Exception Handling

- Exception
- Exception Hierarchy
- Checked Exception
- Un-Checked Exception
- Exception Methods
- Try Block
- Catch Block
- Finally Block

Exception

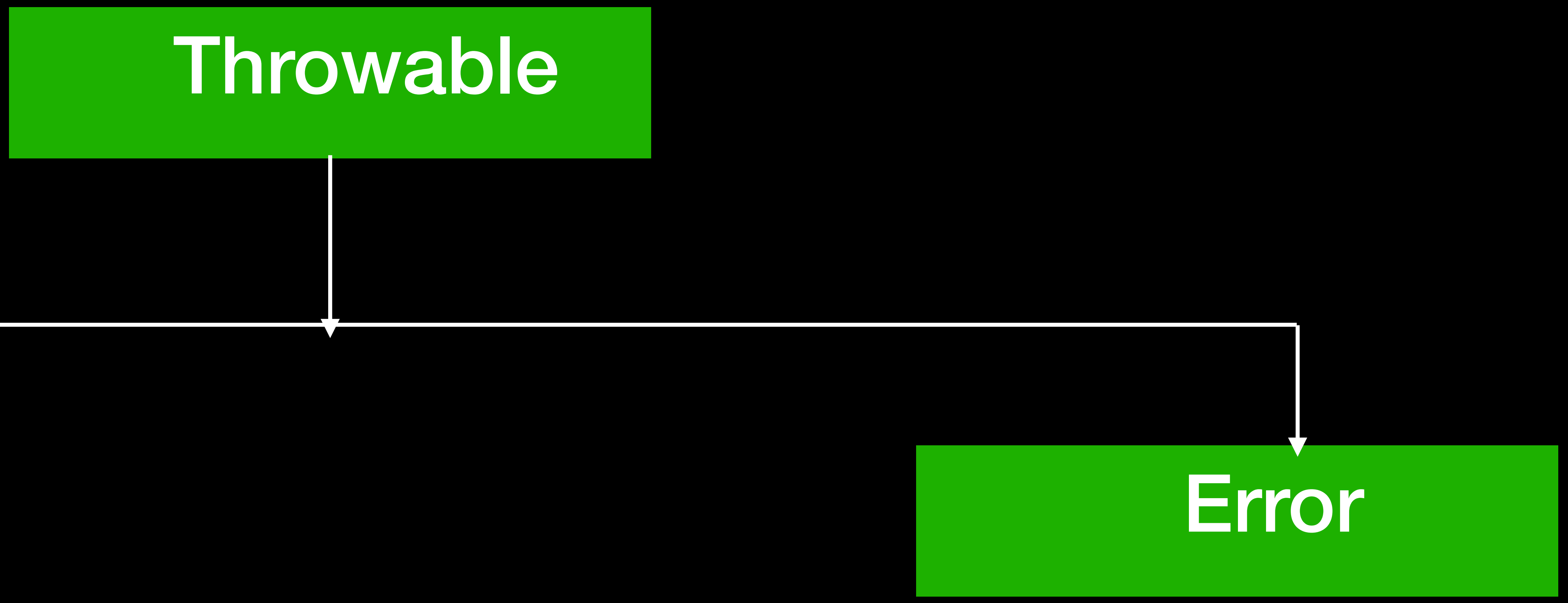
An unwanted unexpected event that disturbs normal flow of the program is called exception.

It is highly recommended to handle exceptions. The main objective of exception handling is graceful (normal) termination of the program

Exception Handling

We have to define alternative way to continue rest of the program normally. This way of defining alternative is nothing but exception handling.

Exception Hierarchy



Exception

Most of the cases exceptions are caused by our program and these are recoverable.

Error

Most of the cases errors are not caused by our program these are due to lack of system resources and these are non-recoverable.

Checked Exception

The exceptions which are checked by the compiler whether programmer handling or not, for smooth execution of the program at runtime, are called checked exceptions

Un-Checked Exception

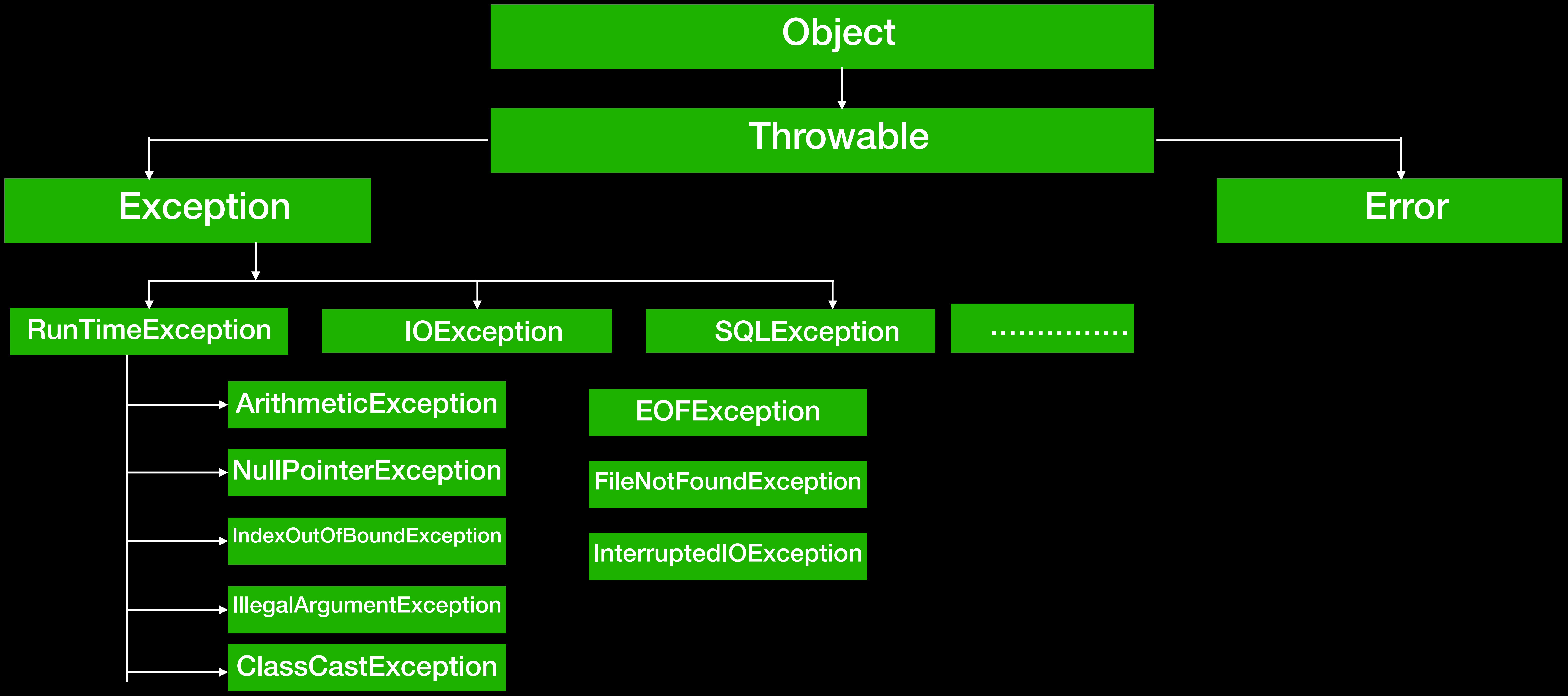
The exceptions which are not checked by the compiler whether programmer handing or not ,are called unchecked exceptions

Fully Checked Exception

A checked exception is said to be fully checked if and only if all its child classes are also checked.

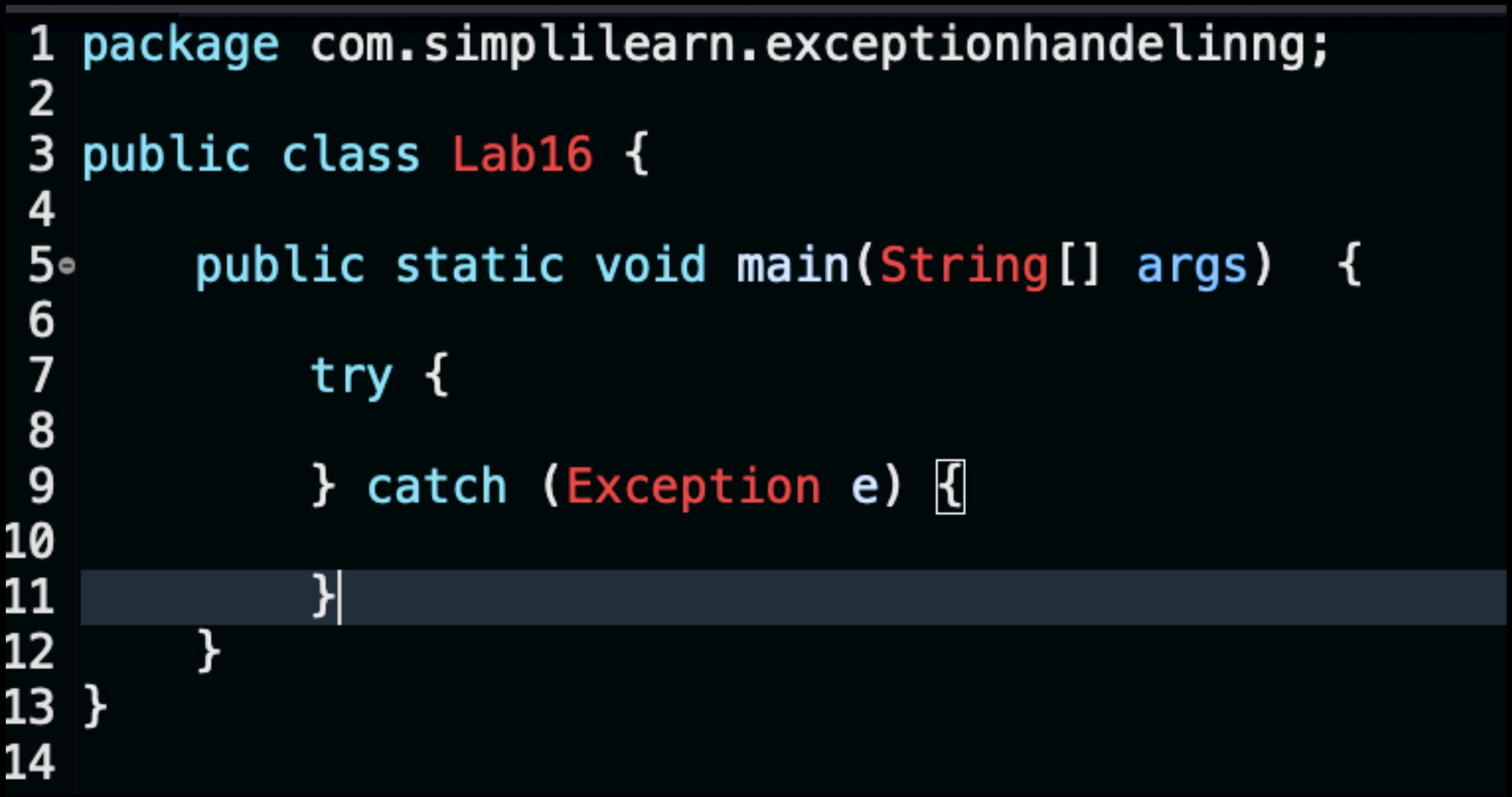
Paratially Checked Exception

A checked exception is said to be partially checked if and only if some of its child classes are unchecked.


Object
Throwable
Exception
Error
RunTimeException
IOException
SQLException
……………
ArithmeticException
NullPointerException
IndexOutOfBoundException
IllegalArgumentException
ClassCastException
EOFException
FileNotFoundException
InterruptedIOException

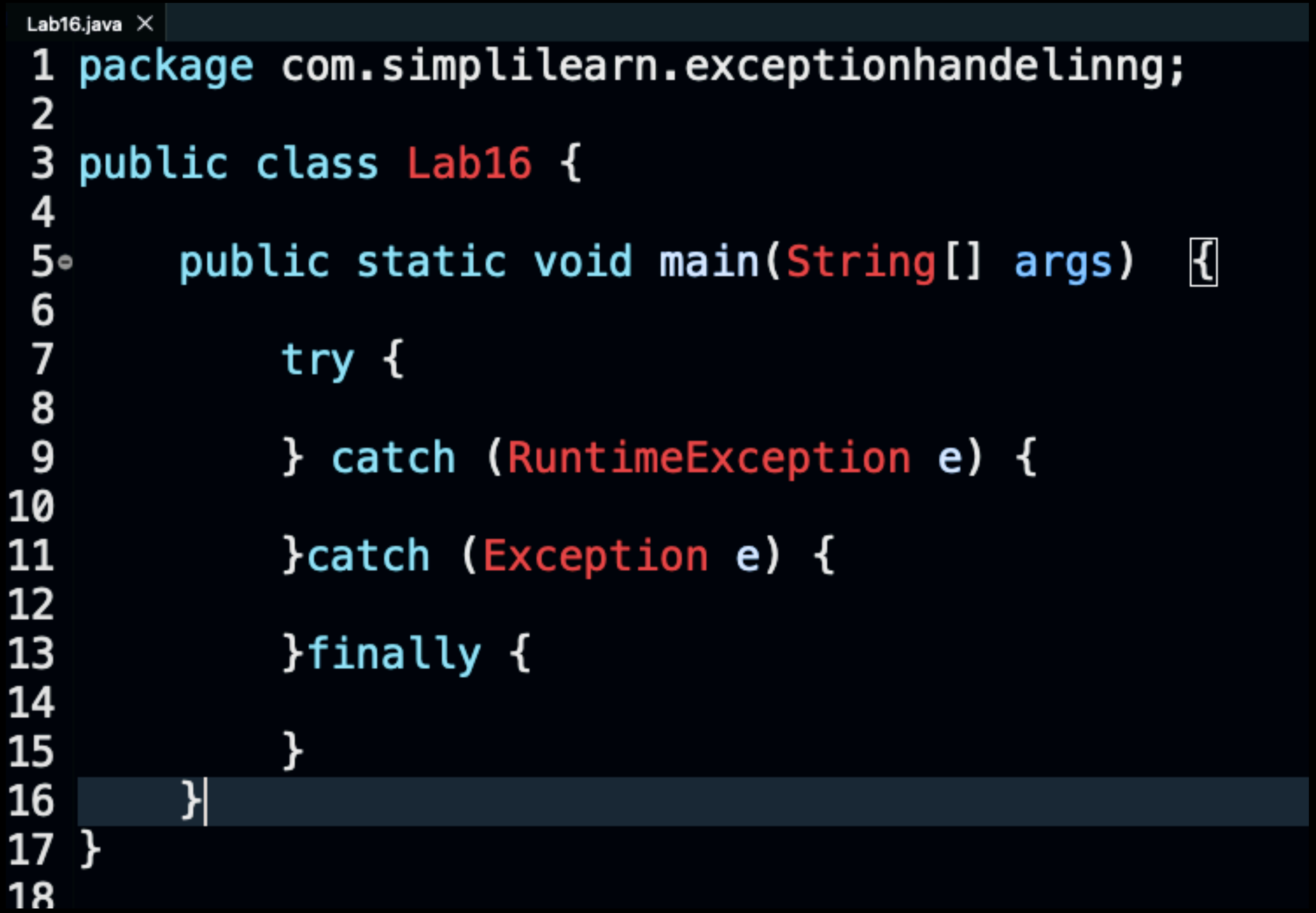
Try …. Catch …Finally

Try with Catch



```
package com.simplilearn.exceptionhandelinng;

public class Lab16 {

    public static void main(String[] args)  {

        try {

        } catch (Exception e) {

        }
    }
}

```

Try with Multi Catch

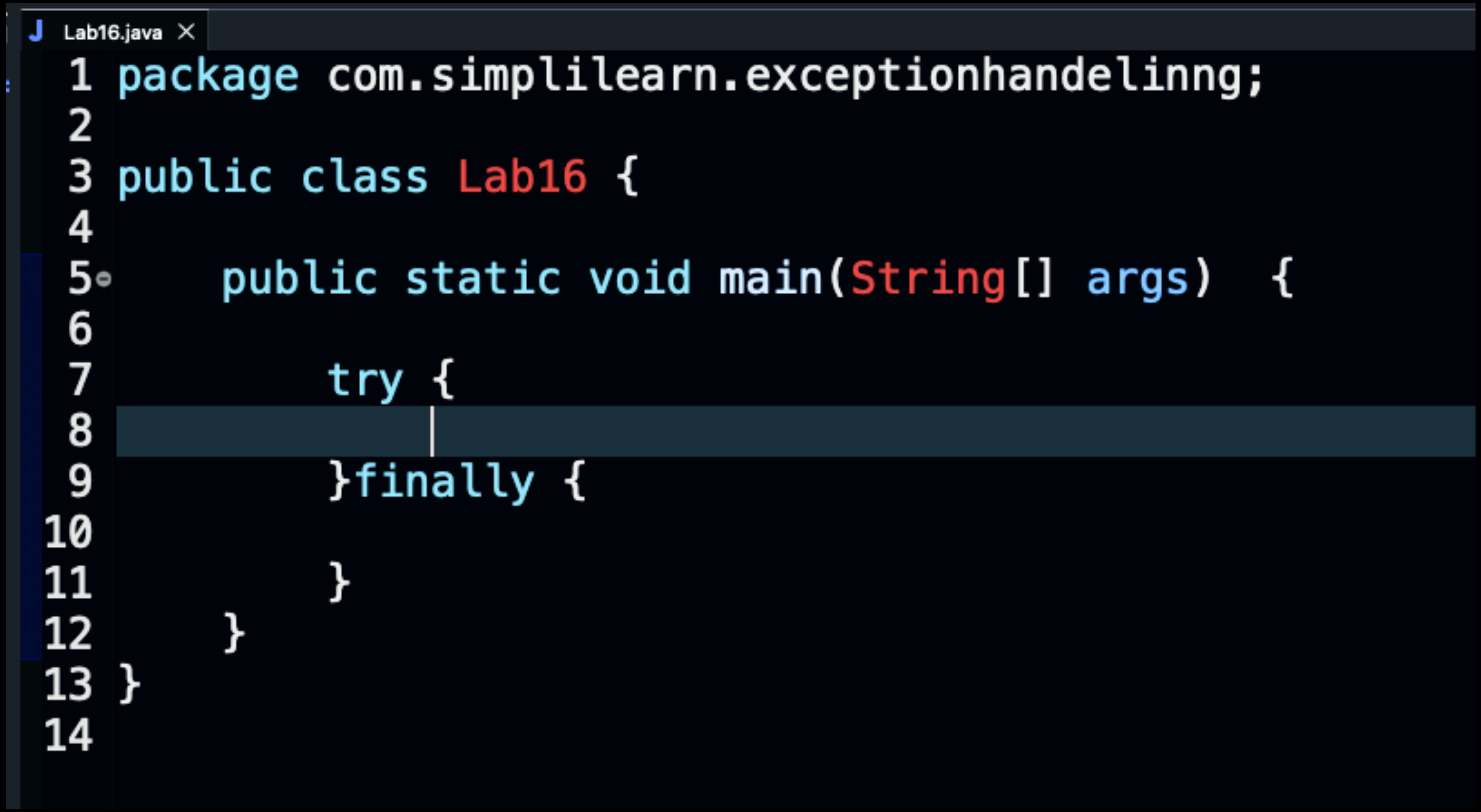


Lab16.java

```
package com.simplilearn.exceptionhandelinng;

public class Lab16 {

    public static void main(String[] args)  {

        try {

        } catch (RuntimeException e) {

        }catch (Exception e) {

        }finally {

        }
    }
}

```

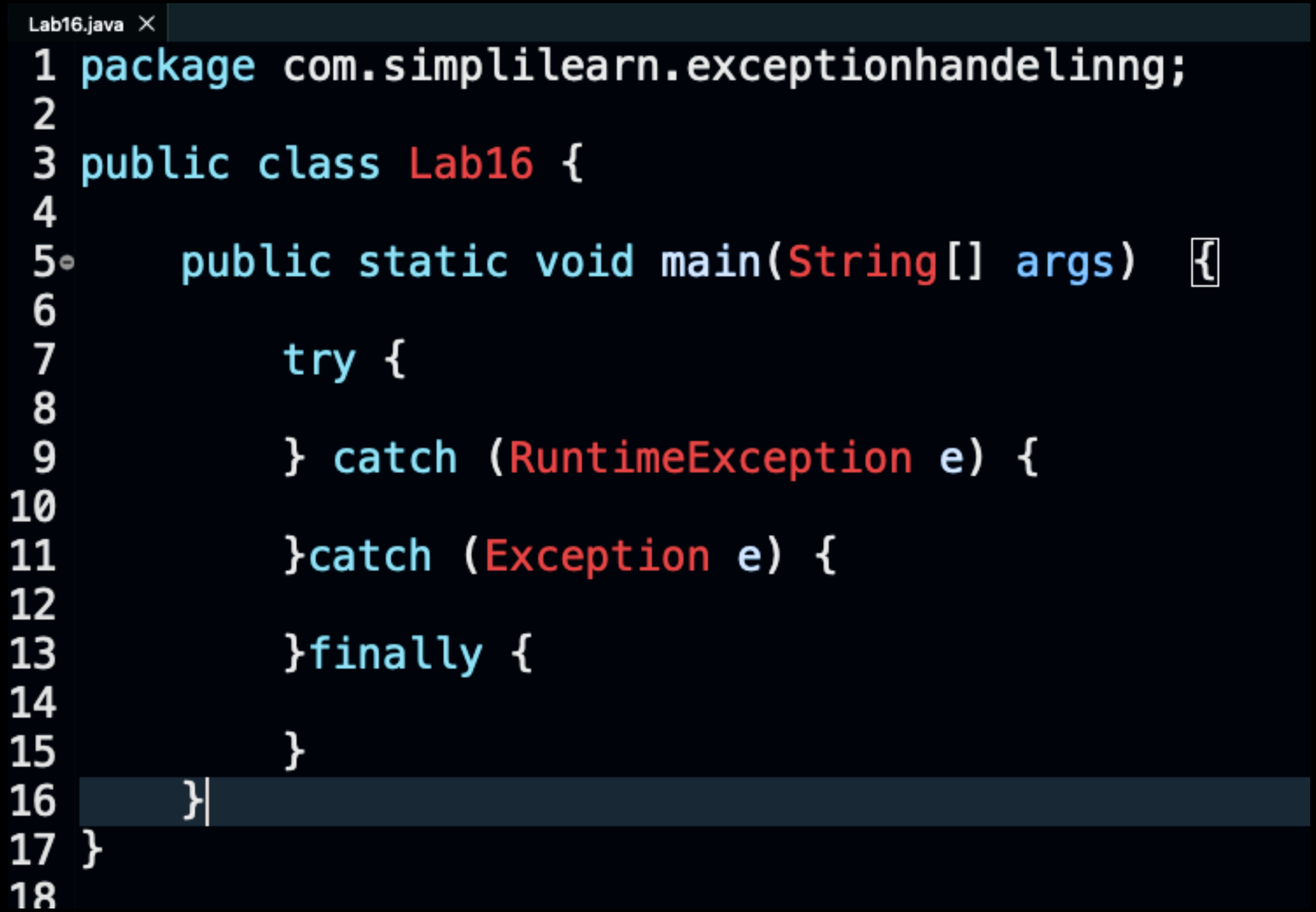
Try …. Catch …Finally

Try with Finally



```
package com.simplilearn.exceptionhandelinng;

public class Lab16 {

    public static void main(String[] args)  {

        try {

        }finally {

        }
    }
}
```

Try with Catch and Finally



```
package com.simplilearn.exceptionhandelinng;

public class Lab16 {

    public static void main(String[] args)  {

        try {

        } catch (RuntimeException e) {

        }catch (Exception e) {

        }finally {

        }
    }
}
```

Methods to print the Exception

```
public void printStackTrace()
```

```
public String toString()
```

```
public String getMessage()
```

Finally Block

- It is not recommended to take clean up code inside try block because there is no guarantee for the execution of every statement inside a try.
- It is not recommended to place clean up code inside catch block because if there is no exception then catch block won't be executed.
- We require some place to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether handled or not handled. Such type of best place is nothing but finally block. Hence the main objective of finally block is to maintain cleanup code .

System.exit(0);

1. This argument acts as status code. Instead of zero, we can take any integer value
2. zero means normal termination , non-zero means abnormal termination.
3. This status code internally used by JVM, whether it is zero or non-zero there is no change in the result and effect is same wrt program

Be Safe

Thank You

Happy Learning

Keep Watching