Exception Handling (Hata İşleme):

Exception: Exception, bir programın yürütülmesi sırasında, yani çalışma zamanında meydana gelen ve programın talimatlarının normal akışını bozan istenmeyen veya beklenmeyen bir olaydır.

Exception Vs Error:

Error: Bir uygulamanın yakalamaya çalışmaması gereken ciddi bir sorunu belirtir.

Exception: Bir uygulamanın yakalamaya çalışabileceği durumları belirtir.

Exception Handling (İstisna İşleme): İstisnaların işlenerek programın akışının sürdürülebilmesidir.

Bir metotta excepiton gerçekleşirse, o metot o exception türünde bir nesne oluşturur ve bunu fırlatır. (throw)

Programcı exception olaylarını 'try, catch, finally, throw ve throws' keywordleri ile kontrol altına alır.

Try: Exception fırtlatma ihtimali olan kısım.

Catch: Fırlatılan exception'ı aldığı parametreye göre yakalayan kısım.

Finally: Try-Catch bloklarından sonra muhakkak çalışır. (Java finally bloğu genellikle dosyayı kapatmak veya aradaki bağlantıları kesmek vs. gibi durumlarda kullanılır.)

Throw: Manuel olarak bir excepiton fırlatmamızı sağlar.

Throws: Excepiton fırlatılabilir anlamı taşır.

Java, kendinden yerleşik gelen (java kitaplığında bulunan) excepitonlara ve kullanıcıların kendilerinin tanımladığı exceptionlara izin veren bir yapıya sahiptir.

Önemli Built-in Excepitonlar:

1. **ArithmeticException**
   Bir aritmetik işlemde istisnai bir durum oluştuğunda atılır.
2. **ArrayIndexOutOfBoundsException**
   Bir diziye geçersiz bir indexle erişildiğini belirtmek için atılır. Index, dizinin boyutuna eşit veya ondan büyükse atılır.
3. **ClassNotFoundException**
   Bu İstisna, tanımı bulunmayan bir sınıfa erişmeye çalıştığımızda ortaya çıkıyor.
4. **FileNotFoundException**
   Bu İstisna, bir dosyaya erişilemediğinde veya açılmadığında ortaya çıkar.
5. **IOException**
   Bir giriş-çıkış işlemi başarısız olduğunda veya kesintiye uğradığında atılır.

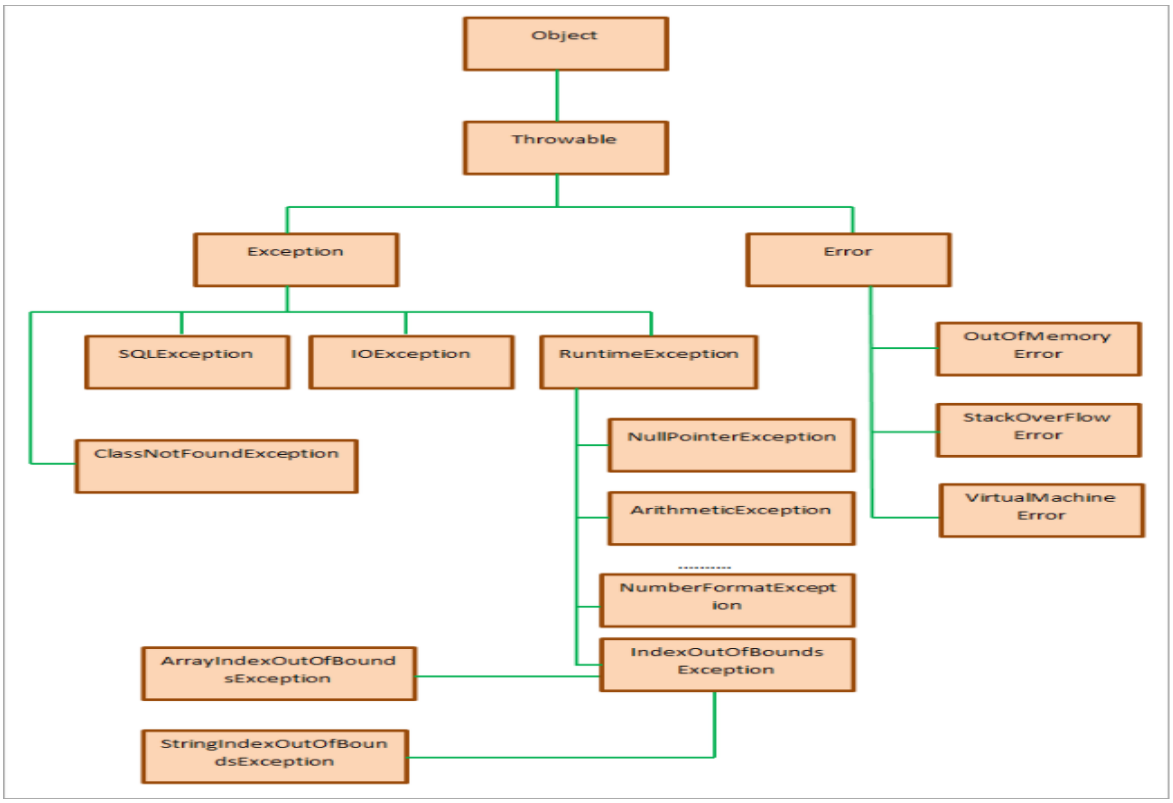
6. **InterruptedException**
   Bir iş parçacığı beklerken, uyurken veya bazı işlemler yaparken atılır ve kesintiye uğrar.
7. **NoSuchFieldException**
   Bir sınıf belirtilen alanı (veya değişkeni) içermediğinde atılır
8. **NoSuchMethodException:** Bulunamayan bir metoda erişirken atılır.
9. **NullPointerException**
   Bu istisna, boş bir nesnenin üyelerine atıfta bulunulduğunda ortaya çıkar. Null hiçbir şeyi temsil etmez
10. **NumberFormatException**
    Bu özel durum, bir metot bir stringi sayısal biçime dönüştüremediğinde ortaya çıkar.(Integer.parseInt("asd"))
11. **RuntimeException**
    Bu, çalışma zamanı sırasında oluşan herhangi bir istisnayı temsil eder.
12. **StringIndexOutOfBoundsException**
    Bir dizinin negatif veya dizenin boyutundan daha büyük olduğunu belirtmek için String sınıfı yöntemleri tarafından atılır.

User-Defined Exception:

Kullanıcının kendi tanımladığı excepiton classlarıdır. Bunun için:

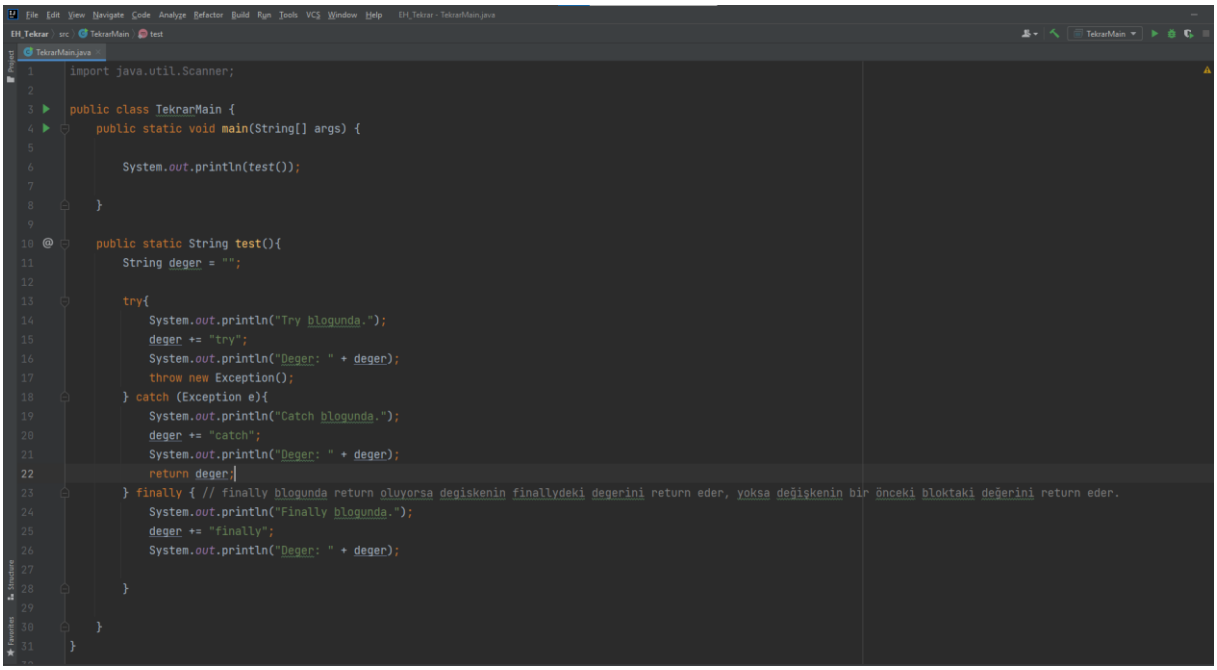
1- Kendi exception classımızı 'Exception' classından türetmeliyiz. (class MyException extends Exception)

2- Kendi exceptionımızı fırlatmak için bu classtan bir nesne üretip bunu fırlatmalıyız.
(MyException nesne = new MyException(); throw nesne;)



Checked Excepiton (Kontrollü Exception): Derleme zamanı kontrol edilen exceptionlardır. Java bu exceptioları try-catch bloklarıyla yakalamamızı ister. (Run Time Exception Ve Error Haricindeki classlar)

UnChecked Exception (Kontrol Edilemeyen Exception): Derleme zamanı kontrol edilemeyen exceptionlardır. Bu nedenle java bunları try-catch ile yakalamamıza bizi zorlamaz. RunTime exceptionlar ve errorlar kontrol edilemeyen exceptionlardır.



```java
import java.util.Scanner;

public class TekrarMain {
    public static void main(String[] args) {

        System.out.println(test());

    }

    public static String test(){
        String deger = "";

        try{
            System.out.println("Try blogunda.");
            deger += "try";
            System.out.println("Deger: " + deger);
            throw new Exception();
        } catch (Exception e){
            System.out.println("Catch blogunda.");
            deger += "catch";
            System.out.println("Deger: " + deger);
            return deger;
        } finally { // finally blogunda return oluyorsa degiskenin finallydeki degerini return eder, yoksa değişkenin bir önceki bloktaki değerini return eder.
            System.out.println("Finally blogunda.");
            deger += "finally";
            System.out.println("Deger: " + deger);

        }

    }
}
```

Override edilen metotlar compiler time exceptionları fırlatıyorsa ediyor ise ana metot ve kalıtımla alınan metotların yanına throws excepiton yazılmalıdır. Run time exception fırlatıyorsa yazmaya gerek yoktur çünkü java bunları yakalamamız için bizi zorlamaz.

```java
class A {
    void f() throws SQLDataException {

    }

    void y(){

    }
}

class B extends A {
    @Override
    void f() throws SQLDataException {
        throw new SQLDataException();
    }

    @Override
    void y() {
        throw new RuntimeException();
    }
}
```