CP213 Review Session 5

EXCEPTION HANDLING, INTERFACES AND INNER CLASSES

Please take this moment to sign in...

UPCOMING EVENTS...

Next Review Session:

Wednesday, November 29 (final exam review)

Useful Links and Follow Along Resources

MATH AND STATS LEARNING SUPPORT

This office hosts drop in and directed homework sessions to give you any extra help you need with course content in most lower level CS courses. You can also get their course widget in MyLS.

CODING SANDBOXES

A really good Java code sandbox is Replit.com.
Follow along or write code on your own time without setting up files by using one of these.

A really good Python, Java, HTML, sandbox (and almost every other language) is Replit.com
https://replit.com/

Another good HTML/CSS/JS sandbox:
https://codepen.io/

Mental Health Resources

We know university is very challenging and difficult. Please don't hesitate to reach out to people if you need help. Listed below are a few useful resources you can access, for additional resources please view the LCS linktree.

SAFEHAWK APP

Connects you with telephone helplines and other mental health resources.

STUDENT WELLNESS CENTRE

Provides comprehensive physical, emotional and mental health services for Waterloo and Brantford Campus students.

DELTON GLEBE COUNSELLING CENTRE:

A holistic counselling facility.

What is Exception Handling?

- EXCEPTION HANDLING IS A WAY TO DEAL WITH UNEXPECTED PROBLEMS THAT MIGHT HAPPEN WHILE THE PROGRAM IS RUNNING
- IT'S LIKE SAYING, "IF SOMETHING GOES WRONG, DO THIS INSTEAD OF JUST CRASHING."

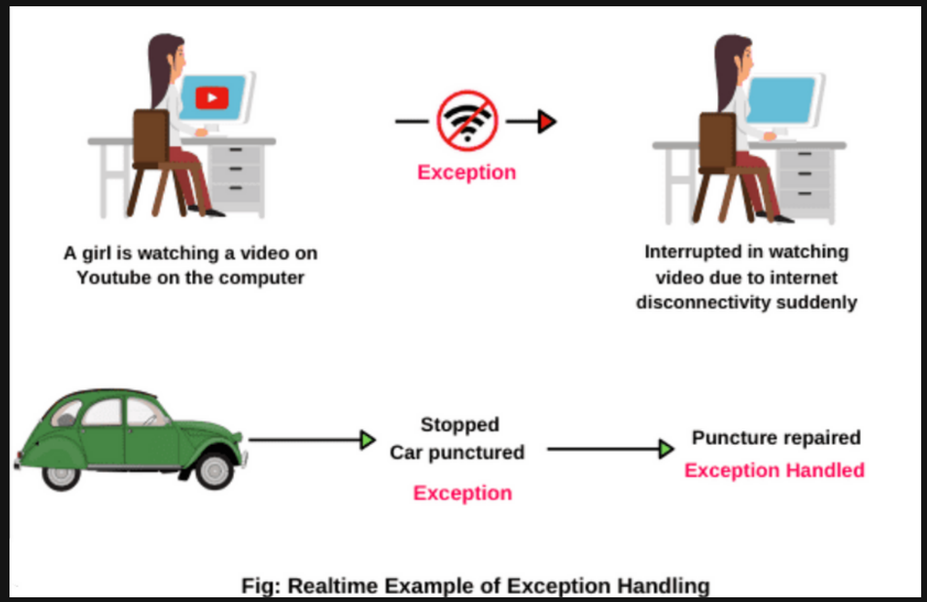


Fig: Realtime Example of Exception Handling

- IT HELPS MAKE PROGRAMS MORE ROBUST AND PREVENTS THEM FROM UNEXPECTEDLY STOPPING.

Try-Throw-Catch Mechanism

- THE BASIC WAY OF HANDLING EXCEPTIONS IN JAVA CONSISTS OF THE TRY-THROW-CATCH TRIO
    - THE "**TRY**" BLOCK IS LIKE A PROTECTIVE SHIELD AROUND A PIECE OF CODE WHERE SOMETHING UNEXPECTED MIGHT HAPPEN. IT'S WHERE WE SAY, "TRY TO DO THIS WITHOUT ANY ERRORS."
    - EX:
        - TRY {
          // CODE THAT MIGHT CAUSE AN EXCEPTION
          INT RESULT = 10 / 0; // THIS WILL CAUSE AN ARITHMETICEXCEPTION
          }

Try-Throw-Catch Mechanism

- THE "**CATCH**" BLOCK IS LIKE A SAFETY NET THAT CATCHES AND HANDLES THE ERRORS THAT OCCUR IN THE "TRY" BLOCK. IT'S WHERE WE SPECIFY WHAT TO DO IF SOMETHING GOES WRONG.
- EX:

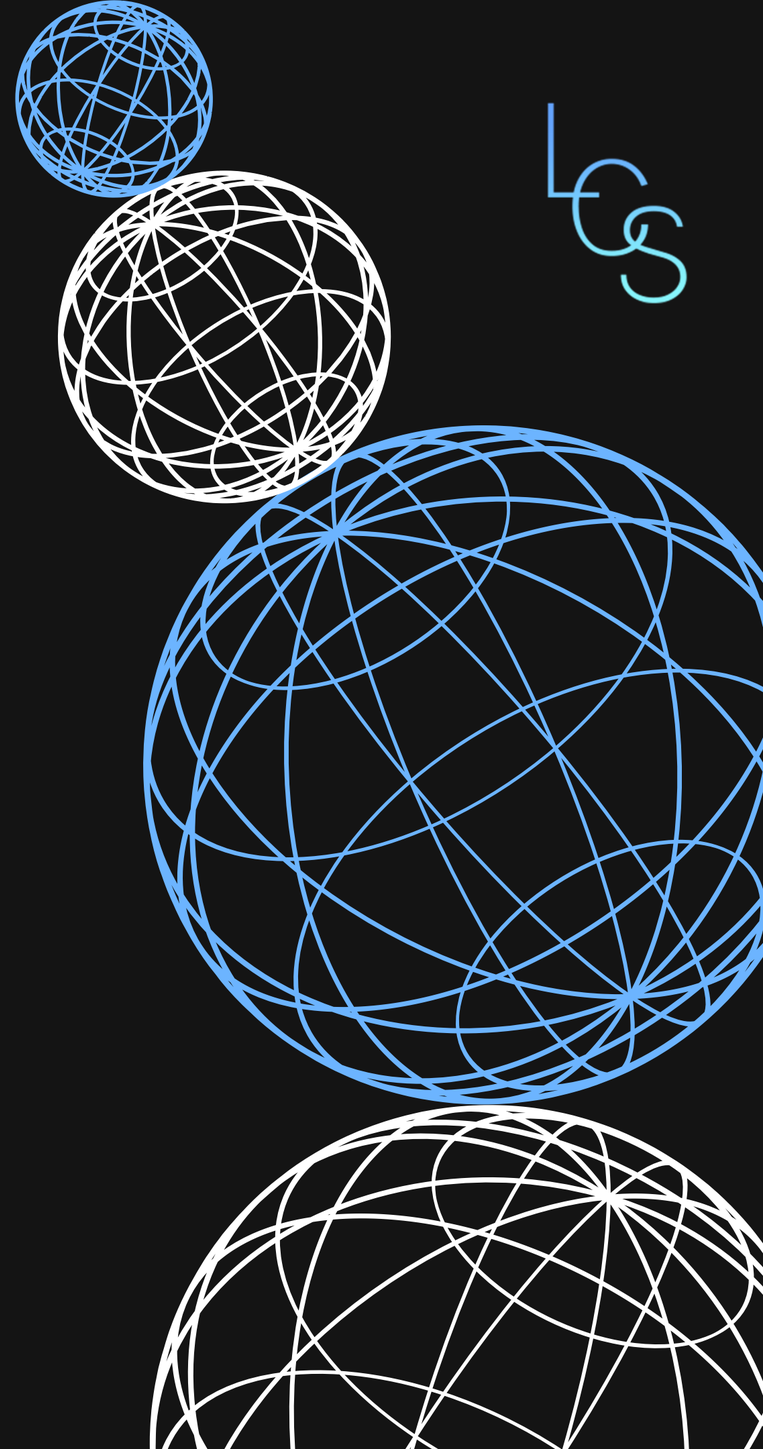
```
Try {
 int result = 10 / 0;
}
Catch (ArithmeticException e) {
 // Handle the exception
 System.out.println("Error: Cannot divide by zero!");
}
```

Try-Throw-Catch Mechanism

- SOMETIMES, WE WANT TO TELL THE PROGRAM THAT SOMETHING UNEXPECTED HAS HAPPENED. WE "THROW" AN EXCEPTION TO SIGNAL THIS, AND IT WILL BE CAUGHT AND HANDLED ELSEWHERE IN THE CODE.
- EX:

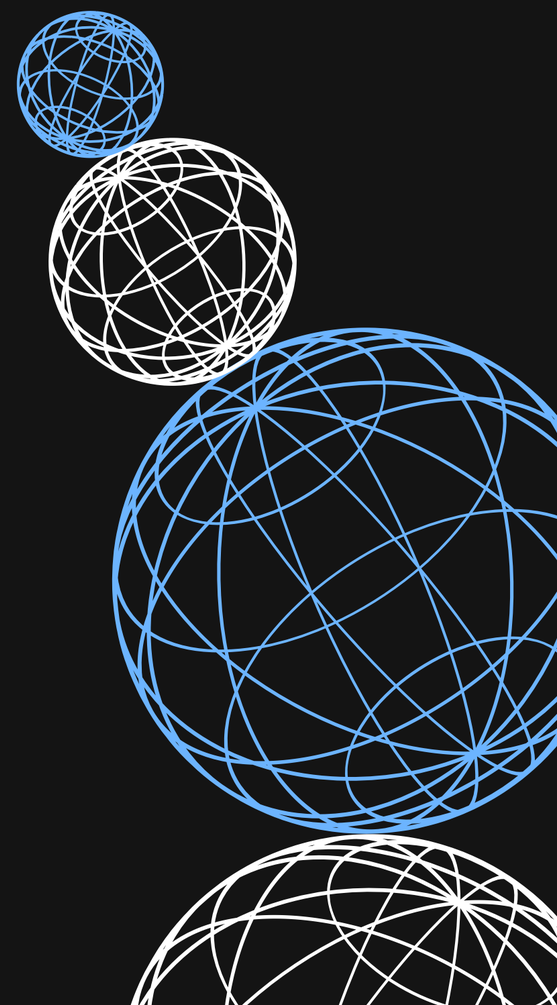
```
void checkTemperature(int temperature) throws
IllegalArgumentException {
    if (temperature > 100) {
      throw new IllegalArgumentException("Too hot!");
      }
}
```

Do you have any questions?



Try-Throw-Catch Example

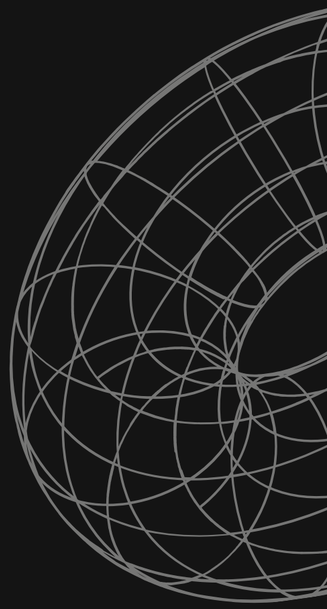
```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            // Try block: Attempting a division that may cause an exception
            int result = divide(10, 0);
            System.out.println("Result: " + result);  // This line won't be reached if an
exception occurs
        } catch (ArithmeticException e) {
            // Catch block: Handling the specific exception
            System.out.println("Error: Division by zero is not allowed.");
        }
    }

    private static int divide(int numerator, int denominator) {
        if (denominator == 0) {
            // Throw block: Throwing an exception if the denominator is zero
            throw new ArithmeticException("Cannot divide by zero");
        }
        return numerator / denominator;
    }
}
```



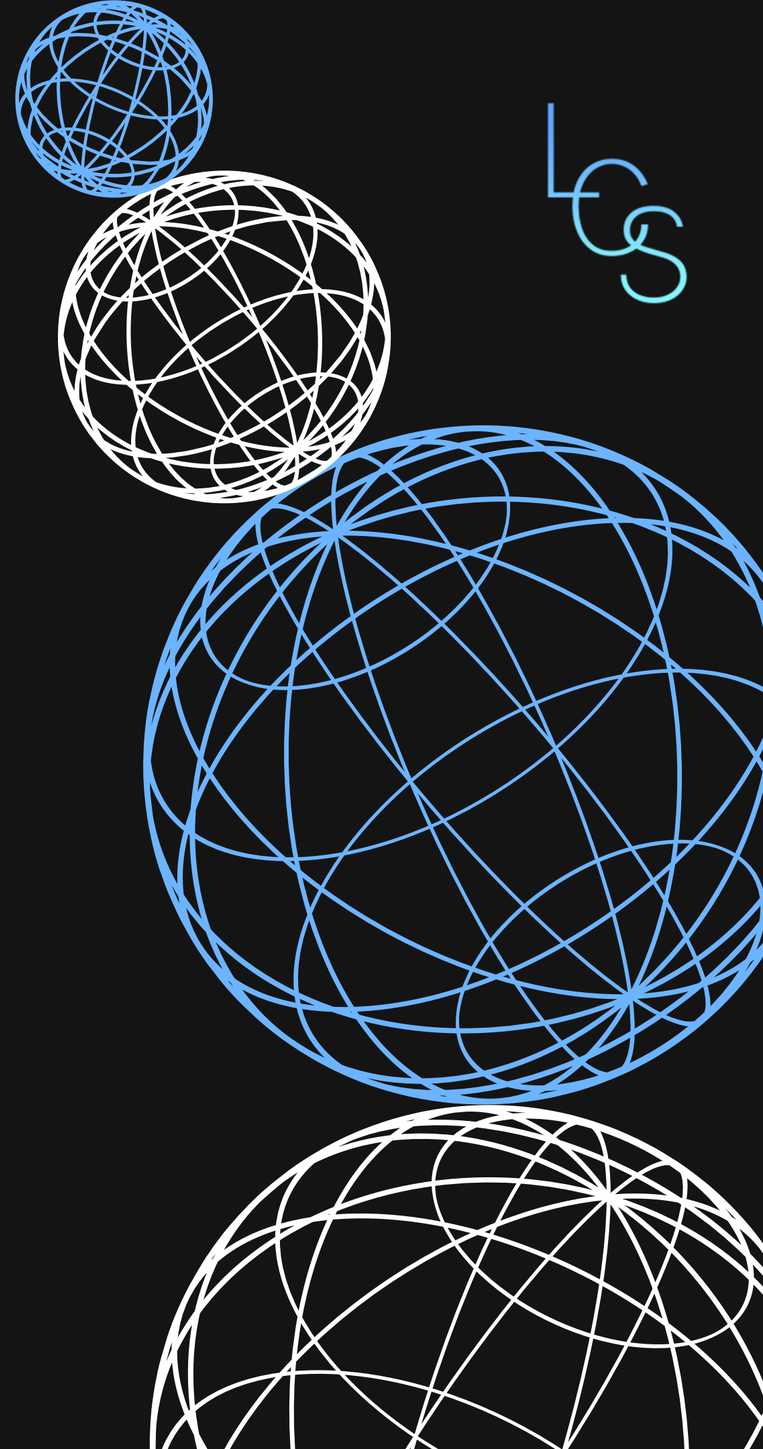
Finally Block

- In Java, a finally block is a piece of code that you can add after a try and catch block. It contains code that will always be executed, regardless of whether an exception occurred or not.
- Ex:

```
public class FinallyExample {
    public static void main(String[] args) {
        try {
            // Risky operation
            System.out.println("Trying something risky...");
            int result = 10 / 0;  // This will cause an exception
            System.out.println("Result: " + result);  // This won't be executed
        } catch (ArithmeticException e) {
            // Catching the exception
            System.out.println("Error: Division by zero is not allowed.");
        } finally {
            // Code in this block will always be executed
            System.out.println("This code always runs, no matter what!");
        }
    }
}
```



Do you have any questions?



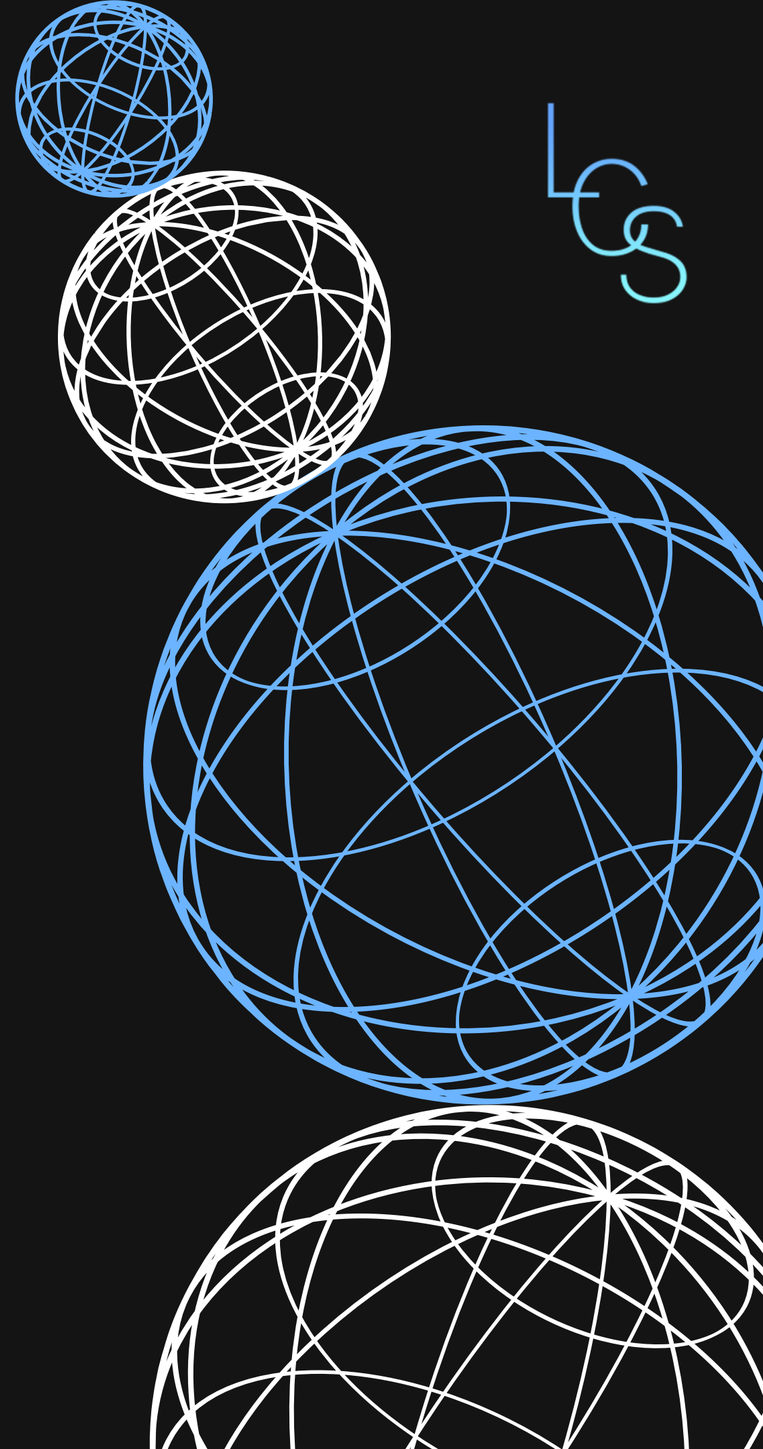
Interfaces

- Interfaces are like templates of classes which you can implement in your own class definitions
  - This helps save the programmer time and provides an oppurtunity for code reusability
- Interfaces just need to have their methods defined with a name and return type, the actual contents of the method code can be written later
  - This is why interfaces are a very helpful blueprint

Interfaces- Example

```
// Define an interface
interface Shape {
    double calculateArea(); // Method signature without implementation
    void display(); // Another method signature
}

// Implementing the interface
class Circle implements Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    @Override
    public void display() {
        System.out.println("This is a circle with radius " + radius);
    }
}
```

Do you have any questions?



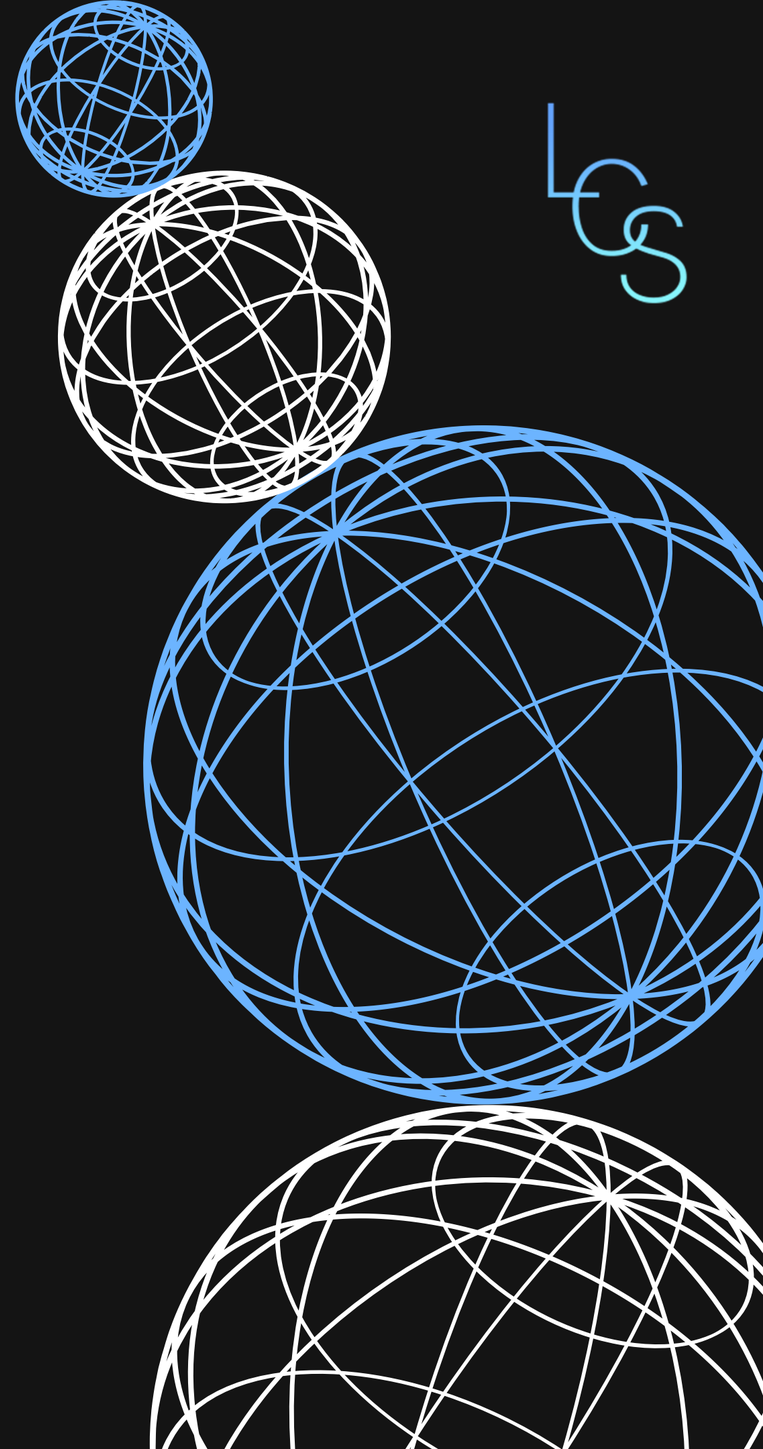
Inner Classes

- Inner classes are classes defined within other classes
- The class that includes the inner class is called the outer class
- There is no particular location where the definition of the inner class (or classes) must be place within the outer class
    - Placing it first or last, however, will guarantee that it is easy to find

Inner Classes - Example

```
class Outer {
    private int outerVar = 5; // Example value

    // Member Inner Class
    class InnerMath {
        int square() {
            return outerVar * outerVar;
        }

        double divideByTwo() {
            return outerVar / 2.0; // Division resulting in a double
        }
    }
}
```

Do you have any questions?



Please fill out this event survey!

Thank you for coming!

Laurier
Computing
Society

@laurier.cs

discord.lauriercs.ca

linktr.ee/lauriercs