Exception Handling in Java

Description

- An Exception is a compile time / runtime error that breaks off the program's execution flow. These exceptions are accompanied with a system generated error message. It is up to the user to write well formed code to catch these exceptions so that program flow is not disturbed and just informs the user that due to this error the program flow cannot continue as expected.

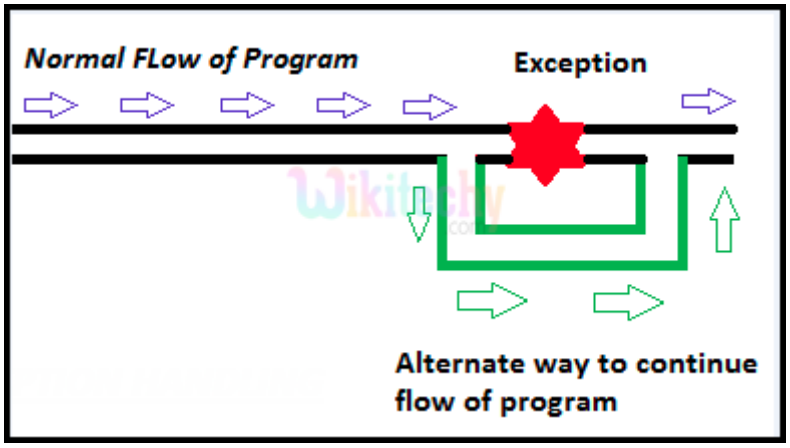


Fig1. Exceptions

History of Exceptions:

- Originated from java.lang.Exception class.

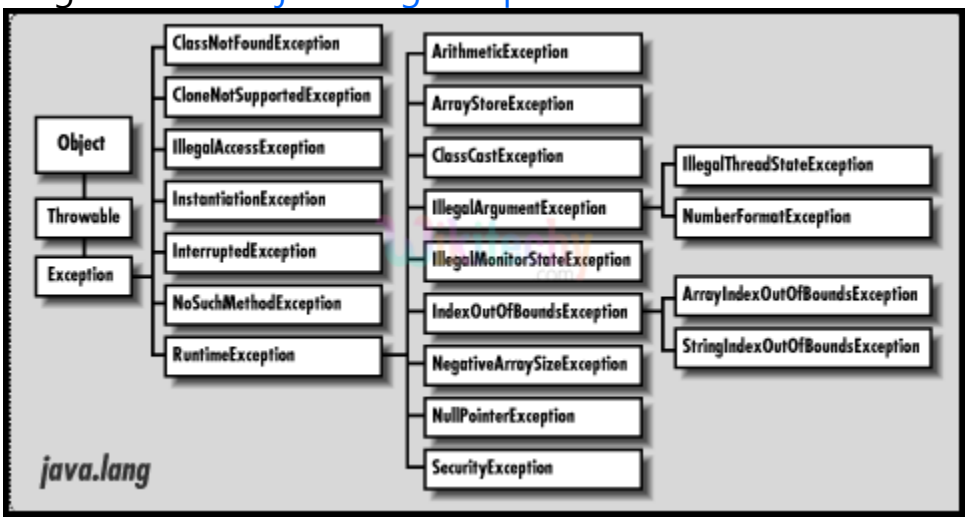


Fig2. Exception Hierarchy

- A subclass of the Throwable class.
- Error – an abnormal situation is as well part of Throwable class.

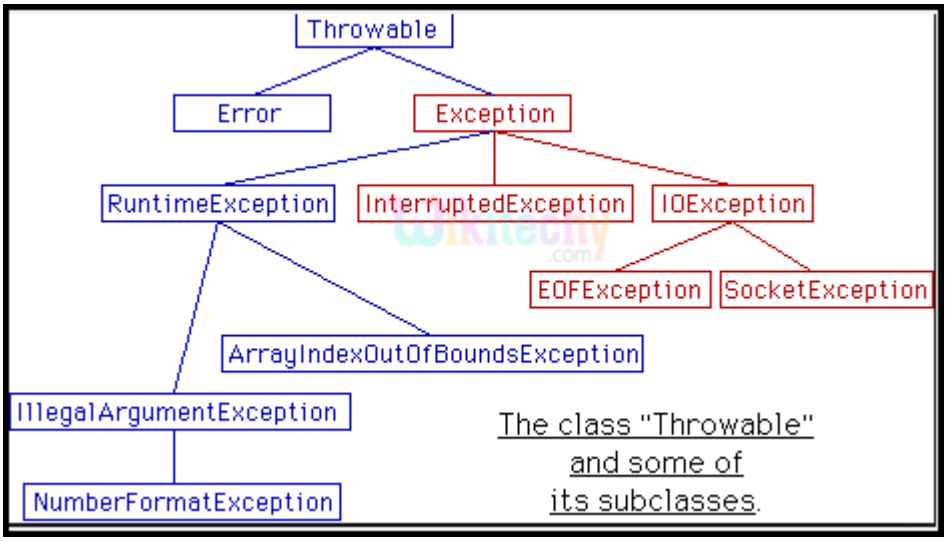


Fig3. Throwable Class

Exception Handling

- Permits to control the normal flow by the way of **try**, **catch** and **finally** blocks.
- Code presented inside the above blocks is termed "**Protected Code**".
- **try** – Place where exception may or may not arise. Occurs only once per try/catch block.
- **catch** – Place where the exception is handled. Occurs multiple times per try/catch block to handle different types of Exception.
- **finally** – Place where the code content is executed for sure, irrespective of an incidence of the Exception. Generally used to place cleanup codes.

Note:

- The object of class System. Exception can handle all types of Exceptions and hence used even when the user is unfamiliar with the exact Handler Class.

Syntax of try catch finally Blocks:

```
try
{
   // code
}
Catch (ExceptionType1 e1)
{
      // code
}
Catch (ExceptionType2 e2)
{
   // code
}
finally
{
  // code
}
```


Exception Handling

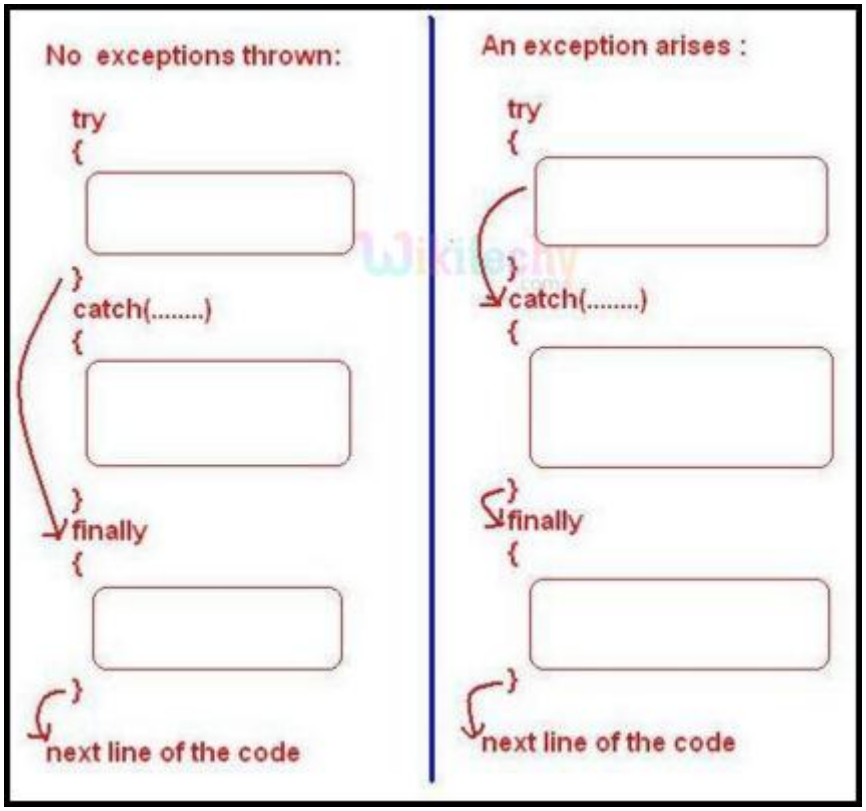


Fig4. Exception Handling

throw/throws keyword:

- Implemented as part of method statements to clearly identify the exceptions that the concerned method might throw.
- Care should be taken to handle exceptions in place of such method-calls.
- throws - delays the exception handling process.
- throw - raises an exception openly.

Sample Code:

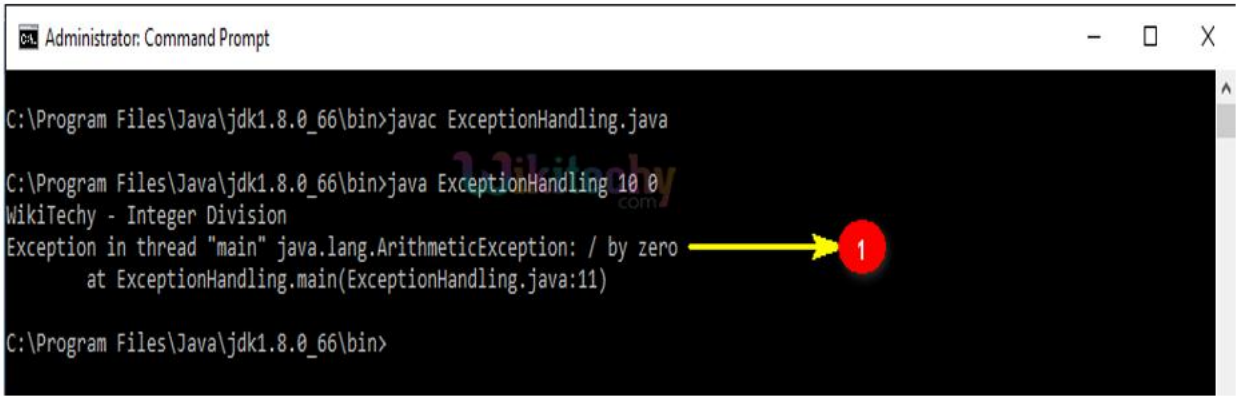
```
import java.io.*;
 public class ExceptionHandling {
  public static void main(String[] args) {

  int x, y, z;
  System.out.println("WikiTechy - Integer Division");

  x = Integer.parseInt(args[0]);
  y = Integer.parseInt(args[1]);
  z = x / y;

  System.out.println("\n First Input = " + x);
  System.out.println("\n Second Input = " + y);
  System.out.println("\n Result of Division = " + z);
  }
}
```


Code Explanation:

```
import java.io.*;

public class ExceptionHandling {
  public static void main(String[] args) {

  int x, y, z;
  System.out.println("WikiTechy - Integer Division");

  x = Integer.parseInt(args[0]);
  y = Integer.parseInt(args[1]);
  z = x / y;

  System.out.println("\n First Input = " + x);
  System.out.println("\n Second Input = " + y);
  System.out.println("\n Result of Division = " + z);
  }
}
```

1. Taking command line arguments as input.
2. Command line arguments are of type String. Hence "Integer.parseInt()" static method is used to convert the command line string argument to integer. The syntax for parseInt() is: public static int parseInt(String s).

Output:



1 Arithmetic Exception – Divide by Zero occurred. Now let's modify the code by placing try, catch and finally blocks to handle the exception.

Sample Code:

```
import java.io.*;
 public class ExceptionHandling {
  public static void main(String[] args) {
   int x, y, z;
  System.out.println("WikiTechy - Integer Division");
   x = Integer.parseInt(args[0]);
  y = Integer.parseInt(args[1]);
  z = x / y;
  System.out.println("\n First Input = " + x);
  System.out.println("\n Second Input = " + y);
  System.out.println("\n Result of Division = " + z);
  }
}
```


Code Explanation:

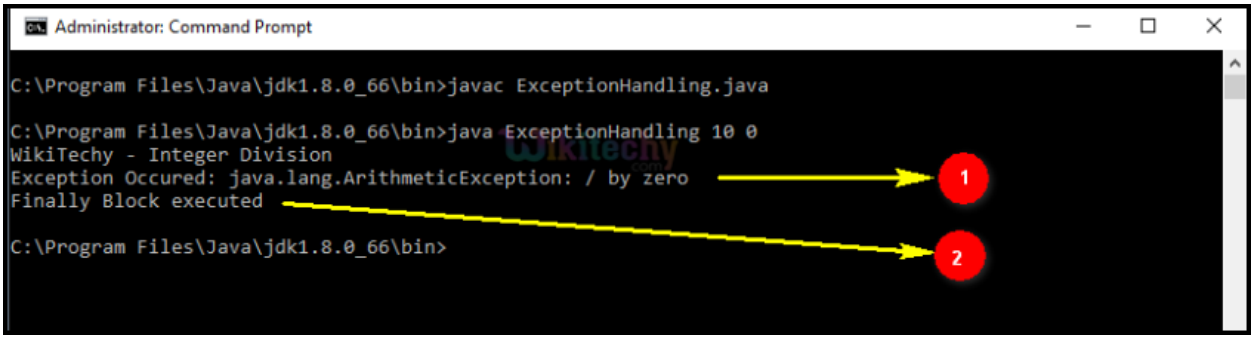
```
import java.io.*;

public class ExceptionHandling {
  public static void main(String[] args) {
  int x, y, z;
  System.out.println("WikiTechy - Integer Division");

  try                                                    1
  {
    x = Integer.parseInt(args[0]);
    y = Integer.parseInt(args[1]);
    z = x / y;
    System.out.println("\n First Input = " + x);
    System.out.println("\n Second Input = " + y);
    System.out.println("\n Result of Division = " + z);
  }
  catch(ArithmeticException e)                           2
  {
    System.out.println("Exception Occured: " + e);
  }
  finally                                                3
  {
    System.out.println("Finally Block executed");
  }
 }
}
```

1. Try – All code logics should be placed in this block.
2. Catch – Exception Handling block. A program can have more than one . catch block as follows:

```
catch(ArithmeticException e)
{
        System.out.println("Exception Occured: " + e);
}
catch(Exception e)
{
        System.out.println("General Exception Handler"); }
```


3 The class Exception handles all types of exception. If the user is not aware of the correct Exception to be captured, the "Exception" class very well is used.

Output:



1 Exception occurred and handled by catch block and hence the catch block print message is displayed here.

2 Finally block is executed and its message as well is printed in the output console.

