CP213 Final Review Session

Please take this moment to sign in...

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:

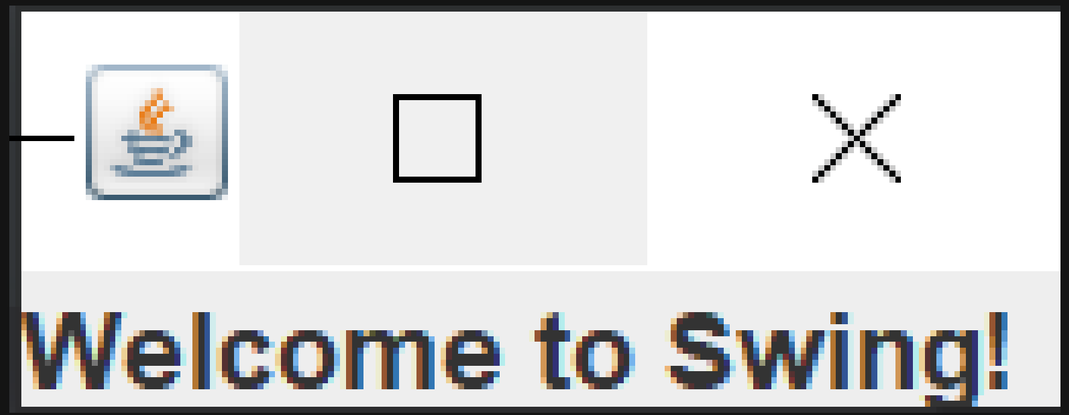
A holistic counselling facility.

Swing GUI Programming

- Swing is an event-driven GUI programming library for the Java programming language
  - Made popular by allowing developers to create software with accessible user interfaces
  - **Event-driven programming:** Programming in which functions are defined and then called based on specific event occurances during the program running
  - **GUI programming:** GUI (graphical user interface) programming is developing software with a graphical user interface
    - An example of a software with a GUI is iOS, and an example of software without a GUI is the command line

Swing Introduction

```java
import javax.swing.JFrame;
import javax.swing.JLabel;

public class SwingSession {
    public static void main(String[] args) {
        // Creating a JFrame with a title
        JFrame frame = new JFrame("Hello, Swing!");
        // Exit the application when the frame is closed
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Adding a JLabel to the frame
        JLabel label = new JLabel("Welcome to Swing!");
        frame.getContentPane().add(label);

        // Packing components and making the frame visible
        frame.pack();
        frame.setVisible(true);
    }
}
```

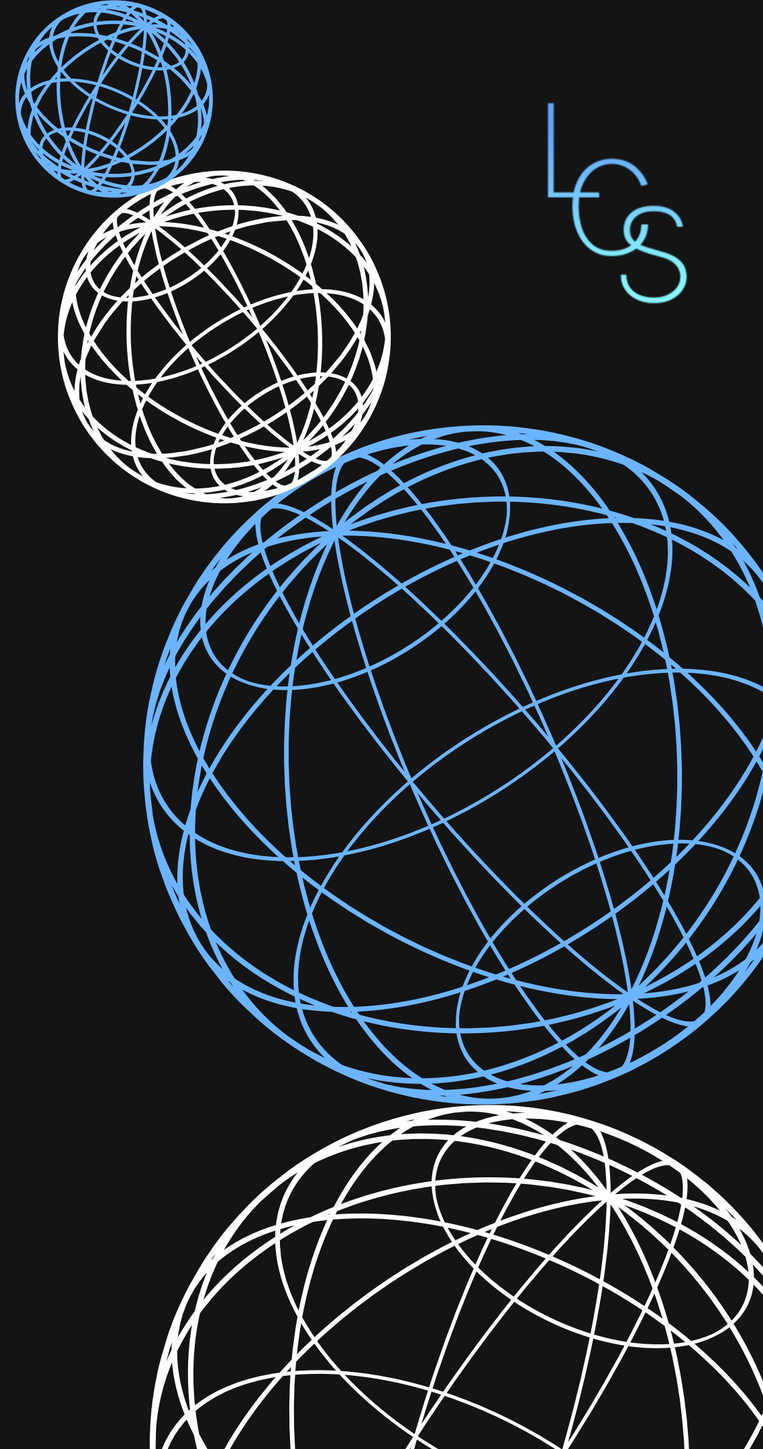


Question:

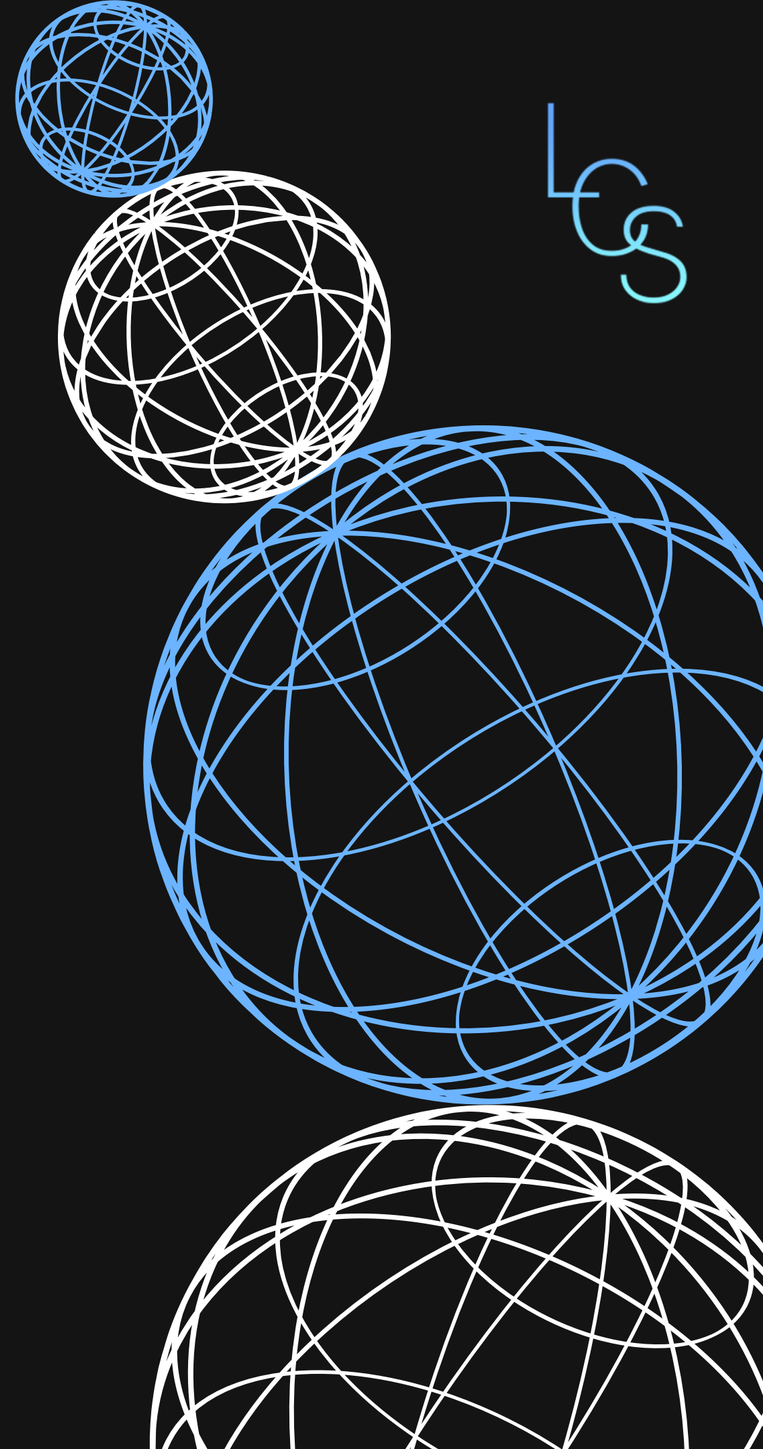
What‘s missing in above screenshot of code?

Answer: the package!

Should say something like:
“package cp213;”

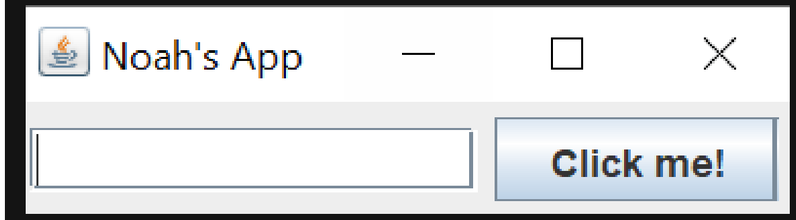


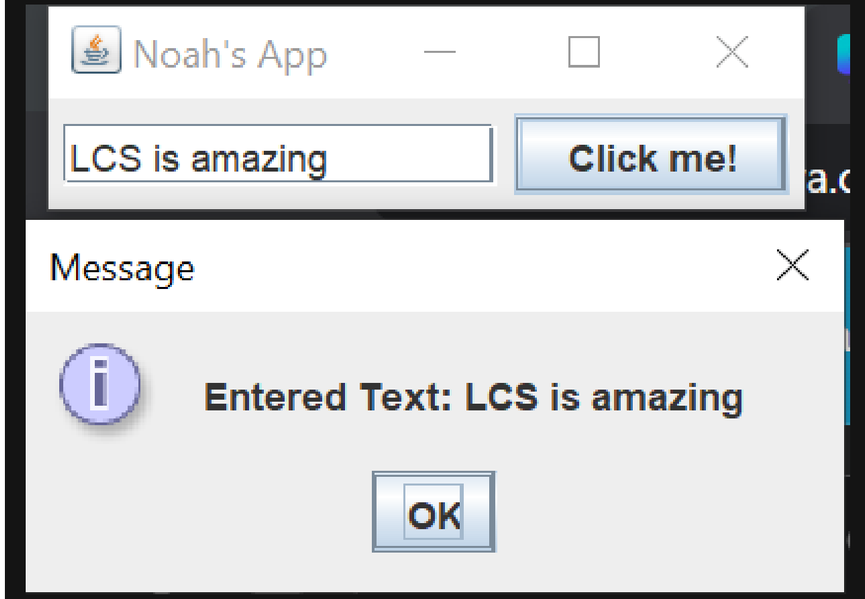
Do you have any questions?



Swing: More Complex Example

```java
package cp213;

import java.awt.FlowLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

public class SwingSession {
    public static void main(String[] args) {
        // Create a JFrame with a title
        JFrame frame = new JFrame("Noah's App");
        // Exit the application when the frame is closed
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Create a JTextField with 15 columns
        JTextField textField = new JTextField(15);
        // Create a JButton with the label "Click me!"
        JButton button = new JButton("Click me!");
        // Add an ActionListener to the button
        button.addActionListener(e -> { // 'e' represents an event object which is passed as a parameter to the
                                        // anonymous function defined in the call of button.addActionListener
            // this anonymous function is called upon the triggering of the event, which
            // would cause the event object to be passed as parameter
            // to the anonymous function defined here
            // Get the text from the JTextField
            String enteredText = textField.getText();
            // Display the entered text in a JOptionPane
            JOptionPane.showMessageDialog(frame, "Entered Text: " + enteredText);
        });
        // Set the layout manager of the frame to FlowLayout
        frame.setLayout(new FlowLayout());
        // Add the JTextField and JButton to the frame
        frame.add(textField);
        frame.add(button);
        // Pack components and make the frame visible
        frame.pack();
        frame.setVisible(true);
    }
}
```

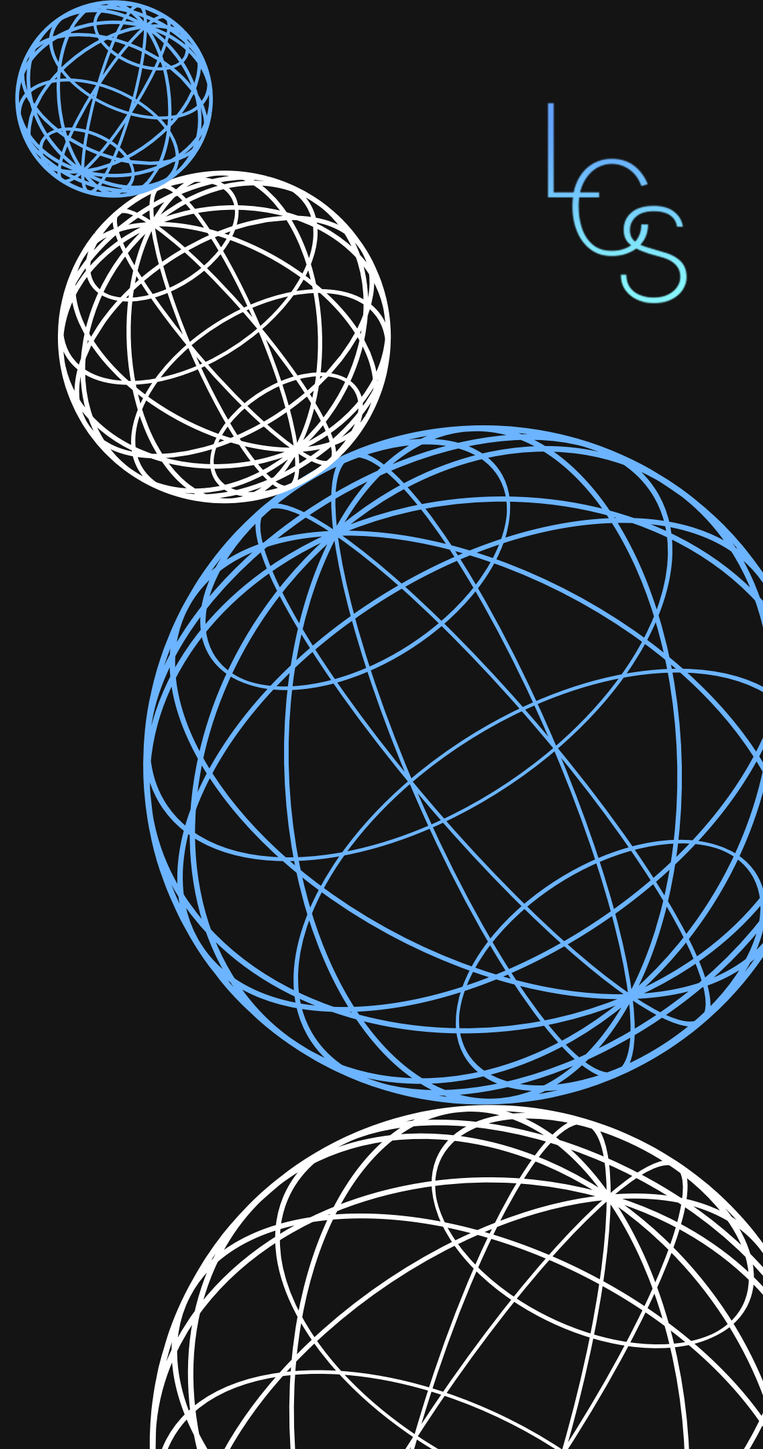




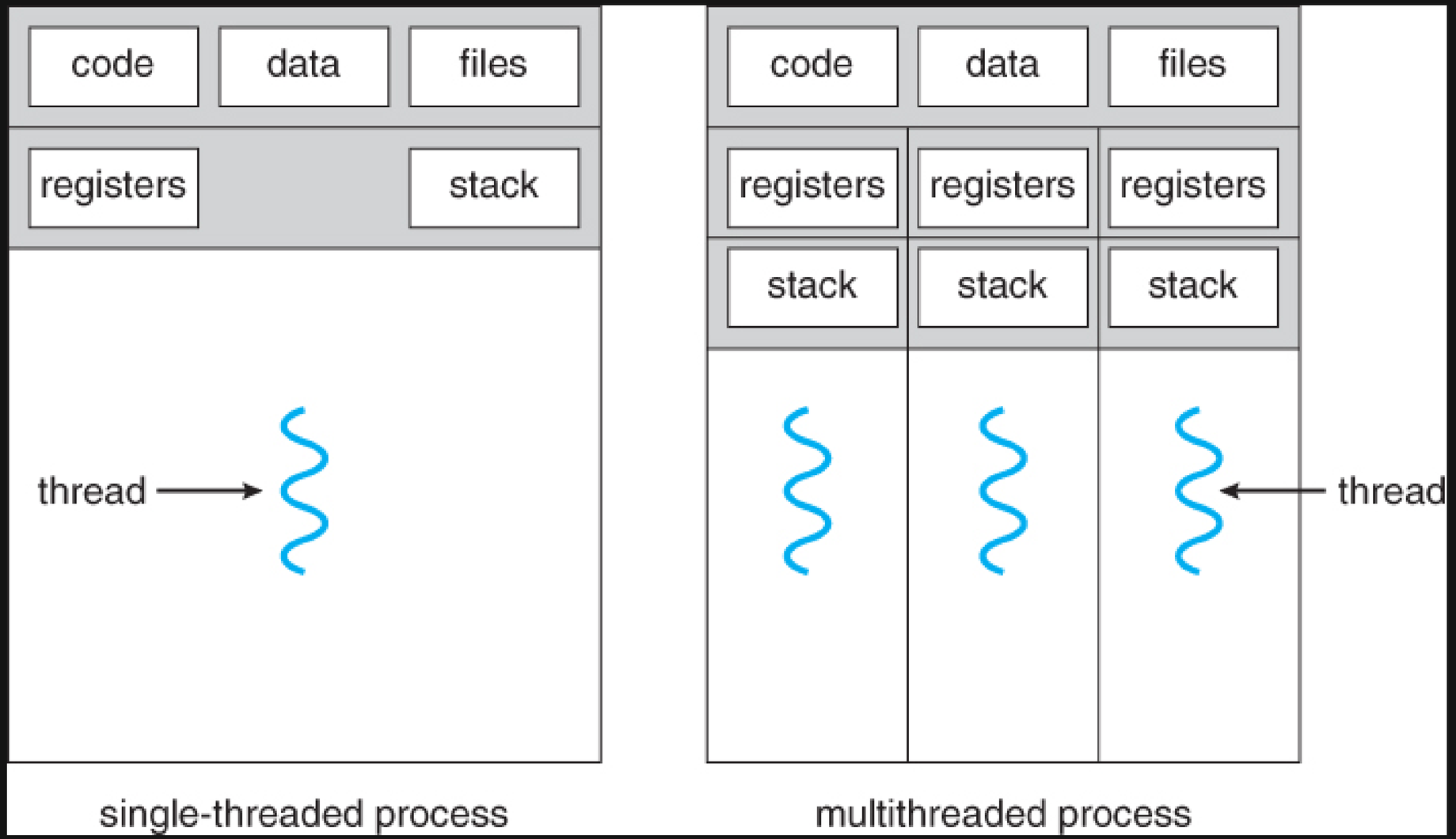
MVC Design Framework

- Explanation of MVC (Model-View-Controller) Architecture:
  - Model: Represents the application's data and business logic.
  - View: Displays the data and interacts with the user.
  - Controller: Handles user input and updates the Model and View accordingly.
- Swing uses the MVC design pattern to separate concerns and make the code modular and maintainable.
  - In Swing, model includes the classes that manage the application's data, it is not concerned with user interface.
  - In Swing, the View is composed of the UI components (such as JFrame, JPanel, JButton, etc.) that users interact with.
  - The Controller in a Swing application includes event listeners and handlers. These components respond to user actions and initiate the appropriate actions in the Model or update the View.

Do you have any questions?

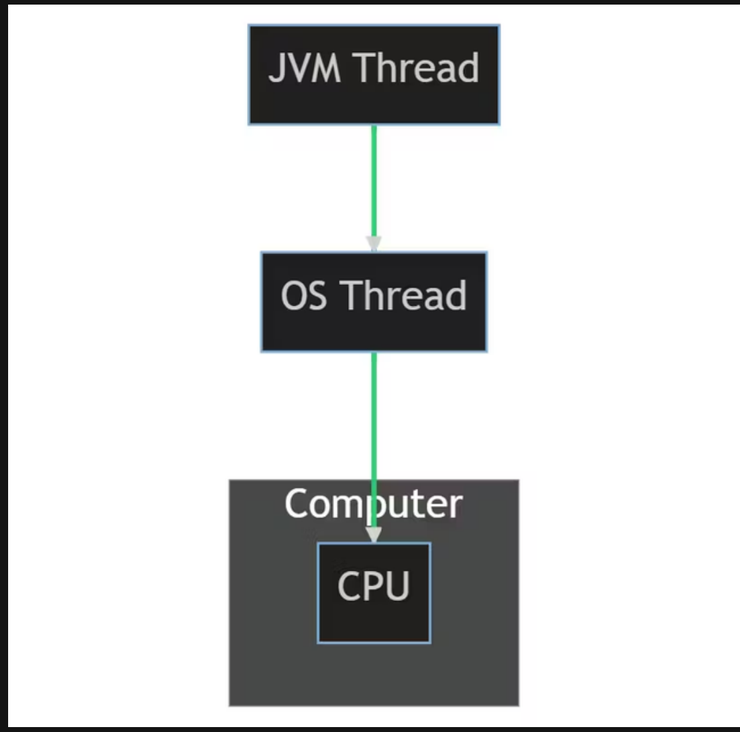


Java for Multithreading

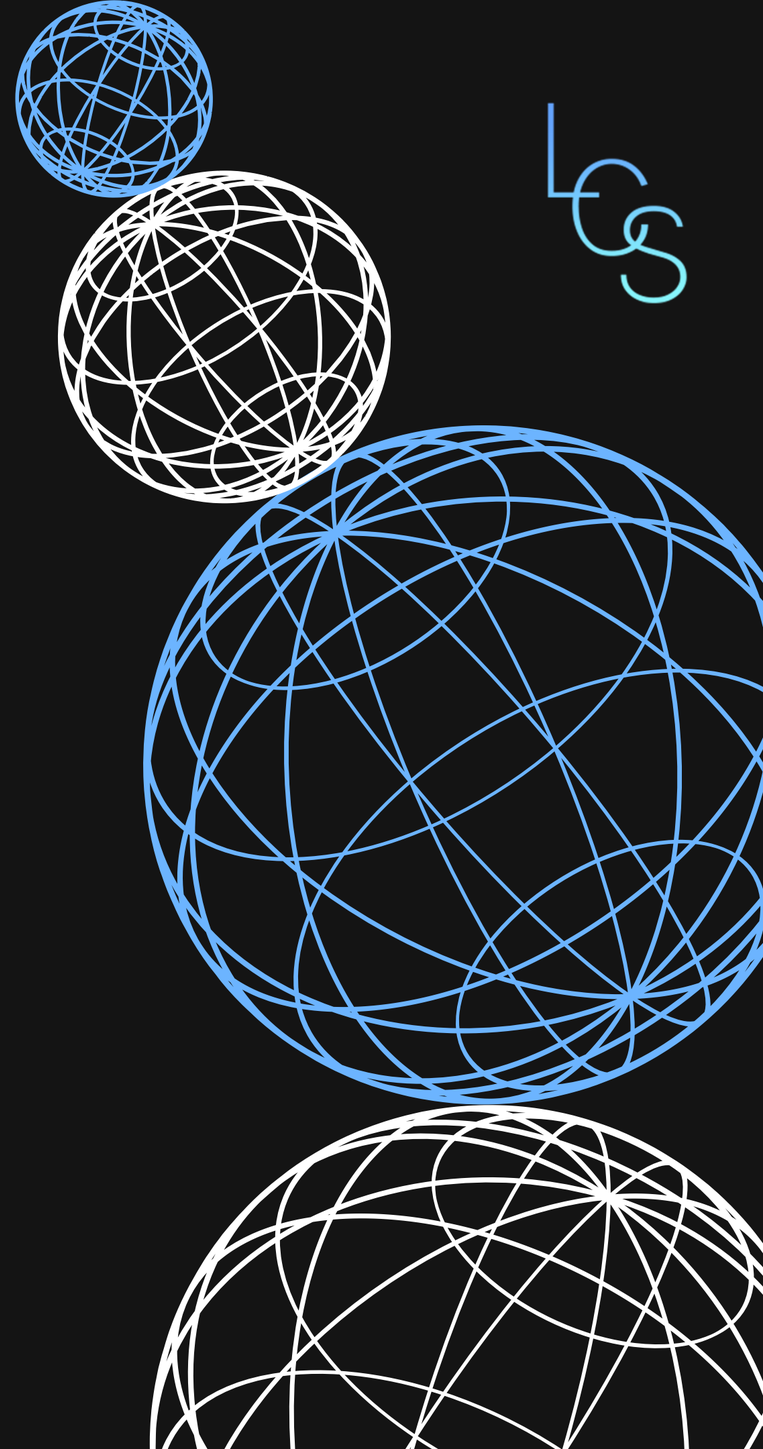


Java for Multithreading

- The Java Thread class is a fundamental component for implementing multithreading in Java applications.
- It provides a way to create and control threads.
- Instances of the Thread class encapsulate the code and resources associated with a separate execution flow.
- The Thread class also offers methods for managing thread execution, such as controlling thread priority and putting threads to sleep for a specified duration.



Do you have any questions?



Thread.sleep() method

- In Java, the Thread.sleep() method is used to temporarily pause the execution of a thread for a specified amount of time.
  - It takes a parameter representing the duration of the pause in milliseconds.
  - This method is commonly employed to introduce delays or control the timing of operations in multithreaded programs, allowing developers to synchronize and coordinate the execution of threads.
  - It is important to note that using Thread.sleep() can impact the responsiveness of the application and should be used carefully to avoid potential performance issues.

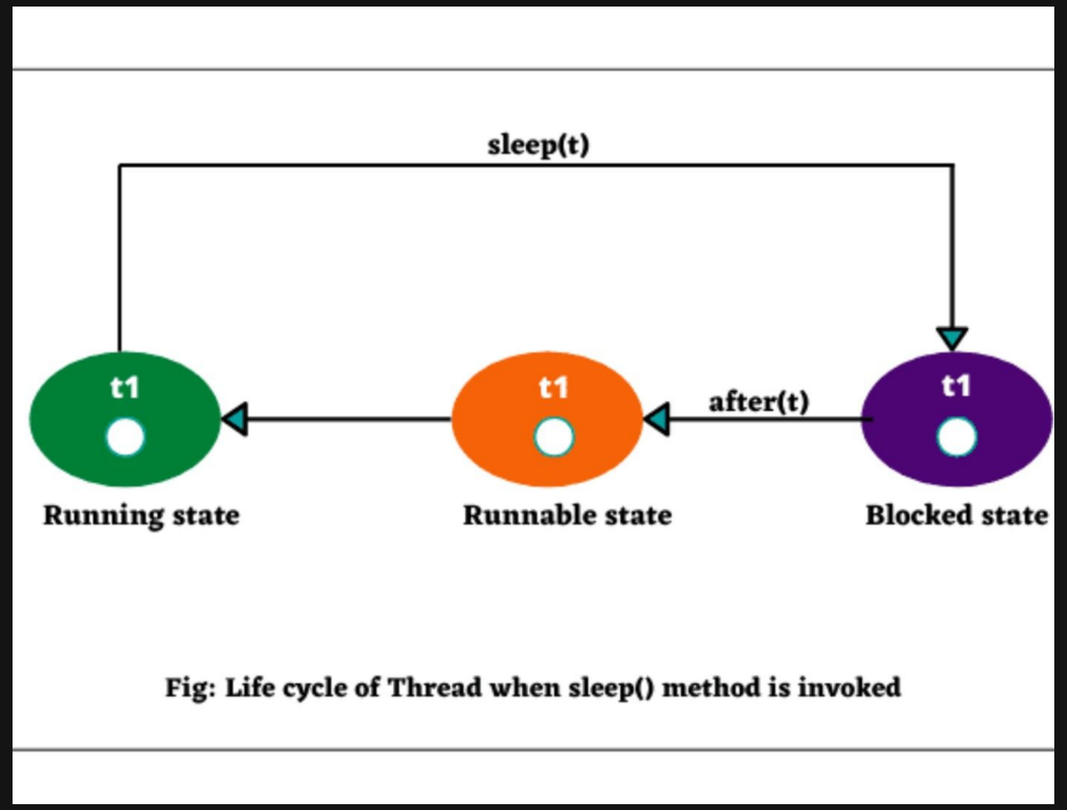


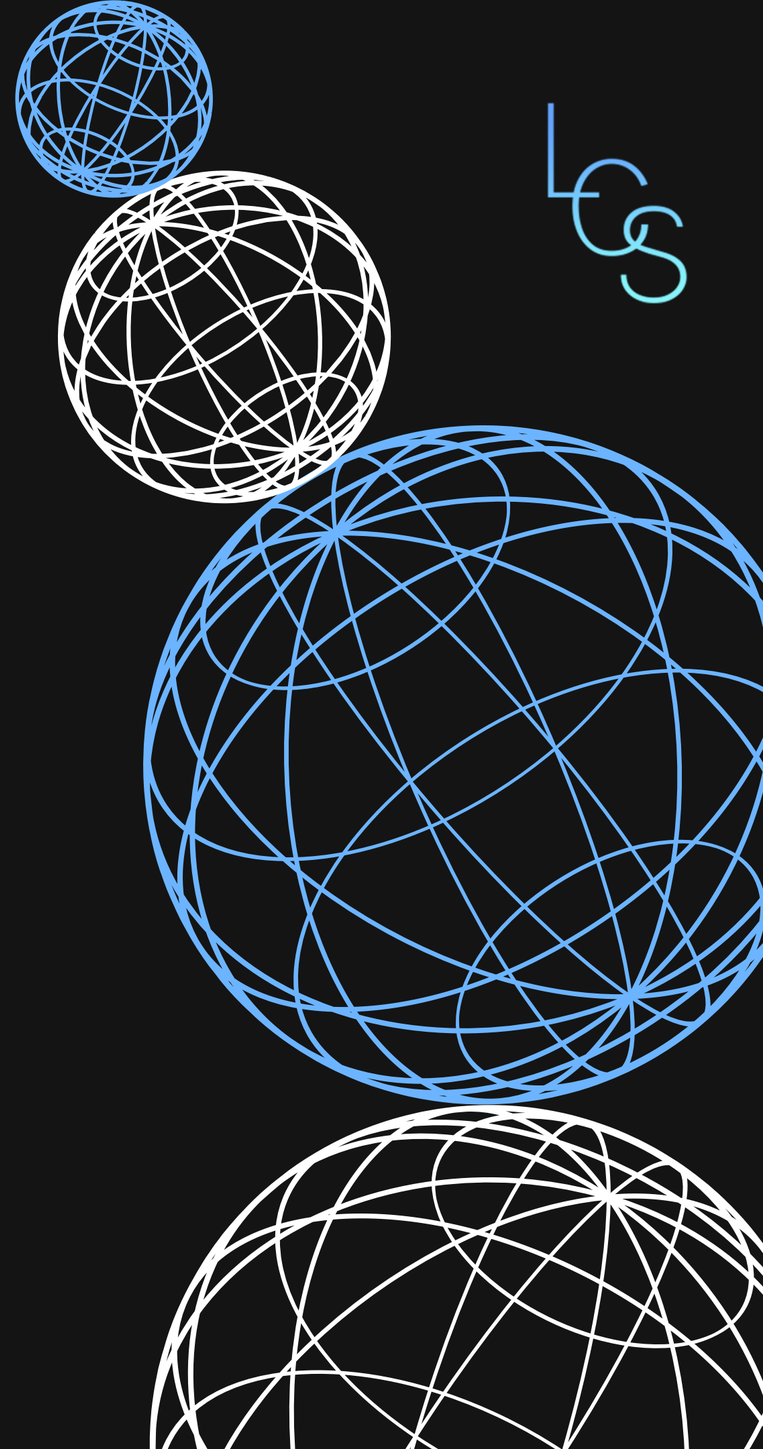
Fig: Life cycle of Thread when sleep() method is invoked

Java for Multithreading

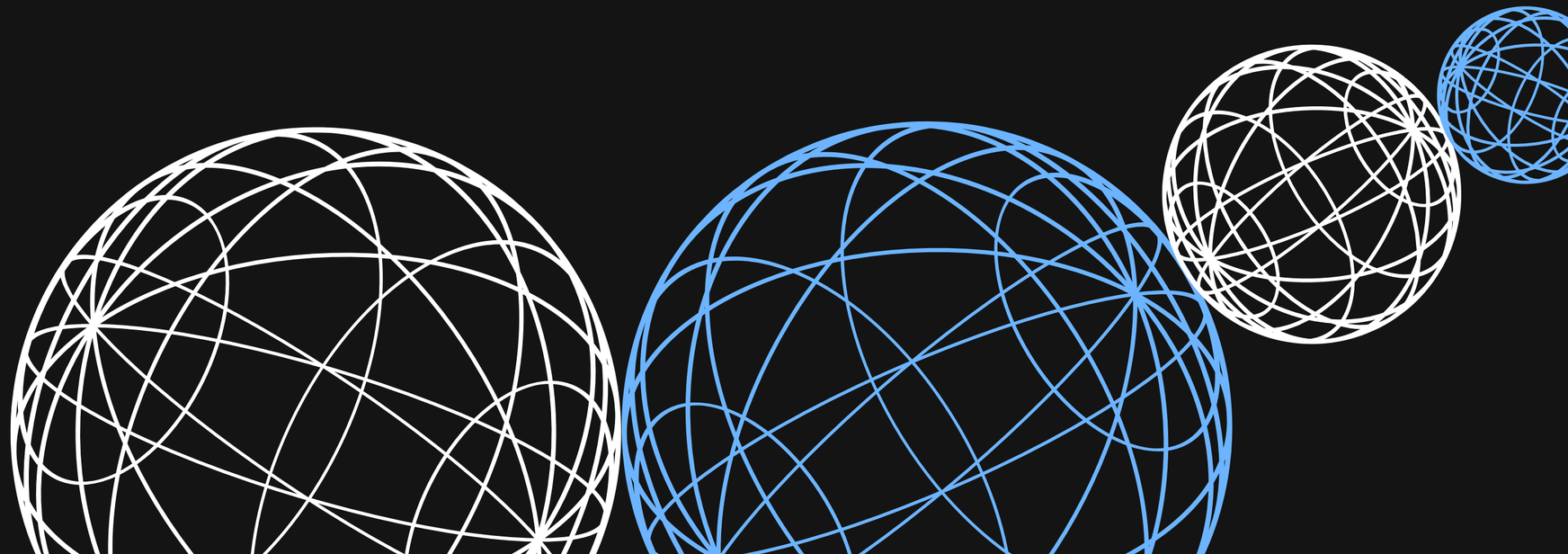
```java
package cp213;

//Define a class for the example
public class ThreadExample {
    // Main method
    public static void main(String[] args) {
        // Create a new thread
        Thread myThread = new Thread(() -> {
            // Loop for 5 iterations in the new thread
            for (int i = 1; i <= 5; i++) {
                // Print a message indicating the current iteration in the new thread
                System.out.println("Thread: " + i);

                try {
                    // Simulate a pause of 500 milliseconds in the new thread
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    // Handle the exception if the sleep is interrupted
                    System.out.println("Thread interrupted.");
                }
            }
        });
        // Start the new thread using start()
        myThread.start();
        // Loop for 5 iterations in the main thread
        for (int i = 1; i <= 5; i++) {
            // Print a message indicating the current iteration in the main thread
            System.out.println("Main: " + i);

            try {
                // Simulate a pause of 300 milliseconds in the main thread
                Thread.sleep(300);
            } catch (InterruptedException e) {
                // Handle the exception if the sleep is interrupted in the main thread
                System.out.println("Main thread interrupted.");
            }
        }
        // Print a message from the main method
        System.out.println("Main method is done.");
    }
}
```

```
<terminated> Thread
Main: 1
Thread: 1
Main: 2
Thread: 2
Main: 3
Main: 4
Thread: 3
Main: 5
Main method is done.
Thread: 4
Thread: 5
```

Do you have any questions?

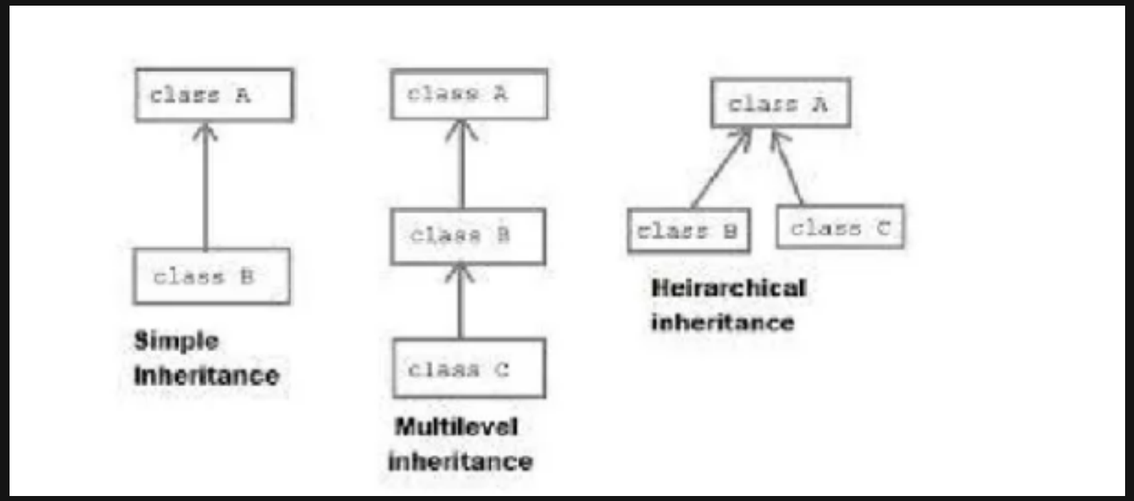


Semester Review



Inheritance

- INHERITANCE IS ONE OF THE MAIN TECHNIQUES OF OBJECT-ORIENTED PROGRAMMING (OOP)
- USING THIS TECHNIQUE, A VERY GENERAL FORM OF A CLASS IS FIRST DEFINED AND COMPILED, AND THEN MORE SPECIALIZED VERSIONS OF THE CLASS ARE DEFINED BY ADDING INSTANCE VARIABLES AND METHODS
    - THE SPECIALIZED CLASSES ARE SAID TO INHERIT THE METHODS AND INSTANCE VARIABLES OF THE GENERAL CLASS



Syntax

```
// Java Program to illustrate Inheritance (concise)

import java.io.*;

// Base or Super Class
class Employee {
    int salary = 60000;
}

// Inherited or Sub Class
class Engineer extends Employee {
    int benefits = 10000;
}
```

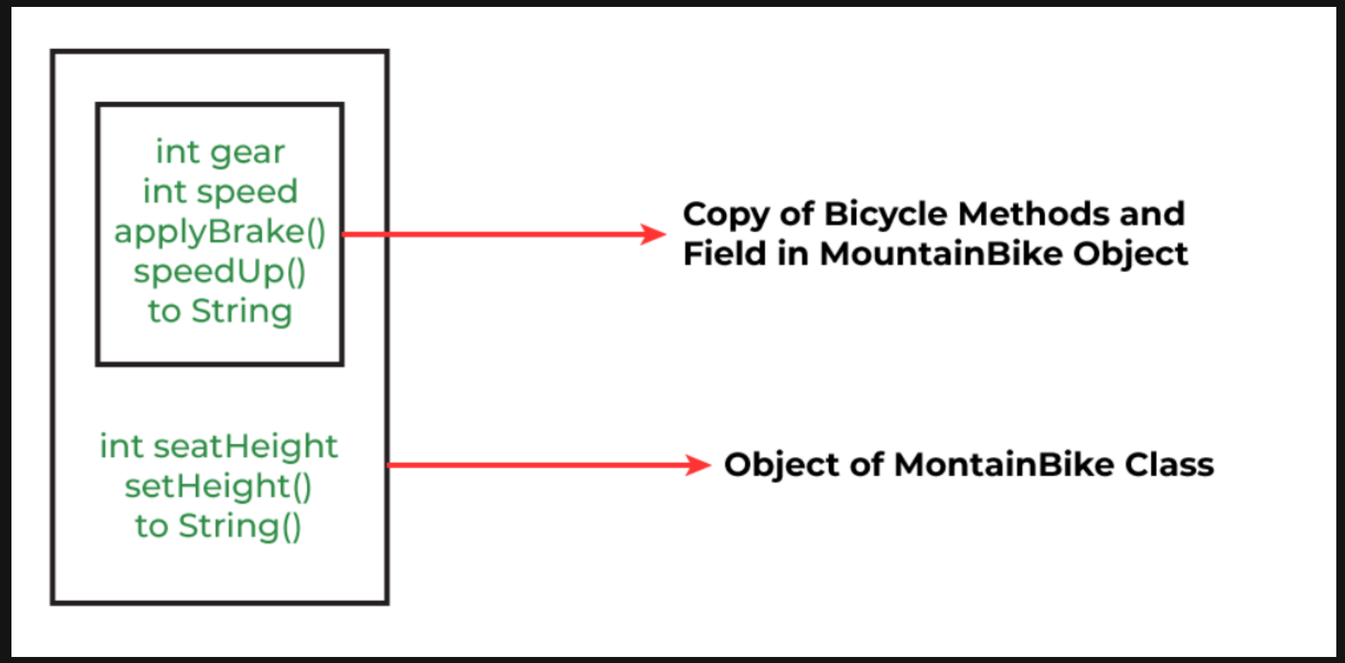
- BY USING THE ‘EXTEND’ KEYWORD WE ARE EXTENDING THE ORGINAL CLASS
- IN THE EXAMPLE ABOVE, WE HAVE A CLASS EMPLOYEE AND WE ARE EXTENDING THAT CLASS TO ENGINEER
- AN ENGINEER IS A TYPE OF EMPLOYEE THAT USES THE BASE CLASS’S GENERIC ATRRIBUTES BUT IT ALSO REQUIRES ITS OWN SET OF ATTRIBUTES

Example

```java
// Java program to illustrate the
// concept of inheritance

// base class
class Bicycle {
    // the Bicycle class has two fields
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int gear, int speed)
    {
        this.gear = gear;
        this.speed = speed;
    }

    // the Bicycle class has three methods
    public void applyBrake(int decrement)
    {
        speed -= decrement;
    }

    public void speedUp(int increment)
    {
        speed += increment;
    }

    // toString() method to print info of Bicycle
    public String toString()
    {
        return ("No of gears are " + gear + "\n"
                + "speed of bicycle is " + speed);
    }
}
```

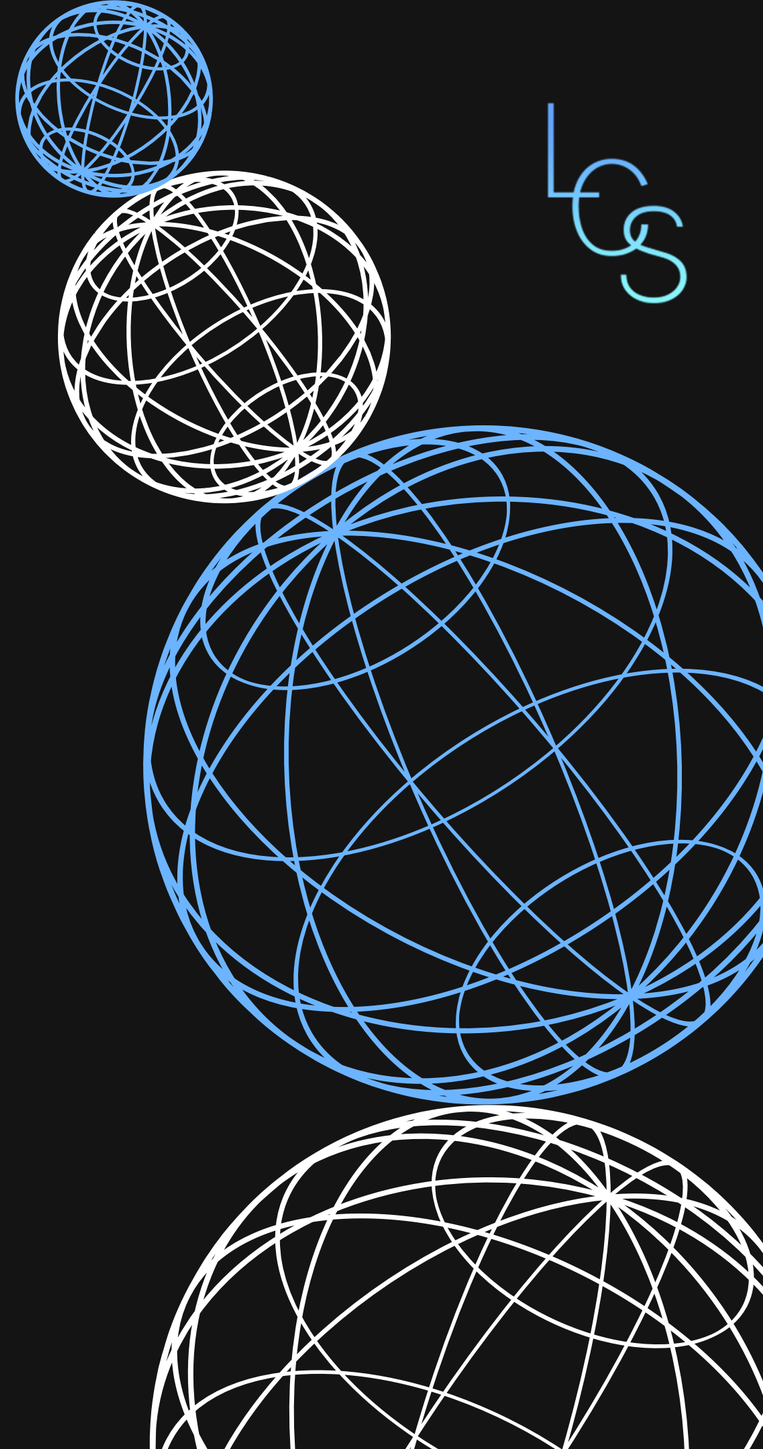
```java
// derived class
class MountainBike extends Bicycle {

    // the MountainBike subclass adds one more field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int gear, int speed,
                        int startHeight)
    {
        // invoking base-class(Bicycle) constructor
        super(gear, speed);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one more method
    public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }

    // overriding toString() method
    // of Bicycle to print more info
    @Override public String toString()
    {
        return (super.toString() + "\nseat height is "
                + seatHeight);
    }
}
```

- CLASS BICYCLE IS A BASE CLASS, CLASS MOUNTAINBIKE IS A DERIVED CLASS THAT EXTENDS THE BICYCLE CLASS



- IN THE ABOVE PROGRAM, WHEN AN OBJECT OF MOUNTAINBIKE CLASS IS CREATED, A COPY OF ALL METHODS AND FIELDS OF THE SUPERCLASS ACQUIRES MEMORY IN THIS OBJECT. THAT IS WHY BY USING THE OBJECT OF THE SUBCLASS WE CAN ALSO ACCESS THE MEMBERS OF A SUPERCLASS.

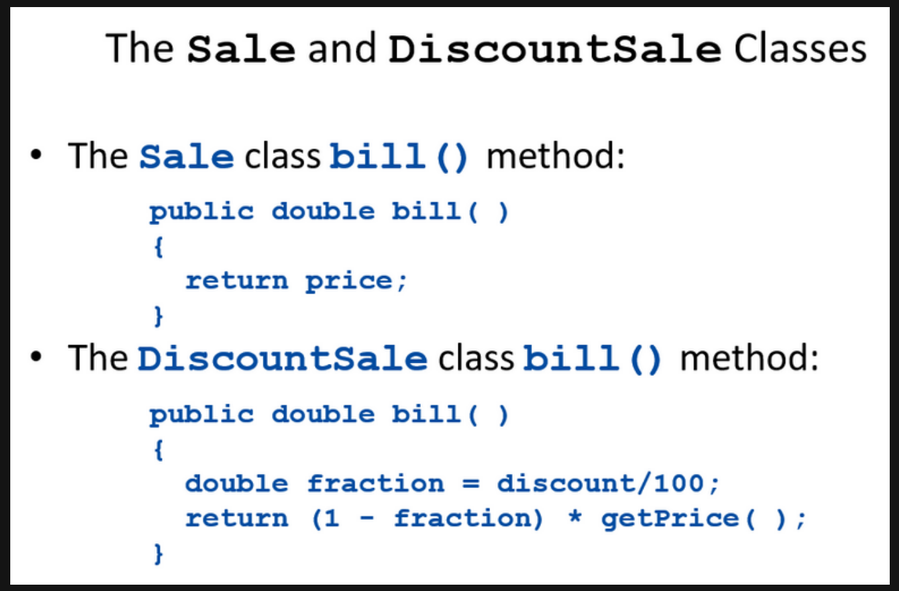
Do you have any questions?



Polymorphism

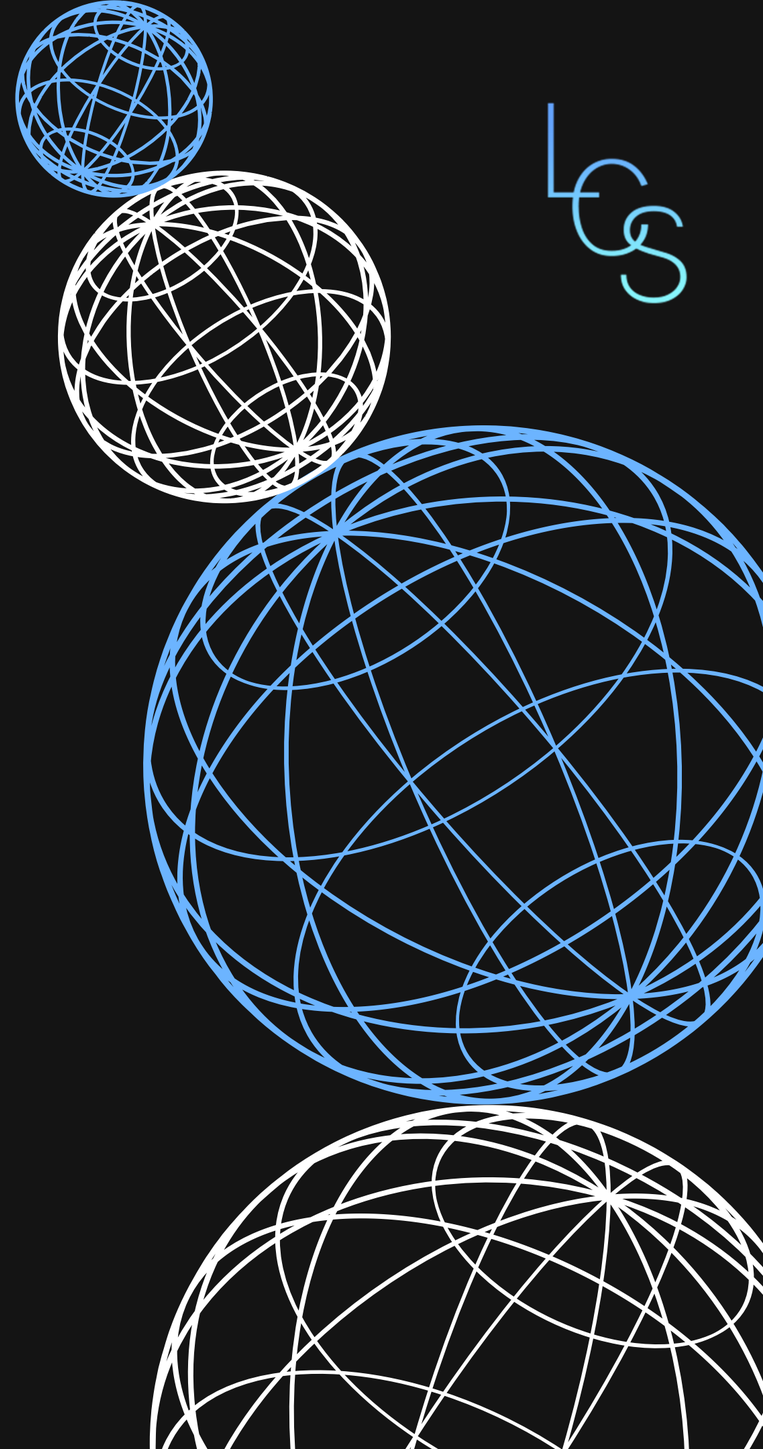
- INHERITANCE ALLOWS A BASE CLASS TO BE DEFINED, AND OTHER CLASSES DERIVED FROM IT
  - CODE FOR THE BASE CLASS CAN THEN BE USED FOR ITS OWN OBJECTS, AS WELL AS OBJECTS OF ANY DERIVED CLASSES
- POLYMORPHISM ALLOWS CHANGES TO BE MADE TO METHOD DEFINITIONS IN THE DERIVED CLASSES; THEY ARE VARIATIONS OF THE SAME METHOD
- METHODS ARE OVERRIDED AND CALLED DEPENDING ON THE SPECIFIC OBJECT BEING USED

- EX: CLASS DISCOUNTSALE EXTENDS CLASS SALE



This is polymorphism, as the method has multiple (“poly”) forms (“morphisms”) between the child and parent classes

Do you have any questions?



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."

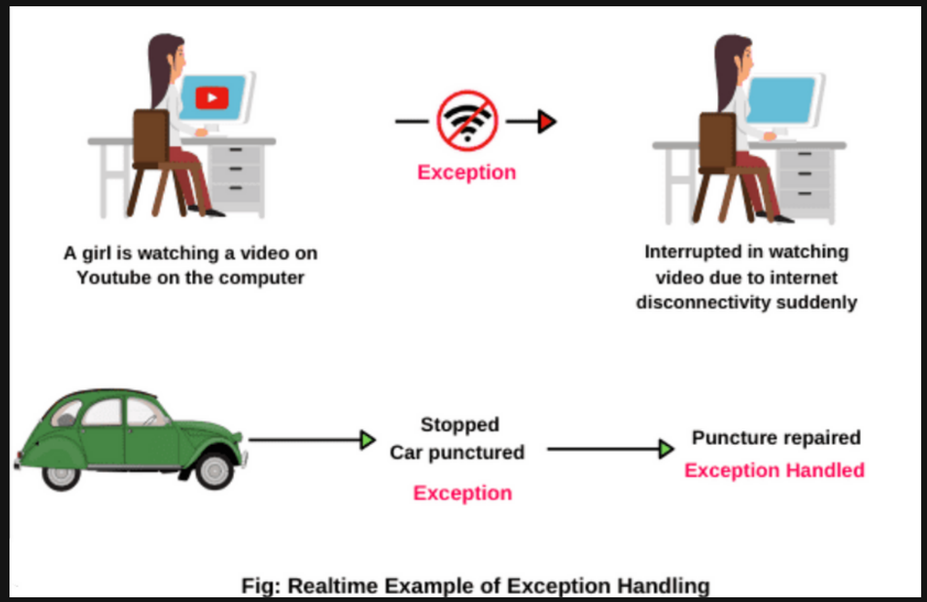


Fig: Realtime Example of Exception Handling

- 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:

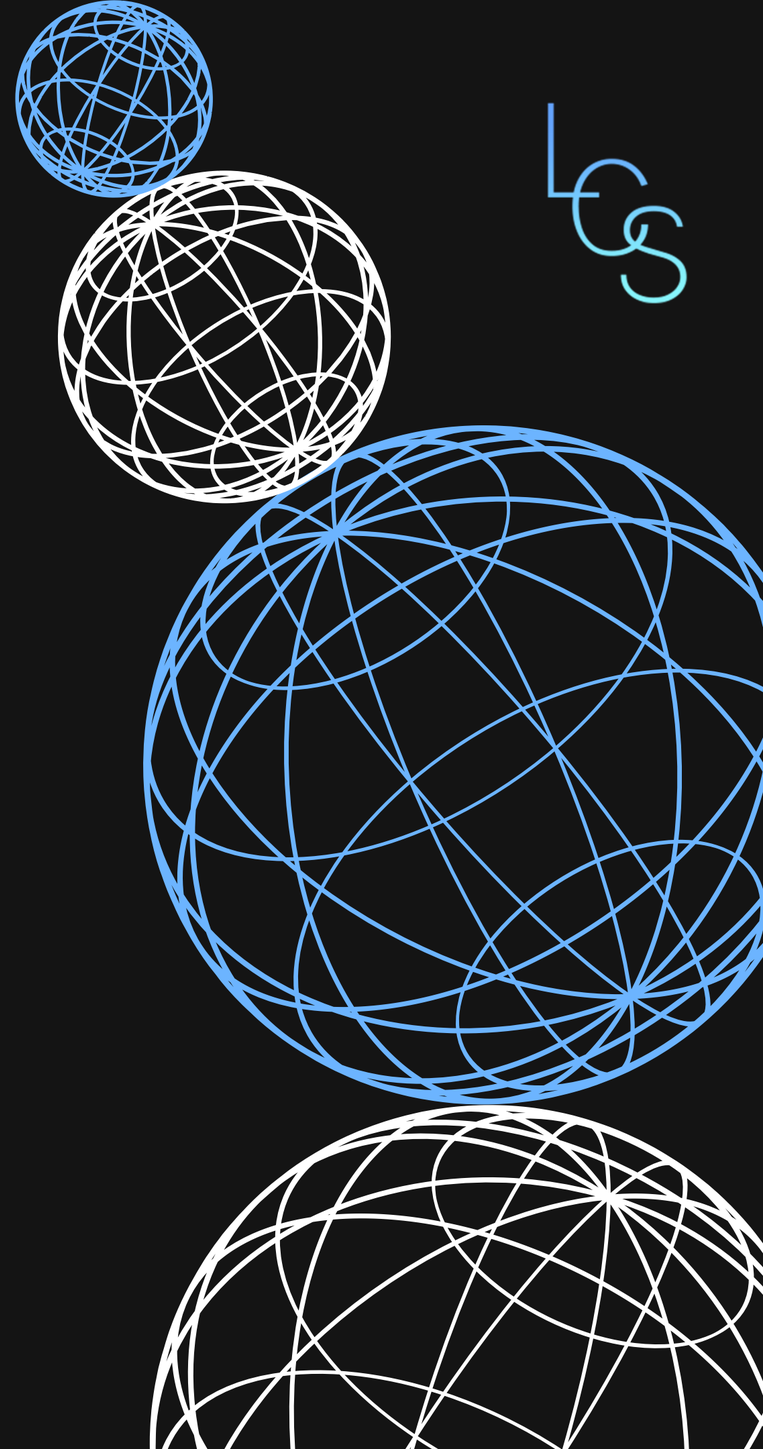
```
Try {
 int result = 10 / 0;
}
Catch (ArithmeticException e) {
 // Handle the exception
 System.out.println("Error: Cannot divide by zero!");
}
```

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:

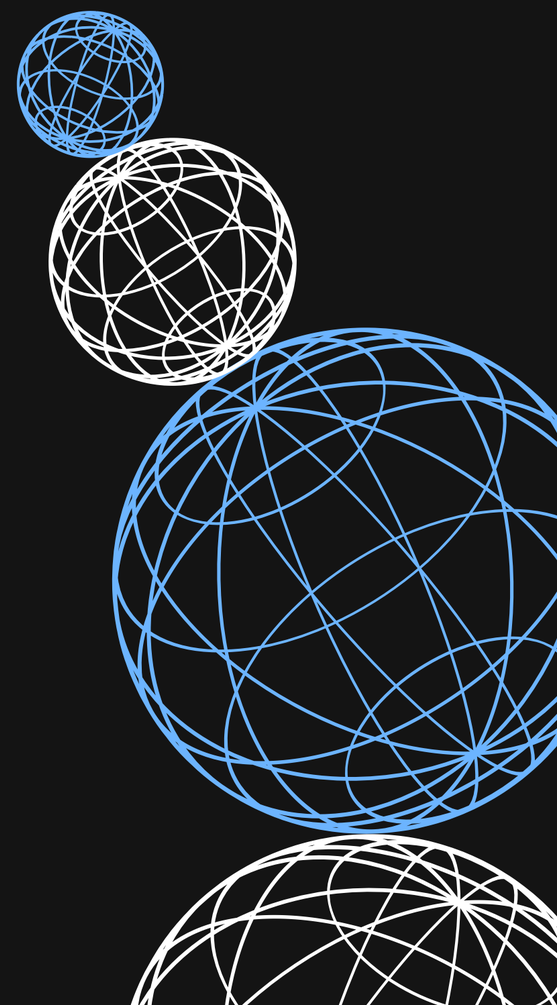
```
void checkTemperature(int temperature) throws
IllegalArgumentException {
    if (temperature > 100) {
      throw new IllegalArgumentException("Too hot!");
      }
}
```

Do you have any questions?



Try-Throw-Catch Example

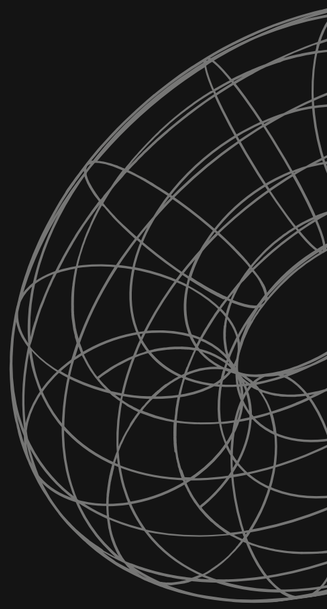
```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            // Try block: Attempting a division that may cause an exception
            int result = divide(10, 0);
            System.out.println("Result: " + result);  // This line won't be reached if an
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;
    }
}
```



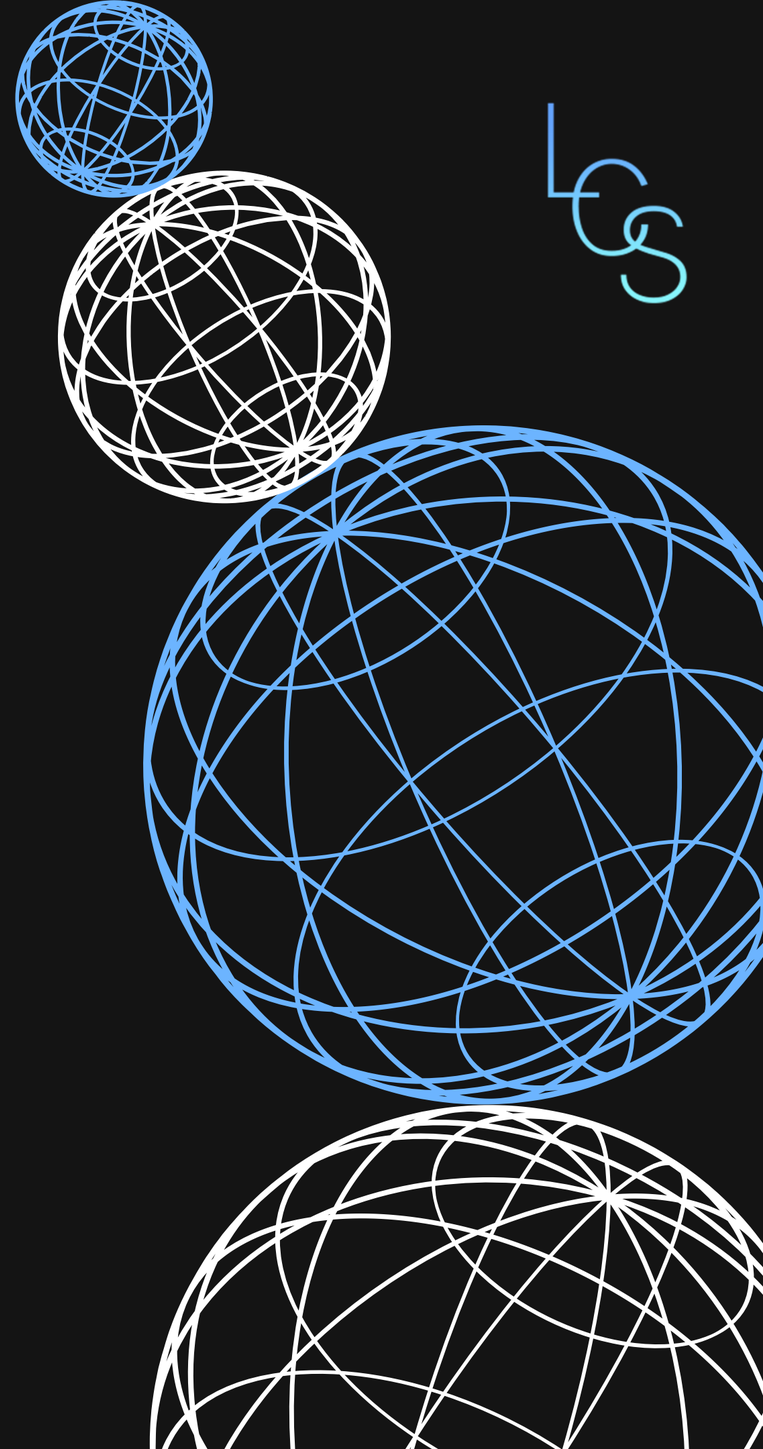
Finally Block

- 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?



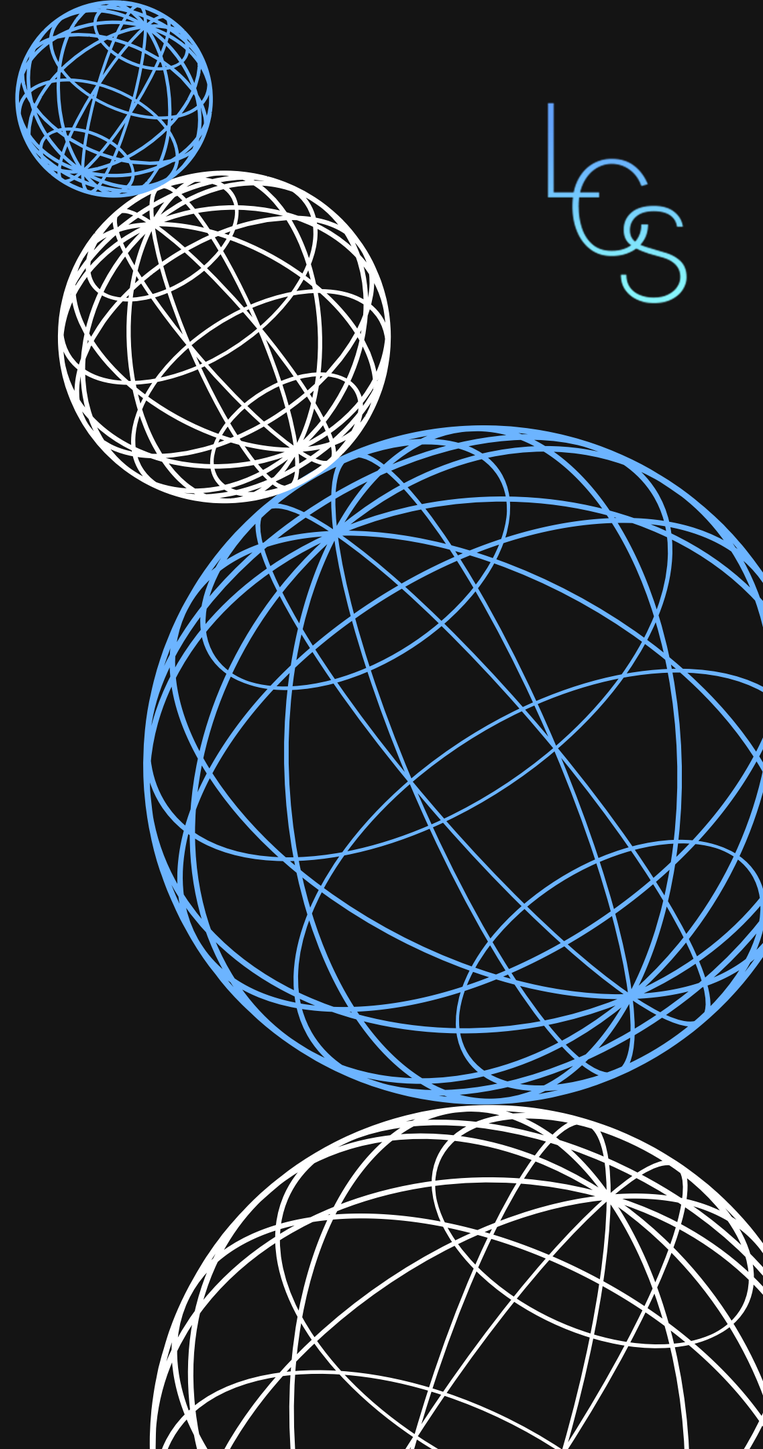
Interfaces

- 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

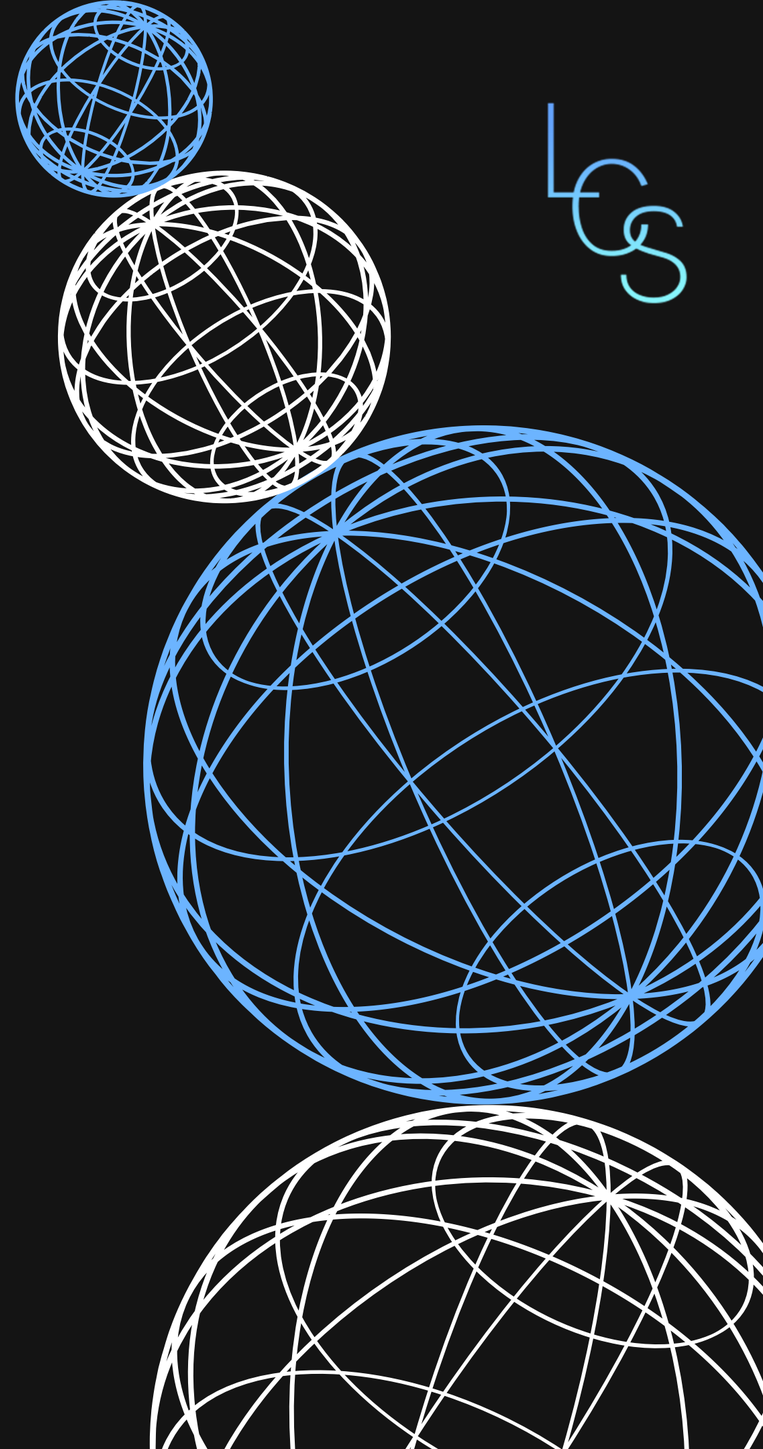
Interfaces- Example

```
// 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?



Live code example!



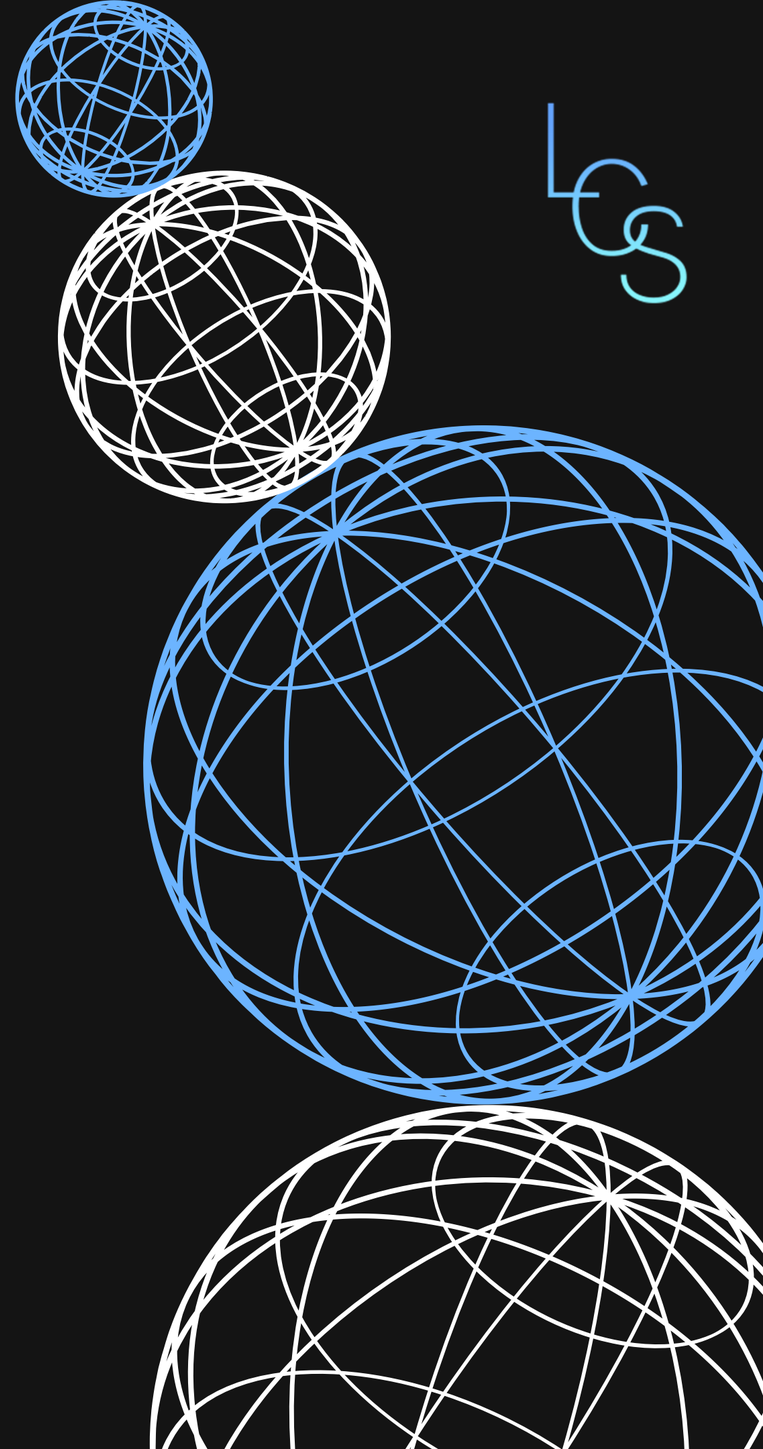
Inner Classes

- 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