Introduction to Java

Using Classes

Writing `class Geometry`

Lecture Contents

- Review:
  - Java Class Library
  - `Math` Methods and Constants
- Writing a Class
  - `Geometry` Class

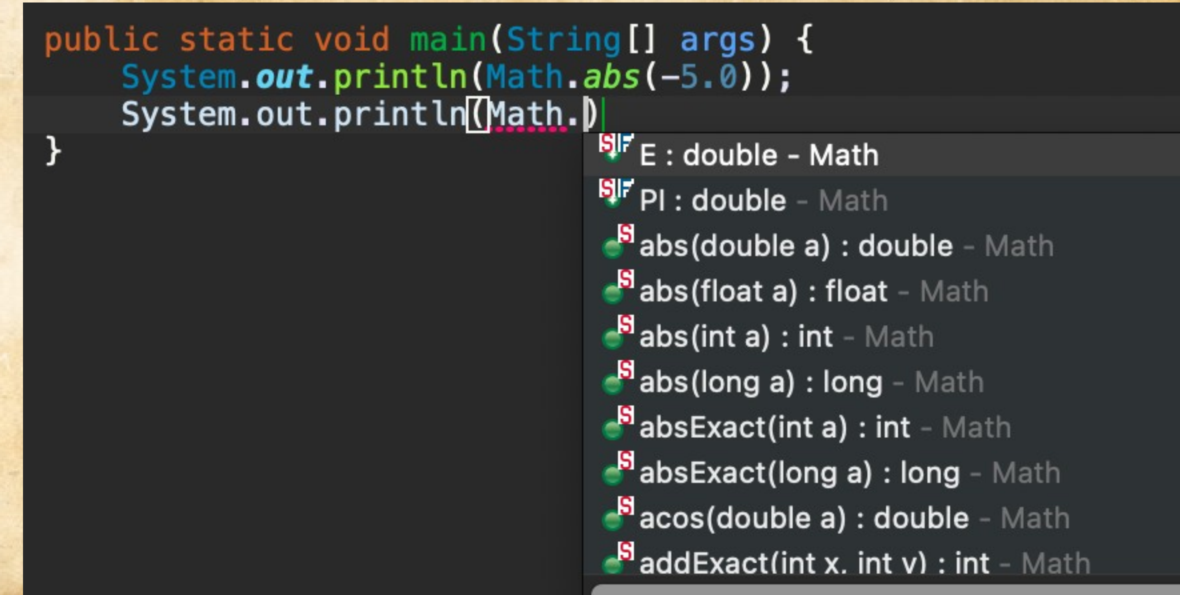
Java Class Library

- Java Class Library (JCL) is part of the Java Development Kit (JDK)
  - A comprehensive collection of pre-built classes and methods
  - Provides essential functionality for Java applications
  - Includes functionality in classes, for example:
    - `Math` (which we’ll discuss in this slide show)
    - `Array` (sorting, etc.)
    - `File` (for file I/O)
    - `ArrayList`
    - `HashMap`

Math Methods and Constants

- **Methods**
  - Math.abs(-5);
  - Math.sqrt(2.0);
  - Math.min(3, 5);
  - Math.max(3, 5);
  - Math.sin(3.14);
  - Math.asin(0.5);
  - Math.pow(2, 5);
  - Math.random();

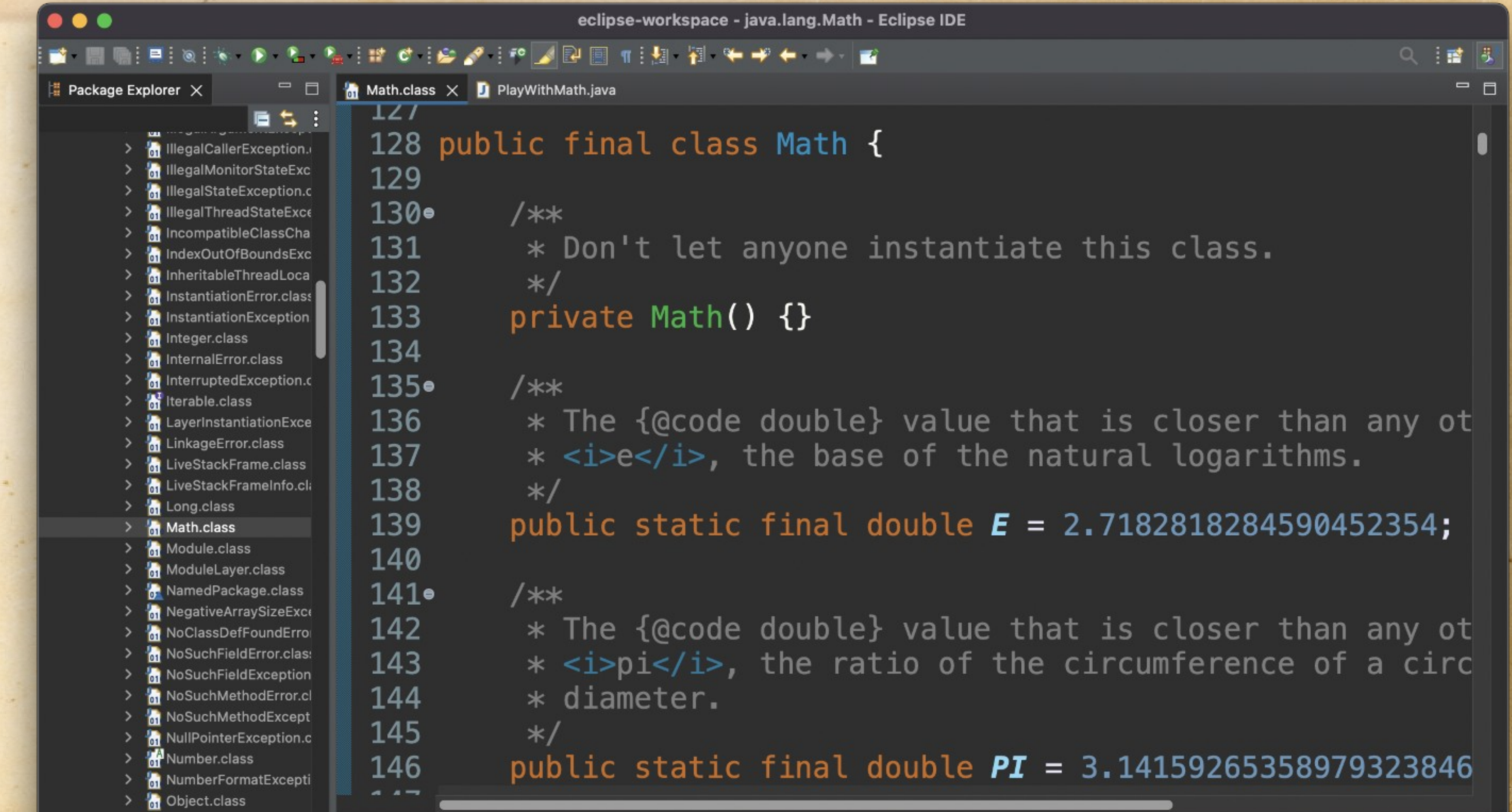
- **Constants**
  - Math.PI
  - Math.E



```
public static void main(String[] args) {
    System.out.println(Math.abs(-5.0));
    System.out.println(Math.)
}
```

E : double - Math
PI : double - Math
abs(double a) : double - Math
abs(float a) : float - Math
abs(int a) : int - Math
abs(long a) : long - Math
absExact(int a) : int - Math
absExact(long a) : long - Math
acos(double a) : double - Math
addExact(int x, int y) : int - Math

- You can view the code of the `Math` class and compare it to the classes we write.



```
public final class Math {

    /**
     * Don't let anyone instantiate this class.
     */
    private Math() {}

    /**
     * The {@code double} value that is closer than any ot
     * <i>e</i>, the base of the natural logarithms.
     */
    public static final double E = 2.718281828459045 4;

    /**
     * The {@code double} value that is closer than any ot
     * <i>pi</i>, the ratio of the circumference of a circ
     * diameter.
     */
    public static final double PI = 3.14159265358979323846
```

Vocabulary - ***refactor***

- To ***refactor*** is to improve the internal structure or design of existing code without changing its behavior.
- *Refactoring* includes:
  - **Renaming** variables, methods or classes to make their purpose more clear
  - **Reorganizing code** to improve structure or readability
  - **Extracting** methods to simplify code (break it into smaller parts)
  - **Consolidating** code by moving duplication into common methods
  - **Improving comments**
  - **Optimizing** algorithms
  - Replacing ***magic numbers*** (literal numbers) with constants.
  - **Simplifying** conditional expressions, loops, etc.

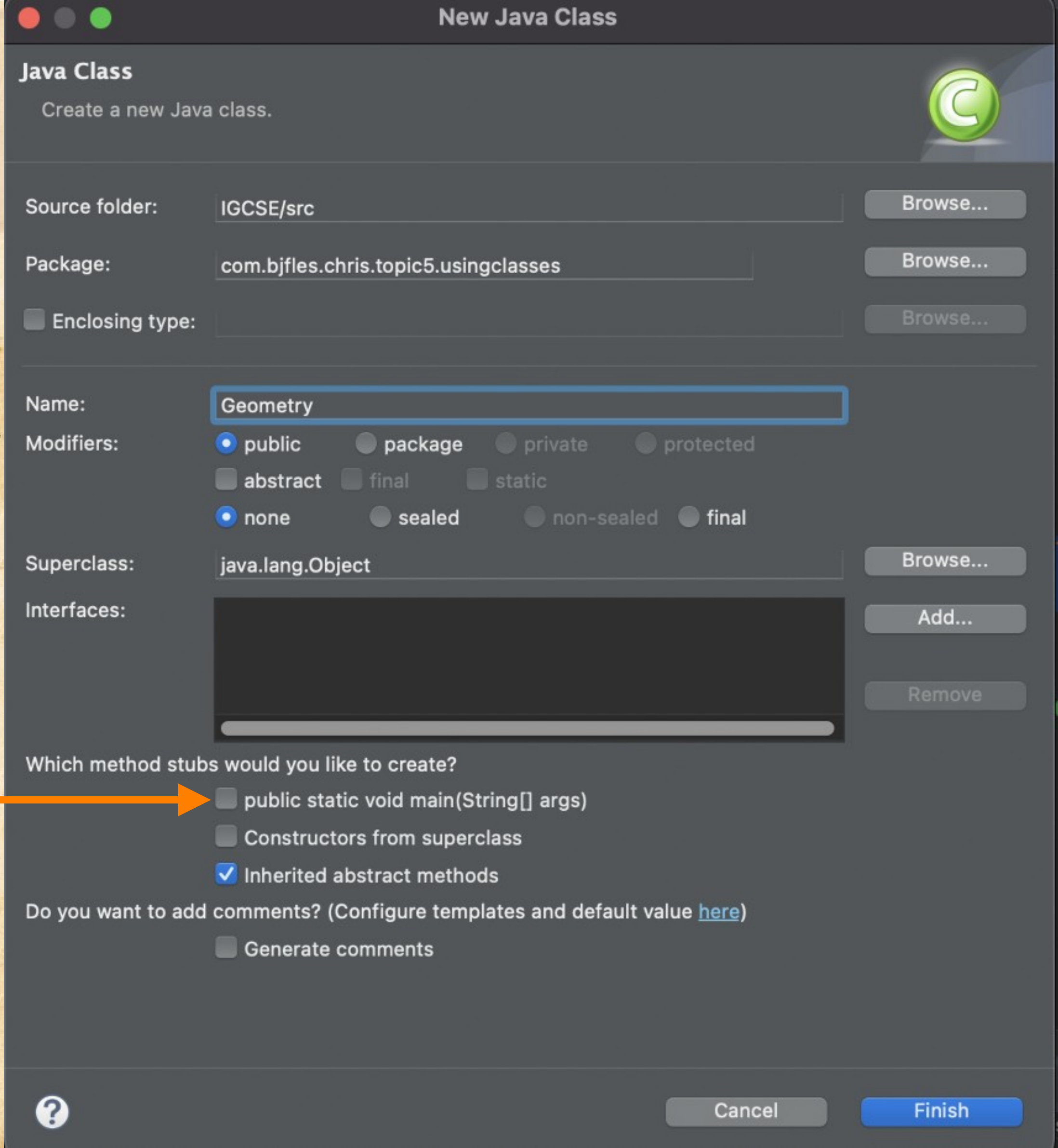
Refactoring `Geometry` Class

- To ***refactor*** is to improve the internal structure or design of existing code without changing its behavior.
- We will ***refactor*** the code from `UsingMath` into two new classes:
  - A `Geometry` class
  - A `UsingGeometry` class.
- This is an exercise in separating code that serves a common function into its own class.

`class Geometry`

- Create a new `Geometry` class, but do ***not*** add a `main` method.

Selecting this will add template code for the **`main`** method, which is where the Java program will start running.



New Java Class

Java Class

Create a new Java class.

Source folder: IGCSE/src

Package: com.bjfles.chris.topic5.usingclasses

☐ Enclosing type:

Name: Geometry

Modifiers: ● public ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

● none ○ sealed ○ non-sealed ○ final

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

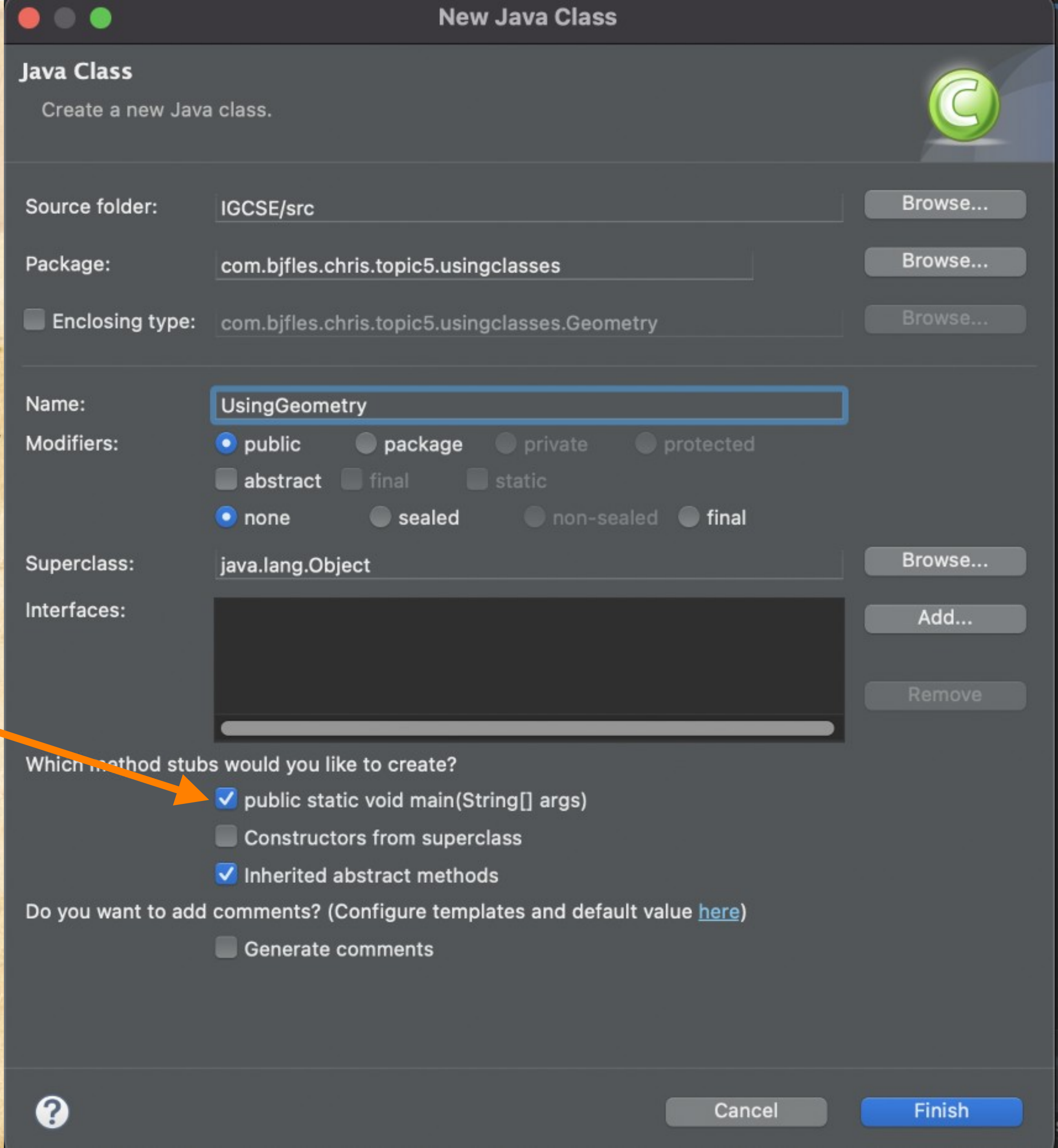
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Geometry Class

- Create a second class named `UsingGeometry`, this time *do* add a `main` method.
  - The `Geometry` and `UsingGeometry` must be in the same package for this exercise.



New Java Class

Java Class

Create a new Java class.

Source folder: IGCSE/src

Package: com.bjfles.chris.topic5.usingclasses

□ Enclosing type: com.bjfles.chris.topic5.usingclasses.Geometry

Name: UsingGeometry

Modifiers: ● public ○ package ○ private ○ protected

□ abstract □ final □ static

● none ○ sealed ○ non-sealed ○ final

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

□ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

Refactoring `class Geometry`

- Copy your methods `calculateCircumference` and `calculateCircleArea` from your `UsingMath` into your `Geometry` class.
- Copy the code from your `main` method in `UsingMath` into your `UsingGeometry`.
- Append “`Geometry.`” to the beginning of each method call in your `main` method in the `UsingGeometry` so that the compiler knows where to find those methods.
- Run `UsingGeometry` and ensure it is working correctly. The output should be the same as it was for your `UsingMath`.

Expanding `class Geometry`

- Add a method `hyptoenuseLength` to your `Geometry` class. Have it take two parameters of type `double`, representing the two legs of a right-angle triangle, and use those two values to calculate and return the length of the hypotenuse, again as type `double`. Recall that formula for this is:

$$h = \sqrt{a^2 + b^2}$$

- Test your method by calling `hypotenuseLength` from your `UsingGeometry` class, perhaps using the values `a = 3.0` and `b = 4.0`, since we know the solution should be `h = 5.0`.

Final Output

- The final output of you UsingGeometry class should be:

```
The value of pi is: 3.141592653589793
The circumference of a circle of radius 5.0 is equal to 31.41592653589793.
The area of a circle of radius 5.0 is equal to 78.53981633974483.
A right-angle triangle with side lengths of 3.0 and 4.0 has a hypotenuse length 5.0.
```

Using Classes

Writing `class Geometry`