JAVA MÜLAKAT HAZIRLIK SORULARI 3

Java Mülakatlara Hazırlık Soruları 3

1. Process (işlem) nedir?

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.

2. Java'da thread (iş parçacığı) nedir?

Thread, bir programın içindeki ayrı bir yürütme yolu ya da işlevidir. Her thread, programın içinde bağımsız bir yürütme akışını temsil eder.

İş Parçacıkları Hafif (Lightweight) ve Aynı Adres Alanını Paylaşır:
Threadleri, hafif yani düşük kaynak tüketen yürütme birimleridir. Bir programın içindeki iş parçacıkları aynı adres alanını paylaşırlar, bu da demektir ki aynı bellek alanını kullanabilirler.

Thread Oluşturmak, İşlem Oluşturmaktan Daha Basittir:
Thread oluşturmak, işlemleri oluşturmaktan daha basittir çünkü iş parçacıkları daha az kaynak tüketirler. Bu nedenle, bir program içinde yeni bir thread oluşturmak daha hızlı ve daha etkili bir yöntemdir.

Threadler Bir İşlem İçinde Varlığını Sürdürür:
Bir işlem (process) en az bir thread içerir. Yani, bir program çalıştığında, en az bir thread bulunur. Threadleri, bir işlem içinde varlığını sürdürür ve bu işlemin içinde birbirleriyle etkileşimde bulunabilirler.
Threadleri, eş zamanlılık ve çok görev yetenekleri sağlayarak programların daha etkili ve hızlı çalışmasına katkıda bulunurlar. Bu nedenle, Java ve diğer çoklu görev destekleyen programlama dillerinde threadler önemli bir kavramdır.

3. İşlem (Process) ve İş parçacığı (Thread) arasındaki farklar nelerdir?

process	thread
Yürütülen bir programdır.	Program içinde ayrı bir yürütme yolunu temsil eder. Bir veya daha fazla threade sahip olabilir ve bu bir veya daha fazla thread bir işlemi oluşturur.
İşlemler daha ağırdır	Threadler daha hafiftir.
İşlemler, kendi bağımsız adres alanlarına sahiptir.	Threadler, aynı işlem içinde aynı adres alanını paylaşır.
İşlemler arasındaki iletişim genellikle maliyetlidir.	Threadler arasındaki iletişim, işlemlere göre daha az maliyetlidir.
İşlemler arasında bağlam değiştirmek maliyetlidir.	Threadler arasında bağlam değiştirmek daha düşük maliyetlidir.

4. Çoklu görev (multitasking) nedir?

Çoklu görev, bilgisayar kullanımında aynı anda birden fazla işlemi gerçekleştirme anlamına gelir. Örneğin, bir elektronik tablo programını kullanırken aynı anda bir hesap makinesi uygulamasını da kullanmak gibi. Bu, bilgisayarın kaynaklarını etkili bir şekilde kullanarak kullanıcılara farklı görevleri aynı anda yerine getirme yeteneği sağlar. Bu, iş verimliliğini artırabilir ve kullanıcıların daha efektif bir şekilde çalışmasına olanak tanır.

5. Çoklu görev (multitasking) türleri nelerdir?

Çoklu görevin farklı türleri şunlardır:

1.İşlem Tabanlı Çoklu Görev (Process Based Multitasking):
Bu tür çoklu görev, iki veya daha fazla programın aynı anda çalıştırılmasına olanak tanır.
İşlem tabanlı çoklu görevde, bir işlem, kodun en küçük parçasını temsil eder.
Örnek: Ms Word ve Ms PowerPoint programlarını aynı anda çalıştırmak.

2.Thread Tabanlı Çoklu Görev (Thread Based Multitasking):

Bu tür çoklu görev, bir programın farklı parçalarının aynı anda çalıştırılmasına olanak tanır.
Thread tabanlı çoklu görevde, bir programın farklı işlevsel parçaları olan iş parçacıkları aynı anda çalışabilir.
Örnek: Bir belgedeki metni biçimlendirirken aynı anda belgeyi yazıcıya göndermek.
Java, thread tabanlı çoklu görevi destekler ve çoklu threadlere dahili destek sağlar.

6. Multithread(çoklu iş parçacıklı) programlamanın avantajları nelerdir?

Çoklu iş parçacıklı programlama, CPU'nun boş zamanını başka bir thread kullanma imkanı sağlar ve bu da programın daha hızlı bir şekilde çalışmasına neden olur. Tek iş parçacıklı bir ortamda her görevin tamamlanması gerektiği için bir sonraki göreve geçilmeden önce CPU boşta bekler.

Örneğin, bir programda tek bir thread kullanılıyorsa, bir görevin tamamlanması beklenmeden bir sonraki görev başlatılamaz. Bu durumda CPU, bir görevin tamamlanmasını beklerken boşta durur ve bu süre içinde başka bir iş yapmaz. Ancak çoklu iş parçacıklı bir programda, bir thread bir görevle meşgulken diğer iş parçacıkları boşta kalan CPU zamanını değerlendirebilir ve başka görevleri gerçekleştirebilir. Bu, işlemcinin daha etkin bir şekilde kullanılmasına olanak tanır ve programın daha hızlı çalışmasını sağlar.

7. Java'da thread (iş parçacığı) kavramını açıklayınız?

Thread Nedir?

Bir program içinde bağımsız bir yürütme yolunu temsil eder. Yani, bir program içinde ayrı bir çalışma akışını ifade eder.
İş parçacıkları, programın içinde aynı anda birden fazla görevi yerine getirmesine olanak tanır.
Thread Yapısı:

Bir thread, üç ana bölümden oluşur: Sanal CPU (Virtual CPU), kod ve veri.
Sanal CPU, threadin yürütme sürecini kontrol eden sanal bir işlemciyi temsil eder.
Kod, threadin yürütülen programı ifade eder.
Veri, threadinn kullanımındaki bellek alanını temsil eder.

Çalışma Zamanında Paylaşım:
Çalışma zamanında, iş parçacıkları aynı kodu ve veriyi paylaşır; yani, aynı adres alanını kullanırlar.
Bu, iş parçacıklarının birbirleriyle iletişim kurmasını ve veri paylaşmasını sağlar.

Java'da Thread:
Java'da her thread, "java.lang.Thread" sınıfının bir nesnesidir.
Java, multi threade olanak tanıyan bir programlama dilidir ve thread oluşturmak ve yönetmek için Java'da özel destek sağlanmıştır.

8. Java'da threadi destekleyen API lar nelerdir?

java.lang.Thread:
Thread sınıfını kullanarak bir thread oluşturmanın bir yoludur. Thread sınıfını genişleterek ve run() metodunu override ederek bir thread oluşturulabilir.

java.lang.Runnable:
Java'da bir arayüz olan Runnable'ı uygulayarak ve run() metodunu override ederek bir thread oluşturmanın başka bir yoludur.

java.lang.Object:
Object sınıfı, tüm Java sınıflarının üst sınıfıdır. Bu sınıfta yer alan wait(), notify() ve notifyAll() gibi metodlar, iş parçacıkları arasında senkronizasyonu sağlamak için kullanılır.

java.util.concurrent:
Bu paket, eş zamanlı programlamayı destekleyen sınıf ve arabirimleri içerir.
Örneğin, Executor arabirimi ve FutureTask sınıfı gibi sınıflar, çoklu iş parçacıklarını daha etkili bir şekilde yönetmeye yardımcı olur.
Bu API'lar, Java dilinde çoklu thread programlamasını kolaylaştırmak ve yönetmek için kullanılır.
"java.lang.Thread" ve "java.lang.Runnable" gibi temel sınıflar, iş parçacıkları oluşturmak ve yönetmek için kullanılan temel araçlardır. "java.util.concurrent" paketi ise daha gelişmiş çoklu thread programlama özelliklerini sunar.

9. Java'da "main thread" (ana thread) kavramını açıklayınız?

Main thread, bir program başlatıldıktan hemen sonra başlayan ilk threaddir(. Main thread'in önemli olmasının nedenleri şunlardır:

Tüm Alt Threadler Ana Threadden Oluşturulur:
Program başladığında, main thread otomatik olarak oluşur ve bu main threadden tüm diğer threadler türetilir.

Main Metodu, Yürütmenin Son Threadidir:
Java programları genellikle main metodunu içerir. Main metodunun tamamlanması, programın yürütmesinin sona ermesi anlamına gelir.
Diğer tüm iş parçacıkları main threadden türetildiği için, main thread tamamlandığında diğer threadler de sona erer.
JVM, main Metodunu Çağırarak Yeni Bir Thread Başlatır:
JVM (Java Virtual Machine), bir Java programını çalıştırdığında, main metodunu çağırarak yeni bir thread başlatır.
main() metodunun içindeki kodlar çalışmaya başlar, ancak bu sırada JVM, yeni bir thread oluşturur ve bu yeni thread üzerinden programın geri kalanını yürütür.
Bu nedenle, main() metodunun içindeki kodlar çalışırken, aslında ana thread geçici bir süre duraklar ve yeni thread çalışmaya başlar.
Java programları genellikle çoklu thread kullanımını destekler ve main thread, programın başladığı noktadır ve diğer iş parçacıklarının temelini oluşturur. Bu, programın eşzamanlı ve etkili bir şekilde çalışmasını sağlamak için kullanılır.

10. Java'da iş parçacıkları (threads) oluşturmanın kaç yolu vardır?

Java'da iş parçacıkları (threads) oluşturmanın iki temel yolu vardır.

1.Thread Sınıfını Extend etme:

Bir thread oluşturmanın bir yolu, Thread sınıfını extend ederek yeni bir sınıf oluşturmaktır.
Bu sınıf, Thread sınıfından türetilir ve run metodunu override ederek threadin ana mantığını tanımlar.

Örneğin:

```
public class MyThread extends Thread {
   public void run() {
      // Threadin ana mantığı burada tanımlanır.
   }
}

// Threadi başlatmak için
MyThread myThread = new MyThread();
myThread.start();
```

Runnable Arayüzünü Uygulayarak:

Başka bir yol, Runnable arayüzünü implemets ederek bir thread oluşturmaktır.
Runnable arayüzü, run metodunu içeren bir arayüzdür. Bu metod, threadin ana mantığını içerir.
Yeni bir sınıf oluşturulur ve bu sınıf Runnable arayüzünü implements eder. Ardından, bir örneği oluşturulan bu sınıf, bir Thread nesnesine parametre olarak verilerek çalıştırılır.
Örneğin:

```
public class MyRunnable implements Runnable {
   public void run() {
      // Threadin ana mantığı burada tanımlanır.
   }
}

// Threadi başlatmak için
Thread myThread = new Thread(new MyRunnable());
myThread.start();
```

Bu iki yöntem de thread oluşturmanın yaygın ve temel yollarını temsil eder.

11. Thread oluşturmanın en iyi yolu hangisidir?

Java'da thread oluşturmanın en iyi yaklaşımı genellikle Runnable arabirimini uygulamaktır. Nedeni şu şekildedir:

Esneklik:
Runnable arabirimini uygulayarak thread oluşturmak, sınıfınızın başka bir sınıfı genişletmesine olanak tanır.
Java'da çoklu kalıtım desteklenmez, bu nedenle Thread sınıfını genişlettiğinizde başka bir sınıfı genişletemezsiniz. Bu durum, sınıfınızın özellikle başka bir sınıfı genişletmesi gerektiğinde sorun yaratabilir.

Arayüz Odaklı Programlama:
Runnable arabirimini uygulamak, arayüz odaklı programlamayı teşvik eder. Bu, sınıflar arasındaki bağımlılığı azaltır ve kodun daha modüler olmasını sağlar.

Thread Pool ve Executor Framework:
Java'da Runnable arayüzünü uygulayan sınıfları kullanarak, Executor framework ve thread pool gibi gelişmiş çoklu thread yönetim araçlarından daha iyi yararlanabilirsiniz.

Kodun Daha Okunabilir Olması:
Runnable arayüzünü uygulamak, kodun daha okunabilir ve bakımı daha kolay olmasını sağlar.

Örnek:

```
public class MyRunnable implements Runnable {
  @Override
  public void run() {
    // Threadin ana mantığı burada tanımlanır.
  }
}

// Threadi başlatmak için
Thread myThread = new Thread(new MyRunnable());
myThread.start();
```

Yukarıdaki örnekte, MyRunnable sınıfı, Runnable arabirimini uygular ve yeni bir thread başlatmak için Thread sınıfını kullanır. Bu, kodun daha esnek ve okunabilir olmasını sağlar.

12. Java'da thread scheduler'ın önemini açıklayınız?

Thread scheduler (thread planlayıcısı), JVM'in bir anda birden fazla thread olduğunda hangi threadin çalıştırılacağını belirlemek için kullandığı bir bölümdür. Sadece "runnable" (çalışabilir) durumdaki iş parçacıkları, thread scheduler tarafından seçilir.

Thread scheduler, öncelikli olan iş parçacıklarına önce mikroprosesör zamanı tahsis eder. Aynı önceliğe sahip iş parçacıkları arasında mikroprosesör zamanını tahsis etmek için ise thread scheduler round robin (dönüş sırası) yöntemini takip eder.

Önemli Noktalar:

Öncelikli İş Parçacıkları:
Thread scheduler, öncelikli iş parçacıklarına öncelik verir. Yüksek öncelikli iş parçacıkları, düşük öncelikli iş parçacıklarından önce mikroprosesör zamanı alır.
Round Robin Yöntemi:
Aynı önceliğe sahip iş parçacıkları arasında zaman paylaşımı yapmak için round robin yöntemini kullanır. Bu, iş parçacıkları arasında adil bir dağılım sağlar.

Runnable Durumdaki İş Parçacıkları:
Sadece "runnable" durumdaki iş parçacıkları, thread scheduler tarafından seçilir. Bu, iş parçacıklarının sadece çalışmaya hazır olduğu durumda mikroprosesör zamanı almalarını sağlar.
Java'da thread scheduler, çoklu iş parçacıklarının etkili bir şekilde çalışmasını sağlamak ve mikroprosesör kaynaklarını adil bir şekilde dağıtmak için kullanılır. Bu sayede öncelikli ve çalışabilir durumdaki iş parçacıkları, belirli bir düzen ve adil bir şekilde mikroprosesör zamanı alabilirler.

13. Java'da thread life cycle (yaşam döngüsünü) açıklayınız?

Yeni (New) Durumu:
Bir thread örneği oluşturulduğunda, thread "New" durumundadır.
Örneğin:

```
Thread t = new Thread();
```

Yukarıdaki örnekte, t isimli thread "new" durumundadır. Thread oluşturulmuş, ancak henüz aktif durumda değildir. Aktif hale getirmek için start() metodunu çağırmamız gerekir.

Çalışabilir (Runnable) Durumu:
Thread, iki şekilde "runnable" durumuna geçebilir:
a) start metodu çağrıldığında.
b) Bir thread, bloke olmuş veya uyku durumundan döndükten sonra da "runnable" durumuna geçebilir.
Çalışan (Running) Durumu:
Thread, thread scheduler tarafından CPU zamanı tahsis edildiğinde "running" durumuna geçer.
Bekleme/Bloke/Uyku (Waiting/Blocking/Sleeping) Durumu:
Thread, kısa bir süre için pasif hale getirilebilir. Bu durumda thread, aşağıdaki durumlardan birinde "bekleme" durumuna geçer:
Thread, bir nesnenin kilidini almayı bekliyor.
Thread, başka bir threadin tamamlanmasını bekliyor.
Thread, diğer bir threadin bildirimini bekliyor.
Bitmiş (Dead) Durumu:
Thread, run metodunun çalışması tamamlandığında "dead" durumuna geçer.
Thread, run metodunun tamamlanmasıyla otomatik olarak sona erer ve thread nesnesi çöp toplayıcı tarafından yok edilir.
Java'daki thread yaşam döngüsü, iş parçacıklarının oluşturulması, çalıştırılması, bekletilmesi ve sonlandırılması gibi farklı durumları içerir. Bu durumlar, iş parçacıklarının etkileşimli ve efektif bir şekilde çalışmasını sağlar.

14. Dead thread tekrar başlatılabilir mi?

Eğer bir thread (thread) ölmüş (dead) durumdaysa ve start methodu kullanılarak tekrar başlatmaya çalışılırsa, bu durumda çalışma zamanı hatası alınır. Çünkü bir kez ölen bir thread, bir daha başlatılamaz.
Thread öldüğünde, run metodunun yürütülmesi tamamlanmış ve thread sona ermiştir. Bu durumda, aynı thread nesnesi üzerinde tekrar start metodunu çağırmak geçerli değildir ve bir IllegalThreadStateException hatası alınır.

```
Thread myThread = new Thread(() -> {
   // Threadnın çalışma mantığı
   System.out.println("Thread çalışıyor.");
});

myThread.start();  // Thread başlatıldı

// ...

myThread.start();  // Hata! Thread öldüğü için tekrar başlatılamaz
```

Yukarıdaki örnekte, myThread adlı thread başlatıldıktan sonra tekrar başlatılmaya çalışıldığında IllegalThreadStateException hatası alınacaktır. Thread bir kez çalışma sürecini tamamladığında, başka bir thread nesnesi oluşturup onu başlatmak gereklidir.

15. Bir thread başka threadi bloke edebilir mi?

Java'da bir thread (thread), başka bir threadi bloke edemez. Ancak, bir thread kendi yürütme akışını geçici olarak durdurabilir (bloke edebilir).

Java'da iş parçacıkları arasında koordinasyon ve senkronizasyon mekanizmaları kullanılarak iş parçacıkları arasında bir tür etkileşim sağlanabilir. Bu mekanizmalar şunları içerir:

Locks (Kilitler): İş parçacıkları arasında ortak bir kaynağa erişimi kontrol etmek için kilitler kullanılabilir. Ancak, bir thread bir kaynağa kilit koyduğunda, diğer iş parçacıkları bu kaynağa erişemez, ancak başka kaynaklara erişebilir.

Wait ve Notify: İş parçacıkları arasında iletişim sağlamak için wait ve notify metodları kullanılabilir. Bir thread, bir koşulu kontrol ederken wait metodunu kullanarak bekleyebilir ve diğer bir thread, bu koşulu sağladığında notify metodunu kullanarak diğer threadi uyandırabilir.

Thread.sleep: Bir thread, belirli bir süre boyunca uykuda kalabilir (Thread.sleep kullanarak). Bu, başka bir thread için kaynağa erişimi geçici olarak durdurabilir, ancak bu blokaj koordinasyon amacıyla değil, zamanlamayla ilgilidir.

Yani, bir thread doğrudan başka bir threadi bloke edemez, ancak senkronizasyon ve koordinasyon mekanizmaları kullanılarak iş parçacıkları arasında bir tür etkileşim sağlanabilir.

16. Başlatılan bir thread tekrar başlatılabilir mi?

Bir thread bir kere başlatıldıktan sonra ikinci kez start yöntemi çağrılırsa (bir kere başlatılan bir thread bir daha başlatılamaz), bu durumda RuntimeException türünden bir IllegalThreadStateException hatası oluşacaktır.

17. Thread pool nedir?

Thread pool, bir uygulamada aynı anda çalışan çok sayıda threadi etkili bir şekilde yönetmek için kullanılan bir tasarım desenidir. Thread pool, genellikle aşağıdaki avantajları sağlar:

Performans İyileştirmesi:
Her thread oluşturulduğunda bir miktar sistem kaynağı (bellek, işlemci zamanı) harcanır. Thread pool, bu maliyeti azaltarak performansı artırır. Zira önceden oluşturulan iş parçacıkları tekrar kullanılır.
Kaynak Yönetimi:
Thread pool, aynı anda çok sayıda threadin çalışmasına izin verirken, aşırı thread oluşturmanın önüne geçer. Bu, sistem kaynaklarının daha etkili kullanılmasını sağlar.
İstikrarlı Performans:
Sabit bir sayıda thread ile çalıştığı için, thread pool, sistemde aşırı iş yükü oluşturarak performans düşüklüğüne yol açan durumları engeller.
Thread Yönetimi:
Thread pool, iş parçacıklarını oluşturma, başlatma, sonlandırma ve takip etme gibi görevleri kolaylaştırır.

18. Volatile keywordu ne işe yarar?

volatile kelimesi, bir değişkenin değerini hafıza modelindeki bazı özelliklere uygun bir şekilde kullanmayı sağlar. volatile anahtar kelimesinin kullanıldığı bir değişken, bir thread tarafından yapılan değişikliklerin diğer iş parçacıkları tarafından hemen görülmesini sağlar. Bu, değişkenin bellek modelindeki ana hafıza konumunu temsil eder.

volatile anahtar kelimesi kullanıldığında, bir değişkenin değeri sadece okuma ve yazma işlemleri sırasında önbellek kullanımı minimize edilerek ana belleğe doğrudan erişimle güncellenir. Bu, diğer iş parçacıklarının güncel değeri hemen görmelerini sağlar.

Bu özellik, özellikle çoklu iş parçacıklı ortamlarda bir değişkenin güncellenmiş değerini diğer iş parçacıklarına hemen göstermesi gerektiği durumlarda kullanılır. Ancak, volatile kullanımı bazı durumlarda yeterli olmayabilir ve daha karmaşık senkronizasyon mekanizmaları (örneğin, synchronized bloklar veya java.util.concurrent kütüphanesindeki araçlar) gerekebilir.

Örnek kullanım:

```
public class SharedResource {
  private volatile boolean flag = false;

  public void setFlagTrue() {
    flag = true;
  }

  public boolean checkFlag() {
    return flag;
  }
```

Bu örnekte, flag adlı değişken volatile olarak tanımlanmıştır. setFlagTrue metodu, flag değişkeninin değerini true yapar. Diğer iş parçacıkları, checkFlag metodu aracılığıyla flag değişkeninin güncel değerini anında görebilirler. volatile kullanılmış olsaydı, diğer iş parçacıkları bu değişikliği hemen fark edemeyebilirdi.

19. Thread oluştururken run methodunu override etmezsek ne olur?

Eğer run metodunu override etmezsek, Thread sınıfının varsayılan run metodunun uygulaması çalıştırılır. Bu durumda, Thread sınıfının run metodu içinde herhangi bir iş yapılması beklenmez ve bu metot boştur.

Eğer bir thread (thread) oluştururken kendi run metodumuzu sağlamazsak, sadece Thread sınıfının run metodunun içindeki varsayılan boş işlem gerçekleşir. Bu durumda, thread hiçbir özel görev gerçekleştirmeyecek ve derlenen programın davranışı değişmeyecektir.

20. Java'da run metodunu overload edebilir miyiz?

Evet, run metodunu overload edebiliriz ancak Thread sınıfının start metodunun her zaman parametre almayan run metodunu çağıracağını unutmamamız önemlidir. Overload run metodunu doğrudan start metodu tarafından çağırmaz.

Yani, eğer bir thread sınıfında birden fazla run metodunu tanımlarsak, Thread sınıfının start metodu yalnızca parametre almayan (run() şeklinde olanı) run metodunu çağırır.