

# Topic 5: Developing Code

Writing a Geometry Class



# 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**

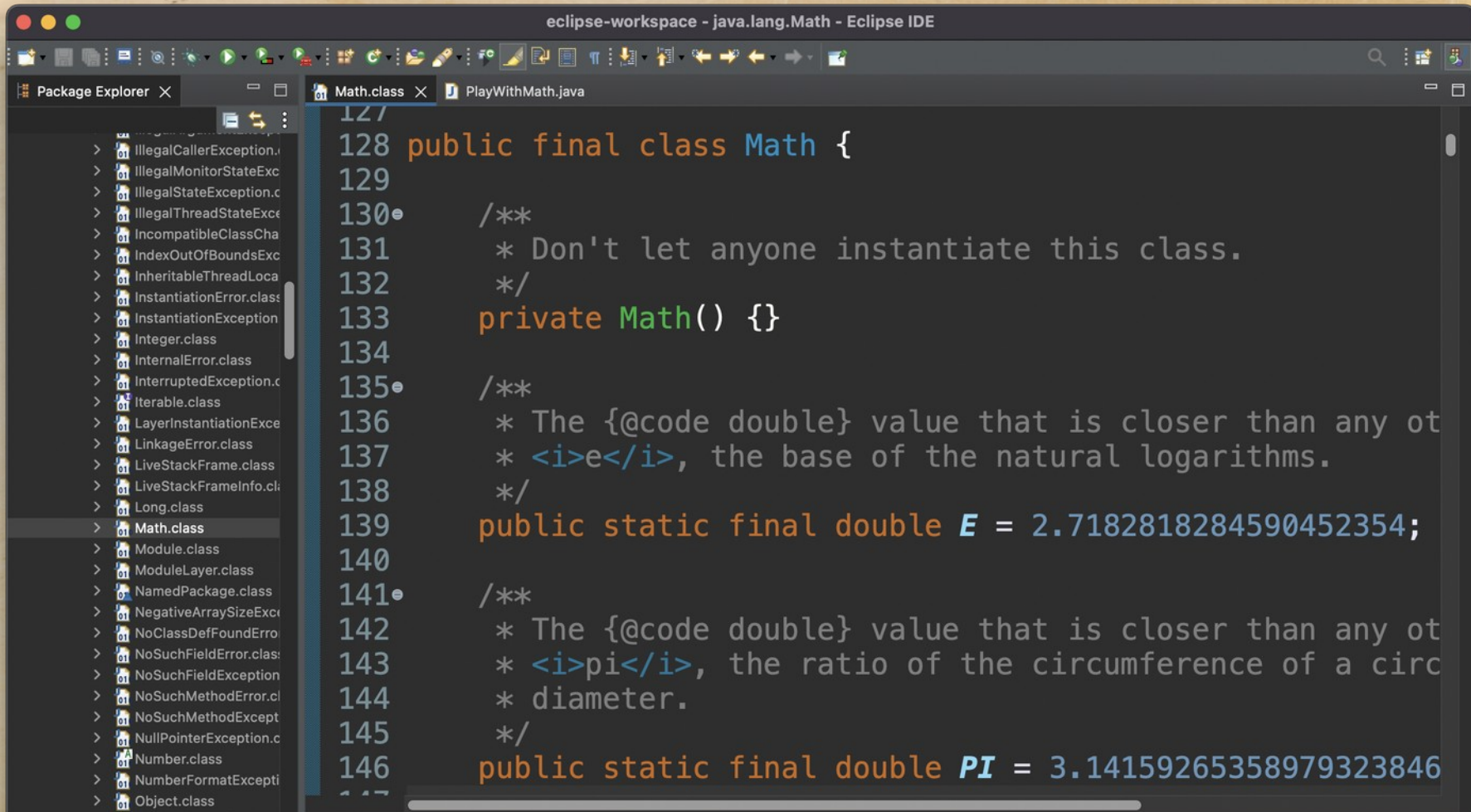
- `Math.PI`
- `Math.E`

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

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.



```
127
128 public final class Math {
129
130     /**
131      * Don't let anyone instantiate this class.
132      */
133     private Math() {}
134
135     /**
136      * The {@code double} value that is closer than any ot
137      * <i>e</i>, the base of the natural logarithms.
138      */
139     public static final double E = 2.7182818284590452354;
140
141     /**
142      * The {@code double} value that is closer than any ot
143      * <i>pi</i>, the ratio of the circumference of a circ
144      * diameter.
145      */
146     public static final double PI = 3.14159265358979323846
147
```

# 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 `UsingMathClass` into two new classes:
  - A `Geometry` class
  - A `UsingGeometryClass` class.
- This is an exercise in separating code that serves a common function into its own class.

# Geometry Class

- 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**

Create a new Java class.

Source folder: IGCSE/src Browse...

Package: com.bjfiles.chris.topic5.usingclasses Browse...

☐ Enclosing type: Browse...

Name: Geometry

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static  
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass: java.lang.Object Browse...

Interfaces: Add...  
Remove

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

? Cancel Finish



# Geometry Class

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

**New Java Class**

Create a new Java class.

Source folder: `IGCSE/src` Browse...

Package: `com.bjfiles.chris.topic5.usingclasses` Browse...

☐ Enclosing type: `com.bjfiles.chris.topic5.usingclasses.Geometry` Browse...

Name: `UsingGeometryClass`

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static  
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass: `java.lang.Object` Browse...

Interfaces: Add... Remove

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

? Cancel Finish

# Refactoring Geometry Class

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



# Topic 5: Developing Code

Writing a Geometry Class