Exception Handling

- Whenever an error occurs while executing a statement, it creates an exception object and then the normal flow of the program halts and JRE tries to find some handler for the raised exception.
- The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurred, type of exception etc.
- When the exception occurs in a method, the process of creating the exception object and handing the exception over to runtime environment is called “throwing the exception”.
- Exception can be checked (compile time) or unchecked (run time).
- Checked exceptions:
  - caught and handled during compile time.
  - If no exception handling code (e.g. try .. catch ..) is provided, then compiler signals a compilation error.
  - Mostly due to faults outside code like unavailable files, illegal class names, network errors, etc.
- Unchecked exceptions:
  - Compiler does not force us to explicitly handle them
  - Occur during the execution (i.e. run time).
  - Normally, due to programming bugs, e.g. logic errors like Divide by Zero.
  - Can be avoided by careful programming.

- Five constructs are used in exception handling:
  - try – a block surrounding program statements to monitor for exceptions
  - catch – together with try, catches specific kinds of exceptions and handles them in some way
  - finally – specifies any code that absolutely must be executed whether or not an exception occurs
  - throw – used to throw a specific exception from the program
  - throws – specifies which exceptions a given method can throw

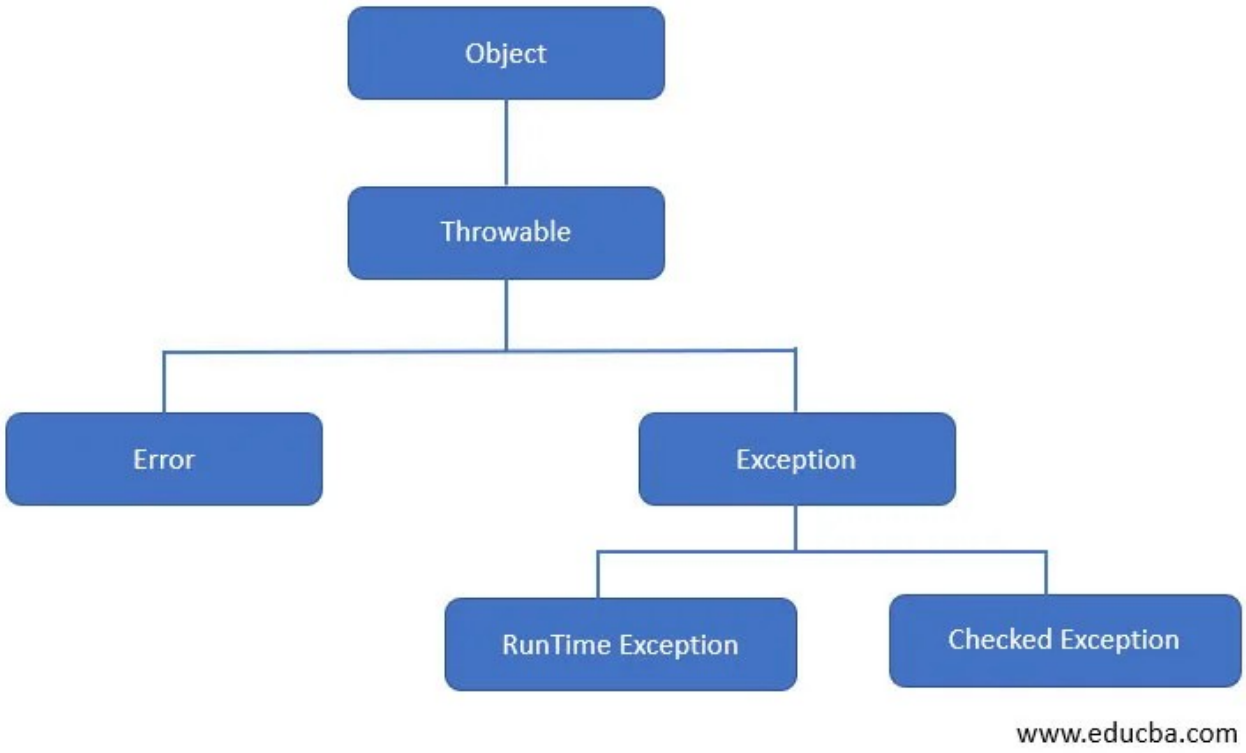
Exception Handling Syntax:

```
try
{
   //code to be tried for errors
}
 catch(ExceptionType1 obj1)
{
  //Exception handler for ExceptionType1
}
  catch(ExceptionType2 obj2)
{
   //Exception handler for ExceptionType2
}
   ...

finally{
 //code to be executed before try block ends.
 // This executes whether or not an exception occurs in the try block.
 }
```

throw	**throws**
Used to throw an exception for a method	Used to indicate what exception type may be thrown by a method
Cannot throw multiple exceptions	Can declare multiple exceptions
Syntax: • `throw` is followed by an object (new *type*) • used inside the method	**Syntax:** • `throws` is followed by a class • and used with the method signature

https://www.w3schools.com/java/ref_keyword_throws.asp


Object
Throwable
Error
Exception
RunTime Exception
Checked Exception
www.educba.com

Example#1: Default Exception Handler

```java
import java.util.Scanner;

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

  int a, b, result;

  Scanner input = new Scanner(System.in);
  System.out.println("Input two integers");

  a = input.nextInt();
  b = input.nextInt();

  result = a / b;

  System.out.println("Result = " + result);
  }
}
```

TODO:

i. Provide Non-Zero integers as input (for a and b) and see the results

ii. Provide Non-Zero (for a) integer and Zero (for b)as input and see the results

Explain what and why you got the output in your report

Example#2: Exception Handling with try and catch

DivisionExp.java

```
1  import java.util.Scanner;
2
3  class DivisionExp {
4    public static void main(String[] args) {
5
6    int a, b, result;
7
8    Scanner input = new Scanner(System.in);
9    System.out.println("Input two integers");
10
11   a = input.nextInt();
12   b = input.nextInt();
13
14   // try block
15   try {
16     result  = a / b;
17     System.out.println("Result = " + result);
18   }
19
20   // catch block
21   catch (ArithmeticException e) {
22     System.out.println("Exception caught: Division by zero.");
23   }
24   }
25 }
26
```

TODO:

Explain what and why you got the output. Also, explain how this example is different than Example#1

Example#3: Finally and call stack

ExpectionTraceDemo.java

```
1  public class ExpectionTraceDemo{
2
3      public static void main(String[] args) {
4
5          printAverage(100, 0);
6          System.out.println("Exit main().");
7      }
8
9      public static void printAverage(int totalSum, int totalNumber) {
10
11         try {
12
13             int average = totalSum/totalNumber;
14             System.out.println("Average = " +
15                 totalSum + " / " + totalNumber + " = " + average);
16
17         } catch (ArithmeticException ae) {
18
19         //The stack trace displayed by the default error handler shows the
20         // sequence of method invocations that led up to the error
21             ae.printStackTrace();
22             System.out.println("Exception handled in " + "printAverage().");
23
24         } finally {
25
26             System.out.println("Finally done.");
27         }
28         System.out.println("Exit printAverage().");
29     }
30 }
```

TODO:

Explain the output in reference with "**finally**" and "**call stack**"

Example#4: Throwing exceptions

- The throw keyword is used to explicitly throw an exception - both checked or unchecked exceptions
- We can even create our own kind of exception and tell the exception object which one is to be thrown
- Exception Instance must be of type Throwable or a subclass of Throwable

```
4  class ThrowDemo {
5
6      //function to check if a GPA value is valid or not
7      public static void validate_gpa(double gpa) {
8          if ((gpa > 4) || (gpa < 0 )) {
9              //throw Arithmetic exception if GPA is over 4.0
10             throw new ArithmeticException("The GPA is NOT valid");
11         }
12         else {
13             System.out.println("The GPA is Valid!!");
14         }
15     }
16
17     public static void main(String args[]) {
18         double myGpa = 5;
19         validate_gpa(myGpa);
20     }
21 }
```

TODO:

i. Provide myGpa = 2 and see the results

i. Provide myGpa = 7 and see the results

Explain what and why you got the output in your report

Example#5: Using throws keyword

- 'throws' keyword is used in the method's signature to tell what exception this method may throw
- The method's caller have to handle the thrown exception using a try-catch block.

ThrowsExample.java

```
1  // The throws keyword tells what kind of exception may be thrown by a method.
2  class ThrowsExample {
3
4      //function to check if a GPA value is valid or not
5      public static void validate_gpa(double gpa) throws ArithmeticException {
6          if ((gpa > 4) || (gpa < 0 )) {
7              //throw Arithmetic exception if GPA is over 4.0
8              throw new ArithmeticException("The GPA is NOT valid");
9          }
10         else {
11             System.out.println("The GPA is Valid!!");
12         }
13     }
14
15     public static void main(String args[]) {
16         double myGpa = 5;
17         validate_gpa(myGpa);
18     }
19 }
20
```

TODO:

Explain the output

Example#6: Declaring our own Exception

- The Exception class is the superclass of all exceptions
- We need to extend the java.lang.Exception to create our custom exception

OwnExceptionDemo.java

```
1
2   class NonGCESException extends Exception{
3
4       NonGCESException(String s){
5           super(s);
6       }
7   }
8
9   class OwnExceptionDemo{
10
11      static void checkStudent(String collage)throws NonGCESException{
12          if( collage != "GCES")
13              throw new NonGCESException(" not a GCES buddy");
14          else
15              System.out.println("welcome to GCES");
16      }
17
18  public static void main(String args[]){
19
20          try{
21              checkStudent("PNC");
22          }
23          catch(Exception e){
24              System.out.println("Exception occured: "+ e);
25          }
26          finally{
27              System.out.println("END");
28          }
29      }
30  }
31
```