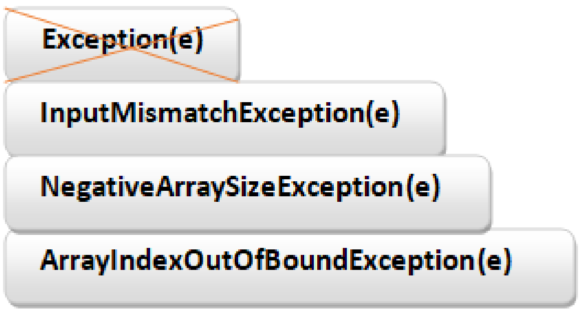
Exception handling continued….

Note:

Can you place a generic catch block on top of specific catch blocks??

In Java, the generic catch block can never be placed on top of the specific catch blocks. If attempted, it would result in an Error.


~~Exception(e)~~
InputMismatchException(e)
NegativeArraySizeException(e)
ArrayIndexOutOfBoundException(e)

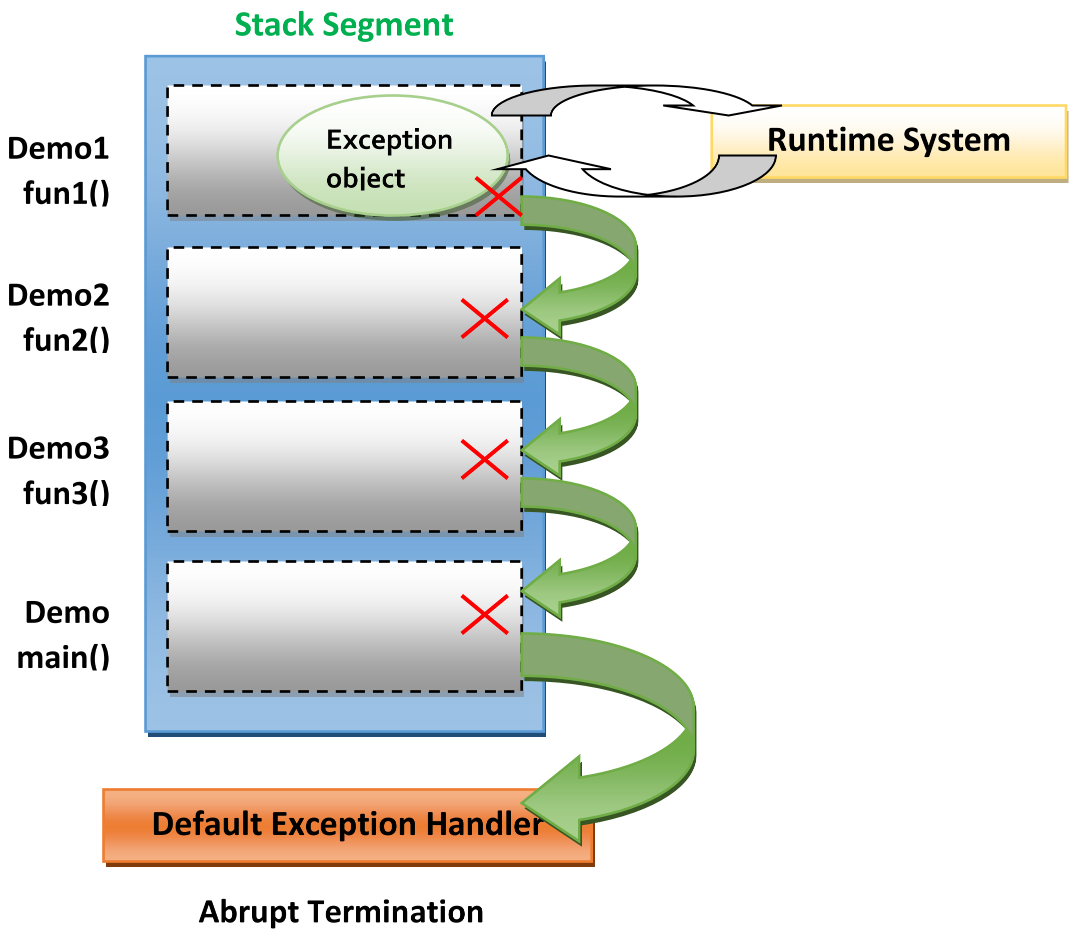
Now let us see how Runtime System works when multiple method calls are involved.

let us consider a program to understand:

```java
import java.util.Scanner;
class Demo1
{
    void fun1()
    {
        System.out.println("Connection4 is estd");
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the numerator");
        int a = scan.nextInt();
        System.out.println("Enter the denominator");
        int b = scan.nextInt();
        int c = a/b;
        System.out.println(c);
        System.out.println("Connection4 is terminated");
    }
}

class Demo2
{
    void fun2()
    {
        System.out.println("Connection3 is estd");
        Demo1 d1 = new Demo1();
        d1.fun1();
        System.out.println("Connection3 is terminated");
    }
}
```

```
class Demo3
{
    void fun3()
    {
        System.out.println("Connection2 is estd");
        Demo2 d2 = new Demo2();
        d2.fun2 ();
        System.out.println("Connection2 is terminated");
    }
}

class Demo
{
    public static void main(String[] args)
    {
        System.out.println("Connection1 is estd");
        Demo3 d3 = new Demo3();
        d3.fun3();
        System.out.println("Connection1 is terminated");
    }
}
```

Let us make use of Stack segment to understand the above program.



Explanation:

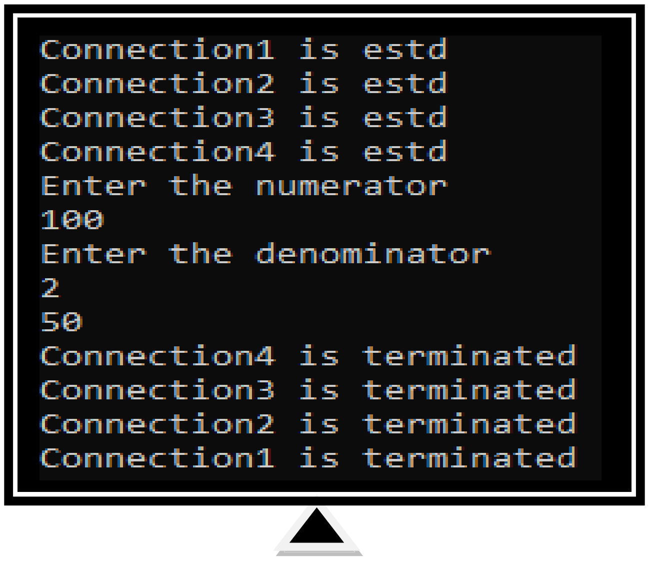
The first method that would get executed is the main method. Whenever a method is called their stack frame gets created as **Demo main()** in the Stack Segment. Next fun3() method is called and its stack frame gets created as **Demo3 fun3().** In the next step, fun3() method calls fun2() method and its stack frame gets created as **Demo2 fun2().** Later, fun2() method calls fun1() method and its stack frame gets created as **Demo1 fun1().**

Whenever there are multiple methods and if within a method an exception object gets generated it handles it to the Run time System and the Run time System checks if there is any try-catch block to handle the exception. If it is not there, then the Exception object does not directly goes to the default exception handler instead it gets propagated below the stack.

If the caller of the method also does not handle the exception with try-catch then it propagates below the stack and this continues until any of the method handles the exception.

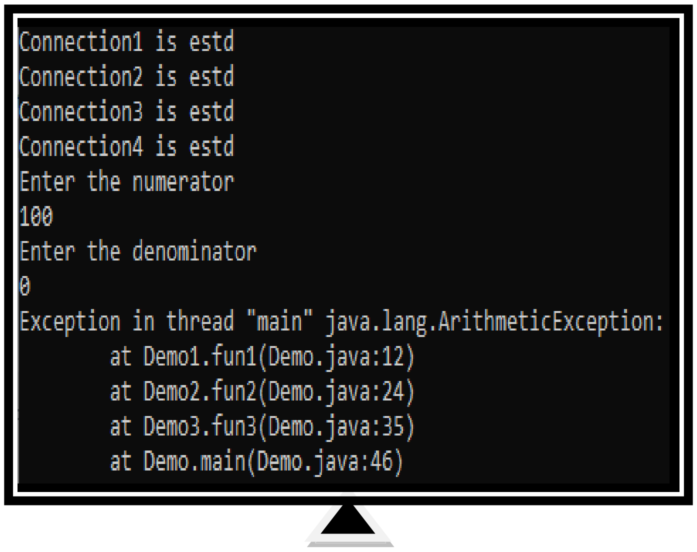
If none of the methods handles the exception then it reaches the Default Exception handler and Abrupt Termination happens.

Normal Termination



```
Connection1 is estd
Connection2 is estd
Connection3 is estd
Connection4 is estd
Enter the numerator
100
Enter the denominator
2
50
Connection4 is terminated
Connection3 is terminated
Connection2 is terminated
Connection1 is terminated
```

Abrupt Termination



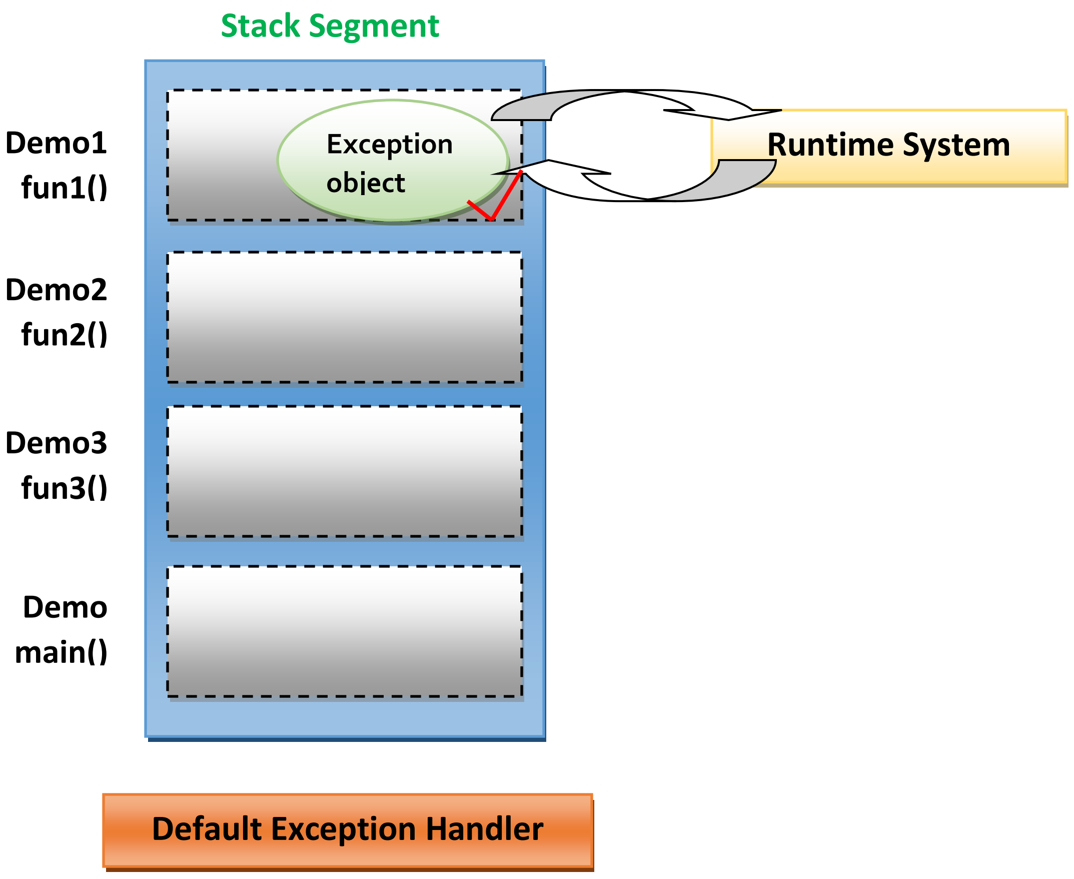
```
Connection1 is estd
Connection2 is estd
Connection3 is estd
Connection4 is estd
Enter the numerator
100
Enter the denominator
0
Exception in thread "main" java.lang.ArithmeticException:
        at Demo1.fun1(Demo.java:12)
        at Demo2.fun2(Demo.java:24)
        at Demo3.fun3(Demo.java:35)
        at Demo.main(Demo.java:46)
```

Let us see another example now with try-catch block given.

```java
import java.util.Scanner;
class Demo1
{
    void fun1()
    {
        System.out.println("Connection4 is estd");
        Scanner scan = new Scanner(System.in);
        try
        {
            System.out.println("Enter the numerator");
            int a = scan.nextInt();
            System.out.println("Enter the denominator");
            int b = scan.nextInt();
            int c = a/b;
            System.out.println(c);
        }
        catch (Exception e)
        {
            System.out.println("Some problem occured");
        }

        System.out.println("Connection4 is terminated");
    }
}

class Demo2
{
    void fun2()
    {
        System.out.println("Connection3 is estd");
        Demo1 d1 = new Demo1();
        d1.fun1();
        System.out.println("Connection3 is terminated");
    }
}

class Demo3
{
    void fun3()
    {
        System.out.println("Connection2 is estd");
        Demo2 d2 = new Demo2();
        d2.fun2 ();
        System.out.println("Connection2 is terminated");
    }
}
```

```
class Demo
{
    public static void main(String[] args)
    {
        System.out.println("Connection1 is estd");
        Demo3 d3 = new Demo3();
        d3.fun3();
        System.out.println("Connection1 is terminated");
    }
}
```

Let us make use of Stack segment to understand the above program.



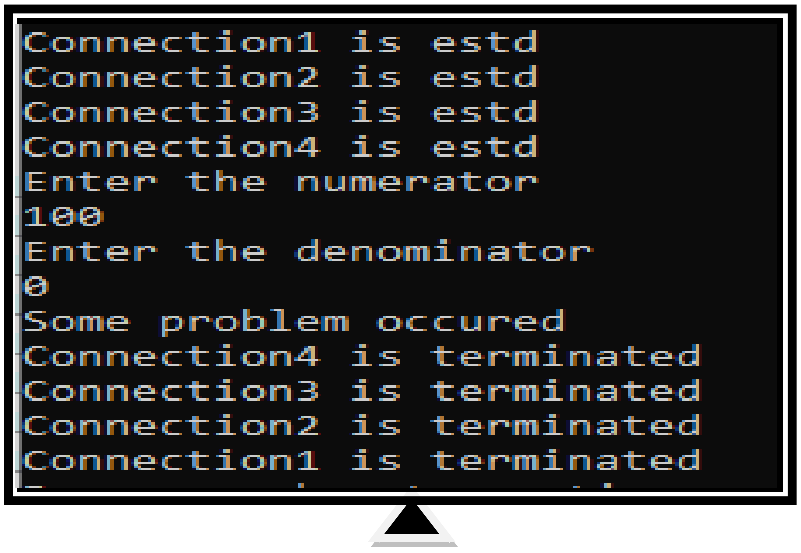
Explanation:

In this example we have given try-catch block inside a fun1() method with an assumption that an exception might occur.

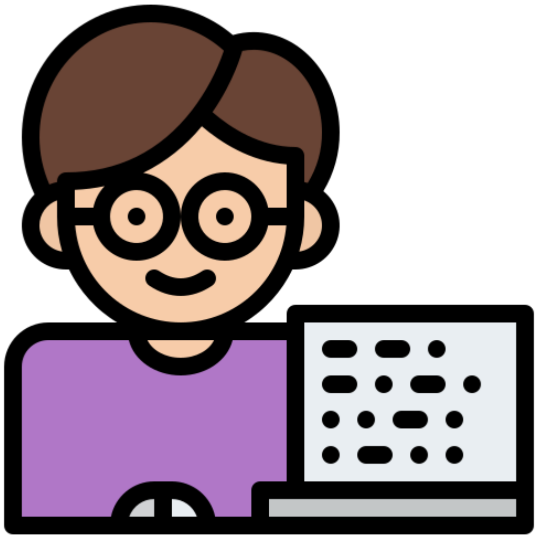
When an exception objects get generated in a method it is handed over to the Runtime System. Then the Runtime System checks if there is any try-catch block to handle the exception. In this example, the exception is caught by the catch block of fun1() method.

Once an exception is caught it does not gets propagate down to the caller methods and normal termination of the program happens.

Output:



```
Connection1 is estd
Connection2 is estd
Connection3 is estd
Connection4 is estd
Enter the numerator
100
Enter the denominator
0
Some problem occured
Connection4 is terminated
Connection3 is terminated
Connection2 is terminated
Connection1 is terminated
```



HE GOT THE OUTPUT. DID YOU??

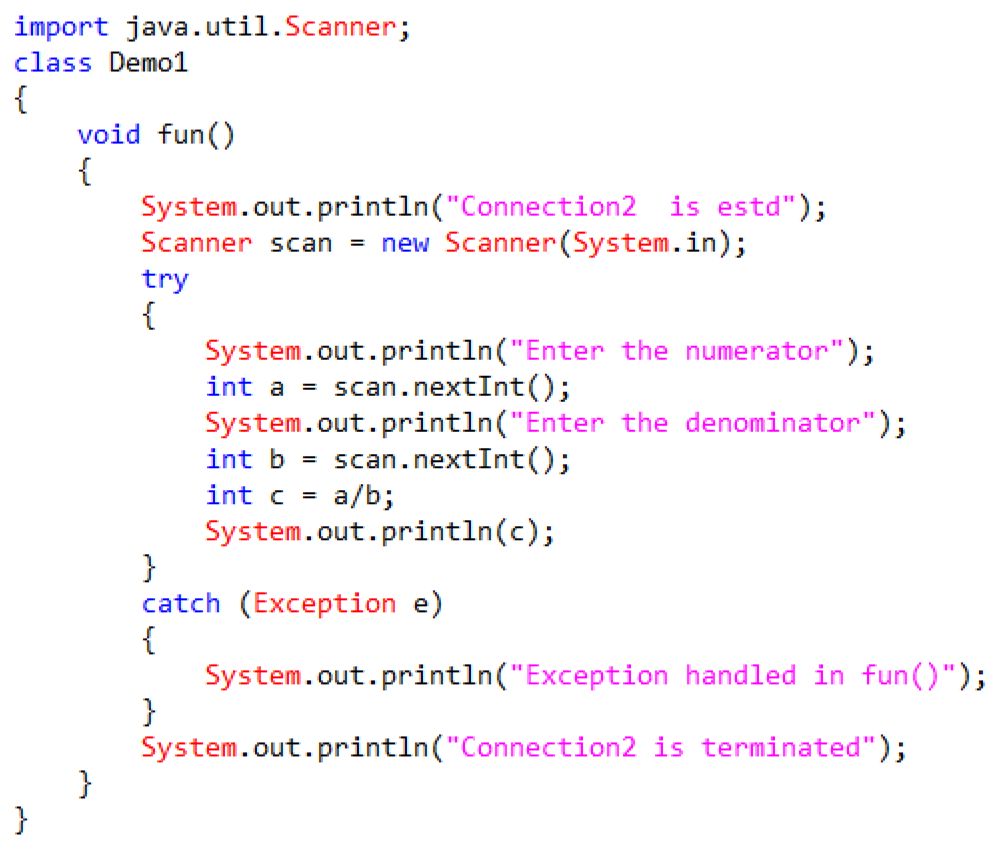
Different ways of handling the Exception.

1) **Handling the Exception(try-catch)**
2) **Re-throwing the Exception(try-catch-throw-throws-finally)**
3) **Ducking the Exception(throws)**

Case-1: Handling the exception.

If the methods in which exception occurs and the method in which handling occurs are one and the same then it is called Handling the exception.

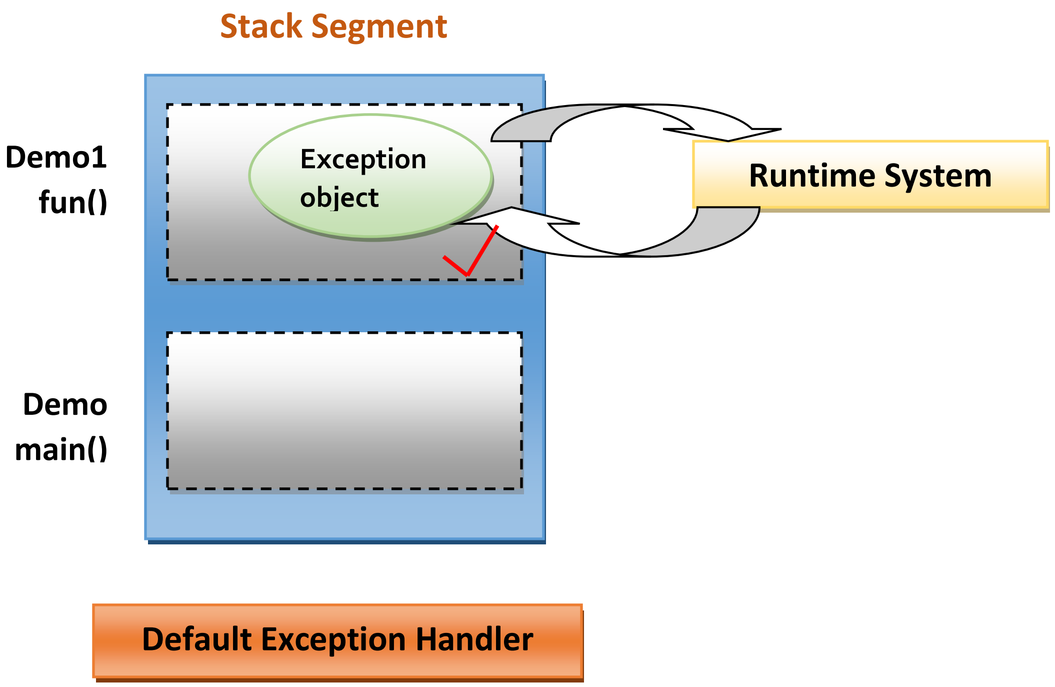
Let us understand this with an example



```java
import java.util.Scanner;
class Demo1
{
    void fun()
    {
        System.out.println("Connection2  is estd");
        Scanner scan = new Scanner(System.in);
        try
        {
            System.out.println("Enter the numerator");
            int a = scan.nextInt();
            System.out.println("Enter the denominator");
            int b = scan.nextInt();
            int c = a/b;
            System.out.println(c);
        }
        catch (Exception e)
        {
            System.out.println("Exception handled in fun()");
        }
        System.out.println("Connection2 is terminated");
    }
}
```

```
class Demo
{
    public static void main(String[] args)
    {
        System.out.println("Connection1 is estd");
        Demo1 d1 = new Demo1();
        d1.fun();
        System.out.println("Connection1 is terminated");
    }
}
```

Let us understand the program through Stack Segment Diagram.



Here in this program the exception gets generated inside fun() method and then it goes to the Runtime System. The Runtime System comes back to fun() methods and checks can the exception be handled in the same method.
Since there is a try-catch block to handle the exception it doesn't goes to the Default exception handler.
So according to our Case-1, exception occurs and gets handled in the same method.