Object-Oriented Principles for Software Development
ENSF380: Lab01

Getting Started with Java Development

Created by: Mahdi Jaberzadeh Ansari (He/Him)
Schulich School of Engineering
University of Calgary
Calgary, Canada
mahdi.ansari1@ucalgary.ca

Week 2, July 2-4, 2024

Contents

1 Introduction

1.1 Objectives

The primary goals of Lab 1 are to:

1. Navigate through the setup of a Java development environment, including installing a Long-Term Support (LTS) version of Java and the Eclipse Integrated Development Environment (IDE).
2. Understand the fundamentals of Java programming, focusing on variables, data types, arithmetic operations, and basic input/output.
3. Learn how to define and invoke functions, gaining insights into code modularization and reuse.
4. Gain hands-on experience with strings and arrays in Java, learning to manipulate collections of data and individual characters.
5. Develop skills in reading from and writing to the console to create interactive Java applications.
6. Get introduced to using Git and GitHub for version control, to start managing and sharing your code effectively.

1.2 Prerequisites

Before starting Lab 1, you should:

1. Have a basic understanding of computer operations and programming concepts.
2. Be familiar with what an Integrated Development Environment (IDE) is and its role in software development.
3. Possess an introductory level of knowledge about version control systems, specifically Git.
4. Note that no prior experience with Java or Eclipse is required, as this lab is designed to introduce you to these tools and concepts from the ground up.

1.3 Forming Groups

- In this lab session, you cannot work with a partner, as each student must prepare their development environment individually.

1.4 Before Submission

- For most of the labs, you will receive a DOCX file that you need to fill in the gaps. Make sure you have downloaded the file to fill in.
- All your work should be submitted as a single file in PDF format. For instructions about how to provide your lab reports, study the posted document on the D2L called 'How to Hand in Your Lab Assignment'.
- If you have been asked to write code, ensure the following information appears at the top of your code:
  - File Name
  - Assignment and exercise number
  - Your names in alphabetic order
  - Submission Date:

  Here is an example of a sample `Application.java` file:

```java
/**
 * <h1>Application.java</h1>
 * <p>
 * This class is designed for processing XXX in Java.
 * It is part of Lab 1 Exercise B.
 * </p>
 * <p>
 * <b>Note:</b> This file represents ...
 * </p>
 *
 * <p><b>Submission Date:</b> May 21, 2030</p>
 *
 * @author Mahdi Ansari
 * @author William Arthur Philip Louis
 * @version 1.0
 */
```

- Some exercises in this lab and future labs might not be marked. Please do not skip them, because these exercises are as important as the others in learning the course material.
- In courses like ENSF380, some students skip directly to the exercises that involve writing code, skipping sections such as 'Read This First,' or postponing the diagram-drawing until later. That is a bad idea for several reasons:
  - 'Read This First' sections normally explain some technical or syntax details that may help you solve the problem or may provide you with some hints.
  - Some lab exercises may ask you to draw a diagram, and most of the students prefer to hand-draw them. In these cases, you need to scan your diagram with a scanner or an appropriate device, such as your mobile phone, and insert the scanned picture of your diagram into your PDF file (the lab report). A possible mobile app to scan your documents is Microsoft Lens, which you can install on your mobile device for free. Please make sure your diagram is clear and readable; otherwise, you may either lose marks or it could be impossible for TAs to mark it at all.
  - Also, it is better to use the Draw.io tool if you need to draw any diagram.
  - Drawing diagrams is an important part of learning how to visualize data transfer between modules and so on. If you do diagram-drawing exercises at the last minute, you will not learn the material very well. If you do the diagrams first, you may find it easier to understand the code-writing exercises and finish them more quickly.
- **Due Dates:**
  - You must submit your solution until 11:59 p.m. on the second session that you have a specific lab. Which means until 11:59 p.m. on Thursdays. Therefore, you almost have two and a half days to work on each lab.
  - Submissions until 24 hours after the due date get a maximum of 50% of the mark, and after 24 hours, they will not be evaluated and get 0.

1.5 Academic Misconduct/Plagiarism

- Ask for help, but do not copy.
  - You can get help from your lab instructor(s), TAs, or even your classmates as long as you do not copy other people's work.
  - If we realize that even a small portion of your work is a copy from a classmate, both parties (the donor and the receiver of the work) will be subject to academic misconduct (plagiarism). Moreover, if you give exercise solutions to a friend, you are not helping them, as they will not

learn the material and will face difficulties during exams or quizzes. Therefore, do not put yourself and your friend in a position of academic misconduct.

  - You can use ChatGPT, but please note that it may provide similar answers for others too, or even the wrong answers. For example, it has been shown that AI can hallucinate, proposing the use of libraries that do not actually exist [1]. We recommend that you imagine ChatGPT as an advanced search engine, not a solution provider.
  - In order to find out who is abusing these kinds of tools, we will eventually push you toward the incorrect responses that ChatGPT might produce. In that case, you might have failed for the final mark and be reported to administration.
  - If we ask you to investigate something, do not forget to mention the source of your information. Reporting without reference can lead to a zero mark, even if you provide a correct answer.

1.6 Marking Scheme

- Do not submit anything for exercises that are not marked.
- In Table 1, you can find the marking scheme for each exercise.

Table 1: Marking scheme

Exercise	Marks
A	20 marks
B	20 marks
C	20 marks
D	20 marks
E	20 marks
Total	100 marks

1.7 Complaints

- Your grades will be posted until 11:59 a.m. on the following Tuesdays, which means they will be accessible at the subsequent lab meeting.
- Normally, the grades for individual labs are assessed by a distinct TA for each lab and section. Kindly refrain from contacting all TAs. If you have any concerns regarding your grades, please direct an email to the TA responsible for that specific lab.

[1] https://perma.cc/UQS5-3BBP

2 Exercise A (Preparing Your Java Development Environment)

2.1 Objectives

In this section, you will:

1. Set up your Java development environment by installing Eclipse and the Java Development Kit (JDK).
2. Learn the basics of Java programming by writing and running a "Hello, World!" program in Eclipse.
3. Discover why Eclipse stands out as a preferred IDE for Java development and explore its potential beyond basic Java programming, including integration with frameworks like Talend for data integration and the ability to develop custom plugins or even your own IDE.

2.2 Prerequisites

1. A computer with internet access.
2. A basic understanding of working with file systems and operating systems.

2.3 Read This First

In this course, we use only Eclipse IDE, and we expect you to use it as part of your learning path. I know that most of you would like to use Visual Studio Code; however, **learning something new and mentioning it in your resume is not harmful**.

2.3.1 Why Eclipse?

Eclipse is a versatile and powerful IDE tailored for Java development, offering an array of features that cater to both beginners and seasoned developers. Its integrated development environment provides comprehensive tools for coding, debugging, and testing Java applications. Unlike other IDEs, Eclipse's plug-in ecosystem allows for extensive customization and expansion, enabling developers to adapt the IDE to their specific needs, whether they are working on Java applications, developing with other programming languages, or even creating their own development tools.

Eclipse stands out for several reasons:

- **Extensive Plugin Ecosystem:** Eclipse's marketplace offers a wide variety of plugins, supporting development in not only Java but also other languages and frameworks.
- **Rich Java Development Tools:** It offers comprehensive tools for Java development, including a powerful code editor, debugger, and testing tools.
- **Versatility:** Beyond Java, Eclipse can be used for developing applications in other programming languages with the appropriate plugins.
- **Customization and Extensibility:** Eclipse allows for the development of custom plugins, enabling you to extend its functionality or even develop your own IDE tailored to specific needs.

2.3.2 Beyond Java Development

Learning Eclipse opens doors to various advanced software development tasks:

- **Talend Integration:** Eclipse serves as the foundation for Talend Open Studio, a powerful tool for data integration and ETL processes.
- **Plugin Development:** You can create plugins to extend Eclipse's functionality, tailoring it to specific project requirements or even developing tools for other developers to use.

- **Custom IDE Creation:** With deep knowledge of Eclipse and its plugin architecture, you can develop specialized IDEs for specific programming languages or frameworks.

By mastering Eclipse, you not only become proficient in Java development but also gain the skills to venture into complex projects, integrate with diverse technologies, and customize your development environment to suit any need.

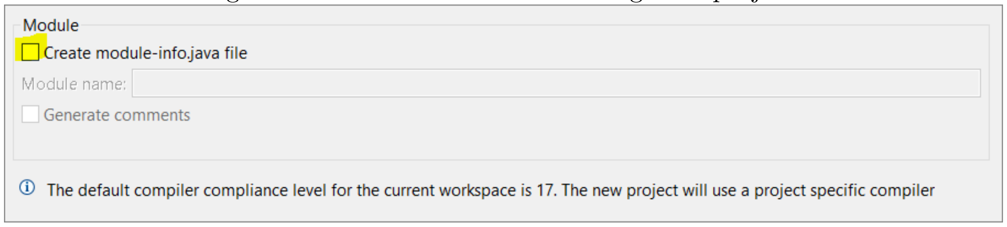
2.4 Setting Up Eclipse and JDK

1. **Download JDK:**
   - Visit the official Oracle JDK website or OpenJDK to download the latest LTS version of JDK.
   - Install JDK by following the on-screen instructions.
2. **Download Eclipse:**
   - Go to the Eclipse Downloads page.
   - Download the latest Eclipse IDE for Java Developers package suitable for your OS.
3. **Install Eclipse:**
   - Run the downloaded installer and follow the prompts to install Eclipse.
   - Launch Eclipse and set your workspace directory when prompted.

2.5 Writing and Running "Hello, World!" in Eclipse

1. **Create a New Java Project:**
   - Open Eclipse.
   - Go to `File` > `New` > `Project...`.
   - In the "`Select a wizard`" dialog, select the **Java** > **Java Project**, to create a Java project.
   - Make sure you uncheck the "`Create module-info.java file`" checkbox; otherwise, you will not be able to run the project as a Java application. **We will not repeat this, it is expected that you do this step for all projects in this course.**

Figure 1: Critical checkbox in making Java projects



Module
☐ Create module-info.java file
Module name:
☐ Generate comments

The default compiler compliance level for the current workspace is 17. The new project will use a project specific compiler

   - Name your project (e.g., `Project01` or `Lab01`) and click **Finish**.
2. **Create a New Java Class:**
   - Right-click on the src folder of your new project in the Project Explorer.
   - Select **New** > **Class**.
   - Name your class `Application` and check the option to include the `public static void main(String[] args)` method.
   - Click **Finish**. It will create a `Application.java` file in your src folder.

3. **Write the "Hello, World!" Code:**

   - In the `Application` class file, inside the `main` method, add the following line:

   ```
   System.out.println("Hello, World!");
   ```

4. **Run Your Program:**

   - Click the run button (green circle with a white arrow) on the toolbar.
   - The console should output `Hello, World!`.

2.6 Deliverable

1. Take a screenshot of your Eclipse IDE and its terminal showing the `Hello, World!` message and add it to your answer sheet file.

3 Exercise B (Git and GitHub)

3.1 Introduction to Git

Read This First: Git is a distributed version control system that allows you to track changes in your code over time. Unlike centralized version control systems, Git gives every developer a local copy of the entire development history, and changes are copied from one repository to another. These data models allow Git to manage a project's history as a series of snapshots.

3.2 Getting Started with Git

3.2.1 Download and Install Git

- Visit the official Git website and download the version for your operating system.
- Follow the installation instructions in the installer window.

3.2.2 Set Up Git

- Open your terminal or command prompt.
- Configure your user name and email with the 'git config' command if you know how, and jump to section 3.3 or follow the following sub-section.

3.2.3 Configuring Git

Open your CLI and follow these steps:

1. **Set Your Username:**

   ```
   git config --global user.name "Your Name"
   ```

   Replace `"Your Name"` with your actual name. This name will be attached to your commits and tags.

2. **Set Your Email Address:**

   ```
   git config --global user.email "your_email@example.com"
   ```

   Replace `"your_email@example.com"` with your email address. Use the same email address associated with your GitHub, GitLab, Bitbucket, or other Git hosting service accounts if you plan to push repositories to these services.

3.2.4 Checking Your Configuration

- To check if your configuration was successful, use:

  ```
  git config --list
  ```

- This command will list all Git configurations, and you should see your `user.name` and `user.email` settings listed.

3.2.5 Scope of Configuration

- The `--global` flag in the commands sets your username and email for all repositories on your system under your user account.
- If you need to set a different username or email for a specific repository, navigate to that repository's directory in your CLI and run the same commands without the `--global` flag. This sets the configuration only for that particular repository.

3.2.6 Why It's Important

- These configurations are critical because every Git commit uses this information, and it's important for accountability and collaboration in team environments.
- Commits you make to repositories hosted on services like GitHub, GitLab, or Bitbucket, will show up under your name and email as specified in these configurations.

Remember, if you're contributing to open-source projects or using public repositories, the username and email you configure here will be publicly visible on those platforms.

3.3 Understanding Git Concepts

3.3.1 Git Snapshots

Git creates 'snapshots' of the repository at each commit. Unlike other systems that track differences in files, Git records the entire content of all files.

Figure 2: Subversion

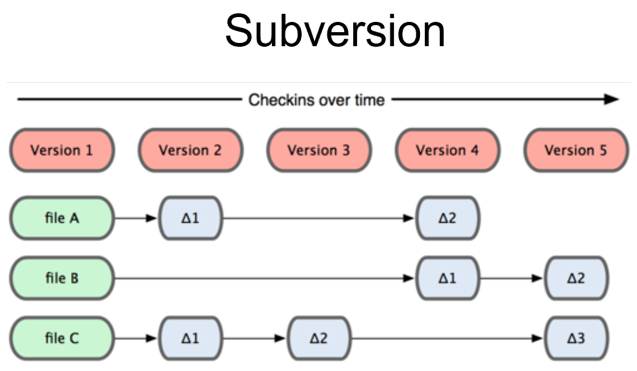
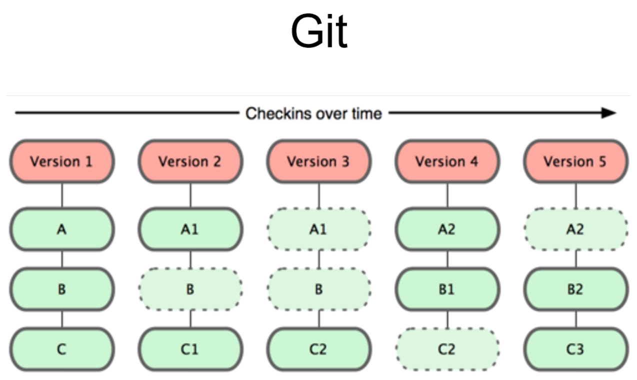


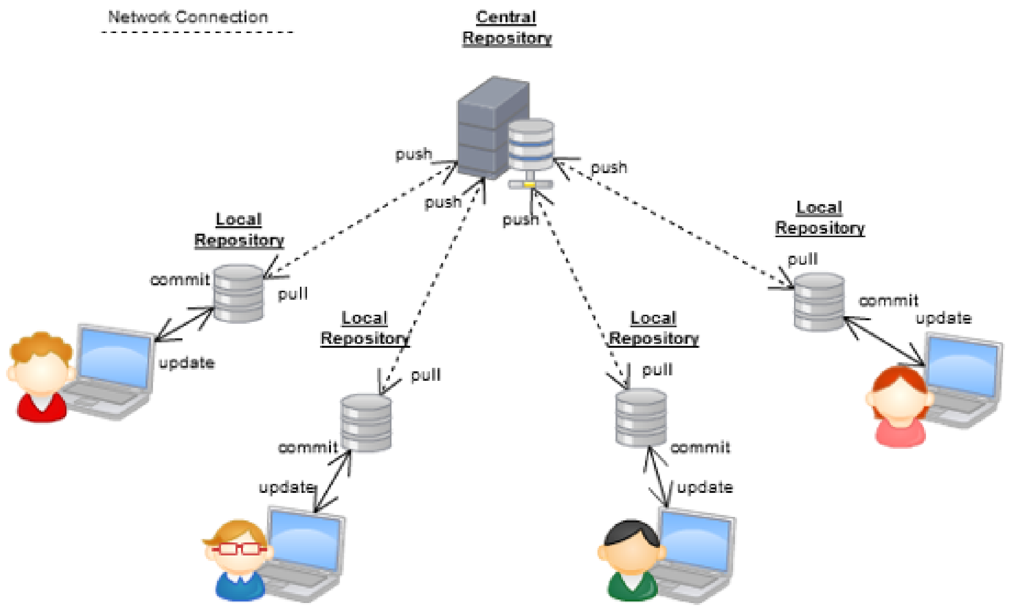
Figure 3: Git



3.3.2 Local vs. Remote Repositories

Local repositories are on your machine, allowing you to work offline. Remote repositories are hosted on servers like GitHub for collaboration. The schema for local and remote repositories is shown in figure 4.

Figure 4: Having remote Git (central repository)



3.3.3 Branches in Git

Branches are used to develop features isolated from each other. The 'main' branch is the default branch in most cases.

3.3.4 Transition from 'master' to 'main'

The Git community has moved from using 'master' to 'main' as the default branch name for new repositories, reflecting a more inclusive language.

3.4 Using Git for a Project

3.4.1 Creating a New Repository

- Open a terminal or a command line.
- Navigate into the directory that you made in section 1, and run 'git init'.

3.4.2 Making Changes

- Make some changes in your `Application.java` file (i.e., change the message, use your first name instead of the word 'world'), use 'git add' to stage it, and use 'git commit' to commit.

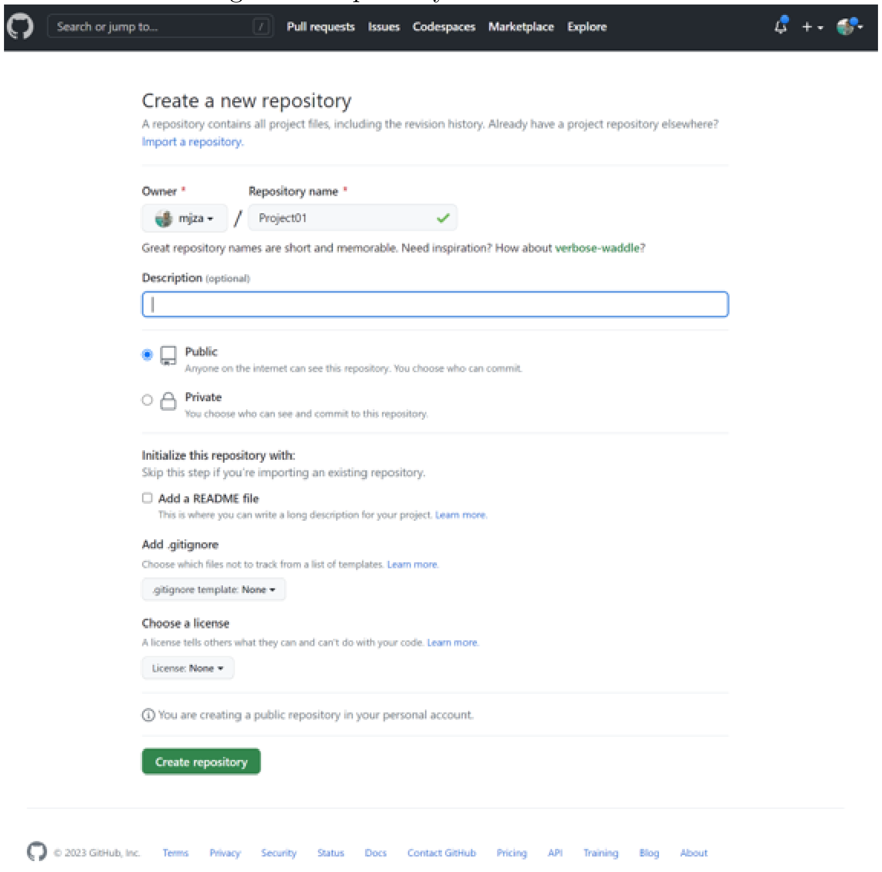
3.4.3 Branching with Git

- Create, switch, and merge branches with 'git branch', 'git checkout', and 'git merge'.

3.5 Collaborating with GitHub

- Set up a GitHub account and create an empty remote repository, like what I did in Figure 5.
- Figure 6 shows a sample repository before pushing to it. In such a situation, it shows the commands that you need to use for pushing and the URL of your brand-new repository.
- Set the current **local** branch to main (in terminal).

Figure 5: Repository creation in GitHub


Search or jump to... Pull requests Issues Codespaces Marketplace Explore

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * mjza / Repository name * Project01

Great repository names are short and memorable. Need inspiration? How about verbose-waddle?

Description (optional)

● Public — Anyone on the internet can see this repository. You choose who can commit.

○ Private — You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ Add a README file — This is where you can write a long description for your project. Learn more.

Add .gitignore — Choose which files not to track from a list of templates. Learn more. .gitignore template: None

Choose a license — A license tells others what they can and can't do with your code. Learn more. License: None

ⓘ You are creating a public repository in your personal account.

Create repository

© 2023 GitHub, Inc. Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About

```
git branch -M main
```

- Add the remote repository.

```
git remote add origin https://url...
```

- Push the latest changes to the remote repository.

```
git push -u origin main
```

- Figure 7 shows a sample repository after pushing to it. If you refresh your repository after pushing, you must be able to see the 'Application.java' file in it.

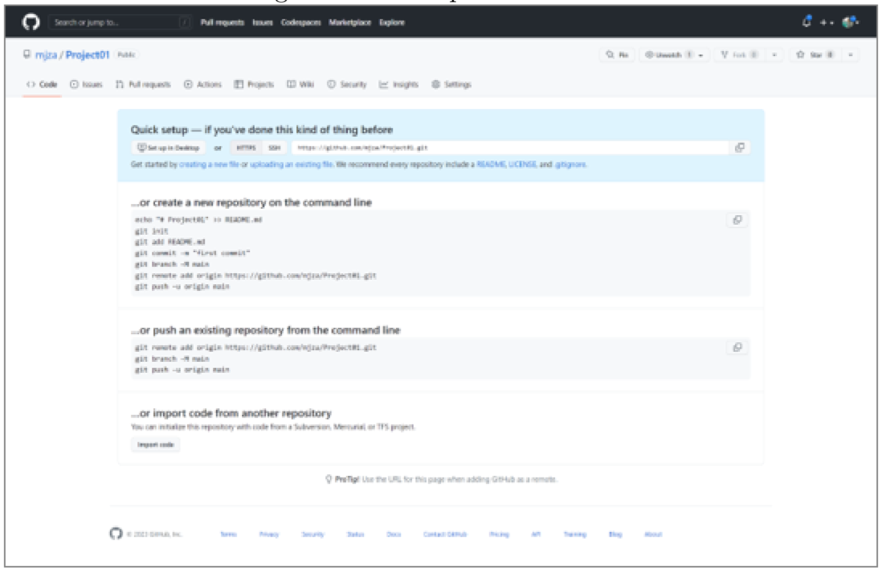
3.6 Resources for Further Learning

- Refer to the Pro Git Book, GitHub Learning Lab.
- Also, Learn Git Branching is a good tutorial for learning Git.

3.7 Basic Git Commands

You need to learn these commands by heart: **git init**, **git add** [**file**], **git commit**, **git clone** [**url**], **git status**, **git push**, **git pull**.

Figure 6: Before push in GitHub



mjza / Project01 Public

Quick setup — if you've done this kind of thing before

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# Project01" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/mjza/Project01.git
git push -u origin main
```

...or push an existing repository from the command line

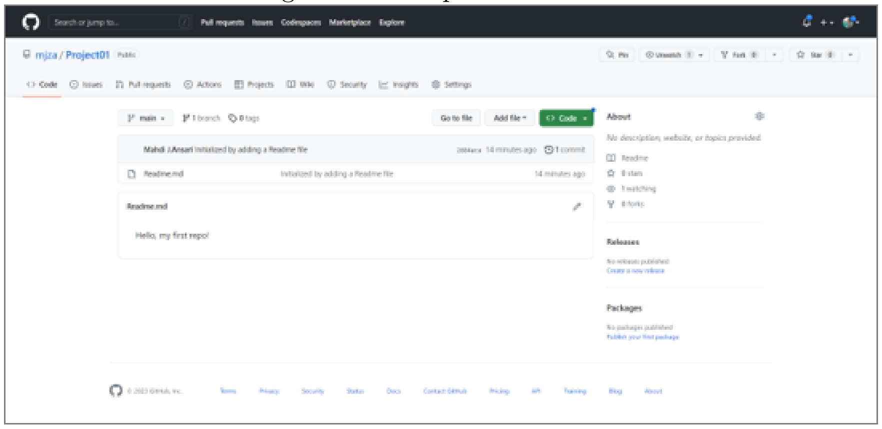
```
git remote add origin https://github.com/mjza/Project01.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

ProTip! Use the URL for this page when adding GitHub as a remote.

Figure 7: After push in GitHub



mjza / Project01 Public

Mahdi J.Ansari Initialized by adding a Readme file · 14 minutes ago · 1 commit

Readme.md · Initialized by adding a Readme file · 14 minutes ago

Readme.md

Hello, my first repo!

About

No description, website, or topics provided.

3.8 Deliverables

1. Take a screenshot from the GitHub repository that you created. It must clearly show the URL of your repository.

2. Copy and paste the URL of your repository in the answer sheet file, and make sure your repository is public.

4 Exercise C (Building a CLI Calculator)

4.1 Objectives

In this section, you will enhance your understanding of Java fundamentals by building a command-line interface (CLI) calculator.

4.2 Starting Point

Assume you have already created a project in Eclipse named `Project1` or `Lab1`, containing an `Application.java` file. This file will be our working space.

4.3 Requirements

- Functions: Define methods for various operations (addition, subtraction, etc.).
- Loops: Use loops to handle multiple operations.
- If/Else: Use conditional statements to determine the operation to perform.
- Recursion: Implement at least one operation using recursion.
- Packages: Organize your code using packages.
- Console I/O: Read user input from the console and display the output.
- Math Class: Utilize Java's Math class for operations like square root, exponents, etc.
- Comments: Document your code with meaningful comments.
- CLI Arguments: Accept arguments from the command line or Eclipse's run configurations.

4.4 Step-by-Step Guide

1. **Setup Your Project Structure**
   - In `Project1` or `Lab1`, ensure your `Application.java` file is ready. You might place it in a package, e.g., `com.yourname.calculator`, for better organization.
2. **Skeleton of `Application.java`**

```java
package com.yourname.calculator;

public class Application {
    public static void main(String[] args) {
        // TODO: Parse CLI arguments or wait for user input
    }

    // TODO: Define methods for calculations
}
```

3. **Parsing Command-Line Arguments**
   - Modify the `main` method to accept CLI arguments for operations. If no arguments are provided, prompt the user for input.

```java
import java.util.Scanner;

public class Application {
    public static void main(String[] args) {
        if (args.length > 0) {
            // Process CLI arguments
        } else {
            // No CLI arguments, ask for user input
            Scanner scanner = new Scanner(System.in);
            System.out.println("Enter operation (e.g., add, subtract,
                multiply, divide):");
            String operation = scanner.next();
            // Further processing based on operation
        }
    }

    // Other Methods' placeholders
}
```

4. **Defining Operations**

   - Implement methods for basic operations. Here, we show addition and a recursive factorial as examples. Add similar methods for subtraction, multiplication, division.

```java
public static double add(double a, double b) {
    return a + b;
}

public static double factorial(double n) {
    if (n <= 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

5. **Implementing Loops and Conditionals**

   - Utilize loops and conditionals within your `main` method or in specific operations to perform calculations based on user input and CLI arguments. Example for handling an operation:

```java
// Inside main or another method after getting operation
switch (operation.toLowerCase()) {
    case "add":
        System.out.println("Enter the first operand:");
        double num1 = scanner.nextDouble();
        System.out.println("Enter the second operand:");
        double num2 = scanner.nextDouble();
        System.out.println("Result: " + add(num1, num2));
        break;
    case "factorial":
        System.out.println("Enter a number:");
        double number = scanner.nextDouble();
        System.out.println("Result: " + factorial(number));
        break;
    // Handle other operations
}
```

6. **Reading and Writing to the Console**
   - You have already seen how to write to the console and read from it using `Scanner`. Continue using these techniques to interact with the user.
7. **Commenting Your Code**
   - Include comments explaining the purpose of each method, what parameters it accepts, and what it returns. Also, comment on significant blocks of code within your methods to describe what they do.
8. **Running Your Program with CLI Arguments in Eclipse**
   - To run your program with command-line arguments in Eclipse: Right-click on the `Application.java` file > Run As > Run Configurations > Arguments tab. Enter your arguments, apply, and run.
   - Test your application and make sure all its functionalities work as expected.

4.5 Deliverable

1. Commit and push your final code to your GitHub repository with this command 'End of Section C'.
2. Take some screenshots of your application for these operations:
   - 5 + 8
   - 9 - 5
   - 2 x 7
   - 8 / 0
   - 5!

   Add your screenshots to your answer sheet file. Your screenshots must show your application works as expected with proper messages and warnings.

5 Exercise D (Extending the Calculator)

In this section, we are going to extend your CLI calculator to support:

1. Scientific operations using the `Math` class:
    - Exponentiation (`pow`)
    - Square root (`sqrt`)
    - Logarithmic (`log` for natural log and `log10` for base-10 log)
    - Trigonometric functions (`sin`, `cos`, `tan`)

5.1 Setup Your Project

Ensure your `Application.java` is ready in `Project1` or `Lab1`. This file will contain all your calculator logic, and we are going to extend our current code.

5.2 Extending Factorial with Progress Bar

Factorial calculation can be implemented recursively. However, as it is a tail recursion, we can get rid of the recursion as well. If you are interested, you can eliminate the recursion by using a loop. For the progress bar, update the `factorial` function with the progress percentage.

```java
// Factorial calculation with progress display
public static long factorial(int num) {
    if (num < 0) {
        System.out.println("Factorial of negative number is undefined.");
        return 0;
    }
    return factorialHelper(num, num);
}

private static long factorialHelper(int originalNum, int num) {
    if (num <= 1) {
        return 1;
    }
    // Calculate progress and update progress bar
    int progress = (int) (((originalNum - num) / (double) originalNum) * 100);
    System.out.print("\rCalculating factorial: " + progress + "%");
    return num * factorialHelper(originalNum, num - 1);
}
```

Note:

It seems that in the above code (i.e., `int progress = (int) (((originalNum - num) / (double) originalNum) * 100);`), the progress bar does not cover the number 1. Can you identify why this might be happening and suggest a way to ensure that the number 1 is included in the progress calculation? Your solution for this issue will be checked in your submitted code in your repository.

5.3 Implement Scientific Operations

Use Java's `Math` class to implement scientific operations. Each method should be straightforward, taking input(s) and returning the Math class method's result.

```java
// Exponentiation
public static double power(double base, double exponent) {
    return Math.pow(base, exponent);
}

// Square root
public static double sqrt(double number) {
    return Math.sqrt(number);
}

// Natural logarithm
public static double log(double number) {
    return Math.log(number);
}

// Base-10 logarithm
public static double log10(double number) {
    return Math.log10(number);
}

// Sine function
public static double sin(double angleRadians) {
    return Math.sin(angleRadians);
}

// Cosine function
public static double cos(double angleRadians) {
    return Math.cos(angleRadians);
}

// Tangent function
public static double tan(double angleRadians) {
    return Math.tan(angleRadians);
}
```

Note:

Please note that the trigonometric functions expect argument in radians. Thus, you need to convert degrees using `Math.toRadians(angleDegrees)`.

5.4 Read Input and Perform Calculations

Implement a method to read input from the console and determine which operation to perform based on user input. Use a loop to keep the program running until the user decides to exit. Here is a sample code:

```java
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    while (true) {
        System.out.println("\nEnter operation (add, subtract, multiply, divide
            , pow, sqrt, log, log10, sin, cos, tan, factorial) or 'exit' to
            quit:");
        String operation = scanner.next();

        if (operation.equalsIgnoreCase("exit")) {
            System.out.println("Exiting calculator...");
            break;
        }

        // For operations requiring two inputs
        if (!operation.equalsIgnoreCase("sqrt") && !operation.equalsIgnoreCase("log") && !operation.equalsIgnoreCase("log10") && !operation.equalsIgnoreCase("sin") && !operation.equalsIgnoreCase("cos") && !operation.equalsIgnoreCase("tan") && !operation.equalsIgnoreCase("factorial")) {
            System.out.print("Enter first number: ");
            double num1 = scanner.nextDouble();
            System.out.print("Enter second number: ");
            double num2 = scanner.nextDouble();

            switch (operation.toLowerCase()) {
                case "add":
                    System.out.println("Result: " + add(num1, num2));
                    break;
                case "subtract":
                    System.out.println("Result: " + subtract(num1, num2));
                    break;
                case "multiply":
                    System.out.println("Result: " + multiply(num1, num2));
                    break;
                case "divide":
                    System.out.println("Result: " + divide(num1, num2));
                    break;
                case "pow":
                    System.out.println("Result: " + power(num1, num2));
                    break;
                default:
                    System.out.println("Invalid operation.");
                    break;
            }
        } else {
            // For operations requiring one input
            System.out.print("Enter number: ");
            double num = scanner.nextDouble();

            switch (operation.toLowerCase()) {
                case "sqrt":
                    System.out.println("Result: " + sqrt(num));
                    break;
                case "log":
                    System.out.println("Result: " + log(num));
                    break;
                case "log10":
                    System.out.println("Result: " + log10(num));
                    break;
                case "sin":
                    System.out.println("Result: " + sin(num));
                    break;
                case "cos":
                    System.out.println("Result: " + cos(num));
                    break;
                case "tan":
                    System.out.println("Result: " + tan(num));
                    break;
                case "factorial":
                    // Factorial is a special case requiring an integer
                    System.out.println("Result: " + factorial((int)num));
                    break;
                default:
                    System.out.println("Invalid operation.");
                    break;
            }
        }
    }

    scanner.close();
}
```

5.5 Deliverable

1. Commit and push your final code to your GitHub repository with this command ‘End of section D’.
2. Take some screenshots of your application for these operations:
   - 5 + 8
   - 10!
   - sin(90)
   - cos(45)

   Add your screenshots to your answer sheet file. Your screenshots must show your application works as expected with proper messages and warnings.

6 Exercise E (Add Permutations to the Calculator)

In this section, we are going to extend your CLI calculator to support permutation, as follows:

1. Create a recursive method which calculates the number of possible permutations without replacement. Permutations are calculated for the values contained in an array.
2. The method should take two variables, the total number of elements in the array and the number of items to be selected.
3. Include error warning if the number of items to be selected is negative or greater than 100. We assume that our array can have a maximum of 100 items. Therefore, you must:
   (a) ensure that the total number of elements is not less than zero.
   (b) ensure that the number of selected items does not exceed the total number of elements in the array.
4. Can you solve this problem without recursion? If yes, provide a second version of your solution.
5. Add a suitable menu to your calculator for calculating permutations of 2 numbers.

6.1 Recursive Algorithm

- To calculate permutations without replacement, multiply the total number of items by the total number of items minus one, and repeat the process until the specified number of selection is reached.
  - Example: If there are 5 items, and we want to select 4 of them, the number of possible permutations is 5 * 4 * 3 * 2
  - Example: If there are 6 items, and we want to select 3 of them, the number of possible permutations is 6 * 5 * 4

6.2 Deliverable

1. Commit and push your final code to your GitHub repository with this command 'End of section E'.
2. Take some screenshots of your application for these operations:
   - permutations(4, 4);
   - permutations(200, 4);
   - permutations(3, 4);
   - permutations(5, 4);
   - permutations(0, 4);
   - permutations(-10, 4);

   Add your screenshots to your answer sheet file. Your screenshots must show your application works as expected with proper messages and warnings.

6.2.1 Conclusion

This lab introduces you to fundamental programming concepts in Java, such as functions, loops, conditional statements, recursion, and working with the console. By completing this project, you will gain practical experience with these concepts and the `Math` class for scientific calculations. You have also learned how to handle user input and execute different operations based on that input, including displaying a progress bar for longer-running tasks.

Remember, practice is key to mastering programming concepts. Experiment with adding new features, improving the user interface, or optimizing your code's performance to deepen your understanding.

7 The End!

Congratulations! You have finished the first lab. Don't forget to save your DOCX file as a PDF file and only submit the PDF file via D2L.