Different ways of handling the Exception continued…

Case-2: Re-throwing the Exception (try-catch-throw-throws-finally)

If an exception occurs within a method and is also handled, but the exception must also **propagate to the caller of the method**, then it is referred to as **re-throwing the exception.**

To achieve this, me must use- try-catch-throw-throws-finally.

Throw keyword is used to **explicitly throw can exception** from a method or any block of code.

The **disadvantage** of throw keyword is that **statements below** throw keyword **do not execute**. This can be **overcome** by placing the statements within finally block.

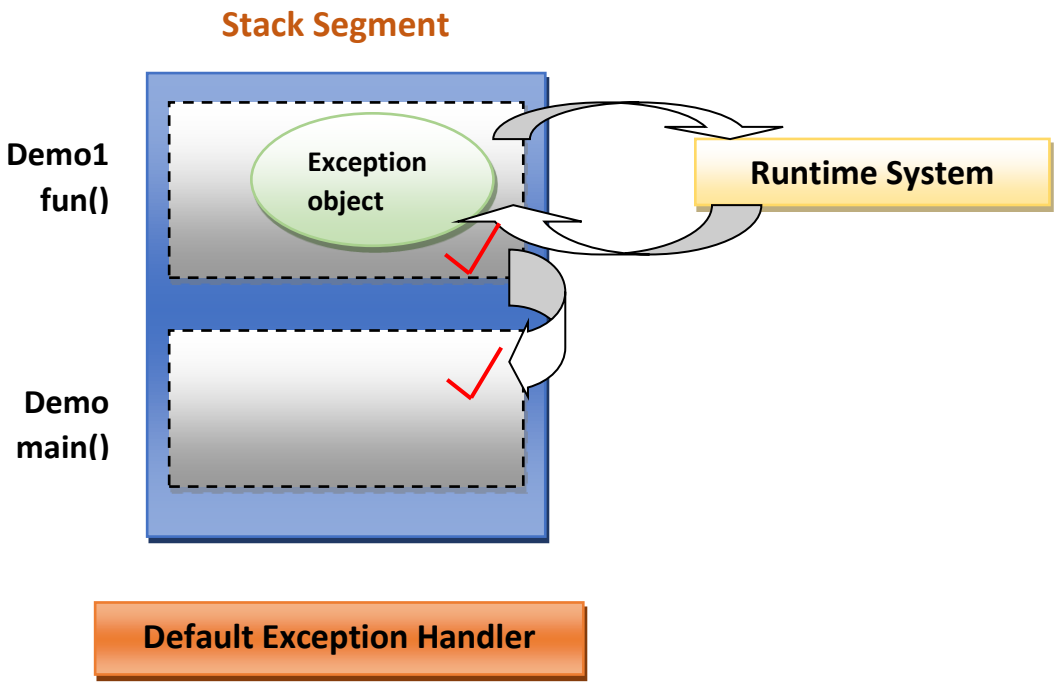
Throws keyword is used in the **signature of method** to indicate that this method Might throw an exception. The caller of this method must handle the exception using try-catch block.

Let us try writing code to understand this in detail


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

```
        finally
        {
            System.out.println("Connection2 is terminated");
        }
    }
}
class  Demo
{
    public static void main(String[] args)
    {
        System.out.println("Connection1 is established");
        try
        {
            Demo1 d1 = new Demo1();
            d1.fun();
        }
        catch (Exception e)
        {
            System.out.println("Exception handled in main()");
        }
        System.out.println("Connection1 is terminated");
    }
}
```

Let us understand the program through Stack Segment Diagram shown below:



Explanation:

Here in this program the exception gets generated inside fun() method in Demo1 class and then it goes to the Runtime System. The Runtime System comes back to fun() method 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 go to the Default exception handler. We are handling the exception and throwing it to main() in class Demo. Before the exception is thrown, the statements inside finally block get executed and then exception is handled again in main() using try-catch.

So according to our Case-2, **exception occurs, gets handled in the same method and is thrown to the caller method, which is nothing but rethrowing the exception.**

Output:

```
Connection1 is established
Connection2 is established
Enter numerator
100
Enter denominator
0
Exception handled in fun()
Connection2 is terminated
Exception handled in main()
Connection1 is terminated
```

Case-3: Ducking the Exception(throws)

If an exception occurs within a method and the **method does not want to handle** it, instead it wants to **handover the responsibility** of handling the exception to **caller of the method**, it is referred to as **ducking an exception**.

To **achieve** this, **throws** keyword is used.

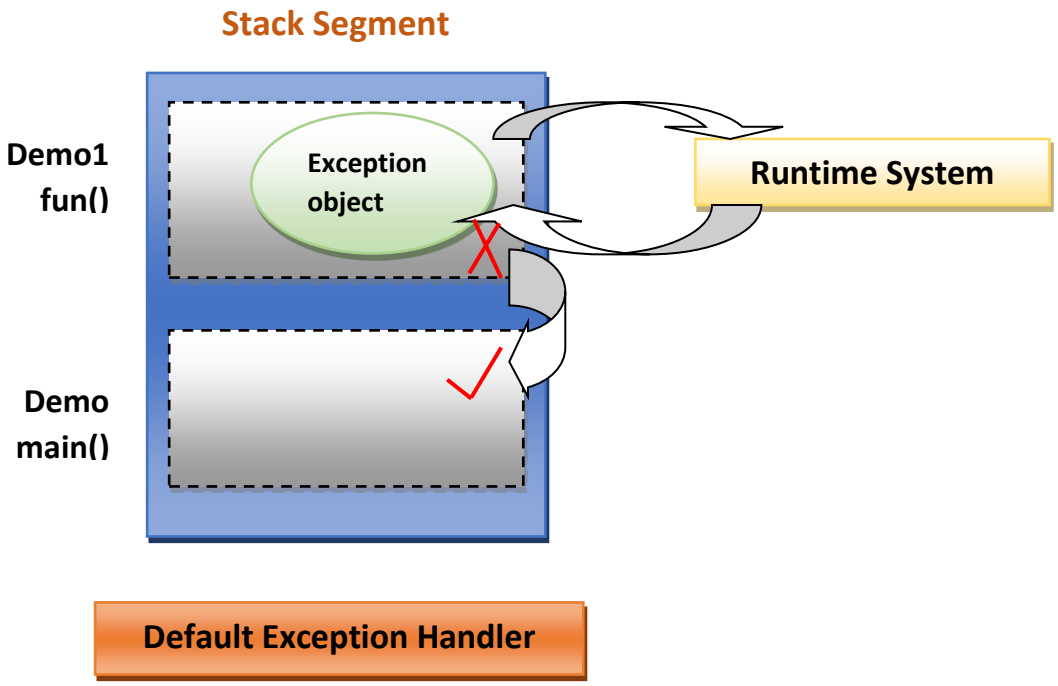
Let us try writing code to understand this case

```
import java.util.Scanner;
class Demo1
{
    void fun() throws Exception
    {
        System.out.println("Connection2 is established");
        Scanner scan = new Scanner(System.in);

            System.out.println("Enter numerator");
            int a = scan.nextInt();
            System.out.println("Enter denominator");
            int b = scan.nextInt();
            int c = a/b;
            System.out.println(c);

            System.out.println("Connection2 is terminated");
    }
}
class  Demo
{
    public static void main(String[] args)
    {
        System.out.println("Connection1 is established");
        try
        {
            Demo1 d1 = new Demo1();
            d1.fun();
        }
        catch (Exception e)
        {
            System.out.println("Exception handled in main()");
        }
        System.out.println("Connection1 is terminated");
    }
}
```

Let us understand the program through Stack Segment Diagram shown below:



Explanation:

Here in this program the exception gets generated inside fun() method in Demo1 class and then it goes to the Runtime System. The Runtime System comes back to fun() method and checks can the exception be handled in the same method.

Since there is throws keyword used in method signature it doesn't go to the Default exception handler, the control is given to main() and it is handled in main() using try-catch. Due to which **connection2 is terminated** is not printed as the control is given to main() once the exception occurs.

So according to our Case-3, **using throws we are not handling the exception in the method in which it occurs instead we are ducking it.**

Output:

```
Connection1 is established
Connection2 is established
Enter numerator
100
Enter denominator
0
Exception handled in main()
Connection1 is terminated
```