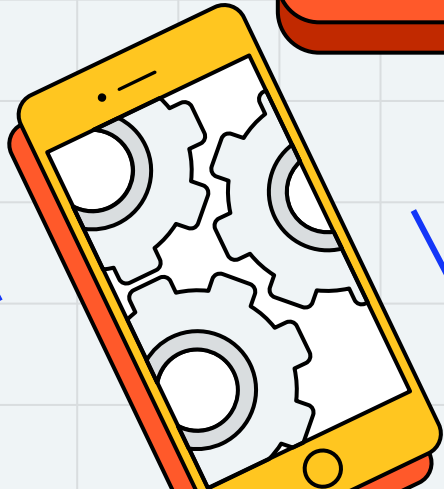


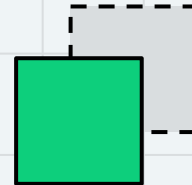


Application

Programming



Hend Alkittawi





Exception Handling

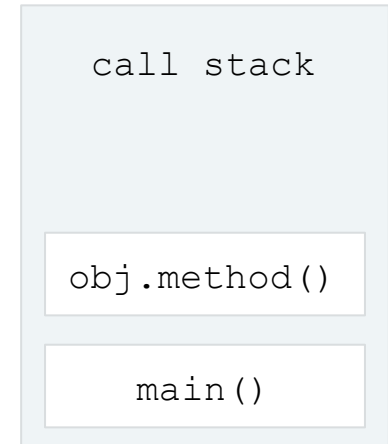
Introduction To Java Errors And
Exceptions

INTRODUCTION

- In Java when things go wrong a `java.lang.Exception` object is created.
- For example,
 - if we add elements to an uninitialized arraylist
 - **NullPointerException**
 - if we try to read from a file that doesn't exist
 - **FileNotFoundException**
 - if we try to read past the end of the file
 - **IOException**
 - if the file changes while we are reading it
 - **IOException**

THE CALL STACK

- When a Java program runs, execution begins in the `main()` method. The `main()` method creates objects and invokes methods on them.
- When execution moves to another method an entry is added to the **call stack**.
- When a method finishes executing, the entry is removed from the call stack, and execution returns to the next line in the `main()` method
 - this continues until the main method finishes



THE CALL STACK

- The call stack entry below, among other things, contains
 - the current method
 - where the call occurred in that method

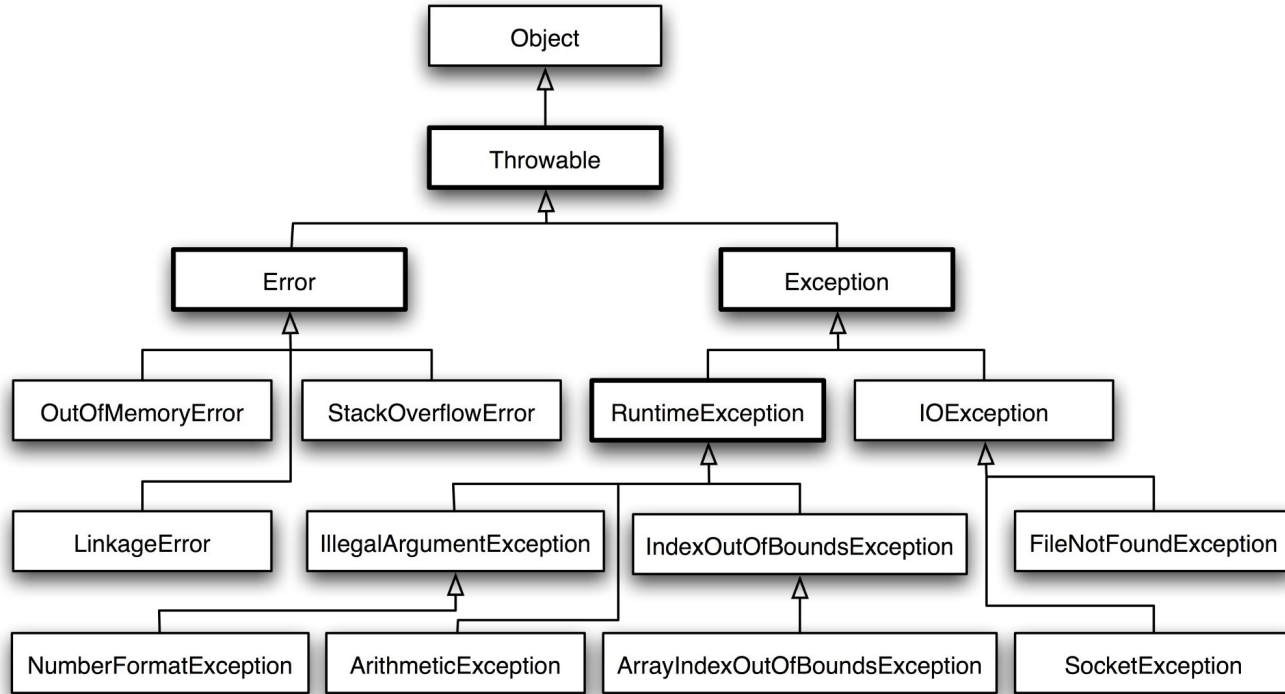
NullPointerException:

```
at Student.getAverage(Student.java:79)
at Student.toString(Student.java:62)
at java.lang.String.valueOf(String.java:2615)
at java.io.PrintStream.print(PrintStream.java:616)
at java.io.PrintStream.println(PrintStream.java:753)
at Student.main(Student.java:120)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:585)
```

EXCEPTIONS IN JAVA

- In Java, all exception classes inherit from the Exception class
- Exceptions in Java are **checked** or **unchecked**!
- Checked exceptions must be caught or thrown. Examples of checked exceptions include:
 - IOException
 - FileNotFoundException
- Unchecked exceptions should never be caught or thrown. Examples of unchecked exceptions include:
 - NullPointerException
 - ArrayIndexOutOfBoundsException

EXCEPTIONS IN JAVA



EXCEPTION HANDLING

- As developers, we must address any problems that might occur.
- For **unchecked** exceptions, your code should follow best practices in order to prevent exceptions occurrences. For example
 - check for array bounds
 - check for null values
- For **checked** exceptions, your code either **throws** the exception or **handles** the exception with a try/catch block.

EXCEPTION HANDLING

- Using try, catch, and finally blocks
 - Wrap all code that can cause a checked exception in try, catch (and optionally finally) blocks

```
try {  
    System.out.println("code here can cause  
                        an exception");  
} catch (Exception e) {  
    System.out.println("handle exception here");  
} finally { // optional  
    System.out.println("code that must be absolutely  
                        executed after try block completes");  
}
```

```
try {  
    System.out.println("reading from a file ...");  
} catch (FileNotFoundException e) {  
    System.out.println("handle exception here");  
} finally { // optional  
    System.out.println("closing the file ...");  
}
```

EXCEPTION HANDLING

- A try block can have multiple catch blocks.
- The order of the catch blocks is important.

```
try {  
    System.out.println("reading from a file ...");  
} catch (FileNotFoundException e) {  
    System.out.println("code here will execute  
    when a FileNotFoundException is thrown!");  
} catch (IOException e) {  
    System.out.println("handle exception here");  
} finally {  
    System.out.println("closing the file ...");  
}
```

```
try {  
    System.out.println("reading from a file ...");  
} catch (IOException e) {  
    System.out.println("code here will execute  
    when a FileNotFoundException is thrown!");  
} catch (FileNotFoundException e) {  
    System.out.println("code here will not execute  
    when a FileNotFoundException is thrown!");  
} finally {  
    System.out.println("closing the file ...");  
}
```

THROWING EXCEPTIONS

- An exception might be thrown, when there is nothing more you can do about it!

```
public void methodA() {  
    try {  
        dangerZone();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
  
public void dangerZone() throws Exception {  
    throw new Exception();  
}  
}
```

EXCEPTION HANDLING EXAMPLE

```
import java.io.FileNotFoundException;
public class DemoExceptions {

    public static void main(String[] args) {

        try {
            method(true);
            System.out.println("returned from method()");
        } catch (FileNotFoundException e) {
            System.out.println("caught the exception, will handle it!");
            e.printStackTrace();
        } finally {
            // code that must be absolutely executed after try block completes
            System.out.println("finally will cleanup!");
        }
    }

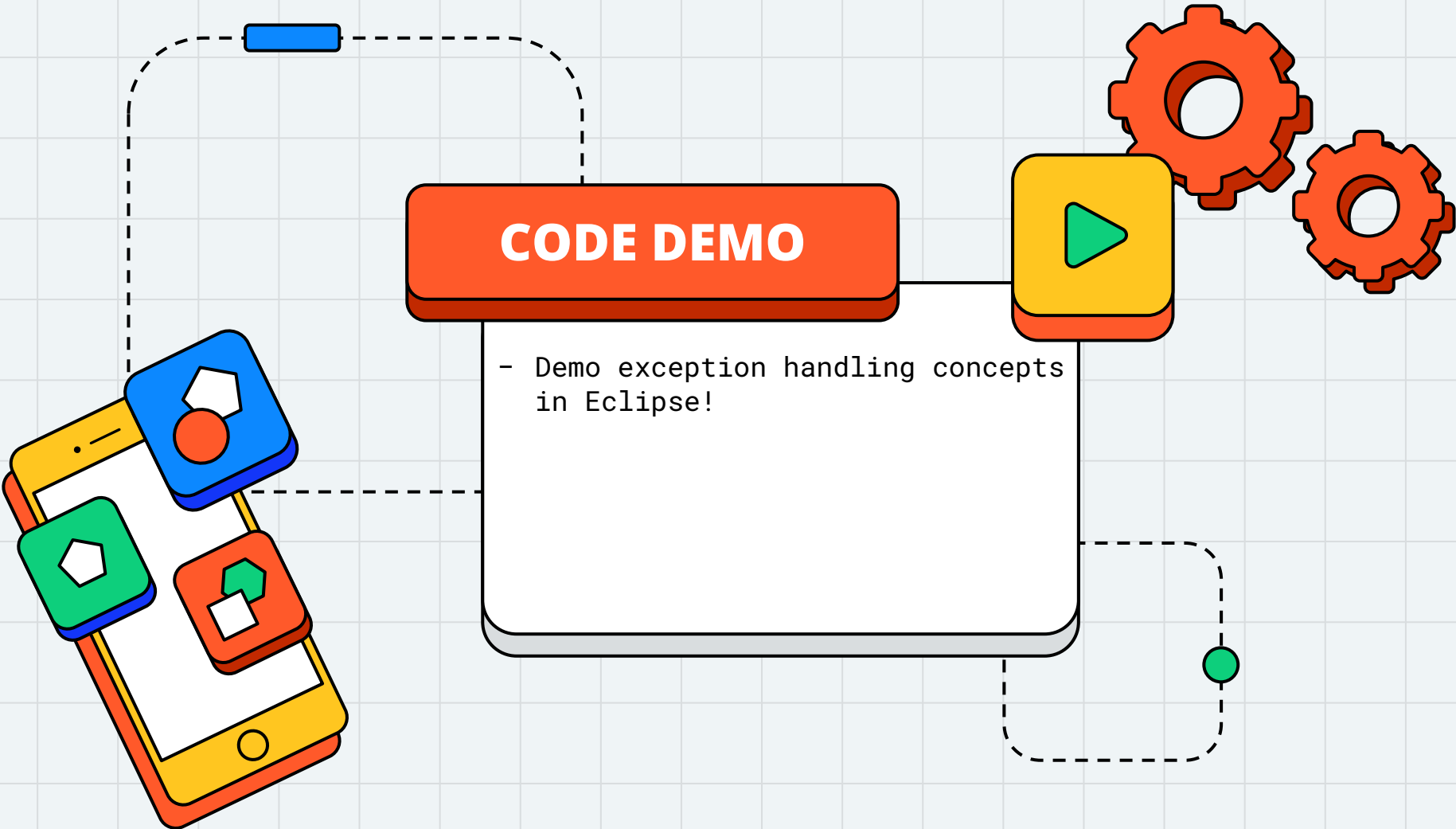
    public static void method(boolean exception) throws FileNotFoundException {
        if(exception)
            throw new FileNotFoundException();
        System.out.println("method 1 executed successfully!");
    }
}
```

HANDLING EXCEPTIONS

- Handling exceptions improves the user experience!
- Consider
 - Where can errors happen caused by our logic?
 - Where can exceptions happen?
 - Where can user error occur?
- For each, how can we prevent or reduce these?
 - What would the user expect?

USER EXPERIENCE!

- Suppose you are exploring with Google earth (`desktop app`), you click a button and it
 - Closes and/or reopens the program/window
 - Changes the size of the window
 - Moves GUI components around the view
 - Does nothing!
- Suppose you are searching Google (`web app`), you enter some text, click the button and
 - The web page refreshes, losing your search text
 - The result page comes up, without results
 - Nothing happens!
- Suppose you are shopping on Amazon (`mobile app`), you tap a button and
 - The app closes (and maybe reopens)
 - The entire style of the app changes
 - GUI components move around the view (unexpectedly)
 - Nothing happens!



CODE DEMO

- Demo exception handling concepts in Eclipse!



THANK

YOU!



DO YOU HAVE ANY QUESTIONS?



hend.alkittawi@utsa.edu



By Appointment



Online