle bir işlem kendi veri, kod ve çalışma zamanı verilerini içerir.

İşlemler genellikle ağır işlemlerdir çünkü her biri kendi bellek alanını, kaynaklarını, ve diğer işletim sistemi kaynaklarını kullanır. İşlemler arasında izolasyon sağlamak için ayrı ayrı çalışırlar. Eğer bir işlem çökerse, diğer işlemler bundan etkilenmez.

İşlemler genellikle bir veya daha fazla "thread"(işlem parçacığı) içerir. Bir "thread", bir işlem içindeki bir yürütme yolunu temsil eder. İşlemin içindeki thread'ler, aynı bellek alanını paylaşır, bu nedenle aynı işlem içindeki thread'ler birbirleriyle veri paylaşabilirler.

Bu nedenle, bir işlem, bir programın yürütülen hali olarak düşünülebilir ve bu işlem, kendi bellek alanına ve kaynaklara sahip, bağımsız bir yürütme birimidir.