L48

Friday, February 4, 2022 9:02 PM

Usage of try, catch and finally

Usage of try, catch and finally

Q1. The syntax for try-catch-finally syntax is as given below:

try **{***// try block***}** catch (*ExceptionName1referenceName1*) **{***// catch block***}** catch (*ExceptionName2referenceName2*) **{***// another catch block***}** finally **{***// finally block***}**

The block of code which is just after the try keyword is called the try block.

The block of code which is just after the catch keyword is called the catch block.

The block of code which is just after the finally keyword is called the finally block.

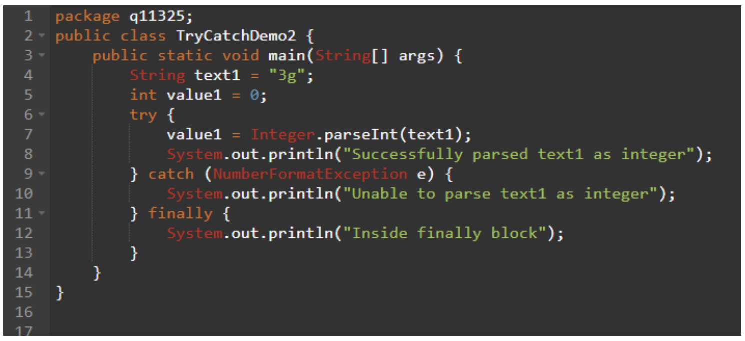
There can be a try block with only catch blocks and without the finally block.

Similarly there can be a try block without catch blocks and with just one finally block.

Below is an example which demonstrates the usage of try, catch and finally.

Note that in the below code **line 6** : Integer.parseInt(text1); will throw an NumberFormatException, since reference text1 has a value "3g" which cannot be parsed into an integer.

When the exception is thrown, the try block does not complete normally, meaning the control flow will jump from **line 6** into the catch block. And after the catch block is executed it will later enter into the finally block and execute the code it.



```
1   package q11325;
2   public class TryCatchDemo2 {
3       public static void main(String[] args) {
4           String text1 = "3g";
5           int value1 = 0;
6           try {
7               value1 = Integer.parseInt(text1);
8               System.out.println("Successfully parsed text1 as integer");
9           } catch (NumberFormatException e) {
10              System.out.println("Unable to parse text1 as integer");
11          } finally {
12              System.out.println("Inside finally block");
13          }
14      }
15  }
16
17
```

Q2. The try block can have zero or more catch blocks. However, a try block can have zero or **only one** finally block.

Note that if a try block has one or more catch blocks, then the finally block should be written only after the last catch block.

The finally block cannot appear before or in between the catch blocks.

Valid
try **{***// try block***}** catch (*ExceptionName1referenceName1*) **{***// catch block***}** catch (*ExceptionName2referenceName2*) **{***// another catch block***}** finally **{***// finally block***}**

Valid
try **{***// try block***}** catch (*ExceptionClassNamereferenceName*) **{***// catch block***}**

Valid
try **{***// try block***}** finally **{***// finally block***}**

Select all the correct usages of try, catch and finally blocks given below:

☐
```
try {
   ...
} finally {
   ...
} catch  (ExceptionClassName  referenceName) {
   ...
}
```

☐
```
catch {
   ...
} finally {
   ...
} try  (ExceptionClassName  referenceName) {
   ...
}
```

☐
```
try {
   ...
} finally (ExceptionClassName  referenceName) {
   ...
} catch {
   ...
}
```

[✓]
```
try {
   ...
} catch (ExceptionClassName  referenceName) {
   ...
} finally {
   ...
}
```

[✓]
```
try {
   ...
} finally {
   ...
}
```

Q3. Below are some important rule on how try, catch and finally blocks work.

If the code in the try block raises an exception and if that exception is caught by a catch block, then the control is abruptly transferred to that catch block.

If no exception is raised by the code in the try block, the code in the try block executes normally.

A finally block if present is always executed. If there is no exception, finally block is executed after the code in the try block is executed.

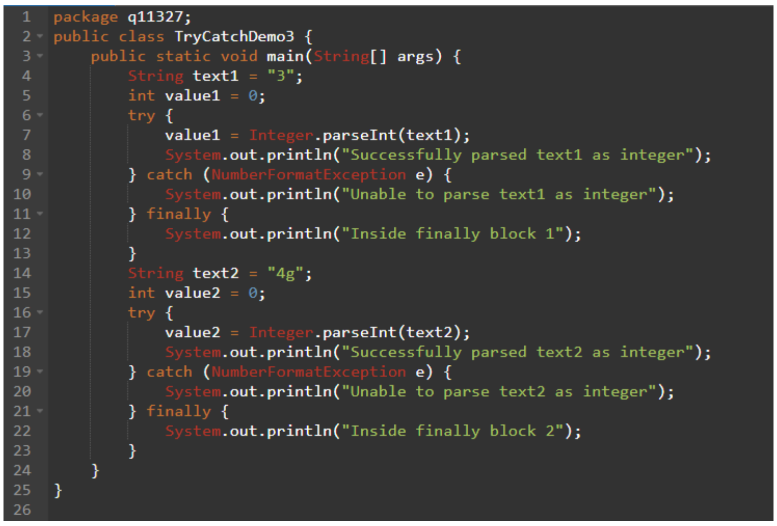
If there is an exception in the try block and there is no corresponding catch block to catch the exception, then the control flow which is abruptly transferred from the try block enters into the finally block.

If there is an exception in the try block and there is a corresponding catch block to catch the exception, then the control flow which is abruptly transferred from the try block enters into the catch block first and later enters into the finally block.

The below code demonstrates the above mentioned rules.

See and retype the below code.

Note : Observe in the output how the catch block is skipped while parsing text1, because no exception is raised.



```java
package q11327;
public class TryCatchDemo3 {
    public static void main(String[] args) {
        String text1 = "3";
        int value1 = 0;
        try {
            value1 = Integer.parseInt(text1);
            System.out.println("Successfully parsed text1 as integer");
        } catch (NumberFormatException e) {
            System.out.println("Unable to parse text1 as integer");
        } finally {
            System.out.println("Inside finally block 1");
        }
        String text2 = "4g";
        int value2 = 0;
        try {
            value2 = Integer.parseInt(text2);
            System.out.println("Successfully parsed text2 as integer");
        } catch (NumberFormatException e) {
            System.out.println("Unable to parse text2 as integer");
        } finally {
            System.out.println("Inside finally block 2");
        }
    }
}
```

Q4. In Java 7 and later versions, multiple catch blocks can be combined into a single block.

For example the two catch blocks in the below code:
try **{***// try block***}** catch (ExceptionClassName1 *referenceName1*) **{***// catch block***}** catch (ExceptionClassName2 *referenceName2*) **{***// another catch block***}** finally **{***// finally block***}**
can be combined into a single catch block as given below:
try **{***// try block***}** catch (ExceptionClassName1 | ExceptionClassName2 *referenceName*) **{***// multi-catch block***}** finally **{***// finally block***}**
Note that only Throwable and its subclasses can be caught in the catch statement.

It is always a good practice to catch the exceptions and also print their stack trace by calling the printStackTrace() method. The **printStackTrace()** method is present in the superclass Throwable, hence it is available in every exception class.

Note there may be situations when you may not want call the **printStackTrace()** method, however let it be a conscious decision.

The finally block is very useful for writing the cleanup code. Since the finally block is always executed after the try block, any code that is written inside the finally block will be executed before the control is transferred either by a **break** statement or a **continue** statement or even by a **return** statement.

In **Java 7** there is a new construct called **try-with-resources**. We will learn more about it later in sections related to streams in **java.io** package.

Select all the correct statements for the below code:

```
public class TrickyExample {
        public static void main(String[] args) {
                String text1 = "3";
                String text2 = "4g";
                System.out.println(getTotal(text1, text2));
        }
        public static int getTotal(String text1, String text2) {
                int value1 = 0;
                int value2 = 0;
                try {
                        value1 = Integer.parseInt(text1);
                        value2 = Integer.parseInt(text2);
                        return value1 + value2;
                } catch (NumberFormatException e) {
                        return -1;
                } finally {
                        return -2;
                }
        }
}
```

☐ The code will print `3`.

☐ The code will print `34g`.

☐ The code will print `-1`.

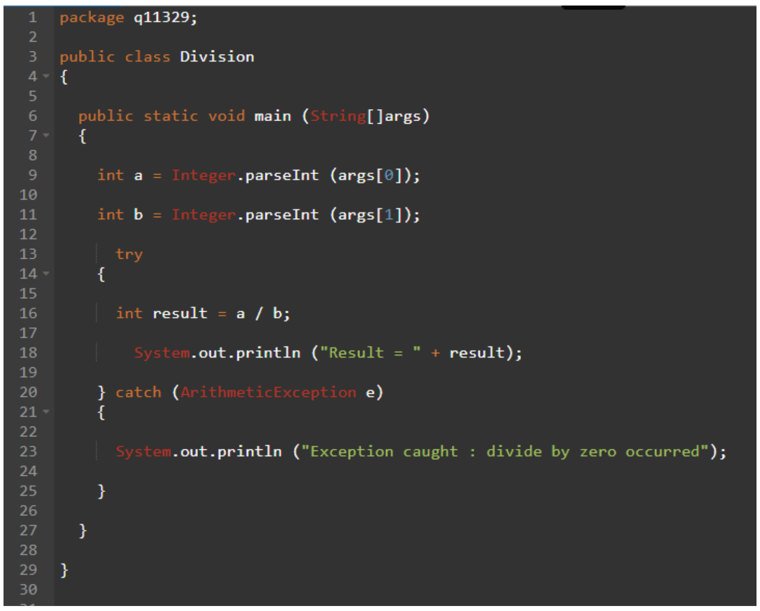
[✓ The code will print `-2`.]

Q5. Write a Java program to handle an ArithmeticException **divide by zero** using exception handling.

Write a class called Division with a **main()** method. Assume that the **main()** method will receive two arguments which have to be internally converted to **integers.**

Write code in the **main()** method to divide the first argument by the second (as integers) and print the result (i.e the quotient).

If the command line arguments to the **main()** method are **"12"**, **"3"**, then the program should print the output as:
```
Result = 4
```
If the command line arguments to the **main()** method are **"55"**, **"0"**, then the program should print the output as:
```
Exception caught : divide by zero occurred
```



```java
package q11329;

public class Division
{

  public static void main (String[]args)
  {

    int a = Integer.parseInt (args[0]);

    int b = Integer.parseInt (args[1]);

      try
    {

      int result = a / b;

        System.out.println ("Result = " + result);

    } catch (ArithmeticException e)
    {

      System.out.println ("Exception caught : divide by zero occurred");

    }

  }

}
```

Q6. Write a Java program to handle an ArithmeticException **divided by zero** by using **try, catch** and **finally** blocks.

Write the **main()** method with in the class MyFinallyBlock which will receive **four** arguments and convert the first two into integers, the last two into float values.

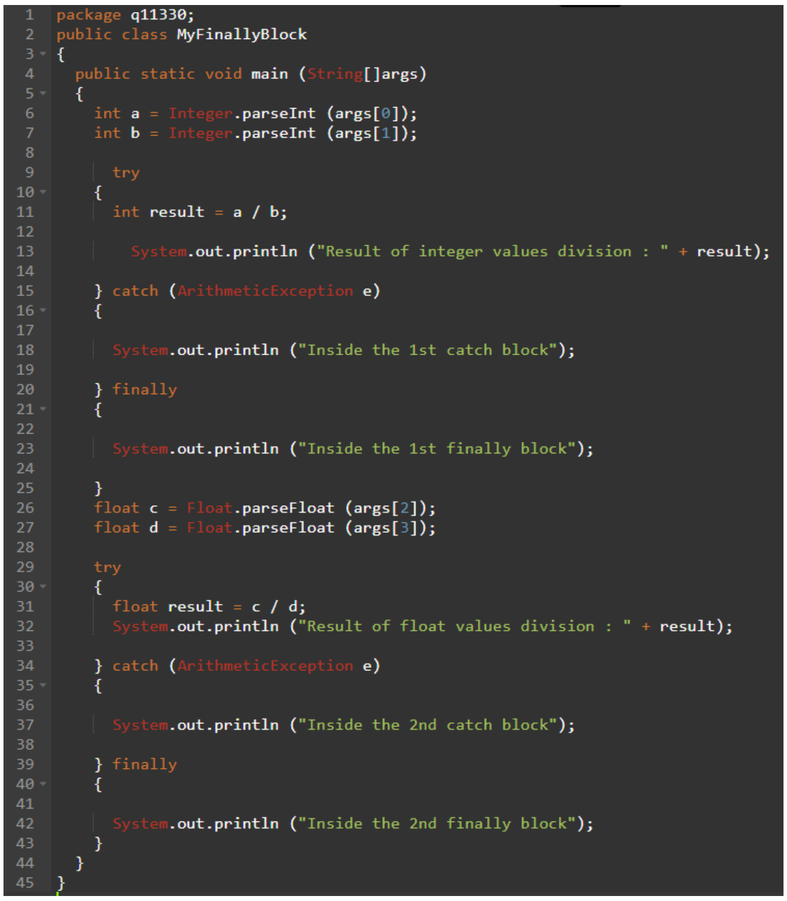
Write the **try, catch** and **finally** blocks separately for finding division of two **integers** and two **float** values.

If the input is given as command line arguments to the **main()** as **"10", "4", "10", "4"** then the program should print the output as:

```
Result of integer values division : 2
Inside the 1st finally block
Result of float values division : 2.5
Inside the 2nd finally block
```

If the input is given as command line arguments to the **main()** as **"5", "0", "3.8", "0.0"** then the program should print the output as:

```
Inside the 1st catch block
Inside the 1st finally block
Result of float values division : Infinity
Inside the 2nd finally block
```



```java
package q11330;
public class MyFinallyBlock
{
  public static void main (String[]args)
  {
    int a = Integer.parseInt (args[0]);
    int b = Integer.parseInt (args[1]);

      try
    {
      int result = a / b;

        System.out.println ("Result of integer values division : " + result);

    } catch (ArithmeticException e)
    {

      System.out.println ("Inside the 1st catch block");

    } finally
    {

      System.out.println ("Inside the 1st finally block");

    }
    float c = Float.parseFloat (args[2]);
    float d = Float.parseFloat (args[3]);

    try
    {
      float result = c / d;
      System.out.println ("Result of float values division : " + result);

    } catch (ArithmeticException e)
    {

      System.out.println ("Inside the 2nd catch block");

    } finally
    {

      System.out.println ("Inside the 2nd finally block");
    }
  }
}
```

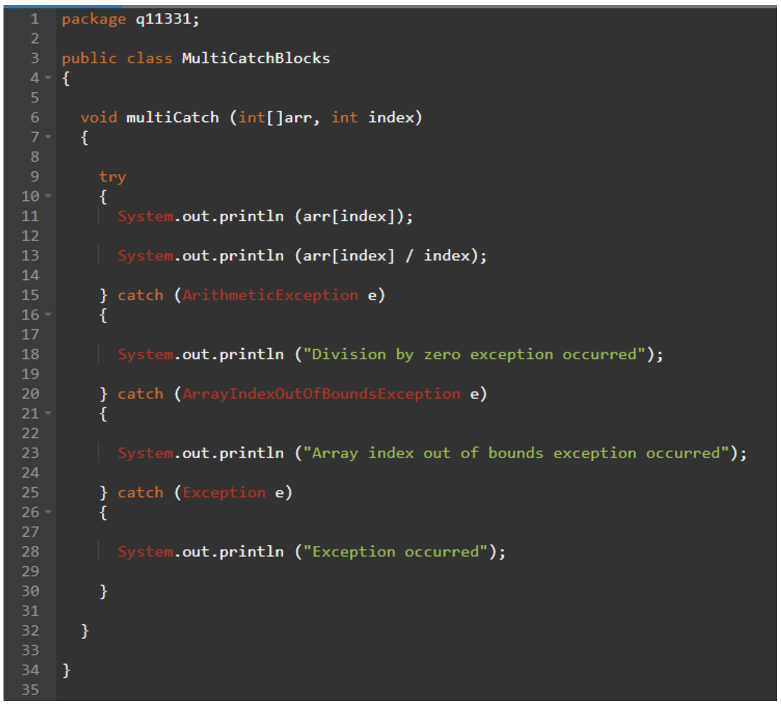
Q7. Write a Java program to illustrate **multiple catch blocks** in exception handling.

Write a method **multiCatch(int[] arr, int index)** in the class MultiCatchBlocks where arr contains integer array values and index contains an integer value.

Write the code in **try** block to print the value of **arr[index]** and also print the division value of **arr[index]** by **index**.

Write the **catch** blocks for

1. ArithmeticException will print "**Division by zero exception occurred**"
2. ArrayIndexOutOfBoundsException will print "**Array index out of bounds exception occurred**".
3. Exception (which catches all exceptions) will print "**Exception occurred**"



```java
package q11331;

public class MultiCatchBlocks
{

  void multiCatch (int[]arr, int index)
  {

    try
    {
      System.out.println (arr[index]);

      System.out.println (arr[index] / index);

    } catch (ArithmeticException e)
    {

      System.out.println ("Division by zero exception occurred");

    } catch (ArrayIndexOutOfBoundsException e)
    {

      System.out.println ("Array index out of bounds exception occurred");

    } catch (Exception e)
    {

      System.out.println ("Exception occurred");

    }

  }

}

```