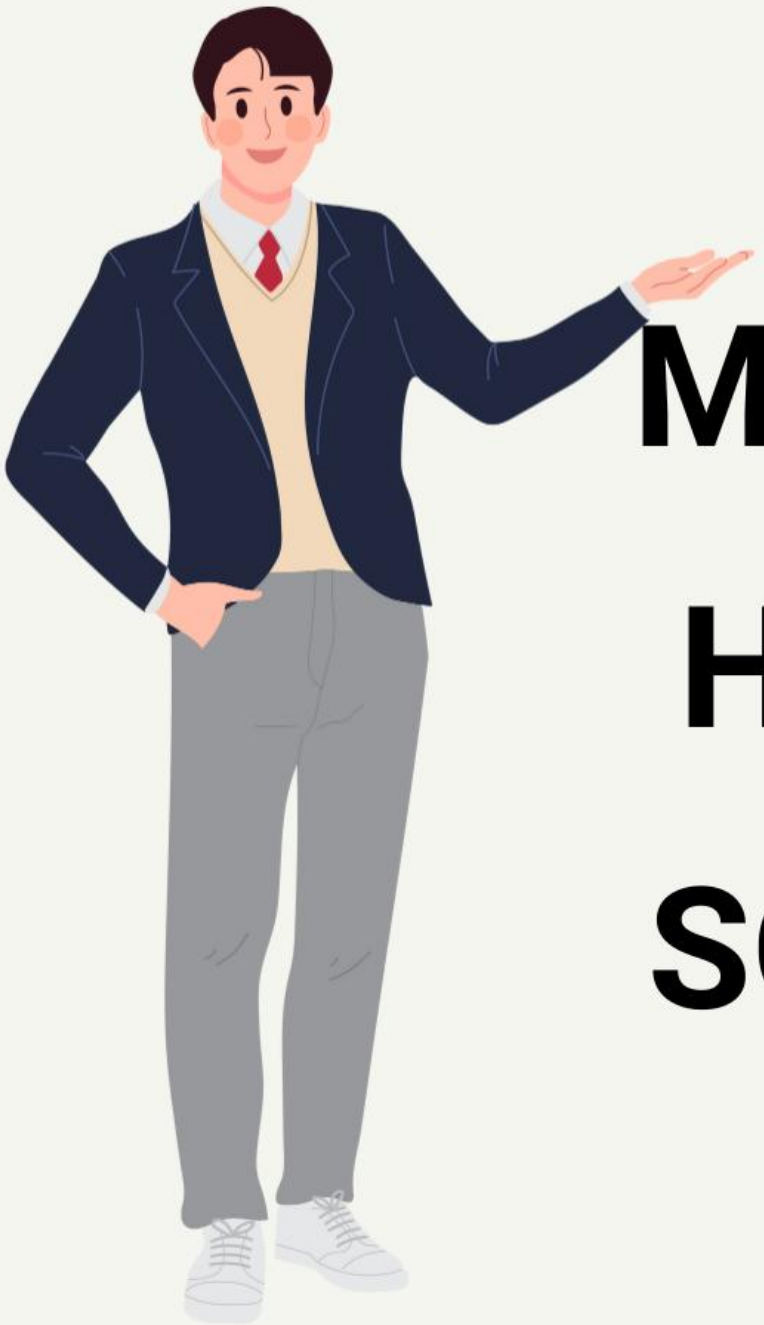

JAVA MÜLAKAT HAZIRLIK SORULARI

Java Mülakatlara Hazırlık Soruları 2

1. **Java'da try catch bloğunu açıklayınız?**

Java'da "try" ve "catch" anahtar kelimeleri, exeption durumlarının ele alınması için kullanılır.

"try" Bloğu:
"try" bloğu içinde, exception oluşturabilen veya fırlatabilecek kodları tanımlarız. Exception meydana geldiğinde bu blok çalışır ve kodunuzu değerlendirmeye alır. Eğer exception oluşmazsa, "try" bloğu normal bir şekilde çalışır.
Örnek "try" blok kullanımı:

```
try {
   // İstisna oluşturabilen kodlar burada bulunur.
} catch (Exception e) {
   // İstisna yakalandığında buradaki kod çalışır.
}
```

"catch" Bloğu:
"catch" bloğu, bir "try" bloğunda meydana gelen exception durumlarını yakalar. Exception durumu "try" bloğundan "catch" bloğuna iletilir ve burada işlenir. "catch" bloğu, exception türüne göre çalışacak kodu içerir. Böylece, exception oluştuğunda hangi işlemin yapılacağını belirleriz.
Örnek "catch" blok kullanımı:

```
try {
   // İstisna oluşturabilen kodlar burada bulunur.
} catch (Exception e) {
   // İstisna yakalandığında buradaki kod çalışır.
   // "e" değişkeni, yakalanan istisnayı temsil eder.
   // Burada istisnayı işleyebilir veya raporlayabiliriz.
}
```

Java'da bir "try" bloğu, bir "catch" bloğu ile tamamlanır ve bu ikisi birlikte bir exception işlem birimi oluşturur. Bir "catch" bloğu, yalnızca kendisinden önceki "try" bloğunda meydana gelen exceptionları yakalayabilir. Başka bir "try" bloğunun oluşturduğu exception, bir "catch" bloğu tarafından yakalanamaz.

Eğer programınızda exception oluşturan kod yoksa veya exception oluşturulmazsa, JVM (Java Sanal Makinesi) "try-catch" bloğunu görmezden gelir ve normal program akışı devam eder.

2. **Java'da catch bloğu kullanmadan try bloğu kullanılır mı?**

Bir "try" bloğunu, Java'da exception bir durumu ele almak veya kaynakları temizlemek için kullanabilirsiniz. Ancak bu "try" bloğunun yalnızca kendisi olmaz; ya bir "catch" bloğu ile hata işleme veya bir "finally" bloğu ile kaynak temizleme gibi bir sonuç sağlamalıdır.

Yani, bir "try" bloğu oluşturuyorsanız, aynı zamanda en az bir "catch" bloğu veya "finally" bloğu eklemeniz gerekmektedir. Sadece "try" bloğu kullanmak veya her ikisini birden atlamak (ne "catch" ne de "finally") derleme hatasına yol açar. Bu nedenle, exception durumları ele almak veya kaynakları düzgün şekilde temizlemek için "try-catch" veya "try-finally" bloklarını kullanmak önemlidir.

3. **Java'da bir "try" bloğu için birden fazla "catch" bloğu kullanılabilir mi?**

Bir "try" bloğu içinde birden fazla "catch" bloğu kullanılabilir çünkü kodunuz birden çok farklı türde exception oluşturabilir. Her "catch" bloğu, belirli bir exception türünü ele alır ve ilgili işlemi gerçekleştirir. Bu şekilde, programınızın farklı türde hatalarla başa çıkabilmesini sağlayabilirsiniz.

Örnek bir "try" bloğu ile birden fazla "catch" bloğu kullanımı:

```
try {
   // İstisna oluşturabilen kodlar burada bulunur.
} catch (ArithmeticException e) {
   // ArithmeticException türündeki istisnaları ele al
   // Bu blok, bölen sıfır hatası gibi matematiksel hataları işleyebilir.
} catch (NullPointerException e) {
   // NullPointerException türündeki istisnaları ele al
   // Bu blok, bir nesnenin olmadığı durumları işleyebilir.
} catch (IOException e) {
   // IOException türündeki istisnaları ele al
   // Bu blok, dosya okuma/yazma hatalarını işleyebilir.
} catch (Exception e) {
   // Genel olarak tüm istisna türlerini ele al
   // Bu blok, yukarıdaki catch bloklarının yakalayamadığı istisnaları işleyebilir.
}
```

Bu "catch" blokları, sırasıyla çalışır. Yani, bir exception meydana geldiğinde, JVM her bir "catch" bloğunu sırayla kontrol eder ve ilk eşleşen "catch" bloğunu çalıştırır. Geri kalan "catch" blokları atlanır. Bu nedenle, "catch" bloklarının sırası çok önemlidir.

Ayrıca, "catch" bloklarının sırasının alt sınıf exceptionlarından üst sınıf exceptionlarına doğru olması gerekir. Yani, daha özel exception türleri daha önce gelmelidir, sonra daha genel olanlar sıralanmalıdır. Bu, herhangi bir "catch" bloğunun çalıştırılmasını sağlar ve genel işlem sırasında daha özel durumları ele almanıza olanak tanır. Bu nedenle, "try" bloğu ile birden fazla "catch" bloğu kullanırken bu sıralamayı dikkate almak önemlidir.

4. **Java'da finally bloğunun önemini açıklayınız.**

"finally" bloğu, Java'da önemli bir rol oynar çünkü şu amaçlar için kullanılır:

Kaynak Temizleme: Genellikle "try" bloğu içinde kaynakların (veritabanı bağlantıları, dosya işlemleri, soketler vb.) kullanıldığı durumlarda, "finally" bloğu bu kaynakların temizlenmesi için kullanılır. Bu, kaynakların güvenli bir şekilde kapatılmasını ve kaynak sızıntılarının önlenmesini sağlar.

Hata Durumlarında İşlem Tamamlama: Eğer "try" bloğu exception oluşturmadan başarıyla çalışırsa, "finally" bloğu "catch" bloğunu atlamadan hemen sonra çalışır. Bu, temizleme veya sonlandırma işlemlerini gerçekleştirmeniz gereken durumlar için kullanışlıdır.

İstisna Durumlarında İşlem Tamamlama: Eğer "try" bloğunda bir exception oluşursa ve bu exception bir "catch" bloğu tarafından ele alınmazsa, yine de "finally" bloğu çalıştırılır. Bu, beklenmeyen hatalar durumunda kaynakları serbest bırakmanız veya temizlemeniz gerektiğinde kullanışlıdır.

"finally" bloğu, "try-catch-finally" yapısı içinde bulunur ve bu yapı, exceptionların düzgün bir şekilde ele alınması ve kaynakların güvenli bir şekilde temizlenmesi için çok önemlidir. Bu sayede, programlarınız daha güvenli ve sağlam hale gelir ve kaynak yönetimi konusundaki hatalar minimize edilir. Bu nedenle, "finally" bloğu Java'da önemli bir yapısal öğedir ve programların daha güvenilir olmasına yardımcı olur.

5. **Java'da "try" ve "catch" blokları arasında herhangi bir kod yazılabilir mi?**

"try" bloğu ve "catch" bloğu arasında herhangi bir kod bulunmamalıdır. "try" bloğu içindeki kod çalıştırılır ve bir exception oluşursa, bu exception "catch" bloğuna yönlendirilir. Eğer "try" bloğu ve "catch" bloğu arasında kod bulunursa, derleme hatası alırsınız. İstisna oluştuğunda, bu kod parçası atlanır ve "catch" bloğu çalıştırılmadan hemen sonra exception işlenir.

Doğru kullanım şu şekildedir:

```
try {
   // try bloğunda istisna oluşturabilen kodlar burada bulunur
} catch (Exception e) {
   // istisna yakalandığında bu blok çalışır
}
```

Yani, "try" bloğu ve "catch" bloğu birbirine bitişik olmalı ve aralarına herhangi bir kod parçası eklenmemelidir. Bu kurala uymak, kodunuzun beklenmedik hatalara karşı daha güvenli ve düzenli olmasına yardımcı olur.

6. Java'da "try" ve "finally" blokları arasında herhangi bir kod yazılabilir mi?

Java'da "try" bloğu ve "finally" bloğu arasında herhangi bir kod parçası yer alamaz. "finally" bloğu, "try" ve (varsa) "catch" blokları tamamlandıktan hemen sonra çalışır. Eğer "catch" bloğu yoksa, "try" bloğu tamamlandıktan hemen sonra "finally" bloğu çalışır.

Doğru kullanım şu şekildedir:

```
try {
   // try bloğunda işlem yapılacak kodlar burada bulunur
} catch (Exception e) {
   // istisna yakalandığında bu blok çalışır
} finally {
   // finally bloğu, try ve catch tamamlandıktan sonra çalışır
}
```

7. Java'da aynı "catch" bloğunda birden fazla exception türü ele alınır mı?

Java 7'den itibaren, aynı "catch" bloğunda birden fazla istisna türünü ele alabiliriz. Örneğin, aşağıdaki gibi bir yaklaşım kullanabiliriz:

```
try {
   // İstisna oluşturabilen kodlar burada bulunur
} catch (ArrayIndexOutOfBoundsException | ArithmeticException e) {
   // Birden fazla istisna türünü aynı "catch" bloğunda ele alabiliriz
   // e değişkeni, her iki istisna türünün de istisnasını temsil eder
   // Bu blok, hem ArrayIndexOutOfBoundsException hem de ArithmeticException istisnalarını işleyebilir.
}
```

Ancak, aynı "catch" bloğunda birden fazla exception türünü ele alırken dikkate almanız gereken bazı önemli noktalar şunlardır:

"catch" parametresi zaten final olarak kabul edilir. Yani, bu değişkene başka bir değer atayamazsınız veya değiştiremezsiniz.

Aynı "catch" bloğunda ele alınan exception türleri arasında bir hiyerarşi olmalıdır. Yani, bir üst sınıfın exceptionsını bir alt sınıfın exceptionsı olarak ele almak mümkün değildir. Örneğin, "catch (Exception e)" ve "catch (RuntimeException e)" aynı "catch" bloğunda kullanılamaz.

Bu tür bir yaklaşım, kodunuzu daha temiz ve daha az tekrarlı hale getirebilir, ancak exception durumları mantıklı bir şekilde ele almak ve kodunuzun okunabilirliğini korumak için dikkatli olmalısınız.

8. Checked Exceptions nelerdir?

Kontrol Edilen Exceptionlar:

- "Throwable" sınıfının alt sınıfları arasında, "error", "RuntimeException" ve onun alt sınıfları haricinde kalan tüm exception türleri "kontrol edilen exception" olarak kabul edilir.
- Kontrol edilen exception türleri, programcıların bu exceptionları işlemek veya bunları bildirmek için özel bir işlem yapmalarını gerektirir.
- Bu exception türlerini işlemek için "throws" anahtar kelimesi ile metot imzasında veya "try-catch" blokları kullanılmalıdır, aksi takdirde derleme hatası alınır.

Örnekler:

IOException: Dosya giriş/çıkış işlemleri sırasında oluşan hataları temsil eder.
SQLException: Veritabanı işlemleri sırasında oluşan hataları temsil eder.
FileNotFoundException: Belirtilen dosya bulunamadığında oluşan exceptionı temsil eder.
InvocationTargetException: Bir yordamın yürütülmesi sırasında exception oluştuğunda bu tür bir exception fırlatılır.
CloneNotSupportedException: Bir nesnenin klonlanmaya çalışıldığında ve klonlama işlemi desteklenmiyorsa oluşan exceptionı temsil eder.
ClassNotFoundException: Bir sınıfın yüklenmeye çalışıldığında ve bu sınıf mevcut değilse fırlatılan exceptionı temsil eder.
InstantiationException: Bir sınıfın bir örneği oluşturulamadığında fırlatılan exceptionı temsil eder.
Bu exception türleri, programın güvenliğini ve istikrarını sağlamak için önemlidir ve programcılar bu tür exceptionları yakalamak ve işlemek için gerekli önlemleri almalıdır.

9. Unchecked Exceptions nelerdir?

Kontrol Edilmemiş İstisnalar (Unchecked Exceptions):
"RuntimeException" sınıfının alt sınıfları, "unchecked exceptions" olarak adlandırılır. Bu tür exception türleri, programın derleyici tarafından exception yönetimi için kontrol edilmesini gerektirmez.
Programlar, bu tür exceptionları ele almasalar bile derlenebilir. Bu, "catch" veya "throws" kullanımına gerek olmadığı anlamına gelir.
Eğer bir kontrol edilmemiş exception meydana gelirse, programın çalışması sona erir. Bu nedenle bu tür exceptionların dikkatli bir şekilde ele alınması önemlidir.
Örnekler:

ArithmeticException: Matematiksel işlemlerde sıfıra bölme veya geçersiz işlemler nedeniyle oluşan exceptionları temsil eder.
ArrayIndexOutOfBoundsException: Dizi erişimi sırasında dizinin sınırlarının dışına çıkma exceptionlarını temsil eder.
ClassCastException: Nesne tür dönüşümü sırasında uyumsuz türler arasında dönüşüm yapma girişimleri nedeniyle oluşan exceptionları temsil eder.
IndexOutOfBoundsException: İndeks sınırları dışına çıkma exceptionlarını temsil eder.
NullPointerException: Null bir nesneye erişme girişimleri nedeniyle oluşan exceptionları temsil eder.
NumberFormatException: Bir sayısal dönüşüm hatası nedeniyle oluşan exceptionları temsil eder.
StringIndexOutOfBounds: Bir dize içinde belirtilen indeksin sınırları dışına çıkma exceptionlarını temsil eder.
UnsupportedOperationException: Bir koleksiyon üzerinde desteklenmeyen bir işlemi yapma girişimleri nedeniyle oluşan exceptionları temsil eder.
Bu tür exceptionlar, programların güvenliğini ve istikrarını sağlamak için önemlidir ve programcılar bu tür exceptionları öngörmeye ve ele almaya çalışmalıdır.

10. Java'da varsayılan handle etme işlemi nedir?

Java'da varsayılan exception işleme, Java Sanal Makinesi (JVM) tarafından otomatik olarak yürütülen bir mekanizmadır. Bir exception oluşturulduğunda veya tespit edildiğinde, JVM aşağıdaki bilgileri içeren yeni bir exception işleme nesnesi oluşturur:

Exception Türü Adı
Exception Hakkında Açıklama
Exception Oluşma Konumu
JVM, bu exception işleme nesnesini oluşturduktan sonra, programın bu exception ile başa çıkıp çıkamayacağını kontrol eder. Eğer programda exception işleme kodu varsa (yani "try-catch" veya "try-finally" blokları), bu kod çalışır ve program devam eder. Ancak, eğer programda uygun bir exception işleme kodu bulunmuyorsa, JVM bu sorumluluğu varsayılan bir işlemciden bekler ve programı beklenmedik bir şekilde sonlandırır.

Varsayılan exception işleyici, exception hakkında bilgi veren bir açıklama, bir exceptionın stacktrace'ini ve exception oluşma konumunu görüntüler. Programın aniden sonlandırılması, bu işlemin dezavantajlarından biridir çünkü programlar düzgün bir şekilde sona ermeli ve kullanıcıya hata mesajları verilmelidir. Bu nedenle, Java programlarında uygun exception işleme kodu eklemek, programın daha güvenilir ve kullanıcı dostu olmasına yardımcı olur.

11. Java’da throw keywordunu açıklayınız?

Java'da "throw" kelimesi, genellikle JVM'nin exceptionları fırlatmasına izin verdiği ve bu exceptionları try-catch blokları kullanarak ele aldığımız durumların dışında, kullanıcı tanımlı exceptionları veya çalışma zamanı exceptionlarını açıkça fırlatmak için kullanılır.

"throw" kelimesinin kullanımı için genel sözdizimi:

throw throwableInstance;
Burada, throwableInstance bir Throwable veya onun alt sınıflarından biri olmalıdır.

"throw" ifadesi çalıştırıldığında, bu ifadeden sonraki ifadeler çalıştırılmaz ve işlem durur. Ardından, JVM fırlatılan exceptionı ele alacak bir catch bloğu olup olmadığını kontrol eder. Eğer yoksa, bir sonraki catch ifadesine geçer ve uygun bir işleyici bulana kadar devam eder. Eğer uygun bir işleyici bulunamazsa, varsayılan exception işleyici programı durdurur ve exception açıklamasını ve konumunu yazdırır.

Genellikle "throw" kelimesini, kullanıcı tanımlı veya özel exceptionları fırlatmak için kullanırız. Bu şekilde, programın belirli durumları ele almasını sağlayabilir ve hata durumlarını düzgün bir şekilde işleyebiliriz.

12. Java’da “throw” kullanımından sonra kod yazılır mı?

“throw” ifadesi çalıştırıldığında, kontrol akışı ilgili catch bloğuna veya uygun bir işleyiciye geçer. throw ifadesinden sonraki ifadeler, exception fırlatıldığı için ulaşılamaz kod olarak kabul edilir. Bu nedenle, throw ifadesinden sonra herhangi bir kod yazmaya çalışırsanız, derleme zamanında "ulaşılamaz kod" hatası alırsınız. Derleyici, throw ifadesinden sonraki kodun hiçbir zaman çalıştırılamayacağını belirler ve bu durumu size bildirmek için bir hata mesajı oluşturur.

13. Java’da throw keywordunun önemini açıklayınız?

Java'da throws anahtar kelimesinin önemi :

Metodun İmzasında Belirtme: throws ifadesi, bir metodun imzasının sonunda kullanılır ve belirli bir türde bir exception bu metodun içinden fırlatılabileceğini belirtir.

İstisna İşleme Sorumluluğunu Delege Etme: throws ifadesinin temel amacı, kontrol edilen exceptionların durumunda exception işleme sorumluluğunu çağıran metotlara devretmektir. Yani, bir metodun içinde bir kontrol edilen exception (checked exception) oluşursa, bu durumu ele almaktan ziyade, metot throws ifadesi ile çağırana bu exception ile başa çıkma sorumluluğunu atar.

Unchecked (Kontrolsüz) İstisnalarda Gerekli Değil: Kontrolsüz exceptionlar (unchecked exceptions) için throws ifadesini kullanmak zorunlu değildir. Bu tür exceptionların genellikle programcı hatası veya beklenmeyen durumlar olduğu düşünülerek, bu exceptionları ele almak yerine programın çökmesine izin verilir.

Sadece Throwable Türleri İçin Kullanılabilir: throws ifadesi sadece Throwable türleri veya bu sınıflardan türetilmiş exception türleri için kullanılabilir. Aksi takdirde, uyumsuz türlerle ilgili bir derleme hatası alırsınız.

Öneri: Yalnızca Belirli İstisna Türlerini Belirtme: Metot içinde fırlatılan exceptionların alt sınıfları yerine, belirli exception türlerini belirtmek daha iyidir. Bu, kodun daha esnek ve anlaşılır olmasına katkıda bulunur.

```
class Test {
  public static void main(String args[]) throws SpecificException {
    // ...
  }
}
```

Bu örnekte, SpecificException veya bu sınıfın alt sınıflarından biri olabilecek bir istisna fırlatılabileceği belirtilmiştir.

14. Finally ve return kullanımı nasıl olur?

finally bloğu ve return ifadesi birlikte kullanıldığında, finally bloğu her iki durumda da (try veya catch bloğu içinde olsun veya olmasın) önemli bir rol oynar. Özellikle finally bloğu, programın normal akışını bozmaksızın son temizleme veya kaynak salma işlemlerini gerçekleştirmek için kullanılır. Bu, programın her durumda düzgün bir şekilde kapatılmasını sağlar.

Önemli bir senaryo şudur:

Return İfadesi Inside Try veya Catch Bloğu:

```
public int exampleMethod() {
  try {
    // Some code that may throw an exception
    return 42;
  } catch (Exception e) {
    // Handle the exception
    return -1;
  } finally {
    // This block will be executed no matter what
    System.out.println("Finally block executed");
  }
}
```

Bu örnekte, try bloğu içinde veya catch bloğu içinde bir return ifadesi bulunmaktadır. Ancak, bu return ifadesi çağrıldığında bile, finally bloğu yine de çalışacaktır.

Finally Bloğu Önceliklidir:
Eğer bir finally bloğu ve return ifadesi birlikte kullanılıyorsa, finally bloğu her zaman önceliklidir. Yani, finally bloğu tamamlandıktan sonra return ifadesi işlenir.

Örneğin, yukarıdaki exampleMethod örneğinde, eğer try bloğu içindeki return 42; satırı çalışırsa, bu değeri döndürmeden önce finally bloğu çalışacaktır. Dolayısıyla, sonuç olarak önce "Finally block executed" yazısı görülecektir, ardından 42 değeri döndürülecektir.

Bu, kaynakları serbest bırakma, temizlik işlemleri veya her iki durumda da yapılması gereken işlemler gibi durumlar için kullanışlıdır. finally bloğu, hata oluşsa bile belirli işlemlerin her zaman gerçekleştirilmesini sağlar. Bu nedenle, finally bloğu, özellikle kaynak yönetimi gibi durumlarda, programın düzgün çalışmasını sağlamak için önemlidir.

15. **Checked exceptions için catch kullanabilir miyiz?**

Java'da, kontrol edilen exceptionların (checked exceptions) işlenmesi için catch blokları eklemek gereklidir. Ancak, bir kod bloğunda kontrol edilen bir exception olasılığı yoksa, yani belirli bir blokta bir hata oluşma ihtimali yoksa, bu durumda bu exceptionı işlemek için bir catch bloğu eklemek zorunlu değildir.

Örneğin:

```
public class Example {
  public static void main(String[] args) {
    try {
      // Kodunuz burada
    } catch (CheckedException ex) {
      // Eğer buraya hiçbir zaman gelinmiyorsa, bu bloğa gerek yok.
      // Bu durumda, bu catch bloğu "unreachable code" hatası alabilir.
    }
  }
}
```

Yukarıdaki örnekte, catch (CheckedException ex) bloğu, eğer CheckedException türünde bir istisna fırlatma olasılığı yoksa gereksiz olacaktır. Bu durumda, bu catch bloğu "ulaşılamaz kod" hatası alabilir.

Bu nedenle, kodunuzda bir istisna olasılığı olup olmadığını dikkatlice değerlendirmek önemlidir. Eğer bir kod bloğunda belirli bir istisna türü olmayacağı kesinse, o istisnayı işlemek için gereksiz bir catch bloğu eklemekten kaçınılmalıdır. Aksi takdirde, derleme sırasında "ulaşılamaz kod" hatası alabilirsiniz.

16. Kullanıcı tanımlı istisnalar (user-defined exceptions) ne demektir?

Kullanıcı tanımlı exceptionlar (user-defined exceptions), programcıların özel durumları belirtmek ve özel hata mesajları oluşturmak için oluşturdukları exception türleridir. Bu tür exceptionlar, hem kontrol edilen (checked) exception türleri olarak hem de kontrolsüz (unchecked) exception türleri olarak oluşturulabilir.

Kontrol Edilen (Checked) User-Defined Exception:
Kullanıcı tanımlı exception türleri, Exception sınıfını genişleterek veya kontrol edilen exception sınıflarından birini genişleterek oluşturulabilir. Kontrol edilen bir exception, programcının bu exception türünü belirtmek için throws ifadesini kullanması gerektiği bir türdür.

```
public class MyCheckedException extends Exception {
  // ...
}
```

Bu durumda, MyCheckedException sınıfı, Exception sınıfını genişleterek kontrol edilen bir exception haline gelir.

Kontrolsüz (Unchecked) User-Defined Exception:
Kullanıcı tanımlı exception türleri, RuntimeException sınıfını genişleterek kontrolsüz bir exception haline getirilebilir. Bu tür exception türleri, programcının throws ifadesini kullanmak zorunda olmadığı, yani yakalanma veya işlenme zorunluluğu olmayan exception türleridir.

```
public class MyUncheckedException extends RuntimeException {
  // ...
}
```

Bu durumda, MyUncheckedException sınıfı, RuntimeException sınıfını genişleterek kontrolsüz bir exception haline gelir.

Not:
Genellikle özel bir durumu temsil etmek için kontrolsüz exception türlerini kullanmak daha yaygındır. Bu, programcının herhangi bir throws ifadesi eklemek zorunda kalmadan exception türünü kullanabilmesine olanak tanır. Bu nedenle, özel exception sınıflarını oluştururken, mümkünse RuntimeException sınıfını genişletmek önerilir.

17. Java'da iç içe try bolakları kullanabilir miyiz ?

Java'da try ifadelerini iç içe yerleştirebiliriz. Bir try ifadesini başka bir try ifadesinin bloğu içinde tanımlayabiliriz. Bu, belirli bir kod bloğunda farklı seviyelerde hata işleme stratejileri uygulamamıza olanak tanır.

Bir örnek:

```
public class NestedTryExample {
  public static void main(String[] args) {
    try {
      // Dış try bloğu
      System.out.println("Outer Try Block");

      try {
        // İç try bloğu
        System.out.println("Inner Try Block");
        int result = 10 / 0;  // Bu satır bir ArithmeticException fırlatabilir
      } catch (ArithmeticException innerException) {
        // İç try bloğundaki ArithmeticException'ı işle
        System.out.println("Inner Catch Block: " + innerException.getMessage());
      }

      System.out.println("After Inner Try-Catch Block");
    } catch (Exception outerException) {
      // Dış try bloğundaki genel istisnayı işle
      System.out.println("Outer Catch Block: " + outerException.getMessage());
    }
  }
}
```

Bu örnekte, dış try ifadesi ile iç içe geçmiş bir iç try ifadesi bulunmaktadır. Eğer iç try bloğunda bir hata oluşursa, iç catch bloğu bu hatayı ele alacaktır. Eğer dış try bloğunda bir hata oluşursa, dış catch bloğu bu hatayı ele alacaktır. Bu, programcılara farklı durumları belirli bloklarda işlemek için esneklik sağlar.

18. Throwable class ve methotlarının önemini açıklayınız?

Throwable sınıfı, Java'daki exceptionların temel sınıfıdır. Bütün exceptionlar, bu Throwable sınıfından türetilir. Throwable sınıfının iki ana alt sınıfı Exception ve Error sınıflarıdır. Throwable sınıfının tanımladığı üç önemli metot şunlardır:

printStackTrace() Metodu:
Bu metot, exception bilgilerini aşağıdaki formatta yazdırır:

İstisna adı
Açıklama
Stack trace (yığın izi)

```
try {
   // ... bir exception fırlatabilecek kod
} catch (Exception e) {
   e.printStackTrace();
}
```

Bu metot, hatanın nerede oluştuğunu belirlemek ve hatayı ayıklamak için çok kullanışlıdır.

getMessage() Metodu:
Bu metot, exception nesnesinin sadece açıklama (description) kısmını yazdırır. Bu, exception hakkında daha spesifik bilgiye ihtiyaç duyulmadığında kullanışlıdır.

```
try {
   // ... bir exception fırlatabilecek kod
} catch (Exception e) {
   System.out.println(e.getMessage());
}
```

Bu şekilde, sadece exception açıklamasını elde edebilirsiniz.

toString() Metodu:
Bu metot, exception nesnesinin adını ve açıklamasını yazdırır. printStackTrace() metodu gibi detaylı bir yığın izini yazdırmaz, sadece exception adı ve açıklamasını döndürür.

```
try {
   // ... bir exception fırlatabilecek kod
} catch (Exception e) {
   System.out.println(e.toString());
}
```

Bu metot, exception hakkında temel bilgi sağlamak için kullanışlıdır.

19. ClassNotFoundException ne zaman fırlatılır, açıklayınız?

ClassNotFoundException, JVM'nin bir sınıfı yüklemeye çalıştığında sınıfın bulunamaması durumunda ortaya çıkan bir exceptiondır. Bir sınıf adı, metin dizesi olarak verildiğinde ve JVM bu sınıfı yüklemeye çalıştığında, ancak belirtilen isimde bir sınıf bulunamadığında bu exception fırlatılır.

Örneğin, aşağıdaki gibi bir durumda ClassNotFoundException ortaya çıkabilir:

```
public class Main {
  public static void main(String[] args) {
    try {
      // Belirtilen sınıf adıyla bir sınıf yüklemeye çalışıyoruz
      Class.forName("BilinmeyenSinif");
    } catch (ClassNotFoundException e) {
      // ClassNotFoundException durumunda buraya düşeriz
      e.printStackTrace();
    }
  }
}
```

Bu örnekte, Class.forName("BilinmeyenSinif"); satırı, belirtilen sınıf adını yüklemeye çalışır. Ancak, "BilinmeyenSinif" adında bir sınıf bulunmadığı için JVM ClassNotFoundException fırlatır. Bu durum, sınıf adının yanlış yazıldığı veya var olmadığı bir durumu temsil eder.

ClassNotFoundException durumu, sınıf adlarını dinamik olarak belirleyerek veya yükleme işlemlerini yürütürken kullanıcı tarafından tanımlanan sınıfların yüklenmesi sırasında sıkça karşılaşılır. Programlar dinamik olarak sınıfları yüklerken veya sınıf adlarını kullanıcı girişi veya dış kaynaklardan alırken bu tür bir durum ortaya çıkabilir.

20. NoClassDefFoundError ne zaman fırlatılır, açıklayınız?

NoClassDefFoundError, JVM'nin bir sınıfı yüklemeye çalıştığında sınıfın tanımının bulunamaması durumunda ortaya çıkan bir hatadır. Bu hatanın sebepleri şunlar olabilir:

Sınıf Tanımının Bulunamaması:

Sınıfın tanımı compile (derleme) aşamasında var olabilir, ancak runtime (çalışma) aşamasında bu tanım bulunamazsa NoClassDefFoundError hatası ortaya çıkar.
Yanlış Sınıf Adı:

Sınıf adının yanlış yazılması veya belirtilen sınıf adının geçerli bir sınıfı işaret etmemesi durumunda bu hata ortaya çıkabilir.
Classpath Problemleri:

Sınıfın byte kodlarını içeren sınıf dosyasının veya sınıf dizininin, classpath üzerinde bulunamaması durumunda bu hatayla karşılaşılabilir.
Byte Kod Dosyasının Kaldırılması veya Değiştirilmesi:

Sınıfın byte kodlarını içeren dosyanın silinmesi veya değiştirilmesi durumunda, JVM sınıfı yükleyemez ve NoClassDefFoundError hatası alınabilir.
Örnek bir durumu göstermek için:

```
public class Main {
  public static void main(String[] args) {
    try {
      // BilinmeyenSınıf adındaki sınıfı yüklemeye çalışıyoruz
      BilinmeyenSinif obj = new BilinmeyenSinif();
    } catch (NoClassDefFoundError e) {
      // NoClassDefFoundError durumunda buraya düşeriz
      e.printStackTrace();
    }
  }
}
```

Bu örnekte, BilinmeyenSinif adında bir sınıfı yüklemeye çalıştık, ancak bu sınıfın tanımı bulunamadığı için NoClassDefFoundError hatası alırız. Bu hatanın genellikle sınıfın adının yanlış yazılması, classpath sorunları veya sınıf dosyasının eksik veya bozuk olması gibi sebepleri vardır.