LAB 8

Exception Handling

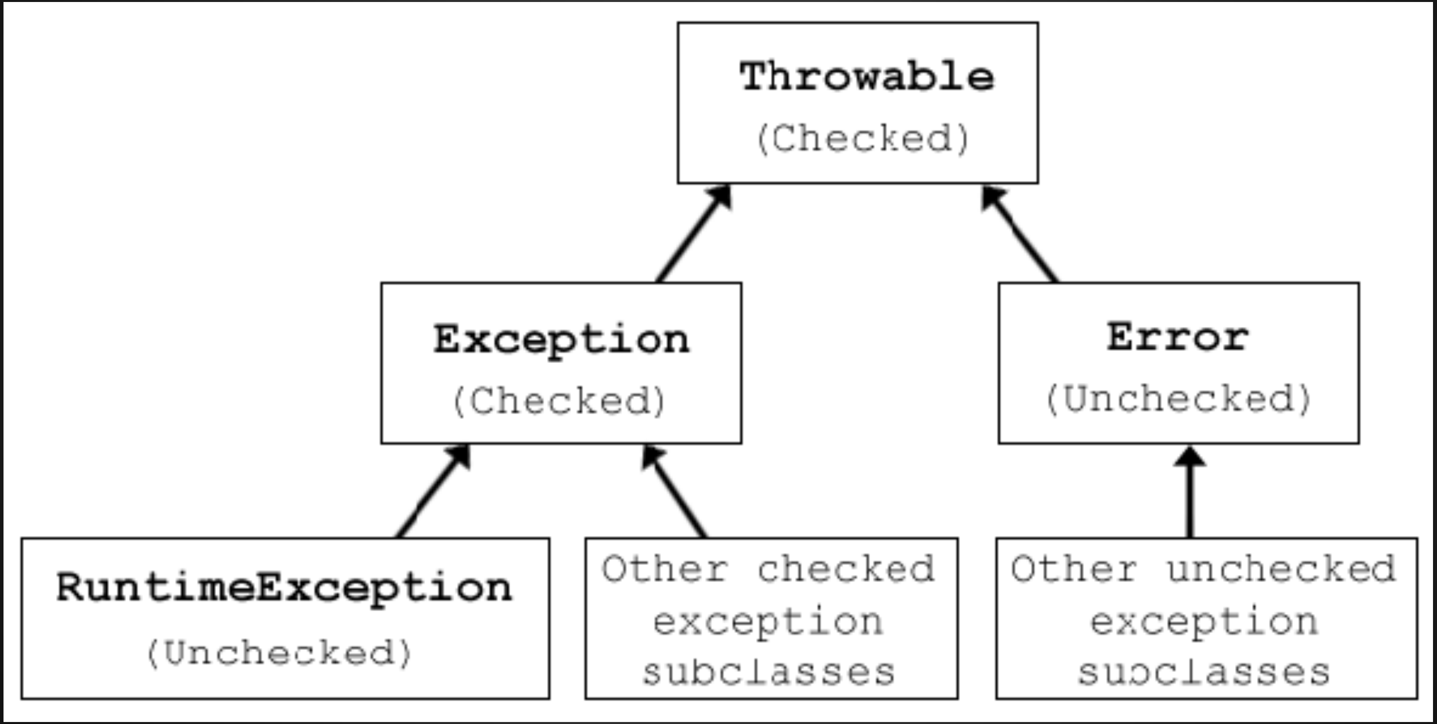
What is exception handling?

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

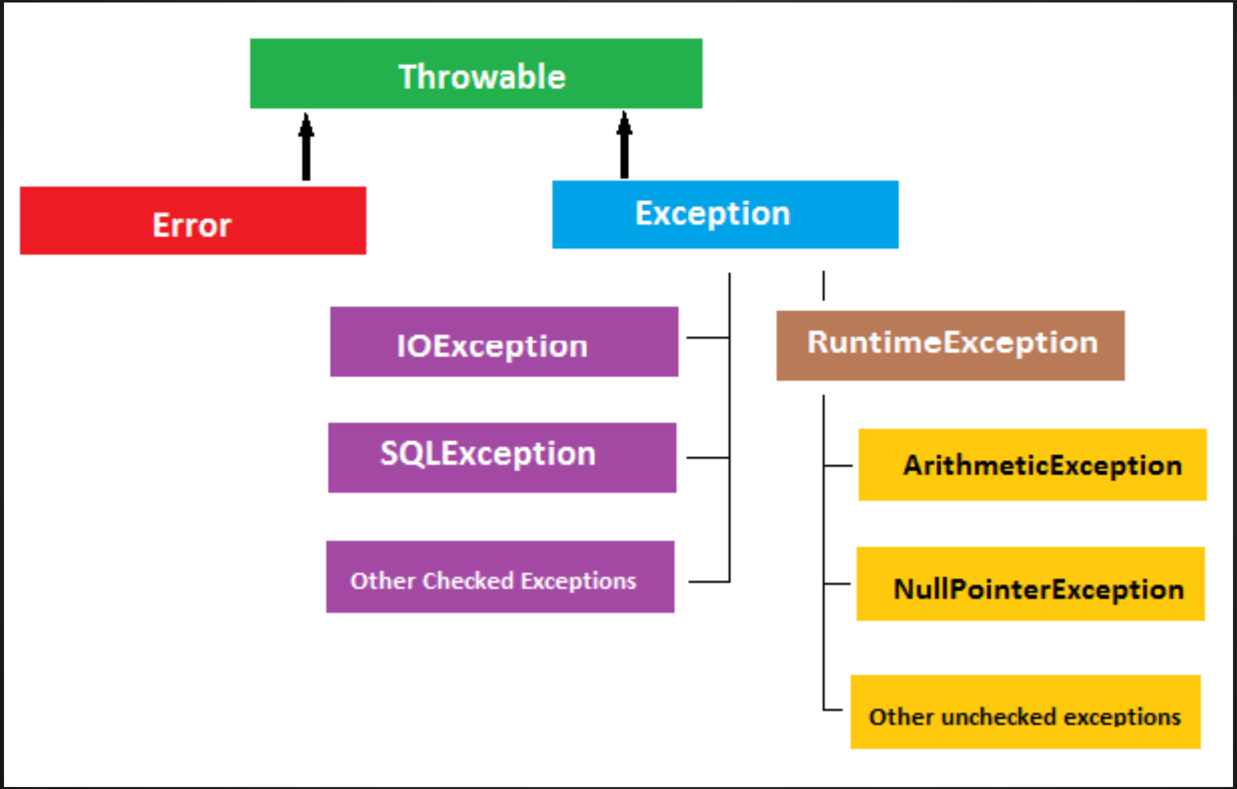
How the program *handles* the exception depends on the severity of the issue.

Understanding Various Exceptions

Exception Hierarchy



Examples


Throwable
Error
Exception
IOException
SQLException
Other Checked Exceptions
RuntimeException
ArithmeticException
NullPointerException
Other unchecked exceptions

Checked Exceptions

Checked exceptions are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using `throws` keyword.

```
try {
    String input = reader.readLine();
    System.out.println("You typed : "+input); // Exception prone area
} catch (IOException e) {
     e.printStackTrace();
}
```

Remember being forced to add the Try and Catch around your Scanner or BufferedReader when file reading? The Java compiler is forcing you to make that checked exception before run time.

Unchecked Exceptions

Unchecked are the exceptions that are not checked at compile time. It is up to the programmers to be civilized and specify or catch the exceptions.

In Java, exceptions under `Error` and `RuntimeException` classes are unchecked exceptions; everything else under throwable is checked.

```
class Main {
   public static void main(String args[]) {
      int x = 0;
      int y = 10;
      int z = y/x;
  }
}
```

It compiles fine, but it throws `ArithmeticException` when run. The compiler allows it to compile, because `ArithmeticException` is an unchecked exception

Runtime Exceptions

A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.

Examples: ClassCastException, NullPointerException, IndexOutOfBoundsException, ArrayIndexOutOfBoundsException.

Errors

These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Catching Exceptions

```
try
{
    //Protected code
} catch(ExceptionName e) {
    //Catch block
}
```

Example

```
import java.io.*;
public class ExcepTest {

    public static void main(String args[]) {
        try {
            int a[] = new int[2];
            System.out.println("Access element four: " + a[3]);
         } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception thrown: " + e);
         }
        System.out.println("Out of the block");
    }
}
```

throw/throws keyword

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

Try to understand the difference between throws and throw keywords.

Example

```
import java.io.*;
public class BankAccount
{
   public void deposit(double amount) throws RemoteException
   {
      // Method implementation
      throw new RemoteException();
   }
   //Remainder of class definition
}
```

Another Example

```
//Other things....
public static void throwException() throws InvalidNumberException {
     String strNumber = “matt”;
      int number;
      try{
           number = Integer.parseInt(strNumber);
      }
      catch (NumberFormatException e){
           throw new InvalidNumberException("You did not enter an Integer.");
      }
}
```

finally keyword

The `finally` keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

Example

```
public class ExcepTest {
   public static void main(String args[]){
      int a[] = new int[2];
      try {
         System.out.println("Access element four: " + a[3]);
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception thrown: " + e);
      } finally {
         a[0] = 6;
         System.out.println("First element value: " +a[0]);
         System.out.println("The finally statement is executed");
      }
   }
}
```

Declaring your own Exception

- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

Example

```
@SuppressWarnings("serial") //may need to add
class InvalidTokenException extends Exception {
    public InvalidTokenException() {}

    public InvalidTokenException(String msg) {
     super(msg);
    }
}
```

Lab Challenge

Write a solution for

```
public static void isWord (String word):
```

Throws InvalidTokenException, will check for the invalid characters such as **!,@,#, and all digits.** When finding one of these invalid characters **throw** an **InvalidTokenException.**