Exception Handling

Error:

An error is a condition that causes a disruption in the execution of a program.

An error can be a result of circumstances, such as incorrect code or insufficient memory resources.

There are three types of errors that can occur in the application:

1.**Compile-time** errors **(Syntax errors):** Occurs when statements are not constructed properly, keywords are misspelled, or punctuation is omitted.

2.**Run-time errors:** Occurs when an application attempts to perform an operation, which is not allowed at runtime.

3.**Logical errors:** Occurs when an application compiles and runs properly but does not produce the expected results.

1.Compile-time errors

The compile-time errors occur when the syntax of a programming language are not followed.

For example, in Java, if you use a keyword name as a variable name, a compile-time error will be raised by the compiler.

And also if you don't follow the proper syntax, it will throws compile time error which is also called as syntax error.

Example:

```
class Project1
{
  public static void main(String[] args)
  {
    System.out.Println("Hello all");
  }
}
```

The above code will throws an compilation error as we are using Println which is not a method in java.

2.Run time errors

The run-time errors occur during the execution of a program.

For example, if you are trying to perform an operation which is not possible, it results in a run-time error. Such an error is known as an exception.

Example:

Division by zero is a run-time error

Index Out of bound exception.

Example 1:

```
class Project1
{
   public static void main(String[] args)
   {
      System.out.println(10/0);
   }
}
```

ArithmeticException: / by zero

The above code will throws an runtime error which is called as exception as we are trying to divide the number by zero which is not possible to do.

Example 2:

```
class Project1
{
   public static void main(String[] args)
   {
      int[] a={10,20,30};
      System.out.println(a[5]);
   }
}
```

The above code will throws an run time error i.e

ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3

as we are trying to access the element from the index 5 which is not existing in the array.

We can handle the above exception by using exception handlers.

i.e try and catch.

3.Logical errors:

These errors depict the flaws in the logic of the program. These errors may occur when a programmer wrongly defined the logical statements of the program.

Logical errors are not detected by compiler or JVM

The programmer is solely responsible for them.

Logical errors will not throws any error. It will gives the output but not the expected output.

Example:

If you are trying to divide two numbers like 20 and 10.

Instead of dividing 20/10 if you divide 10/20 it will gives the result but not the expected output.

```
class Project1
 {
   public static void main(String[] args)
   {
     int a=20;
     int b=10;
     System.out.println(b/a);
   }
}
```

The above code will not throws any error.

It will gives the output but not an expected output.

We are trying to implement the code to divide a/b which must returns 2 but while writing the logic, it was defined in opposite way which return 0 but not 2.

Such kind of errors are called as Logical errors.

An exception can lead to the termination of an application.

Dealing with such abnormal behavior is called exception handling.

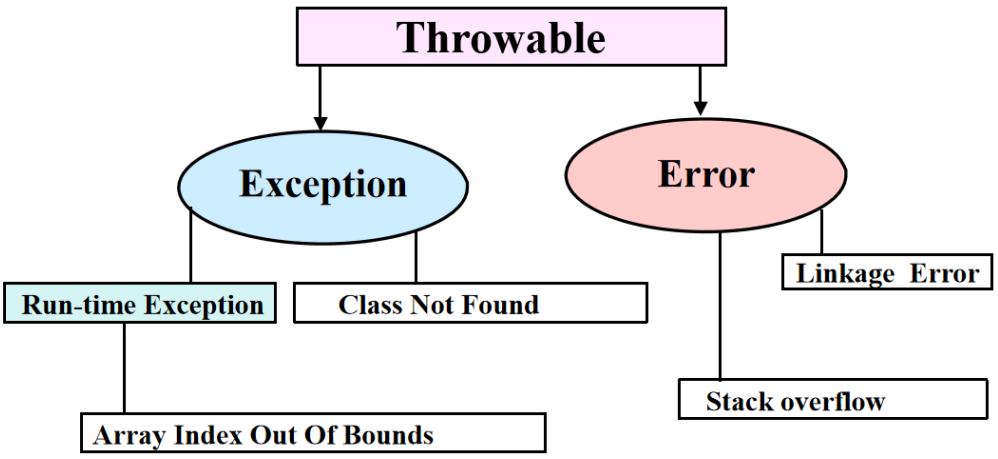
When a run-time error occurs, an exception is thrown by the JVM which can be handled by an appropriate exception handler.

The Java run-time system proceeds with the normal execution of the program after an exception is handled.

If no appropriate exception handler is found by the JVM, the program is terminated.

There are several built-in exceptions that have been identified in Java. In order to deal with these exceptions, Java has various built-in exception classes.

The following diagram explains about the types of errors.



The Throwable class is the base class of exceptions in Java.

You can throw only those exception objects that are derived from the Throwable class.

The following two classes are derived for the Throwable class:

- Exception
- Error

Java exceptions are categorized into the following types:

1. Checked exceptions

2. Unchecked exceptions

Checked Exceptions

These are the invalid conditions that occur in a Java program due to the problems, such as accessing a file that does not exist or referring a class that does not exist.

The checked exceptions are the objects of the Exception class or any of its subclasses excluding the RuntimeException and Error class.

For example, a checked exception IOException is raised, when a program tries to read from a file that does not exist.

The checked exceptions make it mandatory for a programmer to handle them.

If a checked exception is not handled, then a compile-time error occurs.

Unchecked Exceptions

The unchecked exceptions occur because of programming errors.

Therefore, the compiler does not force a programmer to handle these exceptions.

Such errors should be handled by writing bug free codes.

However, there are times such situations cannot be eradicated.

For example in a calculator application, if you divide a number by zero, an unchecked exception is raised.

When an unexpected error occurs, Java creates an exception object.

After creating the exception object, Java sends it to the program by throwing the exception.

The exception object contains information about the type of error and the state of the program when the exception occurred.

You need to handle the exception by using an exception handler and processing the exception.

You can implement exception handling in a program by using the following keywords and blocks:

- try
- catch
- finally

Using try and catch Blocks

A try block encloses the statements that might raise an exception and defines one or more exception handlers associated with it.

If an exception is raised within the try block, the appropriate exception handler that is associated with the try block processes the exception.

In Java, the catch block is used as an exception handler. A try block must have at least one catch block that follows the try block, immediately.

The catch block specifies the exception type that you need to catch.

You can declare the try and the catch block by using the following syntax:

```
try
{
 // Statements that can cause an exception.
}
catch(exceptionname obj)
{
 // Error handling code.
}
```

In the preceding syntax, the catch block accepts the object of the Throwable class or its subclass that refers to the exception caught, as a parameter.

When the exception is caught, the statements within the catch block are executed.

Example:

```
class Project1
 {
    public static void main(String[] args)
    {
      int a=10;
      int b=0;
      System.out.println(a/b);
      System.out.println("Bye");
    }
}
```

The above code will throws an run-time error i.e exception and the program execution will gets terminated because of that the operation (dividing a number by zero).

Because of this exception the remaining part of the code will not gets executed.

If we handle the exception the code will not throws any error and also the other part of the code will gets executed.

```
class Project1
 {
   public static void main(String[] args)
   {
    int a=10;
    int b=0;
    try
    {
        System.out.println(a/b);
    }
    catch(Exception e)
    {
        System.out.println("Something went wrong..!");
    }
    System.out.println("Bye");
   }
}
```

Output: Something went wrong..!"

Bye

In the above code we are just displaying some statement like something went wrong.

We can also display the type of exception which is holding by the exception object e.

```
class Project1
 {
   public static void main(String[] args)
   {
     int a=10;
     int b=0;
     try
     {
         System.out.println(a/b);
     }
     catch(Exception e)
     {
         System.out.println("Something went wrong..!"+e);
     }
     System.out.println("Bye");
   }
}
```

Output:

```
Something went wrong..!java.lang.ArithmeticException: / by zero
Bye
```

Example 2:

```
class Project1
 {
   public static void main(String[] args)
   {
     int[] arr={10,20,30};
     try
     {
         System.out.println(arr[5]);
     }
     catch(Exception e)
     {
         System.out.println("Something went wrong..!"+e);
     }
     System.out.println("Bye");
   }
}
```

Output:

Something went wrong..!java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3

Bye

```
class Project1
 {
    public static void main(String[] args)
    {
      int a=10;
      int b=2;
      int[] arr={10,20,30};
      try
      {
          System.out.println(a/b);
          System.out.println(arr[5]);
      }
      catch(Exception e)
      {
          System.out.println("Cannot divide by zero");
      }
      System.out.println("Bye");
    }
}
```

Output:

5

Cannot divide by zero

Bye

The above code will throws an error i.e cannot divide by zero but that is not the only exception,

There is another exception to be handled

In java, the try-catch block is used to handle exceptions.

It allows you to write code that handles exceptions that may be thrown during the execution of your program.

We can also handle different types of exceptions with multiple catch blocks.

Suppose we write few lines of code and we donot know which line can generate exception but we know which types of exceptions can be generated.

In this case we can use try with multiple catch.

Example:

```
class Project1
 {
   public static void main(String[] args)
   {
    int a=10;
    int b=0;
    int[] arr={10,20,30};
    try
    {
        System.out.println(a/b);
        System.out.println(arr[5]);
    }
    catch(ArithmeticException e)
    {
        System.out.println("Cannot divide by zero");
    }
    catch(ArrayIndexOutOfBoundsException e2)
    {
       System.out.println(e2);
    }
    System.out.println("Bye");
   }
}
```

Handling parents and child exception both:

When catching both child and parent exceptions in a try-catch block, it is generally recommended to catch the child exceptions before the parent exception.

The reason for this is that if you catch the parent exception before the child exception, the catch block for the parent execution will also catch any child exception that are subclasses of the parent exception.

This can make it more difficult to handle the child exceptions separately.

```
class Project1
 {
   public static void main(String[] args)
   {
     int a=10;
     int b=0;
     int[] arr={10,20,30};
     try
     {
         System.out.println(a/b);
         System.out.println(arr[5]);
     }
     catch(Exception e)
     {
         System.out.println("Parent class exception of every exception");
     }
     catch(ArithmeticException e2)
     {
        System.out.println(e2);
     }
     System.out.println("Bye");
   }
```

The above code will give compile time error i.e. error:

ArithmeticException has already been caught

Right way

```
class Project1
 {
    public static void main(String[] args)
    {
      int a=10;
      int b=2;
      int[] arr={10,20,30};
      try
      {
          System.out.println(a/b);
          System.out.println(arr[5]);
      }
      catch(ArithmeticException e1)
      {
         System.out.println(e1);
      }
      catch(Exception e)
      {
          System.out.println("Parent class exception of every exception");
      }
      System.out.println("Bye");
    }
}
```

finally block:

During the execution of a Java program, when an exception is raised, the rest of the statements in the tryblock are ignored.

Sometimes, it is necessary to execute certain statements irrespective of whether an exception is raised.

The finally block is used to execute these required statements.

The statements specified in the finally block are executed after the control has left the try-catch block.

The 'try catch' is finished.

```
class Project1
 {
   public static void main(String[] args)
   {
     int a=10;
     int b=0;
     try
     {
         System.out.println(a/b);
     }
     catch(Exception e)
     {
         System.out.println(e);
     }
     finally
     {
     System.out.println("The 'try catch' is finished");
     }
   }
}
```

Output:

```
java.lang.ArithmeticException: / by zero
The 'try catch' is finished
```

```
class Project1
 {
    public static void main(String[] args)
    {
      int a=10;
      int b=2;
      try
      {
          System.out.println(a/b);
      }
      catch(Exception e)
      {
          System.out.println(e);
      }
      finally
      {
      System.out.println("The 'try catch' is finished");
      }
    }
}
```

Output:

```
5
The 'try catch' is finished
```