Exception Handling

Before we get into anything new, let's review errors and exceptions together.

In Java, exception handling means taking care of **runtime errors,** so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as `ClassNotFoundException` and `IOException`, which refer to missing classes and bad input/output respectively**.**

As we've seen, an **exception** is an unwanted or unexpected event which occurs during the execution of a program (i.e. at runtime), that disrupts the normal flow of the program's instructions. In Java, when an exception occurs within a method, Java creates an object with information about what happened. Predictably, this object is called the `exception` object. It contains information like the name and description of the exception and the state of the program when the exception occurred.

Here's a list of major reasons why **exceptions** occur in a program:

- Invalid user input
- Device failure
- Loss of network connection
- Physical memory limitations (e.g., the user's computer has run out of space to complete the process)
- Coding errors
- Attempting to open an unavailable file

#Errors vs Exceptions

Unlike exceptions, **errors** represent irrecoverable conditions such as the Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc. To clarify: exceptions are errors that we've anticipated and can write code to deal with. Errors, on the other hand, are usually beyond the control of the programmer. Therefore, we don't usually try to handle errors directly–we just write code that tries to keep them from happening.

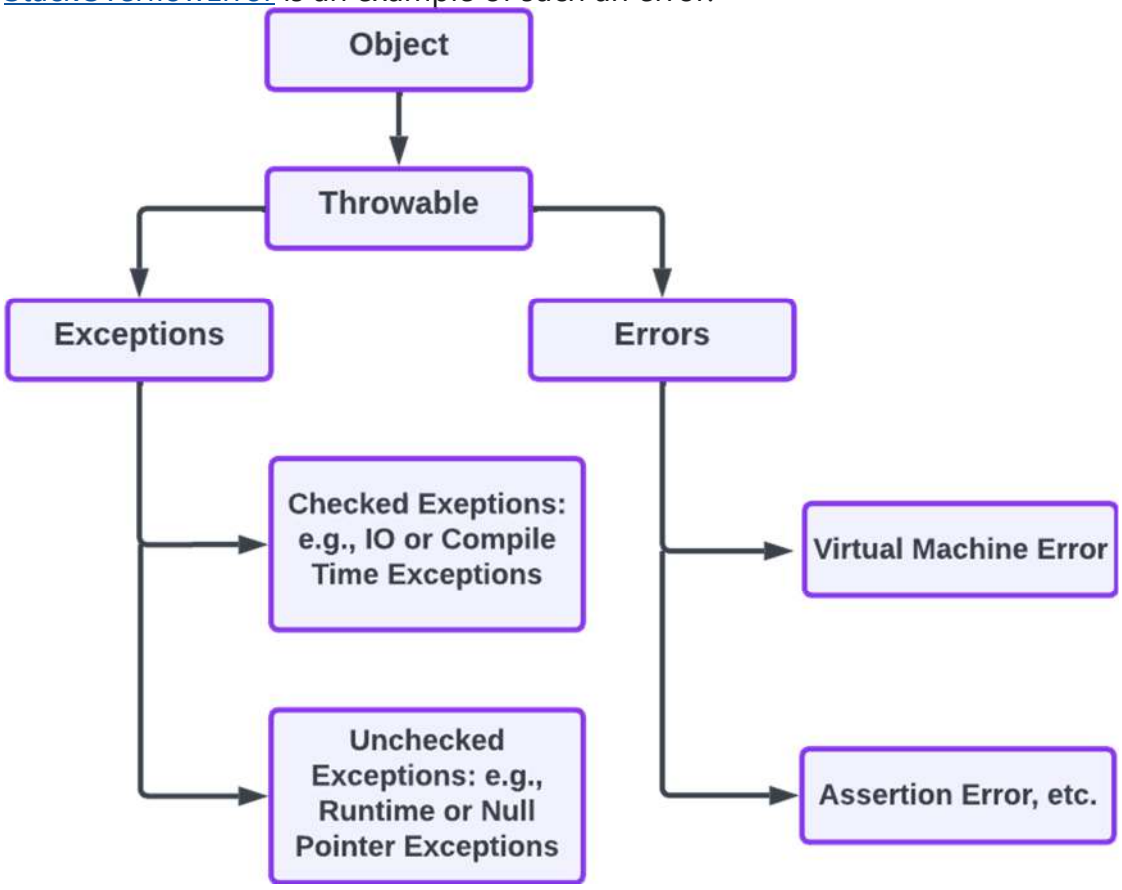
Here are the most important differences between Errors and Exceptions:

- **Error**: An error indicates a serious problem that a reasonable application should not try to catch.
- **Exception**: An exception indicates conditions that a reasonable application might try to catch and appropriately handle, so long as resources and access permit them to do so.

#Exception Hierarchy

All exception and error types are subclasses of the `Throwable` class. One branch of this hierarchy is headed by `Exception`. This class is used for exceptional conditions that our programs should catch. One example of the Exception class is the NullPointerException

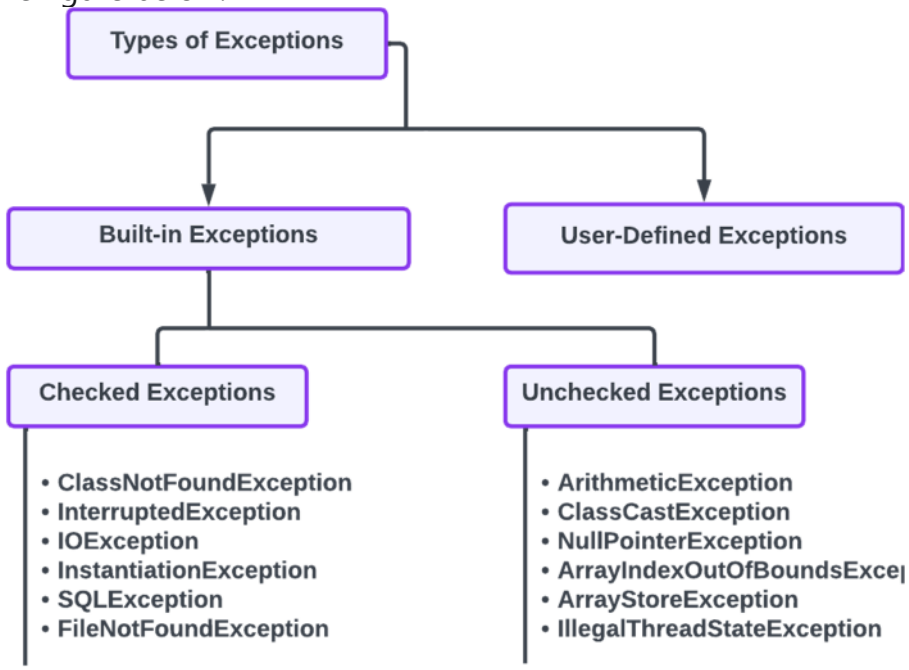
The other major branch of the `Throwable` class, `Error`, is used by the Java run-time system (aka, the Java Virtual Machine or JVM) to indicate errors having to do with the run-time environment itself (aka the Java Runtime Environment or JRE). StackOverflowError is an example of such an error:



Stemming from the original Object class, these are the branches of the Throwable class

#Types of Exceptions

Java defines several types of exceptions that relate to its various class libraries. As shown in the figure below:



Java's built-in errors that branch off from the classes defined above

The branch with no list in the image above is potentially the most expansive: we can define as many or as few of our own exceptions as we need to, and decide the error handling that goes along with them. We'll be going in-depth on defining our own exceptions in a lesson soon.

#Extra Resources

When time permits, watch the following video to see try/catch in action and learn a bit more about how to handle exceptions in your code:
https://youtu.be/adTDlH0lhaA

Problem Statement

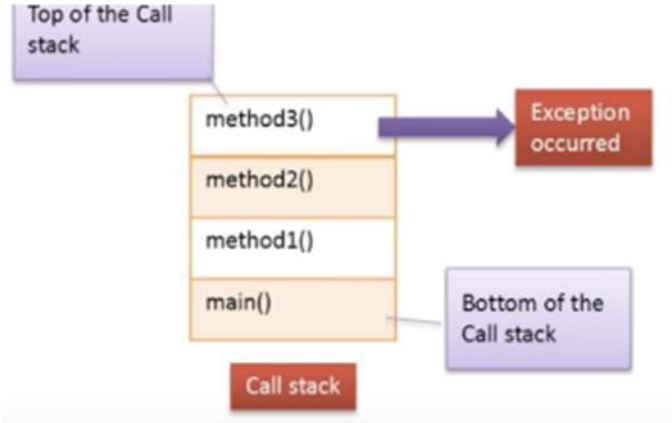
Write a program to call a method from another method. Catch the exception in the caller method if that exception is not handled in the callee method.

Write a program using a `try` - `catch` block. Catch the exception in the outer catch block if the exception is not handled in the inner catch block.

#Explanation

#Exception propagation:

If an exception occurs in a function, it expects to be handled; if not, it will be thrown to all of the methods in the call stack at runtime until it gets handled. The process in which an exception is dropped from the top to the bottom of the stack is called **exception propagation**.



#Steps to be followed:

1. **Create a Java Class:**
   - Define a class named `Main`.
2. **Create a Method:**
   - Implement a method named `method1()`.
3. **Declare an Array and Initialize:**
   - Inside `method1()`, declare and initialize an array `arr` with values [1, 2, 3, 4].
4. **Try to Access an Index Out of Bounds in the array:**
   - Attempt to access an element at index 4 within `method1()`, which is out of bounds.
5. **Create Another Method and Call the First Method:**
   - Implement a method named `method2()` that invokes `method1()`.

6. **Place Try and Catch Blocks in the Second Method:**
   - Inside `method2()`, enclose the call to `method1()` within a try block.
7. **Catch** `ArrayIndexOutOfBoundsException` in Method 2:
   - Add a catch block in `method2()` to catch `ArrayIndexOutOfBoundsException`.
   - Inside the catch block, print "Method2: ArrayIndexOutOfBoundsException handled".
8. **Create a Main Method and Call the Second Method:** [This code is already provided to you in the template]
   - In the `main()` method, create an instance of the `Main` class.
   - Call `method2()` using the created instance.

#Expected Output:

- **Exception Propagation and Handling:**
  - An `ArrayIndexOutOfBoundsException` is thrown within `method1()`.
  - This exception propagates from `method1()` to `method2()`.
  - The catch block in `method2()` catches the propagated exception.

Run

Main.java

```
public class Main {

    // Method to demonstrate exception propagation
    public void method1() {
        int[] arr = {1, 2, 3, 4};
        int value = arr[4]; // Access an index out of bounds
    }

    // Method to handle exception and demonstrate propagation
    public void method2() {
        try {
            method1(); // Call method1 within a try block
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Method2: ArrayIndexOutOfBoundsException handle
d");
        }
    }

    // main() method to run the program
    public static void main(String args[]) {
```

```
        Main exceptionSolutionObject = new Main();
        exceptionSolutionObject.method2(); // Calling method2 to demonstrate e
xception propagation
    }
}
```

#Nested try blocks:

We can use a `try` block within another `try` block. If an exception occurs in an inner `try` block and is not handled in the inner `catch`, it should be caught and handled in the outer `catch` block. Write a function with nested `try` and `catch` blocks according to the following steps.

Steps to be followed:

1. **Create a Java Class:**
   - Define a class named `Main`.
2. **Create a Method with Nested Try-Catch:**
   - Implement a method named `method()`.
3. **Create Nested Try Block:**
   - Inside `method()`, open an outer try block.
4. **Create Inner Try Block:**
   - Inside the outer try block, create an inner try block.
5. **Declare and Initialize an Array:**
   - Inside the inner try block, declare and initialize an array `arr` with values [1, 2, 3, 4].
6. **Print and Access Array Elements:**
   - Print an element of the array using `arr[0]`.
   - Attempt to print an element at index 5, which is out of bounds for the array.
7. **Wrap Inner Try-Catch in Outer Try-Catch:**
   - Wrap the inner try-catch block in the outer try-catch structure.
8. **Catch** `ArithmeticException` in Inner Catch Block:
   - In the inner catch block, catch an `ArithmeticException`.
   - Print "ArithmeticException occurred" in the inner catch block.
9. **Catch** `ArrayIndexOutOfBoundsException` in Outer Catch Block:
   - In the outer catch block, catch an `ArrayIndexOutOfBoundsException`.

- Print "ArrayIndexOutOfBoundsException is handled in the outer catch block" in the outer catch block.

10. **Call the Method to Run the Program:** [This code is already provided to you in the template]
    - Inside the `main()` method, create an instance of the `Main` class.
    - Call the `method()` on the instance to run the program.

Run

Main.java

```
public class Main {
    void method() {
        try {
            try {
                int[] arr = {1, 2, 3, 4};
                System.out.println("Value at index 0: " + arr[0]);
                System.out.println("Value at index 5: " + arr[5]); // Access i
ndex out of bounds
            } catch (ArithmeticException e) {
                System.out.println("ArithmeticException occurred");
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("ArrayIndexOutOfBoundsException is handled in t
he outer catch block");
        }
    }

    // main() method to run the program
```

```
    public static void main(String args[]) {
        Main nestedTryBlockSolutionObject = new Main();
        nestedTryBlockSolutionObject.method();
    }
}
```

Which exception is thrown when the index of some sort (such as to an array, to a string, or a similar data structure) is out of range?

NumberFormatException

NullPointerException

IndexOutOfBoundsException

Exception

InfoWarningTip

`IndexOutOfBoundsException` is a more general exception that covers any attempt to access an index that is out of bounds in an array, list, or similar data structure. However, if we're specifically dealing with an array and its index, we should use `ArrayIndexOutOfBoundsException`