While folks are joining

Get you laptops ready and login to www.crio.do

Start up your workspace.

Crio Sprint: JAVA-2

Session 8

Today’s Session Agenda

- Exception Handling Basics
- Unit Test Exceptions
- QCalc Module Introduction
    - Module 4 - Debug and Handle Exceptions
    - Module 5 - Extend Simple Calculator

QCalc - Module 4: Debug and Handle Exceptions

- In this module:
  - Debug and Handle Exceptions for invalid data.
  - Write unit test to validate the Exception being thrown.

What is an Exception and Exception Handling?

- During the **execution** of a program, things can, and **do go wrong.**
- The user may **enter inappropriate data**,and **program during runtime may throw an error** and terminate**.**
- **Exception handling** is the process designed to **handle such exceptions,** so that the program can **continue gracefully**.

Note: We will only be working with basics of Exception in this session and the QCalc ME. We will go into much more depth in the next Sprint with a dedicated Byte.

Curious Cats

How is Exception different from Error?

- An **Error** indicates **serious problems** that a reasonable application **can't handle or cannot be handled.**
- Nobody can guess or control when it might happen.
  - For. eg, Out of Memory Error
- An Exception "indicates conditions that a reasonable application **might want to handle**."
  - For. eg. Invalid Operations like Divide By Zero.
  - It can be guessed and handled pretty well using try/catch (Will be discussed)

Create a New Demo Java Project

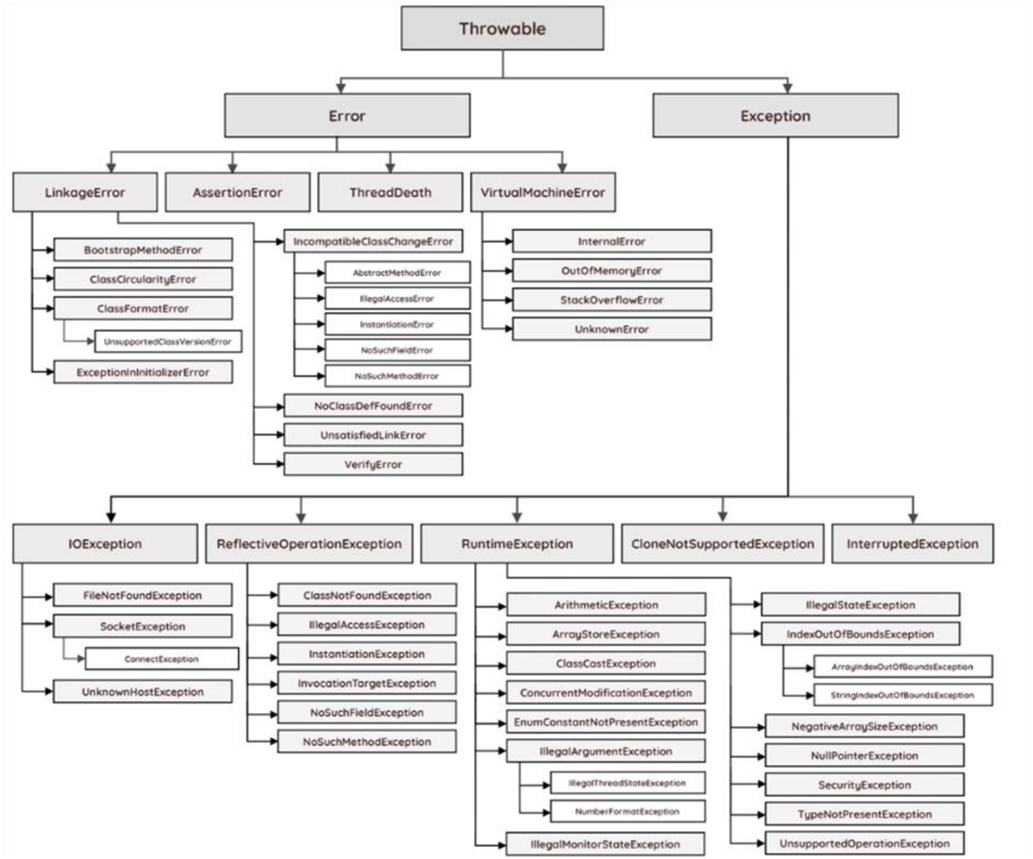
- Create a new “exceptiondemo” Project using the Spring Initializer Extension (cd ~/workspace)
- Create a new file `Rectangle.java` under src > com > crio > exceptiondemo folder.
- Create a new file `RectangleTest.java` under test > com > crio > exceptiondemo folder.
- In this file, we will be writing and executing our unit tests.

Activity 1 - Find the Exception Type

Paste below the snippets in the main method of application that you created

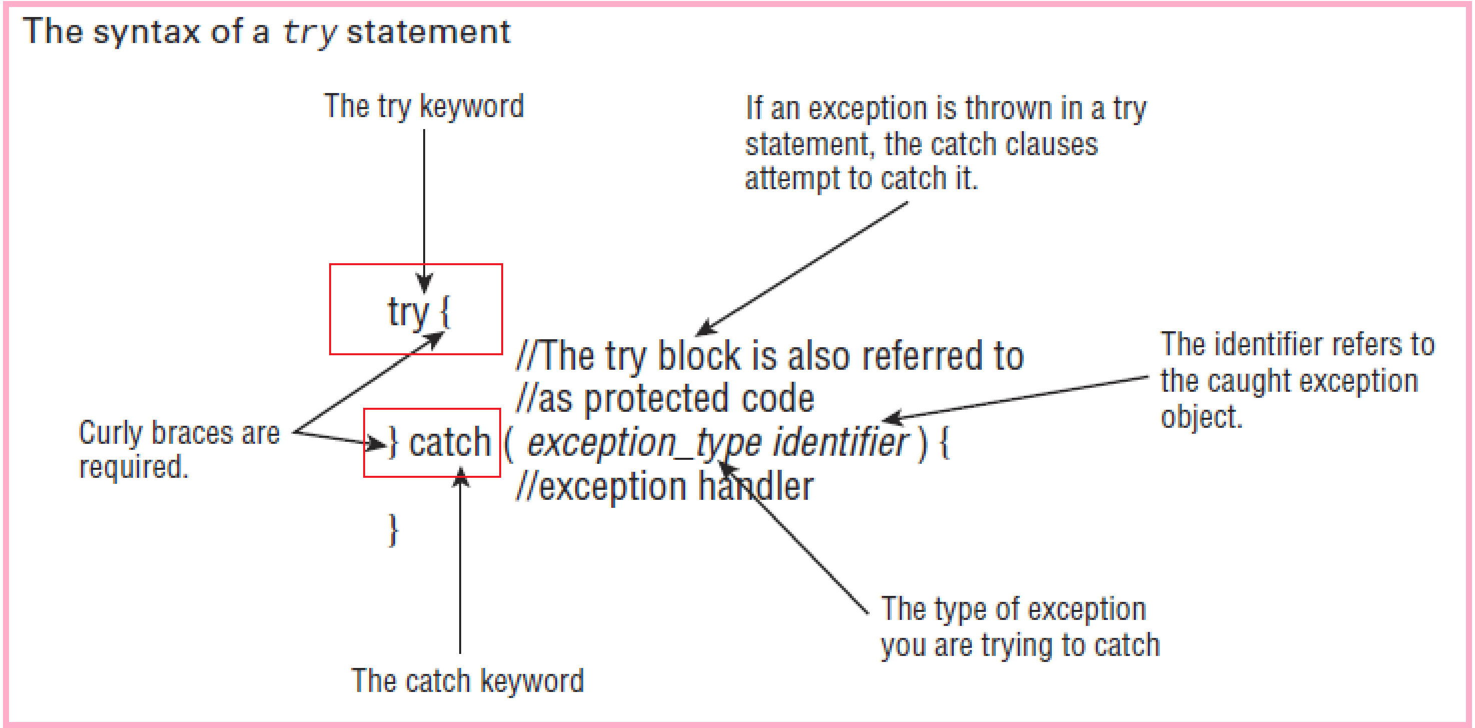
```
int a=50/0;

String s=null;
System.out.println(s.length());

String s="abc";
int i=Integer.parseInt(s);

int a[]=new int[5];
a[10]=50;
```

Java Exception Class Hierarchy



How can you stop the process from crashing?

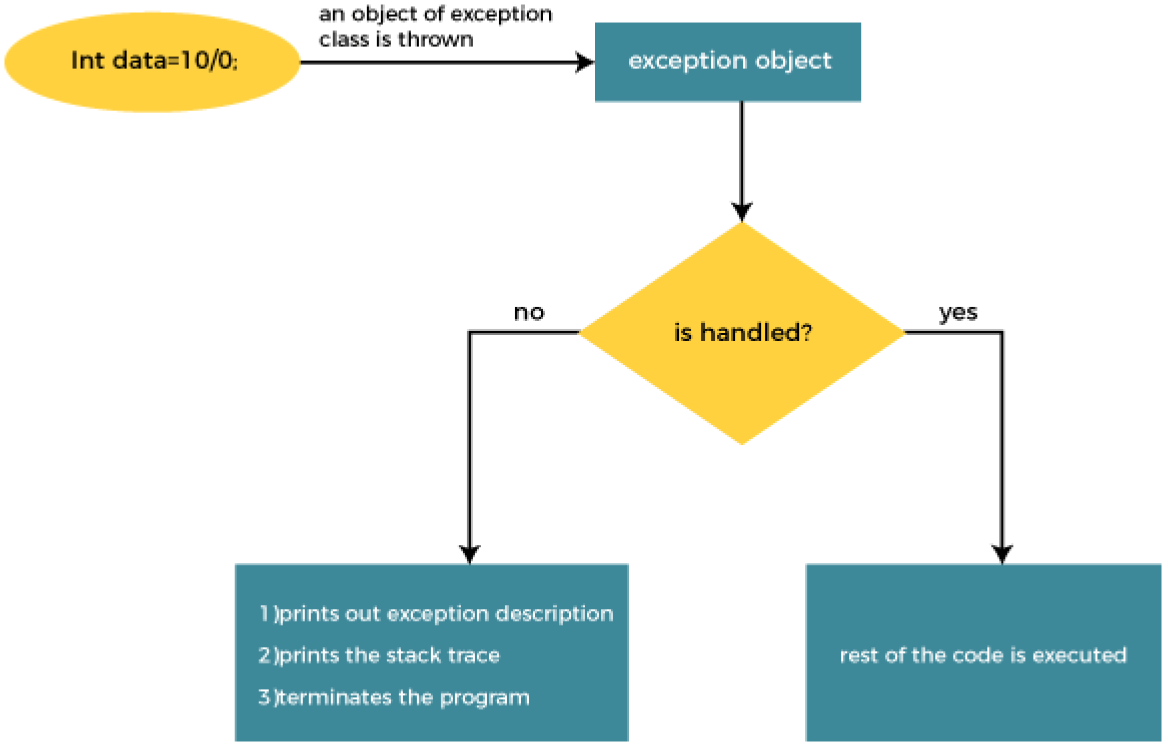
Using the **try catch** syntax

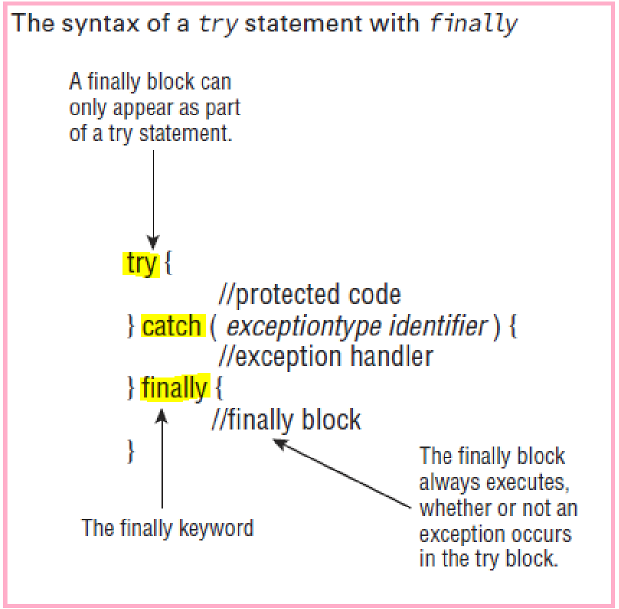

The syntax of a try statement
The try keyword
If an exception is thrown in a try statement, the catch clauses attempt to catch it.
try {
//The try block is also referred to
//as protected code
} catch (exception_type identifier) {
//exception handler
}
The identifier refers to the caught exception object.
Curly braces are required.
The type of exception you are trying to catch
The catch keyword

Try Catch Finally Block

```java
public class Main {

    public static void main(String[] args) {
        try{
            int data=50/0; //may throw exception
        }
        //handling the exception
        catch(ArithmeticException e){
            System.out.println(e);
            e.printStackTrace();
        }finally {
            System.out.println("finally block is always executed");
        }
               System.out.println("Rest of the program can continue after graceful handling");
    }
}
```

Exception Handling using Try-Catch


Int data=10/0;
an object of exception class is thrown
exception object
is handled?
no
yes
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed


The syntax of a try statement with finally
A finally block can only appear as part of a try statement.
The finally keyword
The finally block always executes, whether or not an exception occurs in the try block.

```
try {
        //protected code
} catch ( exceptiontype identifier ) {
        //exception handler
} finally {
        //finally block
}
```

Would we ever need to throw an exception explicitly?

- Sometimes we might want to define our own set of conditions or rules and avoid abnormal behaviour.
    - For.e.g, Person having age less than 18 are not eligible for voting.
    - This abnormal behaviour can be handled using exceptions gracefully.
    - We can throw ArithmeticException with a message “Not eligible for Voting with Age < 18” specifying the correct error.
- We can define our own custom exceptions. We’ll visit this in later sprints.

Throw Exception explicitly

```java
public class Main {
    public static void validate(int age) {
        if(age<18) {
            throw new ArithmeticException("Person is not eligible to vote");
        }
        else {
            System.out.println("Person is eligible to vote!!");
        }
    }
    public static void main(String args[]) {
          validate(13);
          System.out.println("rest of the code...");
     }
}
```

Curious Cats

- When we throw an exception, who should handle it with a try catch?
    - **One who calls the method** (who might throw exception) should handle it **using try/catch block**.
    - **Propagate the exception** to the top to the invoking method.
        - A method may contain several nested methods being called inside it. The exception could be handled anywhere between them or at the top.
- Can we write code to catch an exception of one type and expect the catch block to be executed for any Exception?
    - No. Exception being thrown or the parent class of that Exception type should be present in catch block.

Activity 2 - Let’s test Rectangle Class

```
public class Rectangle {
    private final double width, height; //sides
    public Rectangle() {
        this(1,1);
    }
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
    public double calculateArea() {
        return width * height;
    }
    public boolean isSquare(){
        if(width == height){  return true; }
        return false;
    }
}
```

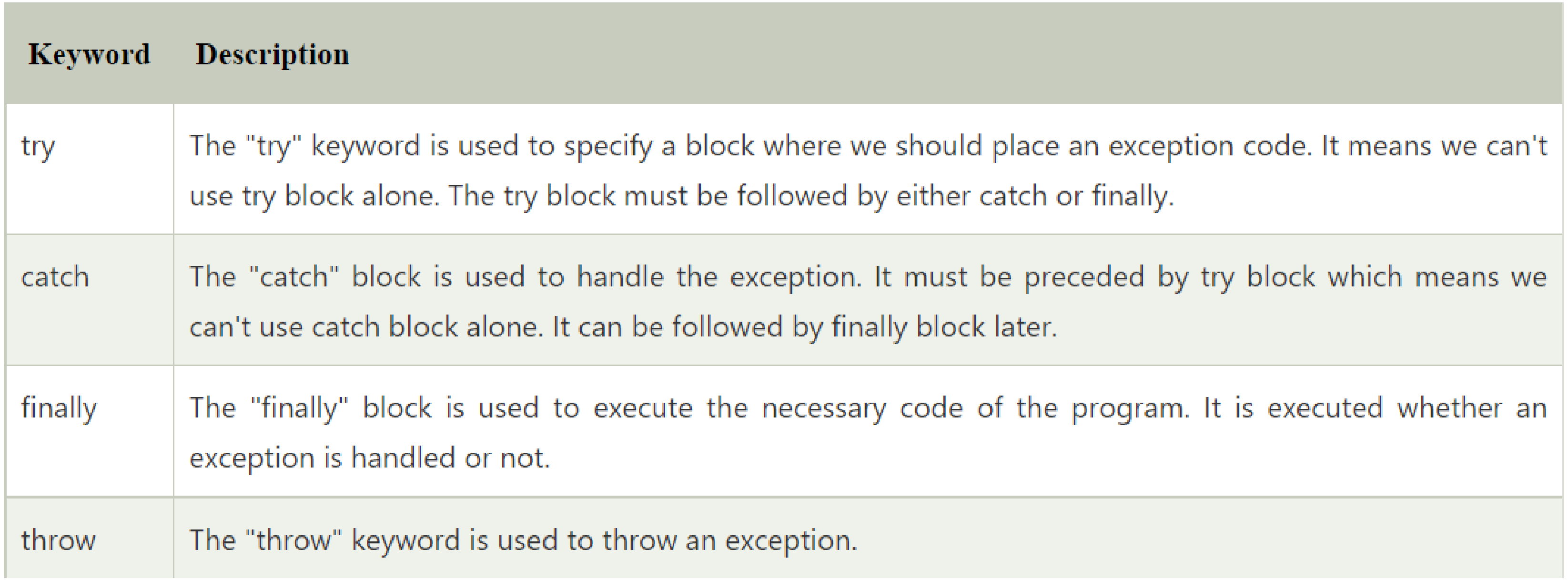
Activity 2.1 - Throw Exception for Invalid Arguments

```
public class Rectangle {
   private final double width, height; //sides
   public Rectangle() {
      this(1,1);
   }
   public Rectangle(double width, double height) {
   if(width <=0 || height <=0){
      throw new ArithmeticException("Width or Height Cannot be Negative or Zero");
   }
      this.width = width;
      this.height = height;
   }
   public double calculateArea() {
      return width * height;
   }
   public boolean isSquare(){
      if(width == height){  return true; }
      return false;
   }
}
```

Activity 2.2 - Handle Exceptions using Try Catch in main

```
public class Main {
   public static void main(String args[]){
   try {
      Rectangle r = new Rectangle(0,-3);
   } catch(Exception e) {
      System.out.println(e.getMessage());
   }
   System.out.println("rest of the code...");
  }
}
```

Java Exception Keywords



Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.

Random Memes

How to test Exceptions?

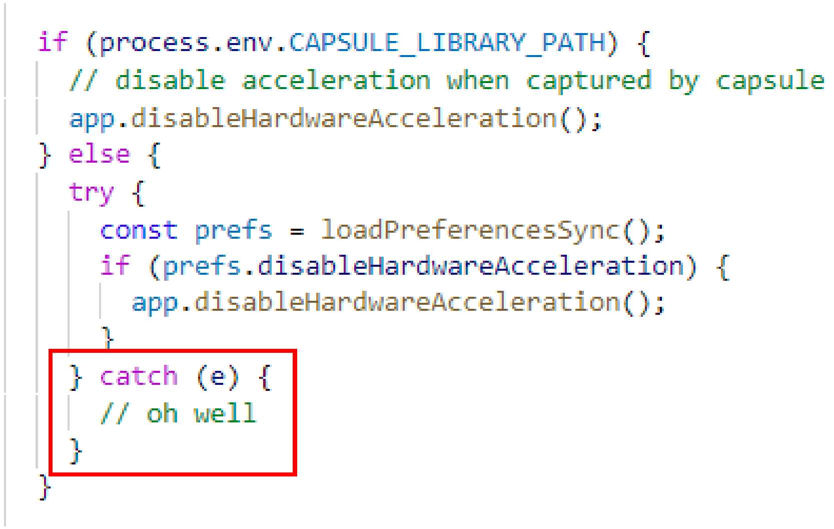
- **assertThrows()** method asserts a method which **throws an exception.**
- If **no exception is thrown** from the executable block then assertThrows() will **FAIL.**
- If an **exception of a different type** is thrown, assertThrows() will **FAIL.**

```
@Test
void testRectangleException() {
   //First argument - specifies the expected exception.
   //Here it expects that code block will throw NumberFormatException
   //Second argument - is used to pass an executable code block
   Assertions.assertThrows(NumberFormatException.class,new Executable(){
      @Override
      public void execute() throws Throwable{
          Integer.parseInt("One");
      }
   });
}
```

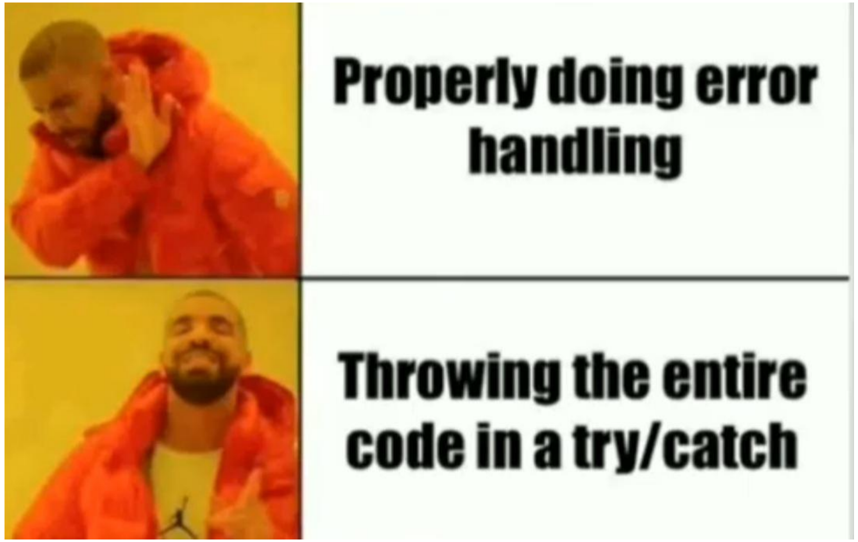
Activity 2.3 - Test Rectangle Exception

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;
import org.junit.jupiter.api.Assertions;
@Test
void testRectangleException() {
    Assertions.assertThrows(ArithmeticException.class,new Executable(){
        @Override
        public void execute() throws Throwable{
            new Rectangle(0,-3);
        }
    });
}
```

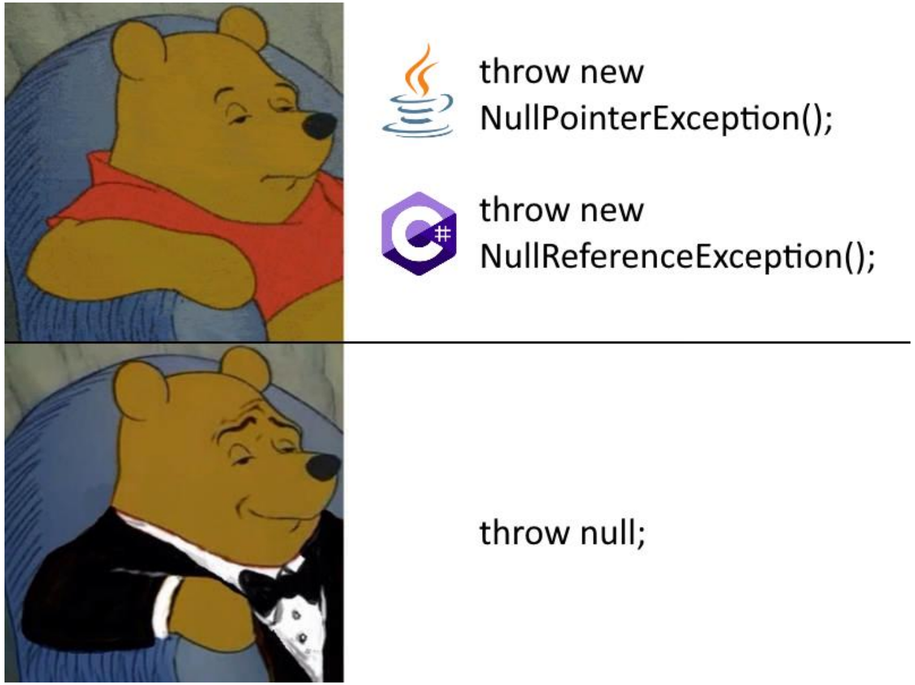
Don’t do this



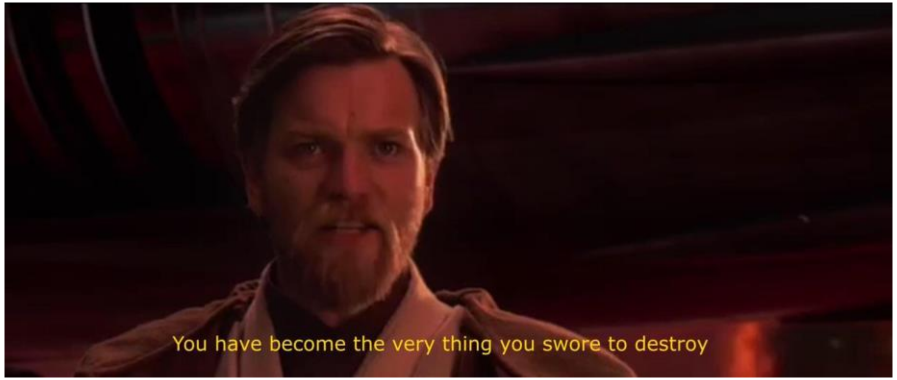
```
if (process.env.CAPSULE_LIBRARY_PATH) {
  // disable acceleration when captured by capsule
  app.disableHardwareAcceleration();
} else {
  try {
    const prefs = loadPreferencesSync();
    if (prefs.disableHardwareAcceleration) {
      app.disableHardwareAcceleration();
    }
  } catch (e) {
    // oh well
  }
}
```


Properly doing error handling
Throwing the entire code in a try/catch

Don’t do this



When the try catch for error handling becomes another error itself.



QCalc - Module 5: Extend Simple Calculator

- In this module:
  - Implement ScientificCalculator class by extending the StandardCalculator class
  - Use Math Java API library for advanced calculations
  - Override a method
  - Prevent method overriding by using final

Common Gradle Tasks

- Compile all the classes
  - $ ./gradlew build
- Running applications
  - $ ./gradlew run
- Running all checks
  - $ ./gradlew check
- Cleanup compiled classes
  - $./ gradlew clean
- $ Running all tests
  - $./gradlew test

Inheritance - Recap

extends

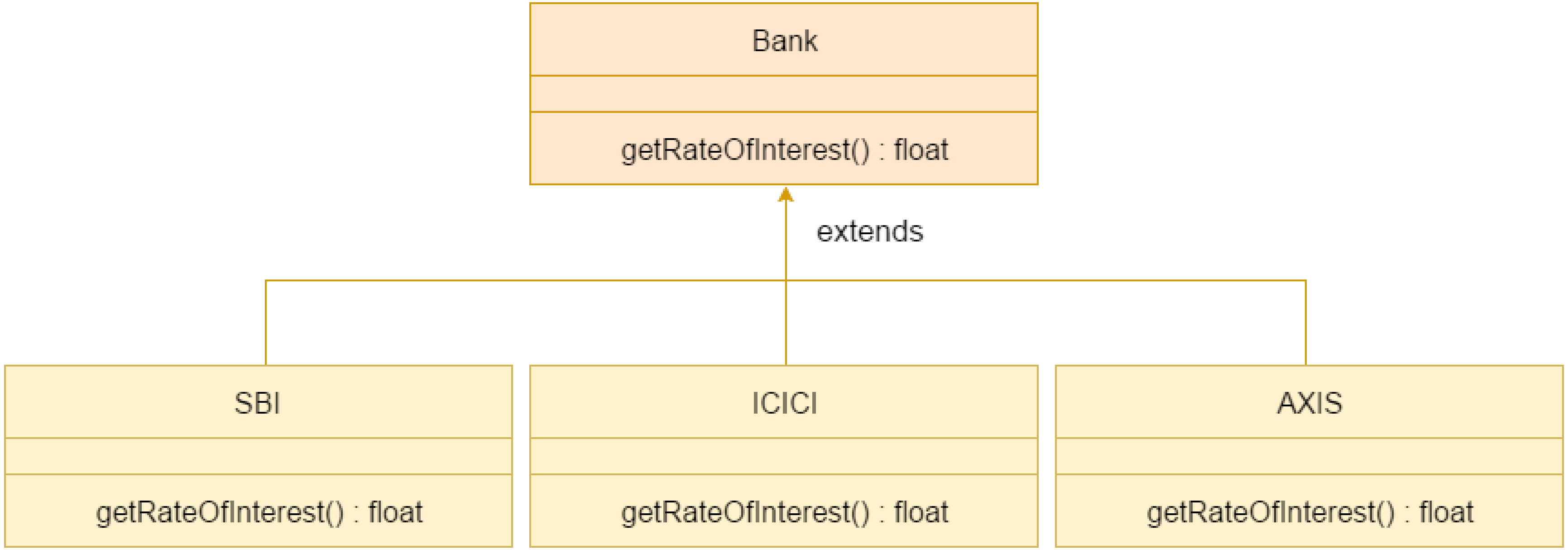
- Used to inherit the properties of a class.

```
class Super {
   .....
   .....
}
class Sub extends Super {
   .....
   .....
  }
```

- **Super**
  - Similar to **this** keyword.
  - Used to:
    - Differentiate the members of superclass from the members of subclass, if they have same names.
    - Invoke the superclass constructor from subclass.

```
super.variable

super.method();

super(values);
```

Method Overriding - Recap


Bank
getRateOfInterest() : float
extends
SBI
getRateOfInterest() : float
ICICI
getRateOfInterest() : float
AXIS
getRateOfInterest() : float

Take home exercises for the session

- You will have to complete the below modules of QCalc Micro-Experience**:**
  - Module 4 - Debug and Handle Exception
  - Module 5 - Extend Simple Calculator
- Try to finish this before the next session on Saturday, 11:00 am.

Feedback

Thank you for joining in today.

We’d love to hear your thoughts and feedback - Feedback for JAVA-2 Session

References

- How to Handle Exceptions in Java - Rollbar
- How to use the Throws keyword in Java (and when to use Throw) - Rollbar
- Types of Exception in Java with Examples - GeeksforGeeks

Thank you