CHAPTER 5: Multithreading and Exception Handling

Course Learning Outcome

- **Perform concept of exception handling.**
  - Describe the concept of exception handling mechanism.
  - Explain the use of exception handling.
  - Explain the different types of exceptions in RuntimeException:
    - a. NumberFormatException
    - b. ArrayIndexOutOfBoundsException
    - c. ArithmeticException
  - Build Java programs using exception handling.

Course Learning Outcome

- **Perform the concept of Threading.**
  - Describe Thread and its uses in Java program.
  - Explain the different types of Thread:
    - a. Single thread
    - b. Multiple thread
  - Differentiate between multitasking and multithreading.
  - State the methods involved in the life cycle of a thread.
  - Build multithreaded application.

Exception Handling

Perform concept of exception handling.

Realtime Example

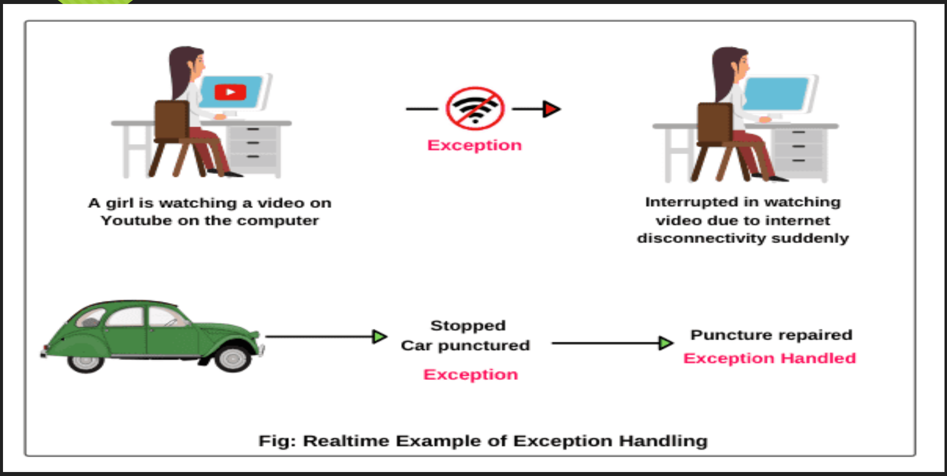

A girl is watching a video on Youtube on the computer
Exception
Interrupted in watching video due to internet disconnectivity suddenly
Stopped
Car punctured
Exception
Puncture repaired
Exception Handled

Fig: Realtime Example of Exception Handling

Source: https://www.scientecheasy.com/2020/08/exception-handling-in-java.html/

Event that might generate exception:

- Opening a non-existing file in your program.
- Reading a file from a disk but the file does exist there.
- Writing data to a disk but the disk is full or unformatted.
- When the program asks for user input and the user enters invalid data.
- When a user attempts to divide an integer value by zero, an exception occurs.
- When a data stream is in an invalid format, etc.

Error Handling

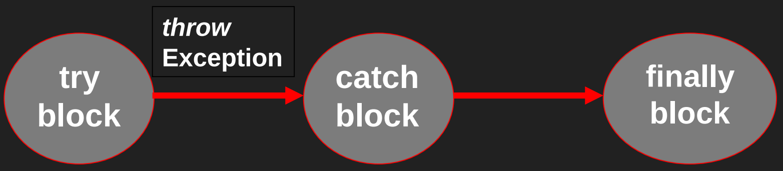
- Errors are common during programming
- They result in
  - Wrong Output
  - Abrupt termination of the program
  - Crashing of the system
- These problems when left undetected, will cause big problems.
- Java provides error handling mechanism to solve these problems.

Exceptions in Java

Exceptions are handled in Java using the five keywords

- try
- catch
- throw
- throws
- finally

Exception Handling Mechanism



Types of Errors

Generally, in any language there are two broad classification of errors:

- Compile Time errors
  - Errors that occur due to wrong syntax in the program.
  - Detected by compilers and .class file not created.
- Runtime errors
  - Errors that occur while executing a program.
  - The .class file are created but may not run properly.

Need for Exception Handling

It is a very important mechanism in all the programming languages.

The needs are :

- To avoid abnormal program termination.
- To avoid system crashes.
- It helps to detect and report the exceptional circumstances, in order to take a necessary action.

Types of Exceptions

ArithmeticException

- Occurs when an **abnormal arithmetic condition** occurs.

ArrayIndexOutOfBoundsException

- Occurs when a program tries to access an array element whose **index value is out of range**

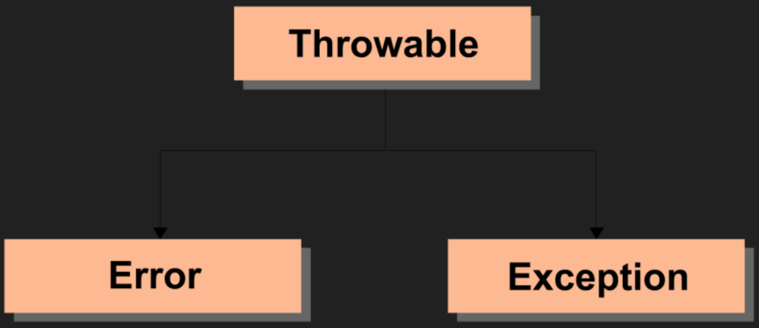
NumberFormatException

- Occurs when you try to **convert a string,** which is not in the form of a number **into numeric.**

Exception Handling

- Java provides superior support for exception handling.
- Java exception can be manually generated
  - by the code
  - by the Java runtime system during the program execution.

Throwable – Sub classes



try,catch, throw blocks

- An exception is an object, which is generated during the execution of the program.
- When an exception arises an exception object of that exception class is created and thrown.
- The exception which is thrown can be caught inside the program to stop the abnormal program termination.

try block

- The exceptions in a program are caught or trapped using a try block.
- A block of code which may generate an exception to terminate the program should be placed inside the *try* block.

try block

Syntax:

```
try
{
    statements ;// code which generates exception
}
```

try block - Example

```
try
{
    int A,B,C;
    A=Integer.parseInt(args[0]);
    B=Integer.parseInt(args[1]);
    C=A/B;
    System.out.println(“The value of C:- “ + C);
}
```

catch block

- A try block should have **at least one catch** block.
- The try block **may or may not** generate an exception.
- The catch block is responsible for catching the exception thrown from the try block.
- Hence, a catch block is **placed immediately after a try block**.

catch block

Syntax:

```
catch(Exceptiontype object)
{
    statements;// code which handles exception
}
```

catch block

Example :

```
catch(ArithmeticException exp)
{
    System.out.println("I have caught the exception");
}
```

Multiple Catch

- Some times a code can generate more than one type of exception.
- If more than one type of exception arises, more than one catch statement should be used to handle those exception types.

finally

- The *finally* block contains statements for doing the final process such as de-allocation of memory etc.
- It may be added immediately after the *try* block or after the last *catch* block.
- A try block should have at least one catch block or finally block immediately following it.

finally

Syntax:

```
finally
{
    statements ;// code to be executed
}
```

finally

Example:

```
finally
{
    System.out.println("Thank You");
}
```