

# BBS515 Nesneye Yönelik Programlama

---

Ders 13 - Collections, Exception Handling

Zümra Kavafoğlu  
*<https://zumrakavafoglu.github.io/>*

# Collections

---

- Koleksiyon(collection), bazen container adını da alır, birçok elemanı tek bir birimde toplayan bir nesnedir.
- Collectionlar, toplu verileri depolamak, çekmek, değiştirmek ve verilerle iletişim kurmak için kullanılır.
- Temsil ettikleri veriler genel olarak doğal bir grup oluşturan verilerdir, bir posta klasörü( mektupların bir koleksiyonu) veya bir telefon defteri (isimlerin telefon numaralarıyla bir eşleştirmesi (mapping))

# Collections Framework

---

- Collections framework koleksiyonları temsil etmek ve değiştirmek için tasarlanmış birleşik bir mimaridir.
- Bir Collections framework'ü aşağıdakileri kapsar:
  - Arayüzler : Koleksiyonları temsil eden soyut yapılardır. Bütün koleksiyonlar için implementasyonlarının ayrıntılarından bağımsız erişim ve değiştirme sağlarlar.
  - Implementasyonlar : Arayüzlerin somut implementasyonlarıdır. Özünde tekrar kullanılabilir veri yapılarıdır.
  - Algoritmalar : Tüm koleksiyon nesnelerinin kullanabileceği, arama ve sıralama gibi faydalı işler için yazılmış metotlardır.

# Java Collections Framework

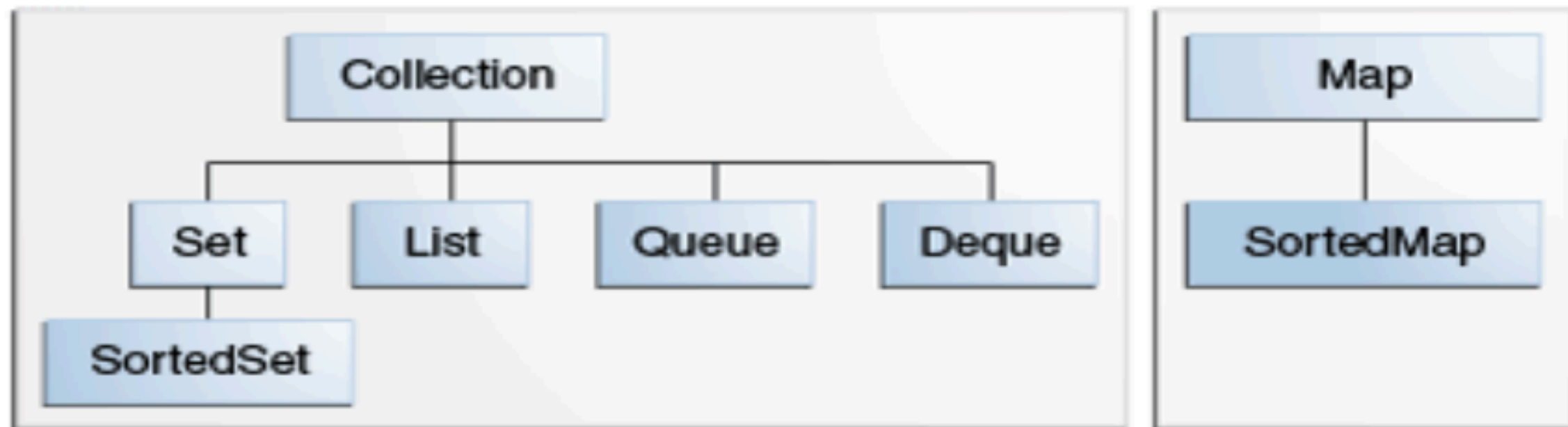
---

- Java Collections Framework aşağıdaki faydaları sağlar
  - Daha hızlı ve kolay programlama yapmayı sağlar: Sunduğu faydalı veri yapıları ve algoritmalarla, programın küçük ayrıntıları yerine daha önemli parçalarına odaklanmayı sağlar
  - Program hızını ve kalitesini artırır: Faydalı veri yapılarının ve algoritmaların yüksek performanslı ve yüksek kalitede implementasyonlarını içerir. Bir arayüzün bir implementasyonu için yazılmış program, başka bir implementasyon için, yalnızca implementasyonun çeşidi değiştirilerek yeniden kullanılabilir. Programcı kendi veri yapılarını ve algoritmalarını yazma yükünden kurtulduğu için, esas programların hızını ve kalitesini arttırmaya odaklanabilir.
  - Yazılımların yeniden kullanılabilirliğini artırır. Standart collection arayüzleriyle uyumlu yeni veri yapılarının üretilmesini sağlar ve bu veri yapıları yeniden kullanılabilir.

# Çekirdek Collection Arayüzleri (Core Collection Interfaces)

---

- Aşağıdaki diagram Java Collections Framework'ünün çekirdek arayüzlerini ve hiyerarşilerini göstermektedir.



# Collection arayüzü

---

- Collection arayüzü collection nesnelerinin kullanımında maksimum genellik sağlar. Bütün koleksiyonların sahip olacağı temel metotlar bu arayüzde deklare edilir.

# Collection arayüzü - Temel metotlar

<code>boolean add (Object obj)</code>	obj nesnesini koleksiyonun sonuna ekler
<code>boolean contains(Object obj)</code>	obj nesnesi koleksiyonda varsa true döndürür
<code>boolean addAll(Collection c)</code>	c'nin tüm elemanlarını koleksiyonun sonuna ekler
<code>void clear( )</code>	Koleksiyonun tüm elemanlarını siler
<code>boolean containsAll(Collection c)</code>	c'nin tüm elemanları koleksiyonda varsa true döndürür.
<code>boolean equals(Object obj)</code>	Koleksiyon ve obj eşitse true döndürür.
<code>boolean isEmpty( )</code>	Koleksiyon boşsa true döndürür.
<code>boolean remove(Object obj)</code>	obj nesnesini koleksiyondan siler.
<code>boolean removeAll(Collection c)</code>	c koleksiyonunun tüm elemanlarını koleksiyondan siler.
<code>boolean retainAll(Collection c)</code>	c'nin elemanları hariç tüm elemanları koleksiyondan siler.
<code>int size( )</code>	Koleksiyonun eleman sayısını döndürür.
<code>Object[ ] toArray( )</code>	Elemanları koleksiyonun elemanlarının kopyası olan bir dizi döndürür.
<code>Iterator iterator( )</code>	Koleksiyon için bir Iterator(tekrarlayıcı) döndürür.

# Collection elemanlarını dolaşmak

---

- Javada bir collection'ı dolaşmanın iki yolu vardır:
  - Iterator kullanmak
  - Gelişmiş for döngüsü (for each) kullanmak



# Collection elemanlarını dolaşmak: Iteratorlar

---

- Iterator bir collectionın elemanlarını dolaşmayı sağlayan bir nesnedir.
- Bir collectionın Iterator'ı onun iterator metodu çağırılarak elde edilir.
- collection bir Collection nesnesi olsun. collection'ın her bir elemanı aşağıdaki biçimde ekrana yazdırılabilir:

```
Iterator <E> it = collection.iterator();  
while(it.hasNext())  
    System.out.println(it.next());
```

# Collection elemanlarını dolaşmak: Gelişmiş for

---

- Gelişmiş for ile bir collectionın elemanları aşağıdaki biçimde alt alta yazdırılabilir:

```
for(Object o : collection)  
    System.out.println(o);
```

- Gelişmiş for ifadesi yalnızca bir kısa yazımdır. Bu ifade de o.iterator() kullanacak biçimde derlenir.

# Iterator Arayüzü

---

- Her bir Iterator nesnesi Iterator arayüzünü implement eden bir sınıftan oluşturulur.
- Iterator arayüzü:

```
public Interface Iterator <E> {  
    boolean hasNext();  
    E next();  
    void remove();  
}
```

**hasNext()** : eğer koleksiyonda gezilecek eleman kaldıysa true döner

**next()** : koleksiyondaki bir sonraki elemanı döndürür. Eğer bütün elemanlar dolaşıldıysa NoSuchElementException hatasını verir.

**remove()**: koleksiyonun o anki elemanını siler. Eğer next() çağırılışından sonra çağırılmazsa IllegalStateException hatasını verir.

# List arayüzü

---

- List arayüzü Collection arayüzünü genişletir ve bir dizi eleman tutan bir koleksiyonu temsil eder.
- Listenin elemanlarına listedeki yerleri kullanılarak erişilebilir.
- Listenin birbirinin aynısı elemanları olabilir.
- Collection arayüzünde tanımlı metotlara ek olarak bir takım yeni metot deklarasyonları içerir.
- ArrayList, LinkedList ve Vector sınıfları List arayüzünü implement eder.
  - ArrayList ve Vector, List arayüzünün boyutu değiştirilebilir dizi implementasyonlarıdır.
  - LinkedList, List arayüzünün bağlantılı liste implementasyonudur.

# List arayüzü - Temel metotlar

<code>void add(int index, Object obj)</code>	obj nesnesini listenin index indisli yerine ekler. O indisteki ve sonrasındaki elemanlar yukarı kaydırılır.
<code>boolean addAll(int index, Collection c)</code>	c'nin tüm elemanlarını listenin index indisli yerine ekler. O indisteki ve sonrasındaki elemanlar yukarı kaydırılır.
<code>Object get(int index)</code>	index indisli elemanı döndürür.
<code>int indexOf(Object obj)</code>	obj nesnesinin listede ilk bulunduğu yerin indisini döndürür. Listedeyoksa -1 döndürür.
<code>int lastIndexOf(Object obj)</code>	obj nesnesinin listede en son bulunduğu yerin indisini döndürür. Listedeyoksa -1 döndürür.
<code>ListIterator listIterator( )</code>	Listenin başlangıcı için bir ListIterator döndürür.
<code>Object remove(int index)</code>	index indisli elemanı listeden siler ve bu elemanı döndürür. Listedeyoksa sonraki elemanları bir geriye kaydırılır.
<code>Object set(int index, Object obj)</code>	obj nesnesini index indisli elemana atar.
<code>List subList(int start, int end)</code>	start indisinden end indisine kadar olan elemanların bir listesini döndürür.

# ArrayList Sınıfı

---

- ArrayList sınıfı List arayüzünü implement eder.
- ArrayList ihtiyaca göre uzunluğu değiştirilebilen dinamik diziler oluşturmayı sağlar.
- Daha önce gördüğümüz Java dizileri sabit uzunluğa sahipti. Bir kere oluşturulduktan sonra genişletilmesi ya da küçültülmesi mümkün değildi. Dolayısıyla dizinin tam olarak kaç elemanının olacağını önceden bilinmesi gerekiyordu.
- ArrayList nesneleri ise bir ilk uzunlukla oluşturulurlar. Uzunluğu aşacak kadar eleman eklenirse otomatik olarak genişletilirler, eleman silindiğinde daraltılırlar.

# ArrayListTest

---

```
import java.util.List;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

public class ArrayListTest{

    public static void main(String[] args){

        String[] colors = { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
        String[] removeColors = { "RED", "WHITE", "BLUE" };

        List< String > list = new ArrayList< String >();
        List< String > removeList = new ArrayList< String >();

        // add elements in colors array to list
        for ( String color : colors )
            list.add( color );

        // add elements in removeColors to removeList
        for ( String color : removeColors )
            removeList.add( color );

        System.out.println( "ArrayList: " );

        // output list contents
        for ( int count = 0; count <list.size(); count++ )
            System.out.printf( "%s ", list.get( count ) );

        // remove colors contained in removeList
        removeColors( list, removeList );

        System.out.println( "\n\nArrayList after calling removeColors: " );

        // output list contents
        for ( String color : list )
            System.out.printf( "%s ", color );

    }
}
```

# ArrayListTest

---

```
public static void removeColors(Collection< String > collection1, Collection< String > collection2){  
    Iterator< String > iterator = collection1.iterator();  
  
    while (iterator.hasNext())  
        if(collection2.contains( iterator.next() ) )  
            iterator.remove();  
}  
// end method removeColors
```

## Çıktı

ArrayList:

MAGENTA RED WHITE BLUE CYAN

ArrayList after calling removeColors:

MAGENTA CYAN



# ListIterator

---

- ListIterator listeler için özelleşmiş Iterator sınıfıdır. ListIterator ile aşağıdaki işlemler yapılabilir:
  - elemanların indisine ulaşmak
  - ileri ve geri dolaşma
  - eleman ekleme ve değiştirme
- Iterator'a ek olarak aşağıdaki metotları implement eder:
  - **add(e)** : O anki pozisyona e elemanını ekler
  - **hasPrevious()** : Listeyi ters yönde gezerken gezilecek eleman kaldıysa true döner
  - **previous()** : Listede o anki pozisyondan bir önceki elemanı döndürür ve bir adım geri gider.
  - **previousIndex()** : Bir önceki elemanın indisini döndürür.
  - **set(e)** : En son next() veya previous() çağırılışında dönen elemanı e ile değiştirir.

# ListIteratorTest

```
import java.util.List;
import java.util.ArrayList;
import java.util.ListIterator;

public class ListIteratorTest {

    public static void main(String[] args) {

        String colors[] = { "black", "yellow", "green", "blue", "violet", "silver" };
        String colors2[] = { "gold", "white", "brown", "blue", "gray", "silver" };

        List< String > list1 = new ArrayList<String>();

        List< String > list2 = new ArrayList< String >();

        for ( String color : colors )
            list1.add( color );

        for ( String color : colors2 )
            list2.add( color );

        list1.addAll( list2 ); // concatenate lists

        list2 = null; // release resources

        printList( list1 ); // print list1 elements

        System.out.print( "\nConverting to uppercase" );
        convertToUppercaseStrings( list1 ); // convert to uppercase string

        printList( list1 ); // print list1 elements

        System.out.print( "\nDeleting elements 4 to 6..." );

        removeItems( list1, 4, 7 ); // remove items 4-7 from list

        printList( list1 ); // print list1 elements

        printReversedList( list1 ); // print list in reverse order

    }
}
```

# ListIteratorTest

---

```
// output List contents
public static void printList(List< String > list) {

    System.out.println( "\nlist: " );

    for ( String color : list )
        System.out.printf( "%s ", color );

    System.out.println();

} // end method printList
```

```
// locate String objects and convert to uppercase
public static void convertToUppercaseStrings(List< String > list) {

    ListIterator< String > iterator = list.listIterator();

    while (iterator.hasNext()) {
        String color = iterator.next(); // get item
        iterator.set( color.toUpperCase() ); // convert to uppercase
    } // end while

} // end method convertToUppercaseStrings
```

# ListIteratorTest

---

```
// obtain sublist and use clear method to delete sublist items
public static void removeItems(List< String > list, int start, int end ) {
    list.subList( start, end ).clear(); // remove items
} // end method removeItems
```

```
// print reversed list
public static void printReversedList( List< String > list ) {
    ListIterator< String > iterator = list.listIterator( list.size() );
    System.out.println( "\nReversed List:" );
    // print list in reverse order
    while (iterator.hasPrevious())
        System.out.printf( "%s ", iterator.previous());
} // end method printReversedList
```

# Set Arayüzü

---

- Set her bir elemandan yalnızca bir tane içeren bir koleksiyondur.
- Collections kütüphanesindeki Set implementasyonlarından bazıları HashSet ve TreeSet'tir.
- HashSet eklenen elemanları sıralamazken, TreeSet eklenen elemanları otomatik olarak küçükten büyüğe sıralar.

# HashSet Örneği : SetTest.java

---

- Bir string dizisindeki tekrarlayan elemanları silen bir program yazalım.

```
import java.util.*;

public class SetTest
{
    public static void main( String args[] ) {
        String colors[] = { "red", "white", "blue",
                            "green", "gray", "orange", "tan", "white", "cyan",
                            "peach", "gray", "orange" };

        List< String > list = Arrays.asList( colors );

        System.out.printf( "ArrayList: %s\n", list );

        printNonDuplicates( list );
    }
}
```

# HashSet Örneği : SetTest.java

---

```
// create set from array to eliminate duplicates
public static void printNonDuplicates( Collection< String > collection) {

    // create a HashSet

    Set< String > set = new HashSet<String>( collection );

    System.out.println( "\nNonduplicates are: " );

    for ( String s : set )
        System.out.printf( "%s ", s );

    System.out.println();

} // end method printNonDuplicates
```

# HashSet Örneği : SetTest.java

---

## Çıktı

```
ArrayList: [red, white, blue, green, gray, orange, tan, white, cyan, peach, gray, orange]
```

```
Nonduplicates are:
```

```
tan green peach cyan red orange gray white blue
```



# SortedSet örneği: SortedSetTest.java

```
import java.util.Arrays;
import java.util.SortedSet;
import java.util.TreeSet;

public class SortedSetTest {
    public static void main(String args[]) {
        String names[] = {"yellow", "green",
            "black", "tan", "grey", "white", "orange", "red", "green"};

        SortedSet<String> tree = new TreeSet<String>(Arrays.asList(names));

        System.out.println("sorted set: ");
        printSet(tree); // output contents of tree

        // get headSet based on "orange"
        System.out.print("\nheadSet (\"orange\"): ");
        printSet(tree.headSet("orange"));

        // get tailSet based upon "orange"
        System.out.print("tailSet (\"orange\"): ");
        printSet(tree.tailSet("orange"));

        // get first and last elements
        System.out.printf("first: %s\n", tree.first());
        System.out.printf("last : %s\n", tree.last());
    }
}
```

# SortedSet örneği: SortedSetTest.java

---

```
public static void printSet(SortedSet<String> set) {  
    for (String s : set)  
        System.out.printf("%s ", s);  
  
    System.out.println();  
} // end method printSet
```

## Çıktı

```
sorted set:  
black green grey orange red tan white yellow  
  
headSet ("orange"): black green grey  
tailSet ("orange"): orange red tan white yellow  
first: black  
last : yellow
```

# Map arayüzü

---

- Map arayüzü değerlerin anahtar-tabanlı bir biçimde tutulmasını sağlayan Collections arayüzüdür.
- Her bir anahtar(key), değer(value) ikilisine bir girdi denir.
- Mapteki her bir anahtardan yalnızca bir tane olabilir.
- Mapteki bir değere anahtarı aracılığıyla ulaşılır.
- Map arayüzünü implement eden sınıflardan bazıları HashMap, LinkedHashMap ve TreeMap'tir.

# Bazı Map metotları

<code>void clear()</code>	Mapteki bütün girdileri siler
<code>boolean containsKey(Object key)</code>	Map key nesnesinin anahtar olduğu bir (anahtar,değer) girdisine sahipse true döner.
<code>boolean containsValue(Object value)</code>	Map value değerli en az bir anahtar,değer) girdisine sahipse true döner.
<code>Object put(Object key, Object value)</code>	Mapte key anahtarı yoksa bir (key,value) girdisi ekler, varsa key'e karşılık gelen değeri value olarak günceller
<code>Object remove(Object key)</code>	Eğer key anahtarlı bir girdi varsa mapten siler.
<code>int size( )</code>	Mapteki tüm girdilerin sayısını döndürür.
<code>Object get(Object key)</code>	Key anahtarı varsa onun gösterdiği değeri döndürür, yoksa null döndürür.
<code>Set keySet( )</code>	Mapteki anahtarları içeren bir Set nesnesi döndürür.

# Map kullanım örneği : WordTypeCount.java

---

- Bir cümlede her kelimedenden kaç adet olduğunu bulup ekrana yazdıran bir program yazalım.
- Yapmamız gereken her bir kelimeye karşılık bir tamsayı değer tutmak
- Bunun için HashMap sınıfından faydalanacağız.
- Bu mapte anahtarlar String tipinde, değerler Integer tipinde olacak.

# Map kullanım örneği : WordTypeCount.java

---

```
import java.util.StringTokenizer;
import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.TreeSet;

import java.util.Scanner;

public class WordTypeCount {

    public static void main(String[] args){

        Map< String, Integer > map = new HashMap< String, Integer >();

        createMap(map); // create map based on user input

        displayMap(map); // display map content

    }
```

# Map kullanım örneği : WordTypeCount.java

---

```
// create map from user input
public static void createMap(Map<String, Integer> map) {

    Scanner scanner = new Scanner( System.in );

    System.out.println( "Enter a string:" ); // prompt for user input

    String input = scanner.nextLine();

    // create StringTokenizer for input
    StringTokenizer tokenizer = new StringTokenizer( input , " ,.:_-");

    // processing input text
    while ( tokenizer.hasMoreTokens() ) // while more input
    {
        String word = tokenizer.nextToken().toLowerCase(); // get word

        // if the map contains the word
        if (map.containsKey(word)) {
            int count = map.get(word); //get current count
            map.put(word, count + 1); // increment count by one
        } else {
            map.put(word, 1); // add new word with a count of 1 to map
        }
    }

} // end method createMap
```

# Map kullanım örneği : WordTypeCount.java

---

```
// display map content
public static void displayMap(Map<String, Integer> map) {
    // get keys
    Set< String > keys = map.keySet();

    // sort keys
    TreeSet< String > sortedKeys = new TreeSet< String >( keys );

    System.out.println( "Map contains:\nKey\t\t\t\tValue" );

    // generate output for each key in map
    for ( String key : sortedKeys )
        System.out.printf( "%-10s%10s\n", key, map.get( key ) );

    System.out.printf("\nsize:%d\nisEmpty:%b\n", map.size(), map.isEmpty());
} // end method displayMap
```



# Map kullanım örneği : WordTypeCount.java

---

## Çıktı

```
Enter a string:
Today the weather is so nice, the birds are so nice, the flowers are so so nice.
Map contains:
Key          Value
are           2
birds         1
flowers       1
is            1
nice          3
so            4
the           3
today         1
weather       1

size:9
isEmpty:false
```

# exception'lar

---

- Bir kod bloğunda çalışma zamanında ortaya çıkan anormal durumlara exception denir. Daha önce de gördüğümüz exception örnekleri:
  - Bir sayıyı sıfır ile bölmek
  - Dizi sınırları dışında bir elemana ulaşmak
- Java exception'ı bir kod parçasında ortaya çıkan exceptional (olağanüstü) bir durumu tarif eden bir nesnedir.
- exceptionlar Java programını çalıştırdığımızda hata olduğu anda oluşturulur ve hatanın olduğu noktaya ulaşabilmemiz için kullanışlı bilgiler ihtiva ederler.
- Exceptionlar Java run-time system tarafından oluşturulabilir veya programcı tarafından kodla üretilebilir.
- Java normal program akışını etkilemeyecek biçimde exceptionlar yakalayıp üstelerinden gelebilir.

## exception örneği : Sıfır ile bölme

```
ExceptionExample.java x
1  public class ExceptionExample {
2
3  public static void main(String[] args){
4
5      int number1 = 0;
6
7      int number2 = 3;
8
9      int number3 = number2 / number1;
10
11      System.out.println("Result: " + number3);
12
13  }
14 }
```

Sıfır ile bölme  
hatası

9. satırda program durdurulur ve aşağıdaki hata mesajı ekrana yazdırılır.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at ExceptionExample.main(ExceptionExample.java:9)
```

# exception örneği : Sıfır ile bölme

---

Peki sıfır ile bölme gibi anormal bir durumla karşılaşıldığında tam olarak ne oluyor:

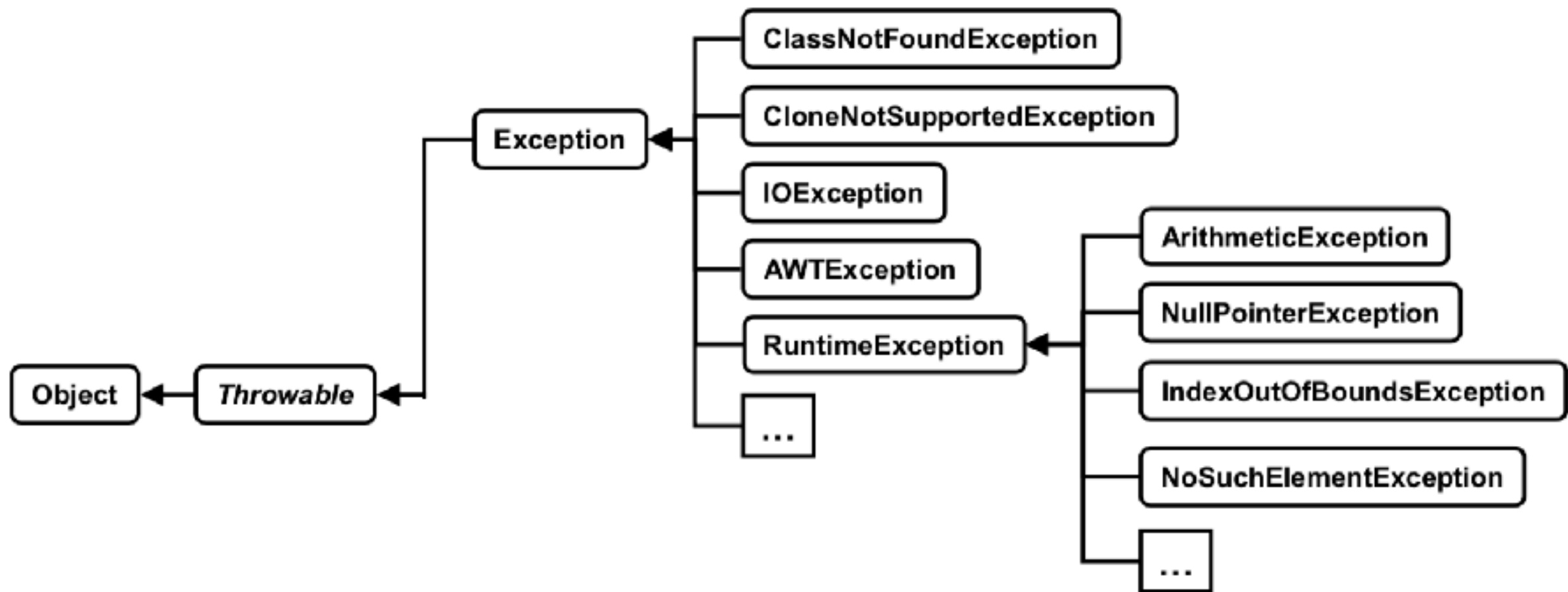
- Java çalışma-zamanı sistemi (run-time system) bu durum için bir exception nesnesi oluşturuyor ve onu fırlatıyor (throws an exception).
- Programda bu exceptionın üstesinden gelecek bir kod parçası olmadığı için, JVM programı durduruyor ve hatanın nereden ve neden kaynaklandığını ekrana yazdırıyor.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at ExceptionExample.main(ExceptionExample.java:9)
```

**Javanın sunduğu “exception” mekanizmasıyla fırlatılan bu exception yakalanıp anormal durumun üstesinden gelinebilir ve programın devam etmesi sağlanabilir. Buna exception handling denir.**

# Exception sınıfları hiyerarşisi

---



# Exception handling

---

- Exception mekanizması programcıya anormal bir durumun üstesinden gelme şansı verir.
- Exception handling bir exceptiona karşılık bir eylem yapmaktır. Yapılabilecek eylemlere örnekler:
  - Programdan çıkmak
  - İşlemi hata vermeyecek biçimde yeni verilerle denemek
  - Bir hata mesajı verip kullanıcıyı yönlendirmek

olabilir.

# Dene-yakala (try-catch) exception handling mekanizması

---

- Dene-yakala (try-catch) mekanizması sayesinde programı çalıştırdığımız anda oluşabilecek muhtemel hataları program akışını etkilemeyecek biçimde yakalayabiliriz. Bu mekanizmanın temel mantığı programda hata oluşturmaları muhtemel kod parçalarını bir try bloğunun içine almak ve bu bloğun hemen ardından hata yakalandığında yapılacak işlemleri içeren bir catch bloğu yazmak.

```
try{  
    //hata oluşturmaları muhtemel kodlar  
}  
catch(ExceptionTip1 exceptionNesnesi){  
    //try içindeki kod bloğunda Exception1 tipinde bir hata olursa  
    çalıştırılacak kodlar  
}
```

# Dene-yakala (try-catch) exception handling mekanizması

```
TryCatchExample.java x
1 public class TryCatchExample {
2
3     public static void main(String[] args){
4
5         int number1, number2, number3;
6         try {
7             number1 = 0;
8
9             number2 = 3;
10
11             number3 = number2 / number1;
12
13             System.out.println("Result: " + number3);
14
15         } catch (Exception e){
16             System.out.println("Exception occurred and handled ! ");
17         }
18
19     }
20 }
```

1. Java run-time sistemi exception hakkında bilgi içeren bir exception nesnesi oluşturur ve fırlatır



# Dene-yakala (try-catch) exception handling mekanizması

```
TryCatchExample.java x
1  ▶ public class TryCatchExample {
2
3  ▶  public static void main(String[] args){
4
5      int number1, number2, number3;
6      try {
7          number1 = 0;
8
9          number2 = 3;
10
11         number3 = number2 / number1;
12
13         System.out.println("Result: " + number3);
14
15     } catch (Exception e){
16         System.out.println("Exception occurred and handled ! ");
17     }
18
19 }
20 }
```

**1.** Java run-time sistemi exception hakkında bilgi içeren bir exception nesnesi oluşturur ve fırlatır

**2.** Bu satırda kodun çalışması durdurulur ve fırlatılan exception tipi için bir catch bloğu aranır

# Dene-yakala (try-catch) exception handling mekanizması

```
TryCatchExample.java x
1 public class TryCatchExample {
2
3     public static void main(String[] args){
4
5         int number1, number2, number3;
6         try {
7             number1 = 0;
8
9             number2 = 3;
10
11             number3 = number2 / number1;
12             System.out.println("Result: " + number3);
13         } catch (Exception e){
14             System.out.println("Exception occurred and handled ! ");
15         }
16     }
17 }
18
19
20 }
```

1. Java run-time sistemi exception hakkında bilgi içeren bir exception nesnesi oluşturur ve fırlatır

2. Bu satırda kodun çalışması durdurulur ve fırlatılan exception tipi için bir catch bloğu aranır

3. Exception sınıfı tüm exception tiplerini kapsadığından fırlatılan exception bu catch bloğu tarafından yakalanır.

# Dene-yakala (try-catch) exception handling mekanizması

```
TryCatchExample.java x
1 public class TryCatchExample {
2
3 public static void main(String[] args){
4
5     int number1, number2, number3;
6     try {
7         number1 = 0;
8
9         number2 = 3;
10
11         number3 = number2 / number1;
12
13         System.out.println("Result: " + number3);
14
15     } catch (Exception e){
16         System.out.println("Exception occurred and handled ! ");
17     }
18
19 }
20 }
```

**1.** Java run-time sistemi exception hakkında bilgi içeren bir exception nesnesi oluşturur ve fırlatır

**2.** Bu satırda kodun çalışması durdurulur ve fırlatılan exception tipi için bir catch bloğu aranır

**3.** Exception sınıfı tüm exception tiplerini kapsadığından fırlatılan exception bu catch bloğu tarafından yakalanır.

**4.** Catch bloğunun içindeki satırlar çalıştırılır.

# Dene-yakala (try-catch) exception handling mekanizması

```
TryCatchExample.java x
1 public class TryCatchExample {
2
3 public static void main(String[] args){
4
5     int number1, number2, number3;
6     try {
7         number1 = 0;
8
9         number2 = 3;
10
11         number3 = number2 / number1;
12
13         System.out.println("Result: " + number3);
14
15     } catch (Exception e){
16         System.out.println("Exception occurred and handled ! ");
17     }
18
19 }
20 }
```

1. Java run-time sistemi exception hakkında bilgi içeren bir exception nesnesi oluşturur ve fırlatır

2. Bu satırda kodun çalışması durdurulur ve fırlatılan exception tipi için bir catch bloğu aranır

3. Exception sınıfı tüm exception tiplerini kapsadığından fırlatılan exception bu catch bloğu tarafından yakalanır.

4. Catch bloğunun içindeki satırlar çalıştırılır.

**Çıktı:**

Exception occurred and handled !

# Dene-yakala (try-catch) exception handling mekanizması

---

- Her catch bloğu yalnızca kendinden bir önceki try bloğundan fırlatılan exceptionları yakalayabilir. Başka bir try bloğundan fırlatılan exceptionları yakalayamaz.
- try ve catch bloklarında yer alan ifadelerin mutlaka süslü parantezler arasında yazılması gerekir.

# java.lang.Exception

---

- java.lang.Exception exception hiyerarşisinin taban sınıfıdır, yani Javadaki her exception sınıfı bu sınıfın alt sınıfıdır.
- java.lang.Exception'ın bir çok alt sınıfı vardır:
  - java.lang.ArithmeticException
  - java.lang.ArrayIndexOutOfBoundsException
  - java.lang.NullPointerException
  - java.io.IOException
  - java.io.FileNotFoundException
- Javanın tanımladığı exception sınıfları dışında kendi özel exception sınıflarımızı da yazabiliriz.

# Birden fazla catch bloğu

- Bir kod bloğunun içinde birden fazla exception oluşma ihtimali olabilir. Örneğin aşağıdaki kod bloğunda kullanıcının girdiği değerlere göre hem sıfırla bölme hem de dizi sınırlarını aşma ihtimali vardır.

```
int arr[] = new int[10];  
int number = 5;  
  
Scanner input = new Scanner(System.in);  
  
System.out.println("Enter the divisor: ");  
  
int divisor = input.nextInt();
```

```
number = number / divisor;
```

divisor değerinin 0 girilme olasılığı var.

```
System.out.println("Enter the array index: ");
```

```
int i = input.nextInt();
```

```
arr[i] = number;
```

i değerinin 0'dan küçük 10'dan büyük girilme olasılığı var.

# Birden fazla catch bloğu

- Bu durumda bu bloğu try içine alıp iki farklı exception tipi için iki farklı catch bloğu yazabiliriz:

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Everything is alright");
```

Bu satırda bir ArithmeticException oluşabilir.

Oluşursa bu catch bloğu tarafından yakalanır



# Birden fazla catch bloğu

- Bu durumda bu bloğu try içine alıp iki farklı exception tipi için iki farklı catch bloğu yazabiliriz:

```
try{  
    System.out.println("Enter the divisor: ");  
    int divisor = input.nextInt();  
    number = number / divisor;  
    System.out.println("Enter the array index: ");  
    int i = input.nextInt();  
    arr[i] = number;  
}
```

Bu satırda bir `ArrayIndexOutOfBoundsException` exception oluşabilir.

```
catch (ArithmeticException e){  
    System.out.println("Division by zero is wrong!");  
}  
catch (ArrayIndexOutOfBoundsException e){  
    System.out.println("Array index is out of bounds!");  
}
```

Oluşursa bu catch bloğu tarafından yakalanır

```
System.out.println("Everything is alright");
```

# Birden fazla catch bloğu: Senaryo 1

```
try{  
    System.out.println("Enter the divisor: ");  
    int divisor = input.nextInt();  
    number = number / divisor;  
    System.out.println("Enter the array index: ");  
    int i = input.nextInt();  
    arr[i] = number;  
}  
catch (ArithmeticException e){  
    System.out.println("Division by zero is wrong!");  
}  
catch (ArrayIndexOutOfBoundsException e){  
    System.out.println("Array index is out of bounds!");  
}  
  
System.out.println("Code must go on!");
```

## Çıktı penceresi

Enter the divisor:

# Birden fazla catch bloğu: Senaryo 1

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

## Çıktı penceresi

```
Enter the divisor:
0
```

# Birden fazla catch bloğu: Senaryo 1

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

## Çıktı penceresi

Enter the divisor:  
0

Sıfır ile bölme hatası:  
ArithmeticException  
nesnesi fırlatıldı ve try  
bloğundan çıkıldı

# Birden fazla catch bloğu: Senaryo 1

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

## Çıktı penceresi

Enter the divisor:  
0

Fırlatılan  
ArithmeticException  
nesnesi burada yakalandı

# Birden fazla catch bloğu: Senaryo 1

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

## Çıktı penceresi

Enter the divisor:  
0  
Division by zero is wrong!

Exception nesnesini  
yakalayan bloğun içine  
girildi



# Birden fazla catch bloğu: Senaryo 1

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}
```

```
System.out.println("Code must go on!");
```

## Çıktı penceresi

```
Enter the divisor:
0
Division by zero is wrong!
Code must go on!
```

**İlk yakalayan kazanır!  
Diğer catch blokları bu  
hatayı artık  
yakalayamaz !**

**catch bloklarından  
sonra gelen satırla  
koda devam edilir**

## Birden fazla catch bloğu: Senaryo 2

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

### Çıktı penceresi

Enter the divisor:



## Birden fazla catch bloğu: Senaryo 2

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

### Çıktı penceresi

```
Enter the divisor:
2
```

## Birden fazla catch bloğu: Senaryo 2

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

### Çıktı penceresi

Enter the divisor:  
2

Bir sorun yok! Koda devam

## Birden fazla catch bloğu: Senaryo 2

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

### Çıktı penceresi

```
Enter the divisor:
2
Enter the array index:
```

## Birden fazla catch bloğu: Senaryo 2

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

### Çıktı penceresi

```
Enter the divisor:
2
Enter the array index:
-3
```

## Birden fazla catch bloğu: Senaryo 2

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

### Çıktı penceresi

```
Enter the divisor:
2
Enter the array index:
-3
```

**Yanlış dizi indisi:  
ArrayIndexOutOfBoundsException  
nesnesi fırlatıldı ve try bloğundan  
çıkıldı**

# Birden fazla catch bloğu: Senaryo 2

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

## Çıktı penceresi

```
Enter the divisor:
2
Enter the array index:
-3
```

Fırlatılan  
ArrayIndexOutOfBoundsException  
nesnesi burada yakalandı



# Birden fazla catch bloğu: Senaryo 2

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

## Çıktı penceresi

```
Enter the divisor:
2
Enter the array index:
-3
Array index is out of bounds!
```

Exception nesnesini  
yakalayan bloğun içine  
girildi

# Birden fazla catch bloğu: Senaryo 2

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}
```

```
System.out.println("Code must go on!");
```

## Çıktı penceresi

```
Enter the divisor:
2
Enter the array index:
-3
Array index is out of bounds!
Code must go on!
```

catch bloklarından  
sonra gelen satırla  
koda devam edilir



## Birden fazla catch bloğu: Senaryo 3

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

### Çıktı penceresi

Enter the divisor:

## Birden fazla catch bloğu: Senaryo 3

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

### Çıktı penceresi

```
Enter the divisor:
2
```

# Birden fazla catch bloğu: Senaryo 3

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

## Çıktı penceresi

Enter the divisor:  
2

Bir sorun yok! Koda devam

## Birden fazla catch bloğu: Senaryo 3

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

### Çıktı penceresi

```
Enter the divisor:
2
Enter the array index:
```

## Birden fazla catch bloğu: Senaryo 3

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

### Çıktı penceresi

```
Enter the divisor:
2
Enter the array index:
4
```

# Birden fazla catch bloğu: Senaryo 3

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}

System.out.println("Code must go on!");
```

## Çıktı penceresi

```
Enter the divisor:
2
Enter the array index:
4
```

Bir sorun yok! Koda devam



# Birden fazla catch bloğu: Senaryo 3

```
try{
    System.out.println("Enter the divisor: ");
    int divisor = input.nextInt();
    number = number / divisor;
    System.out.println("Enter the array index: ");
    int i = input.nextInt();
    arr[i] = number;
}
catch (ArithmeticException e){
    System.out.println("Division by zero is wrong!");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Array index is out of bounds!");
}
```

```
System.out.println("Code must go on!");
```

## Çıktı penceresi

```
Enter the divisor:
2
Enter the array index:
4
Code must go on!
```

Fırlatılan bir exception olmadığı için catch blokları hiç bir şey yakalayamadı!

catch bloklarından sonra gelen satırla koda devam edilir

# Birden fazla catch bloğu ne zaman kullanılmalı?

---

- Eğer amaç yalnızca bir şeylerin ters gittiğini anlamaksa ve her exception için aynı işlem yapılacaksa (ekrana mesaj yazdırmak gibi) tek bir catch bloğu kullanmak yeterlidir.
- Eğer hangi türde exceptionın oluştuğunu bilmek gerçekten gerekliyse, ve her exception için farklı işlemler yapılacaksa, birden fazla catch bloğu kullanılmalıdır.



# İççe try blokları

---

- Bir try bloğu başka try blokları içerebilir.

```
try {  
    ...  
    try {  
        ...  
    } catch (Exception e) {  
        ...  
    }  
    ...  
} catch (Exception e) {  
    ...  
}
```

# İççe try blokları

---

- Bir try bloğu içinde başka bir try bloğu olan bir metot çağırabilir

```
try {  
    ...  
    method();  
} catch (Exception e) {  
    ...  
}
```

```
void method() {  
    try {  
        ...  
    } catch (Exception e) {  
        ...  
    }  
}
```

# İççe try blokları

---

- Bir try bloğunun içinde bir exception oluştuğunda,
  - Eğer try bloğu uygun bir catch bloğuna sahip değilse, dıştaki try bloğunun catch bloklarına bakılır.
  - Eğer bir catch bloğu bulunursa çalıştırılır.
  - Eğer bulunamazsa daha dıştaki try bloklarının catch bloklarına bakılır.
  - Eğer eşleşen hiç bir catch bulunamazsa, exception JVM'nin exception handler'ı tarafından yakalanır.
- **UYARI: Exceptionın fırlatıldığı satıra bir daha geri dönülmez. Yani, bir exception bir catch bloğu tarafından yakalanırsa, catch bloğu çalıştırılır ve catch bloğunun bittiği yerden devam edilir.**

# Soru

---

```
try {  
    statement1;  
    try {  
        statement2;  
    } catch (Exception1 e) {  
        statement3;  
    } catch (Exception2 e) {  
        statement4;  
    }  
    try {  
        statement5;  
    } catch (Exception3 e) {  
        statement6;  
    }  
    statement7;  
} catch (Exception3 e) {  
    statement8;  
}  
statement9;
```

Exception1 ve Exception2  
Exception3'ün alt sınıfları olsun.

Aşağıdaki durumlarda hangi ifadeler  
çalıştırılır:

1. *statement1* Exception1 fırlatırsa
2. *statement2* Exception1 fırlatırsa
3. *statement2* Exception3 fırlatırsa
4. *statement2* Exception1 fırlatırsa ve  
*statement3* Exception2 fırlatırsa

# Soru1: statement1 Exception1 fırlatırsa

---

```
try {  
    statement1;  
    try {  
        statement2;  
    } catch (Exception1 e) {  
        statement3;  
    } catch (Exception2 e) {  
        statement4;  
    }  
    try {  
        statement5;  
    } catch (Exception3 e) {  
        statement6;  
    }  
    statement7;  
} catch (Exception3 e) {  
    statement8;  
}  
statement9;
```



**1.Exception1 fırlatıldı**

# Soru1: statement1 Exception1 fırlatırsa

```
try {
```

```
    statement1;
```

```
    try {
```

```
        statement2;
```

```
    } catch (Exception1 e) {
```

```
        statement3;
```

```
    } catch (Exception2 e) {
```

```
        statement4;
```

```
    }
```

```
    try {
```

```
        statement5;
```

```
    } catch (Exception3 e) {
```

```
        statement6;
```

```
    }
```

```
    statement7;
```

```
    } catch (Exception3 e) {
```

```
        statement8;
```

```
    }
```

```
    statement9;
```

**1.Exception1 fırlatıldı**

**2. try bloğunun catch'lerine bakıldı. Exception1 Exception3 'ün alt sınıfı olduğu için bu catch Exceptionı yakaladı**

# Soru1: statement1 Exception1 fırlatırsa

```
try {
```

```
    statement1;
```

```
    try {
```

```
        statement2;
```

```
    } catch (Exception1 e) {
```

```
        statement3;
```

```
    } catch (Exception2 e) {
```

```
        statement4;
```

```
    }
```

```
    try {
```

```
        statement5;
```

```
    } catch (Exception3 e) {
```

```
        statement6;
```

```
    }
```

```
    statement7;
```

```
    } catch (Exception3 e) {
```

```
        statement8;
```

```
    }
```

```
    statement9;
```

**1.Exception1 fırlatıldı**

**2. try bloğunun catch'lerine bakıldı. Exception1 Exception3 'ün alt sınıfı olduğu için bu catch Exceptionı yakaladı**

**3. catch bloğunun içindeki statement8 çalıştırıldı.**

# Soru1: statement1 Exception1 fırlatırsa

```
try {
```

```
    statement1;
```

```
    try {
```

```
        statement2;
```

```
    } catch (Exception1 e) {
```

```
        statement3;
```

```
    } catch (Exception2 e) {
```

```
        statement4;
```

```
    }
```

```
    try {
```

```
        statement5;
```

```
    } catch (Exception3 e) {
```

```
        statement6;
```

```
    }
```

```
    statement7;
```

```
    } catch (Exception3 e) {
```

```
        statement8;
```

```
    }
```

```
statement9;
```

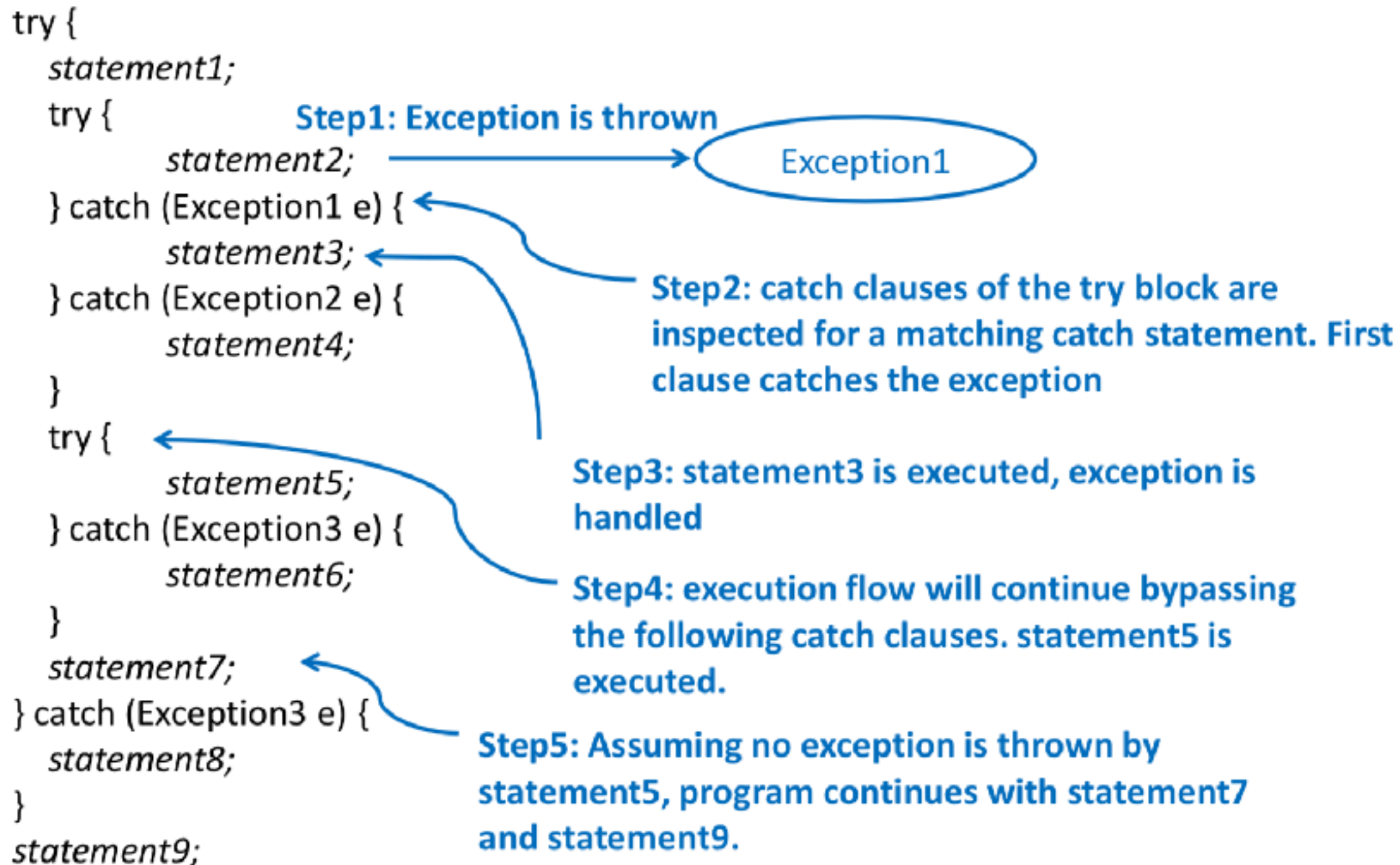
1.Exception1 fırlatıldı

2. try bloğunun catch'lerine bakıldı. Exception1 Exception3 'ün alt sınıfı olduğu için bu catch Exceptionı yakaladı

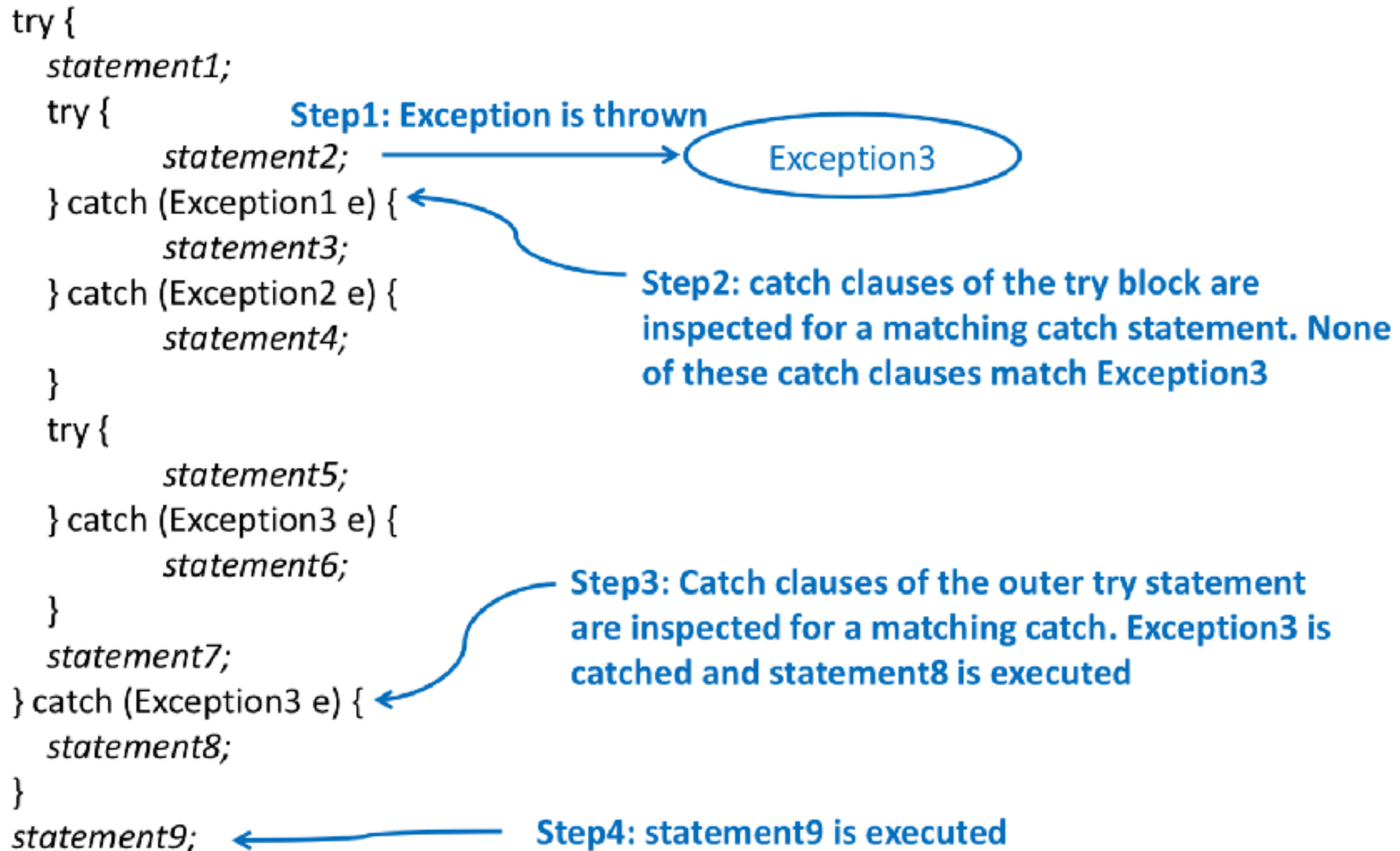
statement9 çalıştırıldı.



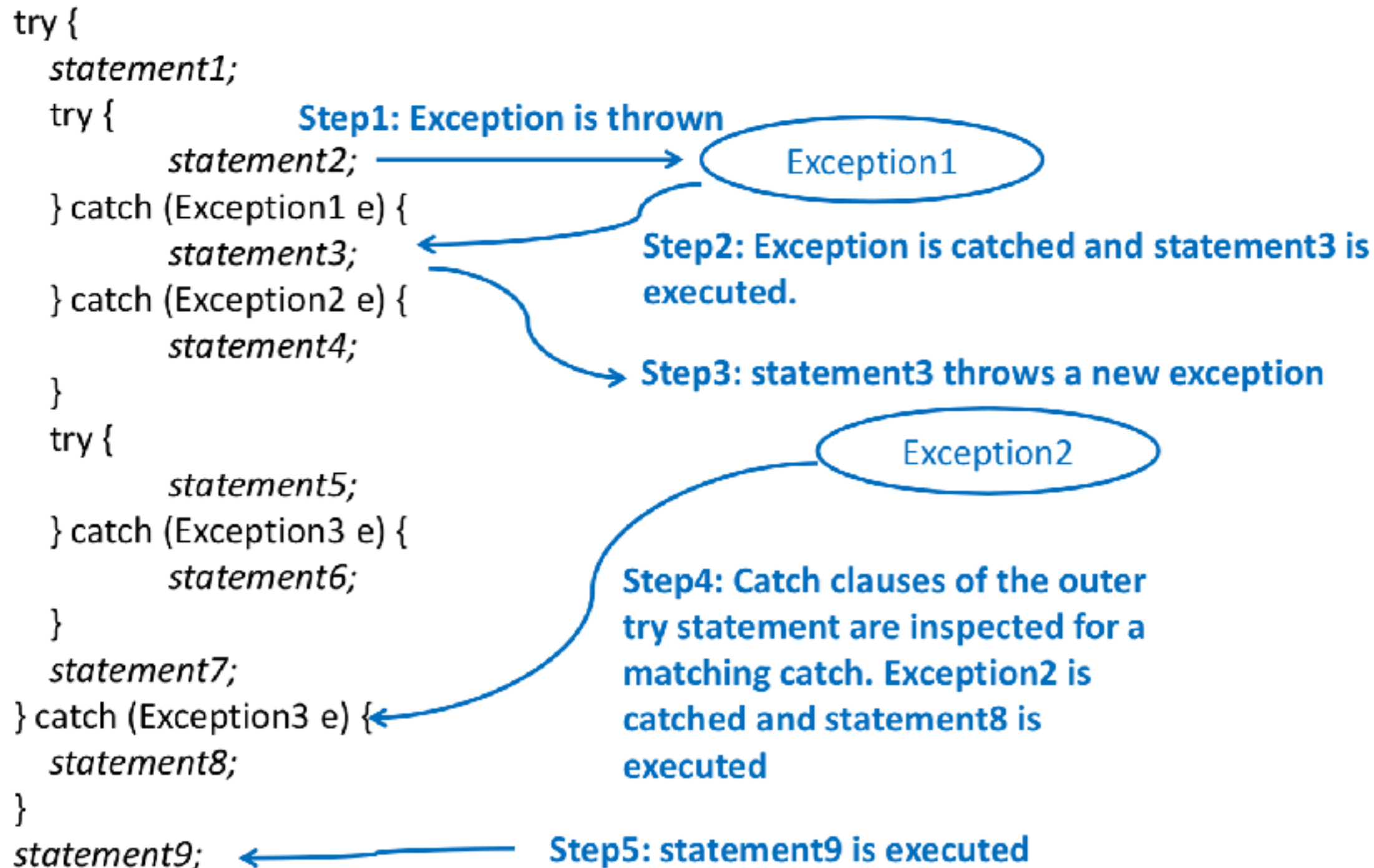
## Soru2: statement2 Exception1 fırlatırsa



## Soru3: statement2 Exception3 fırlatırsa



# Soru4: statement2 Exception1 fırlatırsa ve statement3 Exception2 fırlatırsa



# finally

---

- finally bloğu try/catch bloğu bittikten sonra ve try/catch bloğunu izleyen kod başlamadan önce oluşturulur.
- finally bloğu exception fırlatılsa da fırlatılmasa da çalıştırılır.
- finally bloğu exception yakalansa da yakalanmasa da çalıştırılır.
- finally kapsadığı kod bloğunun her durumda çalıştırılmasını garantiler.

## finally: Soru

---

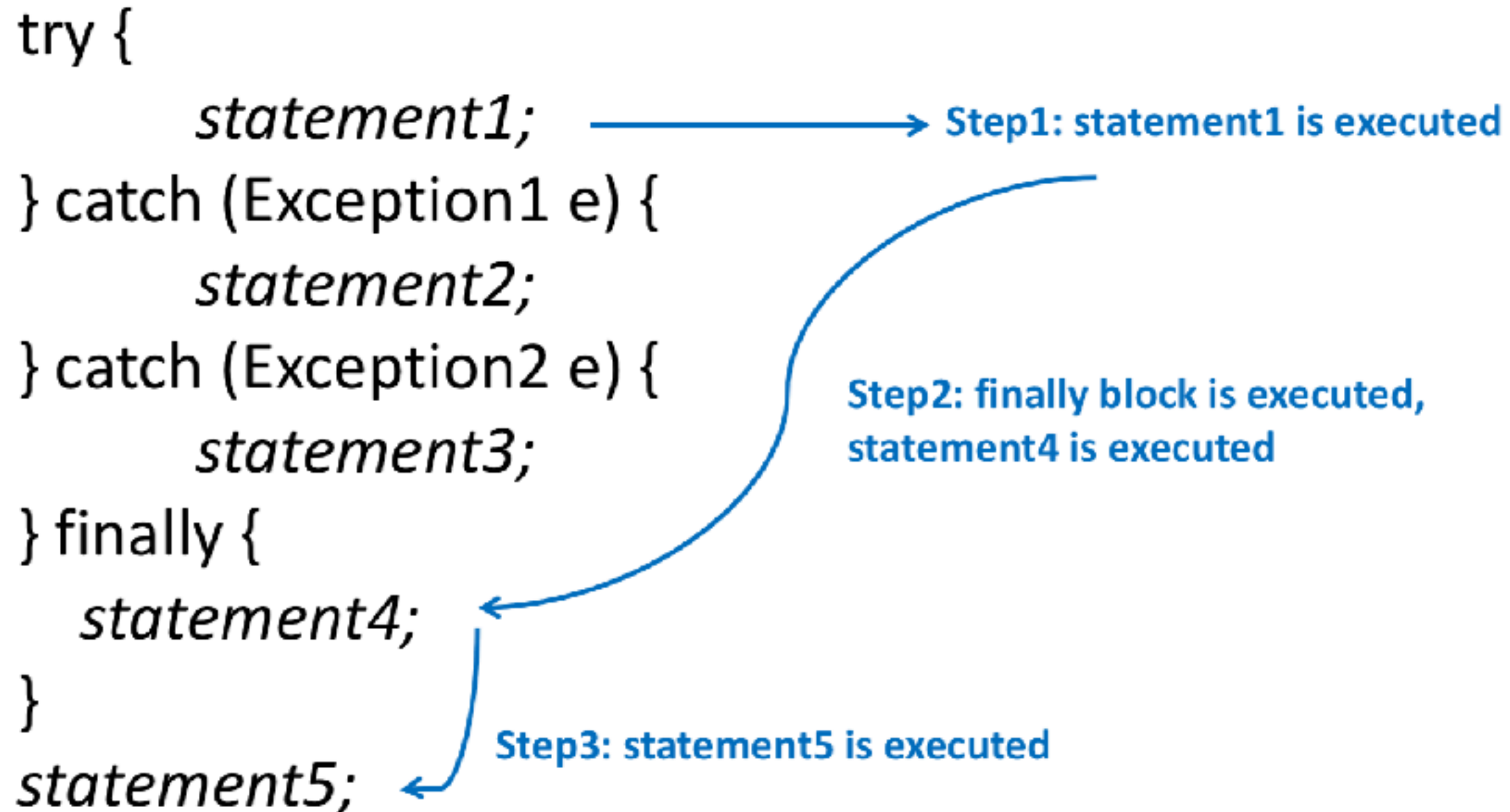
```
try {  
    statement1;  
} catch (Exception1 e) {  
    statement2;  
} catch (Exception2 e) {  
    statement3;  
} finally {  
    statement4;  
}  
statement5;
```

Aşağıdaki durumlarda hangi ifadeler çalıştırılır:

1. hiç exception oluşmazsa
2. *statement1* *Exception1* fırlatırsa
3. *statement1* *Exception3* fırlatırsa

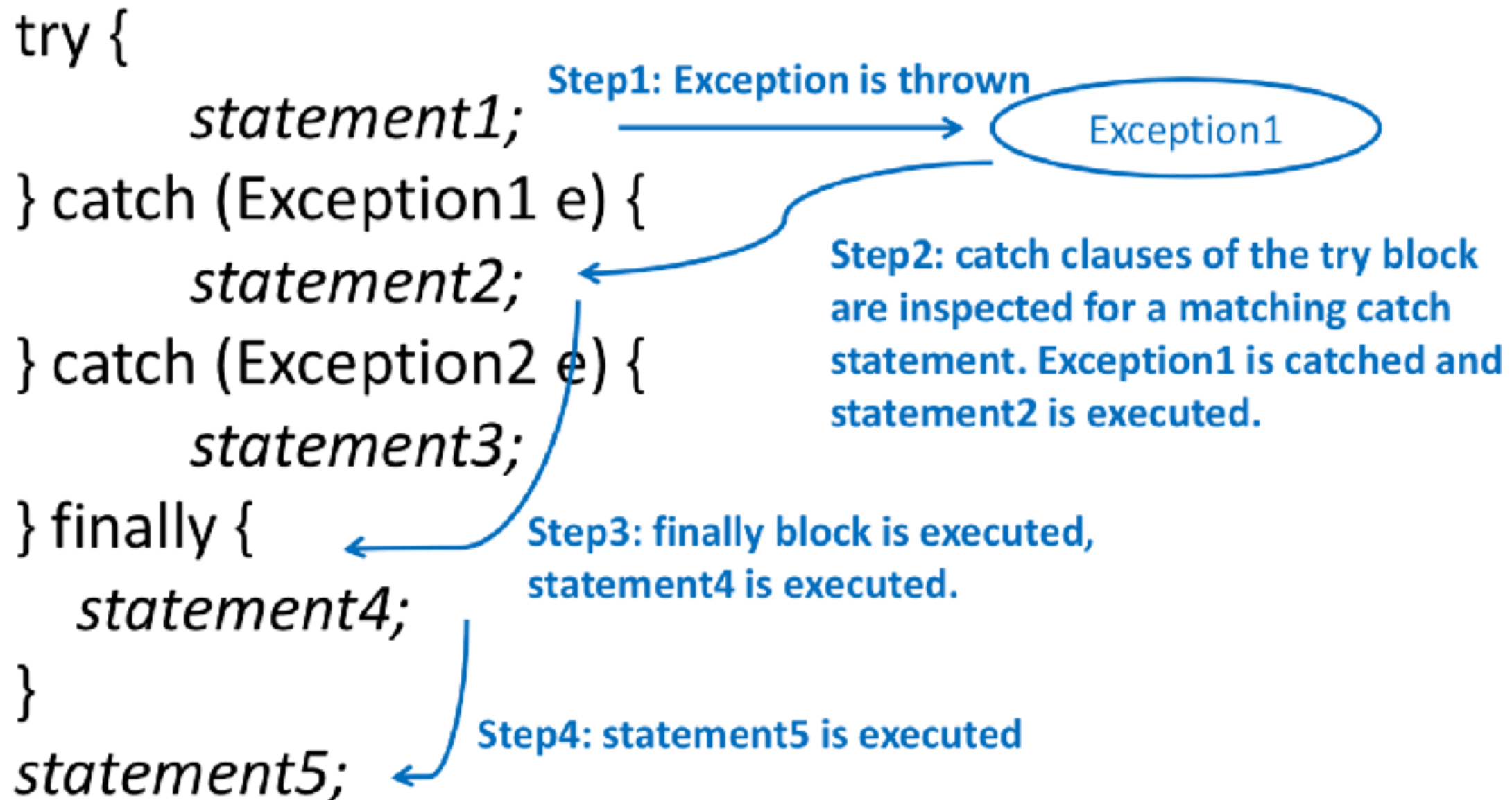
## Soru1: hiç exception oluşmazsa

---



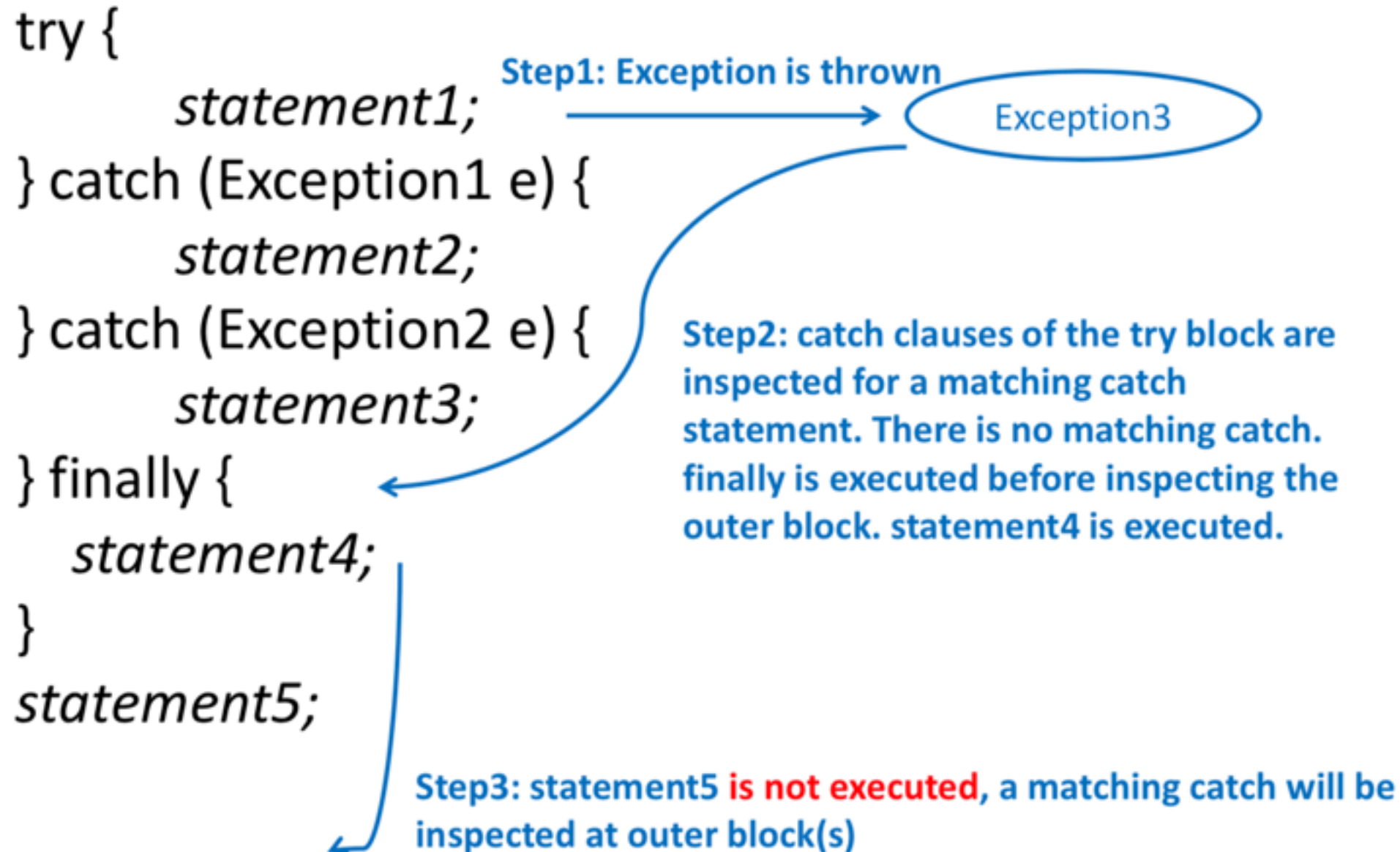
## Soru2: statement1 Exception1 fırlatırsa

---





## Soru3: statement1 Exception3 fırlatırsa





# throw

---

- Programcı manuel olarak da exception fırlatma komutunu verebilir. Bunun için throw anahtar kelimesi kullanılır.

throw ThrowableObject

- ThrowableObject fırlatılacak exception nesnesidir. Java'da tüm exceptionlar Throwable sınıfının alt sınıfıdır. Dolayısıyla ThrowableObject'in de Throwable sınıfının bir alt sınıfının nesnesi olması gerekir.
- Programcı bir exception sınıfının yeni bir nesnesini fırlatabilir veya yakalanmış bir exception yeniden fırlatabilir.

# Fırlatma(Throwing) ve yeniden fırlatma(rethrowing) örneği

```
import java.util.Scanner;

public class ThrowingExample {

    public static void main(String[] args){

        System.out.print("Give me an integer: ");
        Scanner input = new Scanner(System.in);
        int number = input.nextInt();
        try{
            if(number < 0)
                throw new RuntimeException();
            System.out.println("Thank you");
        }catch(Exception e){
            System.out.println("Number is less than zero");
            throw e;
        }
    }
}
```

yeni bir RuntimeException nesnesi oluşturulur ve fırlatılır

catch ile yakalanan e exception nesnesi catch bloğunun içinde yeniden fırlatılıyor.

# Özel exception sınıfları oluşturma

---

- Programcı programındaki anormal durumları yönetmek için kendisi özel exception sınıfları da yazabilir.
- Eğer bir sınıf Throwable sınıfının bir alt sınıfıysa, o sınıfın nesneleri fırlatılabilir.
- Özel exception sınıfları genellikle Exception veya RuntimeException sınıflarının alt sınıfları olarak tanımlanırlar. (Aradaki farkı daha sonra göreceğiz)

# Özel exception sınıfı: InvalidAgeException

```
public class InvalidAgeException extends Exception {  
    InvalidAgeException(String s){  
        super(s);  
    }  
}
```

Exception sınıfının  
yapılandırıcısını  
çağırıyor.

Herhangi bir sınıf gibi  
yapılandırıcıya sahip

Exception sınıfının alt  
sınıfı

# Özel exception sınıfı: InvalidAgeException

```
class TestCustomException1{  
    public static void main(String args[]){  
        System.out.print("Enter your age: ");  
        Scanner input = new Scanner(System.in);  
        int age = input.nextInt();  
        try{  
            if(age < 18)  
                throw new InvalidAgeException("age not valid, must be >= 18");  
        }  
        catch(Exception m)  
        {  
            System.out.println(m);  
        }  
        System.out.println("rest of the code...");  
    }  
}
```

InvalidAgeException nesnesi  
istenilen mesajla oluşturulur ve  
fırlatılır


# Özel exception sınıfı: InvalidAgeException

```
class TestCustomException1{  
    public static void main(String args[]){  
        System.out.print("Enter your age: ");  
        Scanner input = new Scanner(System.in);  
        int age = input.nextInt();  
        try{  
            if(age < 18)  
                throw new InvalidAgeException("age not valid, must be >= 18");  
        }  
        catch(Exception m) → fırlatılan InvalidAgeException nesnesi  
        {                                     burada yakalanır  
            System.out.println(m);  
        }  
        System.out.println("rest of the code...");  
    }  
}
```

# Özel exception sınıfı: InvalidAgeException

---

```
class TestCustomException1{  
    public static void main(String args[]){  
        System.out.print("Enter your age: ");  
        Scanner input = new Scanner(System.in);  
        int age = input.nextInt();  
        try{  
            if(age < 18)  
                throw new InvalidAgeException("age not valid, must be >= 18");  
        }  
        catch(Exception m)  
        {  
            System.out.println(m);  
        }  
        System.out.println("rest of the code...");  
    }  
}
```




m nesnesinin toString() metodu çağırılır

# Özel exception sınıfı: InvalidAgeException

---

## Çıktı

```
Enter your age: 11  
InvalidAgeException: age not valid, must be >= 18  
rest of the code...
```



**toString() metodu otomatik olarak  
Exception nesnesinin tipi ve  
yapılandırıcıda aldığı mesajı içerecek  
biçimde override edilmiştir.**



# getMessage() metodu

---

- Throwable sınıfının getMessage() metodu yapılandırıcıya parametre olarak verilen mesajı döndürür. Örneğin yukarıdaki örnekte catch bloğu şöyle olsaydı

```
catch(Exception m)
{
    System.out.println(m.getMessage());
}
```

Çıktı aşağıdaki gibi olurdu:

```
Enter your age: 11
age not valid, must be >= 18
rest of the code...
```

# printStackTrace() metodu

---

- Throwable sınıfının printStackTrace() metodu hatanın tipini, hata mesajını ve hatanın yerini ekrana yazdırır. Yukarıdaki örnekte catch bloğu şöyle olsaydı:

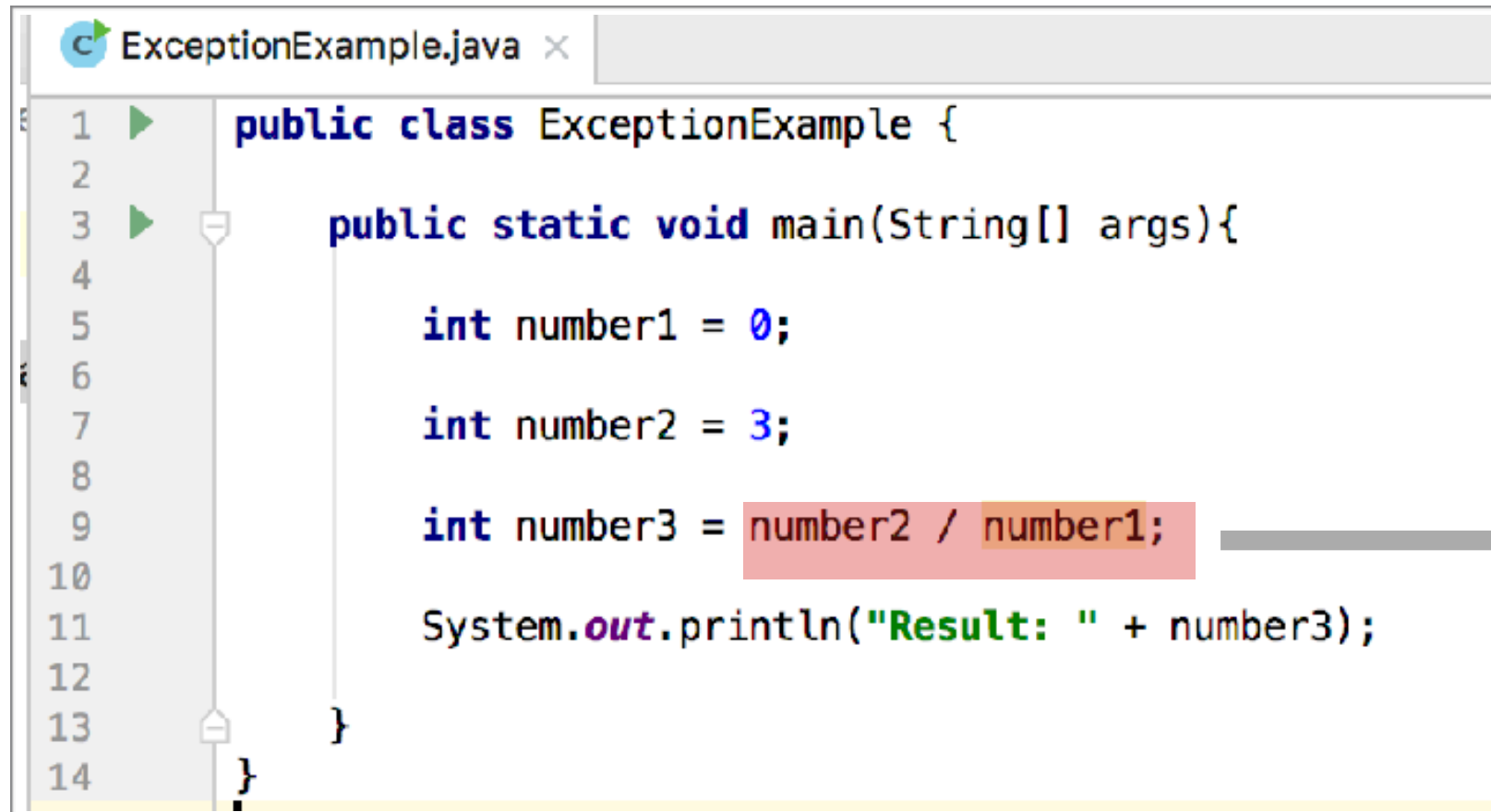
```
catch(Exception m)
{
    m.printStackTrace();
}
```

Çıktı aşağıdaki gibi olurdu:

```
Enter your age: 11
rest of the code...
InvalidAgeException: age not valid, must be >= 18
    at TestCustomException1.main(TestCustomException1.java:15)
```

# printStackTrace() metodu

- printStackTrace metodunun çıktısının bir exceptiondan dolayı program çalışmayı durdurduğunda gördüğümüz çıktılara çok benziyor. Bu benzerliğin sebebi, bir exception programda yakalanmadığında JVM onu yakalar ve stack trace'i ekrana yazdırır. Aşağıdaki örneği hatırlayalım:



```
1 public class ExceptionExample {  
2  
3     public static void main(String[] args){  
4  
5         int number1 = 0;  
6  
7         int number2 = 3;  
8  
9         int number3 = number2 / number1;  
10  
11         System.out.println("Result: " + number3);  
12  
13     }  
14 }
```

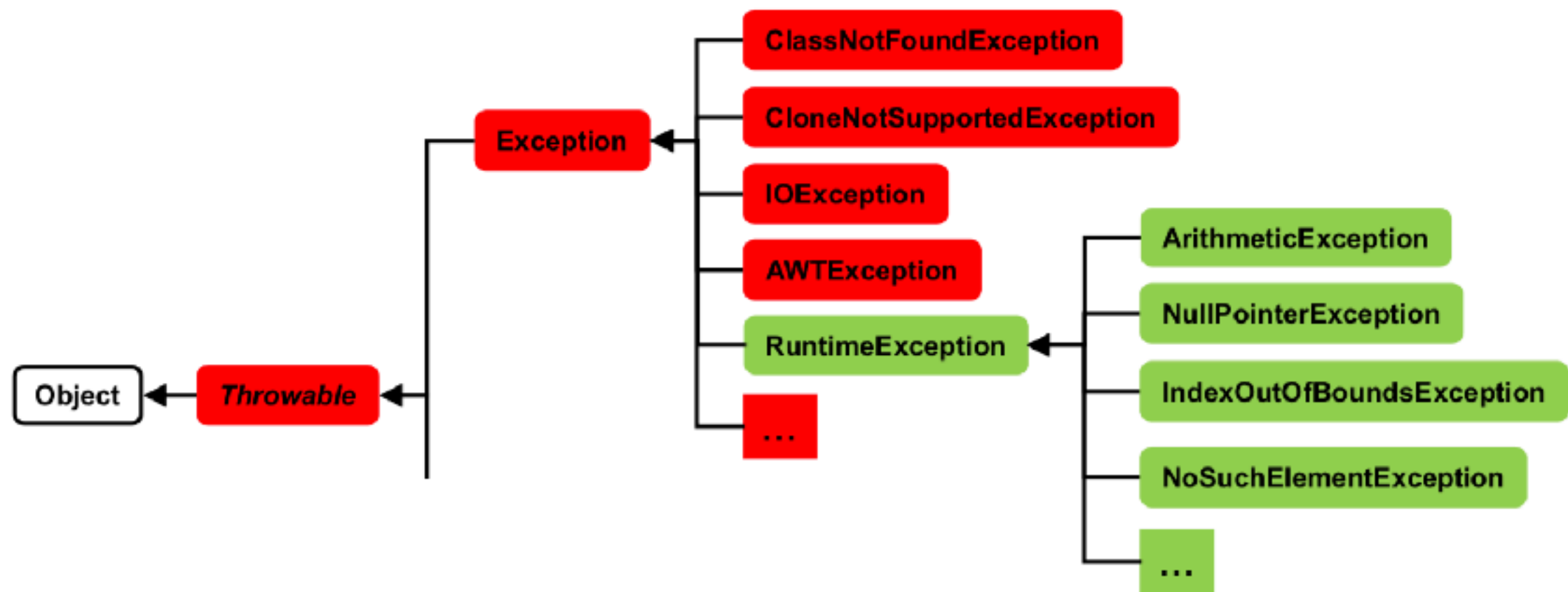
Sıfır ile bölme  
hatası

9. satırda program durdurulur ve aşağıdaki hata mesajı ekrana yazdırılır.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at ExceptionExample.main(ExceptionExample.java:9)
```

# Checked ve UncheckedExceptions

- Exceptionlar checked ve unchecked exception olmak üzere ikiye ayrılırlar. Aşağıdaki diagramda da görüldüğü üzere Unchecked exceptionlar yalnızca RuntimeException sınıfının alt sınıfları olanlardır. Dolayısıyla kendi özel exception sınıfımızı yazarken Exception'dan mı yoksa RuntimeException'dan mı kalıt olduğu önemlidir.



● Unchecked exceptions

● Checked exceptions

# Checked Exceptions

---

- Checked exceptionlar derleme zamanında kontrol edilen exceptionlardır.
- Eğer bir metodun içindeki bir kod checked exception fırlatıyorsa
  - Ya exception için bir try catch bloğu yazılmalıdır
  - Ya da metod throws anahtar kelimesiyle bu tipte bir exception fırlatabileceğini deklare etmelidir.

- **throws** anahtar kelimesi bir metodun exception(lar) fırlatabileceğini deklare etmek için kullanılır.
- Böylece bu metodu çağırانlar bu exception(lar)a karşı önlem alabilirler.
- ```
public void method1() throws Exception1{  
  
}  
  
public void method2() throws Exception1, Exception2, Exception3{  
  
}
```

# Checked Exceptions

---

- Exception sınıfının alt sınıfı olarak yazılan her sınıf bir Checked Exception'dır. Örneğin InvalidAgeException bir Checked exception'dır.

```
public class InvalidAgeException extends Exception {  
    InvalidAgeException(String s){  
        super(s);  
    }  
}
```

# Checked Exceptions

- Eğer InvalidAgeException'a karşı herhangi bir eylem (exception handling) yapılmazsa, derleme zamanı hatası oluşur.

```
class TestCustomException2{  
    public static void main(String args[]){  
        System.out.print("Enter your age: ");  
        Scanner input = new Scanner(System.in);  
        int age = input.nextInt();  
  
        if(age < 18)  
            throw new InvalidAgeException("age not valid, must be >= 18");  
        System.out.println("rest of the code...");  
    }  
}
```

Information: 25.12.2017 16:07 - Compilation completed with 1 error and 0 warnings in 1s 372ms

/Users/zumrakavafoglu/IdeaProjects/Ders14/src/TestCustomException2.java

Error:(15, 13) java: unreported exception InvalidAgeException; must be caught or declared to be thrown

# Checked Exceptions

---

- **Derleme zamanı hatasını engellemek için iki yol izlenebilir:**

1. uygun bir try/catch bloğu yazılır

```
class TestCustomException1{  
    public static void main(String args[]){  
        System.out.print("Enter your age: ");  
        Scanner input = new Scanner(System.in);  
        int age = input.nextInt();  
  
        try{  
            if(age < 18)  
                throw new InvalidAgeException("age not valid, must be >= 18");  
        }  
        catch(Exception m)  
        {  
            System.out.println(m);  
        }  
  
        System.out.println("rest of the code...");  
    }  
}
```



# Checked Exceptions

---

- **Derleme zamanı hatasını engellemek için iki yol izlenebilir:**  
2. yol: metodun bu tipte bir hata fırlatabileceği deklare edilir

```
class TestCustomException2{  
    public static void main(String args[]) throws InvalidAgeException{  
        System.out.print("Enter your age: ");  
        Scanner input = new Scanner(System.in);  
        int age = input.nextInt();  
  
        if(age < 18)  
            throw new InvalidAgeException("age not valid, must be >= 18");  
        System.out.println("rest of the code...");  
    }  
}
```

# Unchecked Exceptions

---

- Unchecked exceptionlar derleme zamanında kontrol edilmeyen exceptionlardır.
- Eğer bir kod bloğunun bir unchecked exception fırlatma ihtimali varsa, programcı aşağıdakilerden herhangi birini yapmaya karar verebilir:
  - exception handling
  - programın durmasına(crash) izin vermek

# Unchecked Exceptions : InvalidHeightException

---

```
public class InvalidHeightException extends RuntimeException {  
    public InvalidHeightException(String message){  
        super(message);  
    }  
}
```

```
▶ public class UncheckedExceptionExample {  
▶     public static void main(String args[]){  
        System.out.print("Enter your height: ");  
        Scanner input = new Scanner(System.in);  
        double height= input.nextDouble();  
        if(height <= 0)  
            throw new InvalidHeightException("height not valid, must be positive");  
        System.out.println("rest of the code...");  
    }  
}
```

# Unchecked Exceptions : InvalidHeightException

```
public class InvalidHeightException extends RuntimeException {  
    public InvalidHeightException(String message){  
        super(message);  
    }  
}
```

```
public class UncheckedExceptionExample {  
    public static void main(String args[]){  
        System.out.print("Enter your height: ");  
        Scanner input = new Scanner(System.in);  
        double height= input.nextDouble();  
        if(height <= 0)  
            throw new InvalidHeightException("height not valid, must be positive");  
        System.out.println("rest of the code...");  
    }  
}
```

Herhangi bir exception handling yapılmıyor, ancak Unchecked exception olduğu için bu bir derleme hatasına sebep olmaz

```
Enter your height: -3  
Exception in thread "main" InvalidHeightException: height not valid, must be positive  
at UncheckedExceptionExample.main(UncheckedExceptionExample.java:13)
```

# Checked ve Unchecked Exceptions

---

- Checked exceptionlar derleme zamanında kontrol edilen exceptionlardır.
- Eğer bir metodun içindeki bir kod checked exception fırlatıyorsa
  - Ya exception için bir try catch bloğu yazılmalıdır
  - Ya da metod throws anahtar kelimesiyle bu tipte bir exception fırlatabileceğini deklare etmelidir.

- **throws** anahtar kelimesi bir metodun exception(lar) fırlatabileceğini deklare etmek için kullanılır.
- Böylece bu metodu çağırانlar bu exception(lar)a karşı önlem alabilirler.
- ```
public void method1() throws Exception1{  
  
}  
  
public void method2() throws Exception1, Exception2, Exception3{  
  
}
```