Exception Handling

What is an Exception?

An exception is something that happens when your program is running, that stops the normal flow of the program. This is usually caused by something that is out of your control, like a user inputting the wrong data type. Instead of crashing the program, you can handle the exception and continue running the program.

Here is an example of an exception:

```java
public ExepctionExample {
    public static void main(String[] args) {
        int myInt = Integer.parseInt("hello");
    }
} // This would crash the app
```

To fix this we use try and catch, the catch will never run if the try is successful.

```java
public ExepctionExample {
    public static void main(String[] args) {
        try {
            int myInt = Integer.parseInt("hello");
        } catch (Exception e) { // To be more specific you can use the
specific exception like NumberFormatException
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

It usually makes sense to have a try and catch on the method that calls it and not the method itself. This is because you can have multiple methods that call the same method, and you don't want to have to put a try and catch on every method that calls it. This is called throwing up the call stack.

```java

public class ExceptionExample {
    public static void main(String[] args) {
        try {
            myMethod();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static void myMethod() {
        int myInt = Integer.parseInt("hello");
    }
}
```

You can catch multiple exceptions

```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            myMethod();
        } catch (NumberFormatException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (NullPointerExeptions | ArrayIndexOutOfBoundsException e )
{
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static void myMethod() {
        int myInt = Integer.parseInt("hello");
    }
}
```

You can also have a finally block, this will always run, even if there is an exception. This is useful for closing a file or database connection.

```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            myMethod();
        } catch (NumberFormatException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (NullPointerExeptions | ArrayIndexOutOfBoundsException e )
{
            System.out.println("Error: " + e.getMessage());
        } finally {
            System.out.println("This will always run");
        }
    }

    public static void myMethod() throws NumberFormatException //This is
how you throw an exception
     {
        int myInt = Integer.parseInt("hello");
    }
}
```