Finally Keyword

Definition:

The *finally* keyword is used to create a block of code that follows a try block. A finally block of code contains all the crucial statements which always executes, whether or not an exception has occurred.

Syntax:

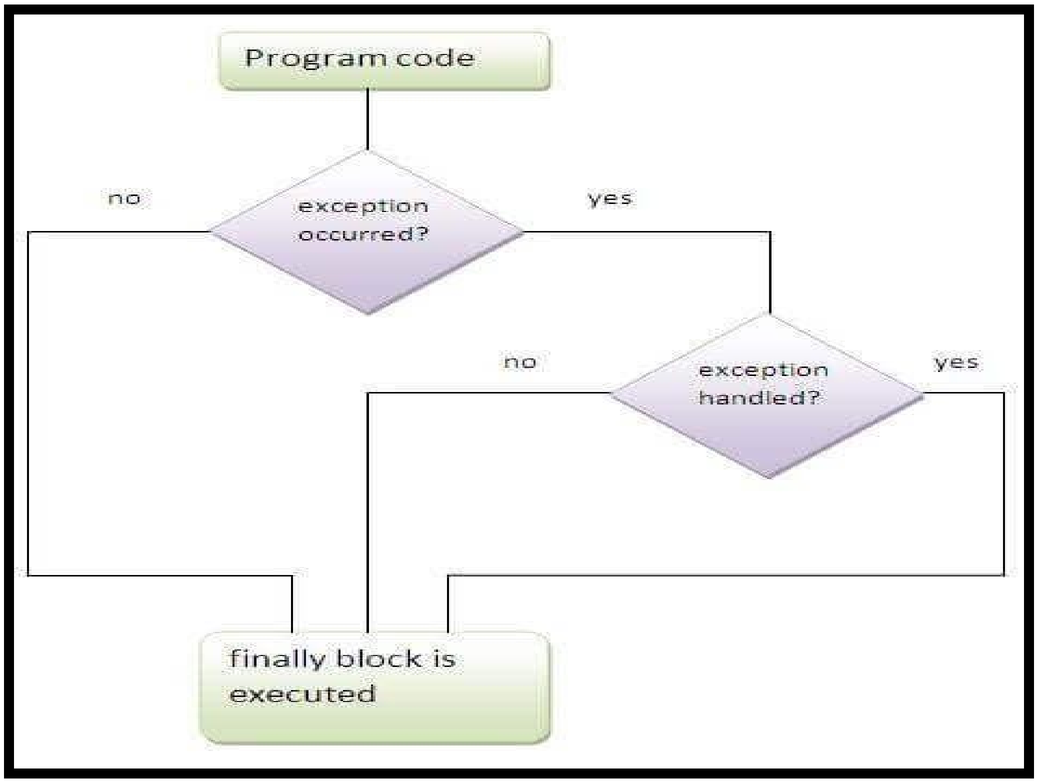
```
try {
    //Statements that may cause an exception
}
catch {
   //Handling exception
}
finally {
   //Statements to be executed
}
```

Need for *finally* keyword:

"finally" block in java can be used to put "cleanup" code such as closing a file, closing connection etc. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

Usage of *finally* keyword:

- *finally* keyword is used to maintain flow of the program.
- It is used after the try block. Even if an exception is not handled, *finally* block maintains the flow without interruption of the program.



Important points to remember:

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. For each try block there can be zero or more catch blocks, but only one *finally* block.
3. An exception in the finally block, behaves exactly like any other exception.

Working of *finally* block:

1. *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.

2. In the normal case when there is no exception in the try block then the finally block is executed after the try block. However if an exception occurs then the catch block is executed before the finally block.
3. *finally* works irrespective of return statements in try-catch block.

Examples:

1. When exception does not occur:

```java
class Main{
    public static void main(String args[])
    {
        try{
            System.out.println("inside try block");
            int data=25/5;  //no error so catch block is not executed
            System.out.println(data);
            }
        catch(NullPointerException e)
        {
            System.out.println("inside catch block");
            System.out.println(e);
        }
    //finally block executed after try block as there is no error occurs
      finally
      {
        System.out.println("finally block is always executed");
      }
      System.out.println("rest of the code...");
    }
}
```

OUTPUT

```
inside try block
5
finally block is always executed
rest of the code...
```

Explanation:

As there is no exception, after the try block, catch block is not executed. The control of the program then goes to *finally* block.

finally block is executed after every try block if present.

Flow of the program:

try - finally - statements after the block.

2. When exception occurs and not handled:

```
class Main{
    public static void main(String args[])
    {
        try{
            System.out.println("inside try block");
            int data=25/0;  //error occurs so catch block is should be
executed
            System.out.println(data);
            }
    //catch block should executed to handle the exception
    //as there is no catch block to handle the exception control comes to
finally block
        finally
        {
            System.out.println("finally block is always executed");
        }
    System.out.println("rest of the code...");
    }
}
```

OUTPUT

```
inside try block
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Explanation:

There is an exception in the try block, catch block should be executed to handle the exception but there is no catch block. So the control of the program then goes to *finally* block. Statements after try-finally block are not executed, since exception is not handled.

Flow of the program:

try - finally.
Error

3. When exception occurs and corresponding catch block matches correctly:

```
class Main{
    public static void main(String args[])
    {
        try{
            System.out.println("inside try block");
            int data=25/0;  //error occurred so catch block should be
executed
            System.out.println(data);
            }
    //corresponding catch block matches
        catch(ArithmeticException e)
        {
            System.out.println("inside catch block");
            System.out.println(e);
        }
    //finally block executed after catch block as error occurs
        finally
        {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}
```

OUTPUT

```
inside try block
inside catch block
java.lang.ArithmeticException: / by zero
finally block is always executed
rest of the code...
```

Explanation:

There is an exception in the try block, catch block should be executed to handle the exception. There is a catch block which matches with corresponding exception. So catch block is executed. Then *finally* block is executed. As there is no interruption while running the program, the last statement after try-catch-finally block is executed.

Flow of the program:

try -catch- finally-statements after the block.

4. When exception occurs and catch block mismatched:

```
class Main{
    public static void main(String args[])
    {
        try{
            System.out.println("inside try block");
            int data=25/0;  //error occurred so catch block should be
executed
            System.out.println(data);
            }
    //corresponding catch block is not matched
    //catch block is not executed
        catch(NullPointerException e)
        {
            System.out.println("inside catch block");
            System.out.println(e);
        }
    //finally block executed after try block as catch block is mismatched
        finally
        {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}
```

OUTPUT

```
inside try block
finally block is always executed
Exception in thread “main” java.lang.ArithmeticException: / by zero
```

Explanation:

There is an exception in the try block, catch block should be executed to handle the exception, there is a catch block, but mismatched. So the control of the program then goes to *finally* block. Statements after try-catch-finally block are not executed, since exception is not handled.

➢ This is as same as if there is no catch block.

Flow of the program:

try - finally.

Error.

5. Exception inside *finally* block:

```
class Main{
    public static void main(String args[])
    {
        try
        {
            System.out.println("inside try block");
        }
    //finally block executed after try block
        finally
        {
            System.out.println("finally block is always executed");
            try
            {
                int data=25/0;  //error occurs so catch block is executed
                System.out.println(data);
            }
            catch(ArithmeticException e)
            {
                System.out.println("inside catch block");
                System.out.println(e);
            }
        }
        System.out.println("rest of the code...");
    }
}
```

OUTPUT

```
inside try block
finally block is always executed
inside catch block
java.lang.ArithmeticException: / by zero
rest of the code...
```

Explanation:

There is an exception in the try block, catch block should be executed to handle the exception. There is a catch block which matches with corresponding exception. So catch block is executed. Then *finally* block is executed. As there is no interruption while running the program, the last statement after try-catch-finally block is executed.

Flow of the program:

try - finally - try - catch - statement after the block.

Circumstances where *finally* block does not execute:

- The death of a Thread
- Using the System. exit() method.
- Due to an exception arising in the finally block and not handled properly.

Example:

System.exit() before *finally* block:

```
class Main{
    public static void main(String args[])
    {
        try{
            System.out.println("inside try block");
            System.exit(0);//program terminates before next statement
            int data=25/0;  //error not occurred since program terminates
            System.out.println(data);
            }
    //catch block is not executed as there is no error
        catch(ArithmeticException e)
        {
            System.out.println("inside catch block");
            System.out.println(e);
        }
    //finally block is also not executed since program terminated before
error occuring statement
        finally
        {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}
```

OUTPUT

```
inside try block
```

Explanation:

This is a special circumstance where the program is terminated before the statement which causes error. *finally* block is not executed.

Flow of the program:

try upto System.exit() statement.

Real time usage:

Basically the use of finally block is **resource deallocation**. Means all the resources such as Network Connections, Database Connections, which we opened in the try block are needed to be closed so that we won't lose our resources as opened. So those resources are needed to be closed in the finally block.

```java
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

class Main
{
    private static final int SIZE = 10;
    public static void main(String[] args)
    {
        PrintWriter out = null;
        try
        {
            System.out.println("Entered try statement");
            // PrintWriter, FileWriter
            // are classes in io package
            out = new PrintWriter(new FileWriter("OutFile.txt"));
        }
        catch (IOException e)
        {
            // Since the FileWriter in
            // try block can throw IOException
        }
    // Following finally block cleans up
    // and then closes the PrintWriter.
        finally
        {
            if (out != null) {
                System.out.println("Closing PrintWriter");
                out.close();
            }
            else
            {
                System.out.println("PrintWriter not open");
            }
        }
    }
}
```

OUTPUT

```
Entered try statement
PrintWriter not open
```

★ The finally block is a key tool for preventing resource leaks. When closing a file or otherwise recovering resources, place the code in a finally block to ensure that resource is always recovered.

Difference between final, finalize and finally keyword:

S. No	**Final**	**Finalize**	**Finally**
1	Final is used to apply restrictions on class, method and variable. Final class can't	Finalize is used to perform clean up processing	Finally is used to place important code, it will be

	be inherited, final method can't be overridden and final variable value can't be changed.	just before the object is garbage collected.	executed whether an exception is handled or not.
2	Final is a keyword.	Finalize is a method.	Finally is a block.