Java Finally block – Exception handling

BY CHAITANYA SINGH | FILED UNDER: EXCEPTION HANDLING

In the previous tutorials I have covered try-catch block and nested try block. In this guide, we will see finally block which is used along with try-catch.
A **finally block** contains all the crucial statements that must be executed whether exception occurs or not. The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.

Syntax of Finally block

```
try {
    //Statements that may cause an exception
}
catch {
   //Handling exception
}
finally {
   //Statements to be executed
}
```

A Simple Example of finally block

Here you can see that the exception occurred in try block which has been handled in catch block, after that finally block got executed.

```
class Example
{
   public static void main(String args[]) {
      try{
          int num=121/0;
          System.out.println(num);
      }
      catch(ArithmeticException e){
         System.out.println("Number should not be divided by zero");
      }
      /* Finally block will always execute
       * even if there is no exception in try block
       */
      finally{
          System.out.println("This is finally block");
      }
      System.out.println("Out of try-catch-finally");
   }
}
```

Output:

```
Number should not be divided by zero
This is finally block
Out of try-catch-finally
```

Few Important points regarding finally block

1. A finally block must be associated with a try block, you cannot use finally without a try block. You should place those statements in this block that must be executed always.

2. Finally block is optional, as we have seen in previous tutorials that a try-catch block is sufficient for exception handling, however if you place a finally block then it will always run after the execution of try block.

3. In normal case when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.

4. An exception in the finally block, behaves exactly like any other exception.

5. The statements present in the **finally block** execute even if the try block contains control transfer statements like return, break or continue.
Lets see an example to see how finally works when return statement is present in try block:

Another example of finally block and return statement

You can see that even though we have return statement in the method, the finally block still runs.

```
class JavaFinally
{
   public static void main(String args[])
   {
      System.out.println(JavaFinally.myMethod());
   }
   public static int myMethod()
   {
      try {
        return 112;
      }
      finally {
        System.out.println("This is Finally block");
        System.out.println("Finally block ran even after return statement");
      }
   }
}
```

Output of above program:

```
This is Finally block
Finally block ran even after return statement
112
```

To see more examples of finally and return refer: Java finally block and return statement
.

Cases when the finally block doesn’t execute

The circumstances that prevent execution of the code in a finally block are:
– The death of a Thread
– Using of the System. exit() method.
– Due to an exception arising in the finally block.

Finally and Close()

close() statement is used to close all the open streams in a program. Its a good practice to use close() inside finally block. Since finally block executes even if exception occurs so you can be sure that all input and output streams are closed properly regardless of whether the exception occurs or not.

For example:

```
....
try{
    OutputStream osf = new FileOutputStream( "filename" );
    OutputStream osb = new BufferedOutputStream(opf);
    ObjectOutput op = new ObjectOutputStream(osb);
    try{
       output.writeObject(writableObject);
    }
    finally{
       op.close();
    }
}
catch(IOException e1){
     System.out.println(e1);
}
...
```

Finally block without catch

A try-finally block is possible without catch block. Which means a try block can be used with finally without having a catch block.

```
...
InputStream input = null;
try {
    input = new FileInputStream("inputfile.txt");
}
finally {
    if (input != null) {
       try {
```

```
         in.close();
       }catch (IOException exp) {
           System.out.println(exp);
        }
    }
}
...
```

Finally block and System.exit()

System.exit() statement behaves differently than **return statement**. Unlike return statement whenever System.exit() gets called in try block then **Finally block** doesn't execute. Here is a code snippet that demonstrate the same:

```
....
try {
   //try block
   System.out.println("Inside try block");
   System.exit(0)
}
catch (Exception exp) {
   System.out.println(exp);
}
finally {
   System.out.println("Java finally block");
}
....
```

In the above example if the **System.exit(0)** gets called without any exception then finally won't execute. However if any exception occurs while calling **System.exit(0)** then finally block will be executed.

try-catch-finally block

- Either a try statement should be associated with a catch block or with finally.
- Since catch performs exception handling and finally performs the cleanup, the best approach is to use both of them.

Syntax:

```
try {
     //statements that may cause an exception
}
catch (…) {
     //error handling code
}
finally {
    //statements to be executed
}
```

Examples of Try catch finally blocks

Example 1: The following example demonstrate the working of finally block when no exception occurs in try block

```
class Example1{
  public static void main(String args[]){
    try{
       System.out.println("First statement of try block");
       int num=45/3;
       System.out.println(num);
    }
    catch(ArrayIndexOutOfBoundsException e){
       System.out.println("ArrayIndexOutOfBoundsException");
    }
    finally{
       System.out.println("finally block");
    }
    System.out.println("Out of try-catch-finally block");
  }
}
```

Output:

```
First statement of try block
15
finally block
Out of try-catch-finally block
```

Example 2: This example shows the working of finally block when an exception occurs in try block but is not handled in the catch block:

```
class Example2{
   public static void main(String args[]){
     try{
        System.out.println("First statement of try block");
        int num=45/0;
        System.out.println(num);
     }
     catch(ArrayIndexOutOfBoundsException e){
        System.out.println("ArrayIndexOutOfBoundsException");
     }
     finally{
        System.out.println("finally block");
     }
     System.out.println("Out of try-catch-finally block");
   }
}
```

Output:

```
First statement of try block
finally block
Exception in thread "main" java.lang.ArithmeticException: / by zero
at beginnersbook.com.Example2.main(Details.java:6)
```

As you can see that the system generated exception message is shown but before that the finally block successfully executed.

Example 3: When exception occurs in try block and handled properly in catch block

```
class Example3{
   public static void main(String args[]){
      try{
         System.out.println("First statement of try block");
         int num=45/0;
         System.out.println(num);
      }
      catch(ArithmeticException e){
         System.out.println("ArithmeticException");
      }
      finally{
         System.out.println("finally block");
      }
      System.out.println("Out of try-catch-finally block");
   }
}
```

Output:

```
First statement of try block
ArithmeticException
finally block
Out of try-catch-finally block
```