Finally Block

A finally block is written after try catch block. Finally block is always executed. It is used to execute important code such as closing connection, stream etc.

Example 1: try-catch-finally

```
1  try
2  {
3     int data = 5;
4     System.out.println(data);
5  }
6  catch(ArtithmeticException e)
7  {
8     System.out.println("number cannot divide by zero");
9  }
10 finally
11 {
12    System.out.println("finally block always executed");
13 }
```

Output:

```
5
finally block always executed
```

Explanation: Since finally block always executes even if there is no exception. So statement inside the finally block will print.

Usage of Finally Block:
The finally block encloses code that is always executed at some point after the try block, whether an exception was thrown or not. This is right place to close files, release your network sockets, connections, and perform any other cleanup your code requires.

Note: If the try block executes with no exceptions, the finally block is executed immediately after the try block completes. It there was an exception thrown, the finally block executes immediately after the proper catch block completes.

Using try without catch

A try block must be followed by either catch or finally block.

Example 2: try without catch, with finally

```
1  try
2  {
3     int data = 5/0;
4     System.out.println(data);
```

```
5  }
6  finally
7  {
8     System.out.println("inside finally block");
9  }
10 System.out.println("resuming program");
```

Output:

```
inside finally block
java.lang.ArithmeticException: / by zero
```

Note: if catch block is not defined with try and an exception occurs then it will not be handled. Finally block will run and program will terminate immediately after that. To handle exception and resume normal program flow it is necessary that try block is followed by catch.

Example 2.2: try with catch and finally

```
1  try
2  {
3     int data=5/0;
4     System.out.println(data);
5  }
6  catch(AtithmeticException e)
7  {
8     System.out.println("divide by 0 not possible");
9  }
10 finally
11 {
12    System.out.println("inside finally block");
13 }
14 System.out.println("resuming program");
```

Output:

```
divide by 0 not possible
inside finally block
resuming program
```

Cases when finally will not execute

Finally block will not be executed if program exits (either by calling System.exit(integer) or by causing a fatal error that causes the process to abort).

Example 3: Skipping Finally

```
1  try
2  {
3     System.out.println("try before exit method");
4     System.exit(0);
5  }
6  finally
7  {
```

```
8      System.out.println("inside finally");
9   }
```

Output:

```
try before exit method
```

Return statement with try-catch-finally

If there is a return statement in the try block, the finally block executes right after the return statement is encountered.

Example 4: Using return with try

```
1   try
2   {
3      System.out.println("try before return statement");
4      return;
5   }
6   finally
7   {
8      System.out.println("inside finally");
9   }
```

Output:

```
try before return statement
inside finally
```

Problem 1	**Give output for each of the following code segments.** **Note: Explanation of each answer is mandatory.**

Code 1.1

```
1   try
2   {
3      int a, b;
4      b = 0;
5      a = 5 / b;
6      System.out.print("A");
7   }
8   catch(ArithmeticException e)
9   {
10     System.out.print("B");
11  }
12  finally
13  {
14     System.out.print("C");
15  }
```

Code 1.2

```
1  try
2  {
3     int i;
4     System.out.println("Hello");
5  }
6  catch(Exception e)
7  {
8     System.out.println("Hello India");
9  }
10 finally
11 {
12    System.out.println("Hello Indian people");
13 }
```

Code 1.3

```
1  try
2  {
3     System.out.print("Inside try");
4     String str = "hello";
5     str.charAt(str.length() + 1);
6  }
7  finally
8  {
9      System.out.print("Cleaning");
10 }
```

Code 1.4

```
1  try
2  {
3     System.out.print("Inside try");
4     int[] arr = new int[2];
5     arr[2] = 5;
6  }
7  catch(StringIndexOutOfBoundsException e)
8  {
9     System.out.println("Exception Occurred");
10 }
11 finally
12 {
13     System.out.print("Cleaning");
14 }
15 System.out.println("Outside");
```

Code 1.5

```
1  try
2  {
3     System.out.print("Inside try");
4     int[] arr = new int[2];
```

```
5      arr[2] = 5;
6   }
7   catch(ArrayIndexOutOfBoundsException e)
8   {
9      System.out.println("Exception Occurred");
10  }
11  finally
12  {
13     System.out.print("Cleaning");
14  }
15  System.out.println("Outside");
```

Common Errors

Example 5: Common Error

```
1   try
2   {
3      int num = 100/0;
4   }
5   System.out.println("Statement between try and catch");
6   catch(ArithmeticException e)
7   {}
```

Output: compilation error
Explanation: Never write any statement in between try and catch. Same condition applies with finally.

Multiple Catch Blocks

If your try block can raise multiple exceptions at runtime. And you need to handle each differently, then you need to use multiple catch blocks.

Example 6: Demonstrating working of multiple catch blocks

```
1   int[] arr=new int[5];
2   try
3   {
4      arr[6] = 25/5;
5      arr[2] = arr[3]/0;
6   }
7   catch(ArithmeticException e)
8   {
9      System.out.println("divide by 0 not possible");
10  }
11  catch(ArrayIndexOutOfBoundsException e)
12  {
13     System.out.println("array index is not available");
14  }
```

Output:

```
array index is not available
```

Explanation: In case of multiple exceptions, catch block corresponding to the exception that occurs first will execute. In this example if we make change in line number 4 like arr[0]=25/0, then it will be ArithematicException and line no 9 will print.

Nested try-catch blocks

Nesting try-catch-finally block inside another try/catch/finally is possible. Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Example 7.1: Nested try catch

```
1  try                                                          //outer try
2  {
3     try                                                       //nested try
4     {
5           System.out.println("going to divide");
6           int b = 39/0;
7     }
8     catch(ArithmeticException e)
9     {
10          System.out.println("number divided by zero");
11    }
12
13    System.out.println("other statement");
14 }                                                            //end outer try
15 catch(Exception e)
16 {
17          System.out.println("outside try");
18 }
19 System.out.println("normal flow");
```

Output:

```
going to divide
number divided by zero
other statement
normal flow
```

Example 7.2: Nested try catch – executing outer catch

```
1  try                                                          //outer try
2  {
3     try                                                       //nested try
4     {
5           System.out.println("going to divide");
6           int b = 39/0;
7     }
8     catch(StringOutOfBoundsException e)
9     {
10          System.out.println("wrong exception handler");
11    }
12    System.out.println("other statement");
13 }                                                            //end outer try
```

```
14 catch(Exception e)
15 {
16          System.out.println("outside try catch");
17 }
18 System.out.println("normal flow");
```

Output:

```
going to divide
outside try catch
normal flow
```

Explanation: In the above example code inside nested try is throwing ArithmeticException however the corresponding catch accepts only StringIndexOutOfBoundsException. We'll check in hierarchy towards top if there is an exception handler that handles our exception. If no handler is found then our program will exit. However, in this case our outside catch handles the correct exception hence, program runs normally.

Problem 2	**Give output for each of the following code segments.** **Note: Explanation of each answer is mandatory.**

Code 2.1

```
1  int [] a = new int[3];
2  try
3  {
4      a[2] = 23;
5      System.out.println("Hello");
6      a[3] = 34;
7      System.out.println("World");
8  }
9  catch(ArrayIndexOutofBoundsException ex)
10 {
11     System.out.println("Handled");
12 }
```

Code 2.2

```
1  try
2  {
3      try
4      {
5          "a".charAt(1);
6      }
7      catch(StringIndexOutOfBoundsException e)
8      {
9          System.out.println("Catch 1");
10     }
11     finally
12     {
13         System.out.println("Finally 1");
14     }
15     System.out.println("Outside 1");
16 }
```

```
17 catch(StringIndexOutOfBoundsException e)
18 {
19         System.out.println("Catch 2");
20 }
21 finally
22 {
23         System.out.println("Finally 2");
24 }
```

Code 2.3

```
1  int [] a = new int[3];
2  try
3  {
4     String str = "Course";
5     System.out.println("not");
6     System.out.println("yet");
7  }
8  System.out.println("complete");
9  catch(StringIndexOutofBoundsException ex)
10 {
11    System.out.println("Keep Coding");
12 }
```

Code 2.4

```
1  try
2  {
3      try
4      {
5          "a".charAt(1);
6      }
7      catch(StringIndexOutOfBoundsException e)
8      {
9          System.out.println("Catch 1");
10     }
11 }
12 catch(StringIndexOutOfBoundsException e)
13 {
14         System.out.println("Catch 2");
15 }
16 finally
17 {
18         System.out.println("Finally 2");
19 }
```

Code 2.5

```
1  try
2  {
3      try
4      {
```

```
5           "a".charAt(1);
6       }
7       catch(ArrayIndexOutOfBoundsException e)
8       {
9           System.out.println("Catch 1");
10      }
11  }
12  catch(StringIndexOutOfBoundsException e)
13  {
14          System.out.println("Catch 2");
15  }
16  finally
17  {
18          System.out.println("Finally 2");
19  }
```

Problem 3	What is try-catch-finally Exception Handling? Explain in your own words.