Your First Java Scientific Calculator Project!

Welcome, aspiring Java engineers! This project will guide you through building a command-line scientific calculator. You'll learn how to take user input, leverage Java's built-in mathematical functions, structure your code with methods, and manage your project with Git and GitHub.

Let's break down the foundational concepts first.

1. Taking Input in Java

In Java, to read input from the user (like numbers or text typed into the console), you primarily use the **`Scanner`** class.

How it works:

1. **Import `Scanner`:** The `Scanner` class is part of the `java.util` package, so you need to import it at the beginning of your Java file:

   Java

   ```
   import java.util.Scanner;
   ```

2. **Create a `Scanner` object:** You need to create an instance of the `Scanner` class, typically linked to `System.in` (which represents the standard input stream, usually your keyboard).

   Java

   ```
   Scanner scanner = new Scanner(System.in);
   ```

3. **Read different data types:** The `Scanner` object provides various methods to read different types of input:
   - `nextInt()`: Reads the next **integer**.
   - `nextDouble()`: Reads the next **double-precision floating-point number**.
   - `next()`: Reads the next **word** (token) as a string.
   - `nextLine()`: Reads the **entire line** of input until a newline character is encountered. This is useful for reading sentences.

Example of taking input:

Java

```
import java.util.Scanner; // Don't forget this import!

public class InputExample {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the console
        Scanner inputReader = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String name = inputReader.nextLine(); // Reads the whole line

        System.out.print("Enter your age: ");
        int age = inputReader.nextInt(); // Reads an integer
```

```
        System.out.print("Enter your height in meters (e.g., 1.75): ");
        double height = inputReader.nextDouble(); // Reads a double

        System.out.println("\nHello, " + name + "!");
        System.out.println("You are " + age + " years old.");
        System.out.println("And you are " + height + " meters tall.");

        // It's good practice to close the scanner when you're done with it
        inputReader.close();
    }
}
```

---

2. The Java `Math` Class

Java provides a powerful **`Math`** class (part of `java.lang` package, so no import needed!) that contains methods for performing common mathematical operations. All methods in the `Math` class are **`static`**, meaning you can call them directly on the class name without creating an object.

Key `Math` functions for your calculator:

- **`Math.sqrt(double a)`**: Returns the **square root** of a `double` value `a`.
  - Example: `double result = Math.sqrt(25.0); // result is 5.0`
- **`Math.pow(double base, double exponent)`**: Returns the value of the `base` raised to the power of the `exponent`.
  - Example: `double result = Math.pow(2.0, 3.0); // result is 8.0 (2 * 2 * 2)`
- **`Math.sin(double angle)`**: Returns the trigonometric **sine** of an angle. **Important:** The angle must be in **radians**.
  - Example: `double result = Math.sin(Math.toRadians(90)); // Sine of 90 degrees (which is pi/2 radians)`
- **`Math.cos(double angle)`**: Returns the trigonometric **cosine** of an angle. **Important:** The angle must be in **radians**.
  - Example: `double result = Math.cos(Math.toRadians(0)); // Cosine of 0 degrees`
- **`Math.tan(double angle)`**: Returns the trigonometric **tangent** of an angle. **Important:** The angle must be in **radians**.
  - Example: `double result = Math.tan(Math.toRadians(45)); // Tangent of 45 degrees`
- **`Math.toRadians(double deg)`**: Converts an angle measured in **degrees to radians**. Use this before `sin`, `cos`, `tan` if your user inputs degrees.
  - Example: `double radians = Math.toRadians(180);`
- **`Math.toDegrees(double rad)`**: Converts an angle measured in **radians to degrees**.
- **`Math.log(double a)`**: Returns the **natural logarithm** (base e) of a `double` value `a`.
  - Example: `double result = Math.log(Math.E); // result is 1.0`
- **`Math.log10(double a)`**: Returns the **base 10 logarithm** of a `double` value `a`.
  - Example: `double result = Math.log10(100.0); // result is 2.0`
- **`Math.abs(type a)`**: Returns the **absolute value** of a number. This method is overloaded for `int`, `long`, `float`, and `double`.
  - Example: `int result = Math.abs(-10); // result is 10`
- **`Math.round(double a)`**: Returns the **closest `long`** to the argument. For `float` it returns `int`.
  - Example: `long result = Math.round(3.6); // result is 4`
  - Example: `long result2 = Math.round(3.4); // result is 3`
- **`Math.ceil(double a)`**: Returns the smallest (closest to negative infinity) `double` value that is **greater than or equal to** the argument and is equal to a mathematical integer (rounds up).

    - Example: `double result = Math.ceil(4.1); // result is 5.0`
- **`Math.floor(double a)`**: Returns the largest (closest to positive infinity) `double` value that is **less than or equal to** the argument and is equal to a mathematical integer (rounds down).
    - Example: `double result = Math.floor(4.9); // result is 4.0`
- **`Math.min(type a, type b)`**: Returns the **smaller** of two numbers. Overloaded for `int`, `long`, `float`, `double`.
    - Example: `int result = Math.min(5, 10); // result is 5`
- **`Math.max(type a, type b)`**: Returns the **larger** of two numbers. Overloaded for `int`, `long`, `float`, `double`.
    - Example: `int result = Math.max(5, 10); // result is 10`

---

3. The Scientific Calculator Task

Your goal is to create a command-line scientific calculator in a single Java class.

Requirements:

1. **Single Class:** All your code will reside in one Java class (e.g., `ScientificCalculator.java`).
2. **Main Loop:** The calculator should run continuously, presenting a menu of options to the user until they choose to exit. This will require a **`while` loop**.
3. **Menu Driven:** Display a clear menu of operations (e.g., "1. Add", "2. Subtract", "3. Square Root", etc.).
4. **User Input:** Use the `Scanner` class to read the user's choice and the numbers for calculations.
5. **Separate Functions (Methods):** For *every* calculation, you *must* create a separate method. This promotes good code organization and reusability. For example, `public static double add(double num1, double num2)`, `public static double calculateSquareRoot(double num)`.
6. **Conditional Logic (`switch` statement):** Use a `switch` statement in your `main` method to handle the user's menu choice and call the appropriate calculation method.
7. **Error Handling (Basic):** Implement basic error handling for invalid menu choices or non-numeric input where appropriate (e.g., using `try-catch` blocks for `InputMismatchException` or simple `if` checks to ensure positive numbers for square root, etc.).
8. **Output:** Display the results of calculations clearly to the user.

Specific Functionalities to Implement and `Math` Class Mapping:

Category	Operation	Math Class Function (if applicable)	Method Signature Example	Notes
Basic Arithmetic	Addition (+)		public static double add(double num1, double num2)	Handles standard addition of two numbers.
	Subtraction (-)		public static double subtract(double num1, double num2)	Handles standard subtraction of two numbers.
	Multiplication (*)		public static double multiply(double num1, double num2)	Handles standard multiplication of two numbers.
	Division (/)		public static double divide(double num1, double num2)	Implement a check for division by zero.

Scientific Operations	Square Root (√x)	Math.sqrt()	public static double calculateSquareRoot(double num)	Use Math.sqrt(). Handle negative input.
	Power (x^y)	Math.pow()	public static double calculatePower(double base, double exponent)	Use Math.pow().
	Sine (sin(x))	Math.sin()	public static double calculateSine(double degrees)	Use Math.sin(). Convert degrees to radians using Math.toRadians().
	Cosine (cos(x))	Math.cos()	public static double calculateCosine(double degrees)	Use Math.cos(). Convert degrees to radians using Math.toRadians().
	Tangent (tan(x))	Math.tan()	public static double calculateTangent(double degrees)	Use Math.tan(). Convert degrees to radians using Math.toRadians(). Handle undefined angles (e.g., 90°, 270°).
	Natural Logarithm (ln x)	Math.log()	public static double calculateNaturalLogarithm(double num)	Use Math.log(). Handle non-positive input.
	Logarithm Base 10 ($\log_{10}$ x)	Math.log10()	public static double calculateLogarithmBase10(double num)	Use Math.log10(). Handle non-positive input.
	Absolute Value (	x	)	Math.abs()
	Round (to nearest long)	Math.round()	public static long roundNumber(double num)	Use Math.round().
	Ceiling (round up)	Math.ceil()	public static double ceilingNumber(double num)	Use Math.ceil().
	Floor (round down)	Math.floor()	public static double floorNumber(double num)	Use Math.floor().
	Minimum of two numbers	Math.min()	public static double findMin(double num1, double num2)	Use Math.min().
	Maximum of two numbers	Math.max()	public static double findMax(double num1, double num2)	Use Math.max().
Control & Utilities	Display Menu		public static void displayMenu()	Prints all available options to the console.
	Helper methods		private static void performOperation(Scanner scanner)	Encapsulate user interaction: prompting for numbers, calling method, printing result, handling input errors.
	Exit			Allows the user to exit the calculator program.

These methods will encapsulate the user interaction (prompting for numbers, calling the calculation method, printing result, handling input errors) for each specific operation. | | Exit | | | The ability for the user to exit the calculator program. |

Example: The `ScientificCalculator` Class Structure (Conceptual)

Your class will generally look like this:

Java

```
import java.util.Scanner;
import java.util.InputMismatchException; // Needed for handling non-numeric input

public class ScientificCalculator {

    // You might declare constants here, like PI or E

    public static void main(String[] args) {
        // Create Scanner object
        // Loop for continuous calculator operation (while loop)
        //    Display menu (call displayMenu() method)
        //    Get user choice
        //    Use switch-case statement to call appropriate method based on choice
        //    Handle invalid choices
        //    Handle exit condition
        // Close scanner
    }

    // --- Menu Display Method ---
    public static void displayMenu() {
        // Print all calculator options
    }

    // --- Basic Arithmetic Methods ---
    public static double add(double num1, double num2) {
        // Logic for addition
        return 0; // Placeholder
    }

    // public static double subtract(...) { ... }
    // public static double multiply(...) { ... }
    // public static double divide(...) { ... }

    // --- Scientific Math Methods ---
    // public static double calculateSquareRoot(...) { ... }
    // public static double calculatePower(...) { ... }
    // public static double calculateSine(...) { ... }
    // ... and so on for all required functions

    // --- Helper Methods for User Interaction (calling from main's switch-case) ---
    // These methods will get input from the user specifically for each operation
    private static void performAddition(Scanner scanner) {
        // Prompt for first number
        // Prompt for second number
        // Call add() method
        // Print result
        // Implement try-catch for InputMismatchException here!
    }

    // private static void performSubtraction(Scanner scanner) { ... }
    // ... and so on for all required functions
}
```

---

4. Git and GitHub Workflow

Now, let's learn how to manage your project using **Git** and store it on **GitHub**. Git is a version control system that tracks changes in your code, and GitHub is a cloud-based hosting service for Git repositories.

Prerequisites:

- **Git Installed:** Make sure Git is installed on your computer. You can download it from git-scm.com.
- **GitHub Account:** You need an account on github.com.

Steps to get your project on GitHub:

Step 1: Initialize a Local Git Repository

1. **Create a Project Folder:** Create a new folder on your computer for your calculator project. Let's call it `ScientificCalculatorProject`.

   Bash

   ```
   mkdir ScientificCalculatorProject
   cd ScientificCalculatorProject
   ```

2. **Create your Java file:** Inside this folder, create the `ScientificCalculator.java` file. You will be writing your code into this file.
3. **Initialize Git:** Open your terminal or command prompt, navigate to your `ScientificCalculatorProject` folder, and initialize a new Git repository:

   Bash

   ```
   git init
   ```

   This command creates a hidden `.git` folder, which Git uses to track your project's history.

Step 2: Create a New Repository on GitHub

1. Go to github.com and log in.
2. Click the **"+" icon** in the top right corner (or "New" button on the left sidebar) and select **"New repository"**.
3. **Repository name:** Enter `ScientificCalculator` (or any name you prefer).
4. **Description (optional):** Add a brief description, e.g., "A command-line scientific calculator in Java."
5. **Visibility:** Choose **"Public"** (recommended for learning projects) or "Private."
6. **Do NOT initialize with a README, .gitignore, or license** for this exercise, as you'll be pushing existing files.
7. Click **"Create repository"**.
8. GitHub will then show you instructions for pushing an existing repository. Look for the section "push an existing local repository from the command line." It will look something like this:

   Bash

   ```
   git remote add origin https://github.com/YOUR_USERNAME/ScientificCalculator.git
   ```

```
git branch -M main
git push -u origin main
```

Copy these lines.

Step 3: Link Local Repository to GitHub and Push Initial Structure

1. Back in your terminal (still in `ScientificCalculatorProject` folder), paste the `git remote add origin ...` command you copied from GitHub. This links your local repository to your new remote repository on GitHub.

   Bash

   ```
   git remote add origin https://github.com/YOUR_USERNAME/ScientificCalculator.git
   ```

   (Replace `YOUR_USERNAME` with your actual GitHub username).

2. Now, write the basic structure of your `ScientificCalculator.java` file. This includes:
   - The `import java.util.Scanner;` statement.
   - The `public class ScientificCalculator { ... }` declaration.
   - The empty `public static void main(String[] args) { ... }` method.
   - The empty `public static void displayMenu() { ... }` method.
3. **Stage your Java file:** Add your `ScientificCalculator.java` file to the staging area. This tells Git which changes you want to include in your next commit.

   Bash

   ```
   git add ScientificCalculator.java
   ```

   You can also use `git add .` to stage all new and modified files in the current directory.

4. **Commit the initial structure:** Now, commit these staged changes with a descriptive message. This creates a snapshot of your project at this point.

   Bash

   ```
   git commit -m "feat: Initial project setup for ScientificCalculator"
   ```

   Good commit messages are crucial! They should be concise but informative. "feat" usually means a new feature.

5. **Push to GitHub:** Push your committed changes from your local `main` (or `master`) branch to the `main` branch on GitHub.

   Bash

   ```
   git branch -M main # Renames your default branch to 'main' (common practice)
   git push -u origin main
   ```

   The `-u` flag sets the upstream branch, so future `git push` commands will automatically know where to push.

Now, if you refresh your GitHub repository page, you should see your `ScientificCalculator.java` file! 🎉

Step 4: Commit Each Functionality Separately (The Core of the Task!)

This is the most important part of the Git workflow for this task. You will make small, atomic commits for *each* function you add or significant part of the code.

Example workflow for adding a function:

Let's say you decide to implement the `add` function and its `performAddition` helper first.

1. **Make changes:** Write the `add` and `performAddition` methods in your `ScientificCalculator.java` file. Update your `main` method's `switch` statement to include the addition option and call `performAddition`.
2. **Check status:** See what changes you've made.

   Bash

   ```
   git status
   ```

   You'll see `ScientificCalculator.java` listed under "modified files."

3. **Stage the changes:** Add the specific file (or just the relevant changes if you made multiple changes you want to separate) to the staging area.

   Bash

   ```
   git add ScientificCalculator.java
   ```

4. **Commit the changes with a meaningful message:** This commit should describe *only* what you just implemented.

   Bash

   ```
   git commit -m "feat: Implement addition functionality (add, performAddition)"
   ```

5. **Push the commit:** Send your new commit to GitHub.

   Bash

   ```
   git push origin main
   ```

Repeat this process for each major functionality:

Sample Git Commit Messages & Commands:

After your initial commit, continue developing and committing like this:

- **Implementing Subtraction:**

  Bash

```
# (Write your subtract and performSubtraction methods in ScientificCalculator.java
and update main)
git add ScientificCalculator.java
git commit -m "feat: Add subtraction functionality (subtract, performSubtraction)"
git push origin main
```

- **Implementing Multiplication:**

  Bash

  ```
  # (Write your multiply and performMultiplication methods and update main)
  git add ScientificCalculator.java
  git commit -m "feat: Implement multiplication (multiply, performMultiplication)"
  git push origin main
  ```

- **Implementing Division with Error Handling:**

  Bash

  ```
  # (Write your divide and performDivision methods, including the check for division
  by zero, and update main)
  git add ScientificCalculator.java
  git commit -m "feat: Add division functionality with zero division error handling"
  git push origin main
  ```

- **Implementing Square Root:**

  Bash

  ```
  # (Write your calculateSquareRoot and performSquareRoot methods and update main)
  git add ScientificCalculator.java
  git commit -m "feat: Implement square root function using Math.sqrt()"
  git push origin main
  ```

- **Implementing Power Function:**

  Bash

  ```
  # (Write your calculatePower and performPower methods and update main)
  git add ScientificCalculator.java
  git commit -m "feat: Add power function using Math.pow()"
  git push origin main
  ```

- **Implementing Sine (Degrees):**

  Bash

  ```
  # (Write your calculateSine and performSine methods, remember Math.toRadians!, and
  update main)
  git add ScientificCalculator.java
  git commit -m "feat: Implement sine function (degrees to radians conversion)"
  git push origin main
  ```

- **Implementing Cosine (Degrees):**

  Bash

```
# (Write your calculateCosine and performCosine methods and update main)
git add ScientificCalculator.java
git commit -m "feat: Implement cosine function (degrees to radians conversion)"
git push origin main
```

- **Implementing Tangent (Degrees):**

  Bash

  ```
  # (Write your calculateTangent and performTangent methods, including undefined
  check, and update main)
  git add ScientificCalculator.java
  git commit -m "feat: Implement tangent function with undefined angle handling"
  git push origin main
  ```

- **Implementing Natural Logarithm:**

  Bash

  ```
  # (Write your calculateNaturalLogarithm and performNaturalLogarithm methods and
  update main)
  git add ScientificCalculator.java
  git commit -m "feat: Add natural logarithm (ln) using Math.log()"
  git push origin main
  ```

- **Implementing Base 10 Logarithm:**

  Bash

  ```
  # (Write your calculateLogarithmBase10 and performLogarithmBase10 methods and
  update main)
  git add ScientificCalculator.java
  git commit -m "feat: Implement base 10 logarithm (log10) using Math.log10()"
  git push origin main
  ```

- **Implementing Absolute Value:**

  Bash

  ```
  # (Write your calculateAbsoluteValue and performAbsoluteValue methods and update
  main)
  git add ScientificCalculator.java
  git commit -m "feat: Add absolute value function using Math.abs()"
  git push origin main
  ```

- **Implementing Round, Ceiling, Floor:**

  Bash

  ```
  # (Write your roundNumber, ceilingNumber, floorNumber methods and their respective
  perform methods, and update main)
  git add ScientificCalculator.java
  git commit -m "feat: Implement round, ceil, and floor functions"
  git push origin main
  ```

- **Implementing Min and Max:**

Bash

```
# (Write your findMin, findMax methods and their respective perform methods, and
update main)
git add ScientificCalculator.java
git commit -m "feat: Add min and max functions using Math.min/max()"
git push origin main
```

- **Refactoring/Improvements (if any):**

  Bash

  ```
  # (Made some general improvements, added comments, refined input handling, etc.)
  git add ScientificCalculator.java
  git commit -m "refactor: Improve input validation and code readability"
  git push origin main
  ```

By following this commit strategy, your Git history will tell a clear story of how your calculator was built, feature by feature. This is an invaluable skill for any software engineer! Good luck! 🚀