

CS 46B
Fall 2023
Lab 12¹
10 points



In this lab, you will build a tree structure in Java to represent the family tree of the Baggins family of hobbits. You will also learn how to use git and Github to collaborate on programming tasks. Familiarity with git and GitHub will be very helpful in future CS classes and when looking for internships and jobs. Applications will ask for the link to your Github repository, where you can showcase your programming talents and the different projects you've worked on. Think of it as an extension of your LinkedIn profile specifically for computer scientists. ***Please make sure to carefully read each part of the lab before you start programming.***

Learning Outcomes

During this lab you will learn how to do the following.

- Learn how to use git commands to backup your code
- Create a Github account and push to a remote repository
- Collaborate with your group mates to jointly develop code
- Familiarize yourself with different methods on trees

¹ Modified from material provided by Dr. Philip Heller and Dr. Cay Horstmann and Dr. Chakarov

Lab Grade

Labs are a core component of the course, and a lot of your learning happens when you have to take what you learned in lecture and apply it in practice. Labs are a required part of the course, and **missing more than two labs will result in failing the course**. To receive the grades for the missing labs, you can complete two lab activities out of the lab and take the exit interviews during your lab instructor's office hours and receive the grades for them (only twice).

Your lab grade will be based on two components:

- Collaboration: 4 points
- Exit Interview: 6 points.

Working in groups to solve problems is an important skill that computer scientists embrace. It is important to not leave group members behind or to each just do the work independently. Collaboration will be assessed on a 4-point scale using the following rubric.

4	3	2	1
<ul style="list-style-type: none">● Group members discuss the problems and work at a similar pace.● No one is left behind or significantly ahead.● Group members make a plan for how they will solve the problem before diving into the code● Group members help each other debug their programs before seeking help from the lab instructor or learning assistants.	<ul style="list-style-type: none">● Group members discuss once one of them has a problem● Group members are all at about the same place in the lab● Group members don't make a joint plan to solve the problem	<ul style="list-style-type: none">● One group member is left behind or is significantly ahead of the other two● Group members look to the lab instructors and learning assistants before trying to help each other● Little discussion occurs between the group members● Group members don't make a joint plan to solve the problem	<ul style="list-style-type: none">● Group members do not discuss the problems● Each group member works independently● Group members do not help each other

The exit interview will be assessed on a 6 point scale using the following rubric.

6	4	2
<ul style="list-style-type: none">* Group has read and discussed the exit interview questions* Group has completed all of the lab	<ul style="list-style-type: none">* Group has completed the majority of the lab* Group is prepared to answer most of the interview questions.	<ul style="list-style-type: none">* Group did not discuss exit interview questions beforehand and is not prepared to answer them* Group missed portions of the lab

Exit Interview

To receive credit for this lab, your group will complete an exit interview. To get an idea of the kinds of questions that will be asked, look at the questions **highlighted in blue** that you encounter as you complete the lab instructions. ***To help you prepare for the exit interview, I suggest tackling the questions when you encounter them in the lab instruction, discussing them as a group, and then writing down what you think the answer is.***

Setup

Create a Java project in an Eclipse workspace, called lab12. Create a package called trees. Load the 2 starter files (FamilyTree.java and TreeException.java) into your package.

Overview of Family Trees

Open BagginsFamilyTree.txt file and look at its contents. The BagginsFamilyTree.txt file describes family relationships. Each line consists of a name, followed by a colon, followed by a list of names. The first name is the parent. The following names are the parent's children. There is no whitespace. Example: Mungo Baggins had 5 children: Bungo, Belba, Longo, Linda, and Bingo. This is represented in the file by:

```
Mungo:Bungo,Belba,Longo,Linda,Bingo
```

Setting up Git

Use the terminal if you are on a Mac or PowerShell if you are on a Windows machine. Type `git --version` if this command works, then you are good to go. If this command doesn't work and you are on a Mac you will be prompted to install git through XCode. If you are on a Windows machine you can install git [here](#)

1. Tell git who you are. Run these commands: Replacing "Your name" with your name and `username@mailserver.com` with your email address. If you already have a github account, use the same email address you use on Github. I recommend using a personal email address since you will be creating a Github account today.

```
git config --global user.name "Your name"
git config --global user.email "username@mailserver.com"
```

2. Make a public SSH key. First check whether you already have one.

```
ls ~/.ssh
```

- a. If you get a directory listing showing (among others) a file `id_rsa.pub`, then skip the key generation (step b)

- b. Otherwise, run

```
ssh-keygen -t rsa -C "Your name <username@mailserver.com>"
```

- i. If it asks (Enter file in which to save the key(/Users/path): enter the path you see in the parentheses
- ii. When prompted for a passphrase, just hit Enter to skip this.
- iii. *Do not* change the name of the key that is being generated. It should be `id_rsa`.

3. Type

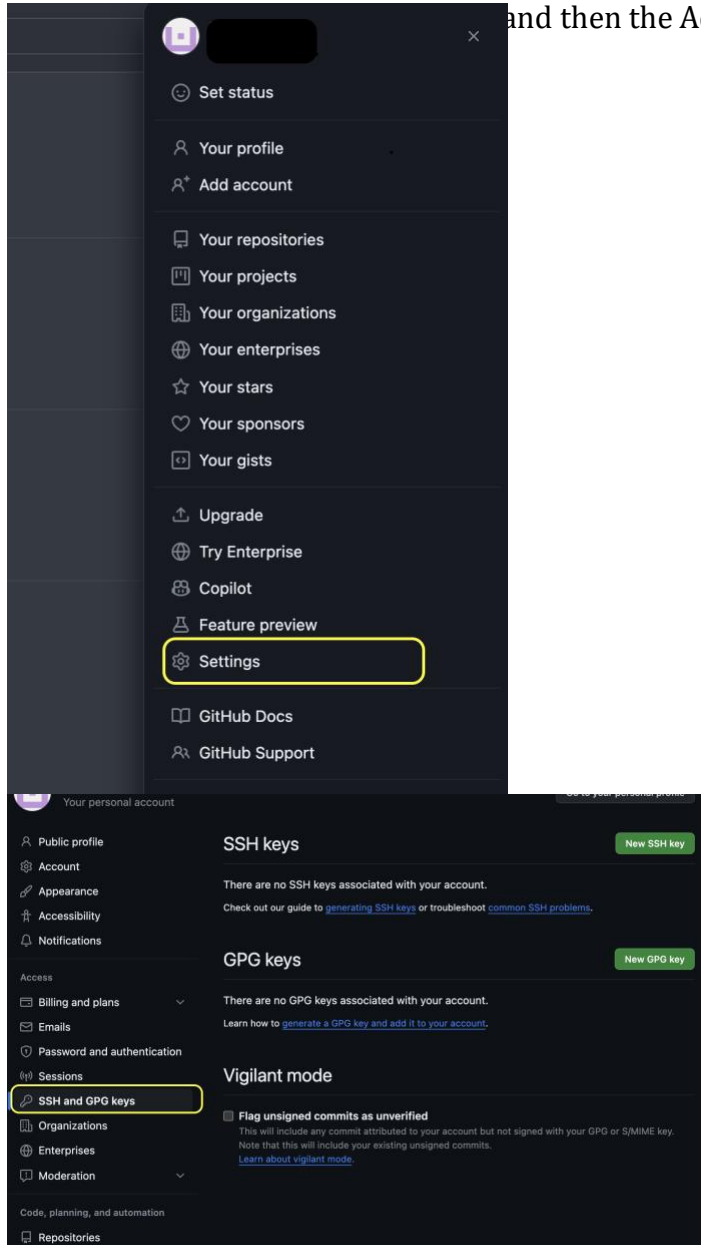
```
cat ~/.ssh/id_rsa.pub
```

This key is needed to allow you access to the git server. (Later, you will copy the result and add it to your GitHub account)

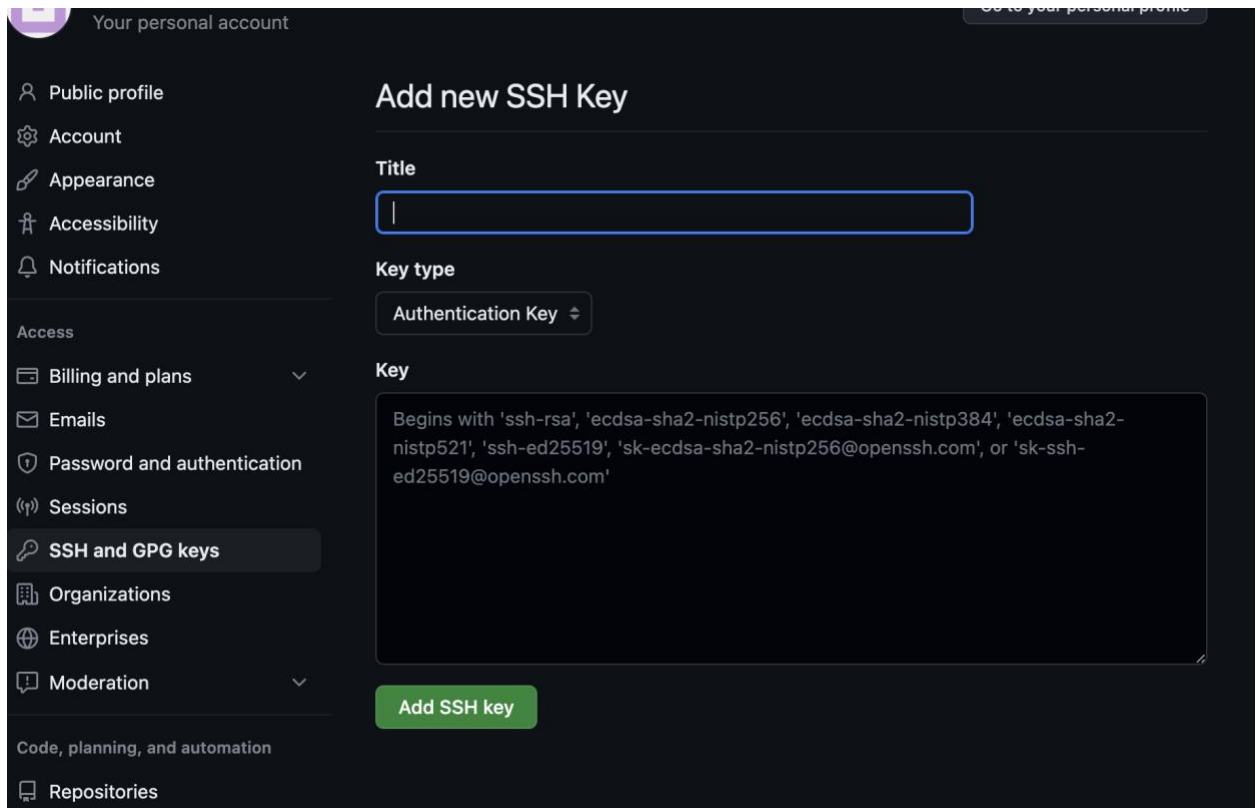
4. Create an account on <https://github.com/>

- a. I recommend using your personal email since you will probably want to keep this account after you graduate from SJSU.

5. and then the Add SSH key button.



6. Give it some title (e.g. my-laptop-2022), and then paste what you get from running



7. Create a new repository called `lab12`. Leave it public for now--you can junk it after you are done with this lab.

Let's Program

You will complete the starter code so that your app can read the file, build a tree structure, and print the tree. The `main()` method in `FamilyTree` searches the tree for the most recent common ancestor of Bilbo and Frodo. ***It is up to you to figure out how to test your code.*** Your most powerful tools are the debugger, Logging, `println()` statements. Use them! If your program does something wrong and you can't figure out why, you need more information. The debugger is a great way to help you explore how the tree changes over time.

Part 1: Setting up your git repository

A version control system remembers the changes that you made to the files in a project. This is useful (even essential) for two reasons:

- You can go undo changes that turned out not to be so good
- You can collaborate with others, merging the changes that you made.

The database that contains the changes to a project is called a *repository*. When you interact with a version control system, you contribute your changes to a repository, fetch the latest changes that are contributed by others, or occasionally undo some changes.

Many version control systems are in common use. In this lab, you will learn about a version control system called git which is used in many projects.

Open a terminal window and use the `cd` command to change to the directory containing your lab12 project [That is, the directory containing the `src` and `bin` subdirectories.] Using Eclipse, create a file called `.gitignore` in the lab12 project folder (**Note:** that the file has a ``.`` at the beginning of the filename). Right-click on the folder name and click new File. Save the file after updating the contents.

```
bin
.settings
```

Now follow the below steps.

1. Run `git init`
2. Run `git status` **What output do you get?**
3. The `git` command interacts with a git repository.
The `git init` command *initializes* a repository. You issue it once when you start a project, or when you decide to add version control to an existing project.
The `git status` command shows the current status of the files in your project. At the beginning, you are told that `.gitignore` and the `src` directory are untracked.
4. Add them: `git add .gitignore src`
5. Now run `git status` again. **What output do you get?**
6. As you just saw, all files in the `src` subdirectories that are not excluded in the `.gitignore` file are added.

Git Rule #1: Add any files that you create to the tracking set with `git add`.

7. Just adding the files to the tracking set does not add them to the repository. That only happens when you make a “commit” to the repository. Do it now:
`git commit -a -m "initial checkin"`. **What output do you get?**
8. The `-a` option means to add changes to the repository. (Don't confuse `-a` with `add`, which means to add files to the tracking set.) It is not useful to run `git commit`

without `-a`.

The `-m` option lets you put a brief message describing the reason for the commit. (You usually commit after adding a feature or fixing a bug.) A message is required.

Git Rule # 2: Always use `-a -m` with `git commit`.

9. Now run `git status` again. What output do you get?
10. When `git status` has nothing to report, then all files are up-to-date.

TreeException.java

Complete this simple class. Replace ??? with the appropriate superclass name. Inside the constructor, add a line of code to pass the argument to the superclass constructor.

Store out results in our git repository.

1. Run `git status` again. What output do you get?
2. Run `git commit` without a `-a`. What happens?
3. Now run `git commit -a -m "..."`
Now the repository contains both the original version and your latest changes.

Now let's add them to our Github repository.

1. Try `git remote add origin git@github.com:githubusername/projectname.git`
 - a. **If that doesn't work** try running the following two commands
 - i. `git branch -M main`
 - ii. `git remote add origin git@github.com:githubusername/projectname.git`
 - b. **If this doesn't work**, try...
 - i. `git remote add origin https://github.com/username/projectname.git`
2. Run `git push -u origin main`
 - a. The `-u` flag sets the remote origin as the default. This lets you later easily just do `git push` and `git pull` without having to specify an origin since we always want GitHub in this case.
 - b. If you received this message, answer yes to it:
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
3. Now, go to your repository page on GitHub. What happened? If you don't see any changes, make sure you refresh the page.

STOP

Make sure everyone in your group has successfully implemented the contains method before moving on to Part 2.

Part 2: TreeNode class in FamilyTree.java

Complete this nested class TreeNode by completing the methods addChild, getNodeWithName, and collectAncestorsToList. Comments in the methods tell you what to do.

After you complete the methods run the following commands to add the updated files to your repository. Run the following commands.

1. `git pull`
 - a. This command makes sure your local repository is up to date.

Git Rule #3: Always pull before you push.

2. `git commit -a -m "updated TreeNode class"`
3. `git push`
4. Now go to your repository page on Github. **What happened?** If you don't see any changes, make sure you refresh the page.

STOP

Make sure everyone in your group has successfully implemented the contains method before moving on to Part 3.

Part 3: FamilyTree.java

Now it's time to see how Github can be used for collaborative coding. You will import one of your partner's lab 14 repositories and complete the methods in their code. They will do the same for you. Follow the instructions in the [Appendix](#).

Now complete this FamilyTree class in the lab12-partner project by finishing the constructor and methods addLine and getMostRecentCommonAncestor. Comments in the methods tell you what to do. When you're done, run this class as an app. Look at the printout of the family tree, and find Bilbo's and Frodo's most recent common ancestor. Did main() find the right ancestor?

STOP

Make sure everyone in your group has successfully implemented the append method before moving on to Part 4.

Part 4: Updating your changes to Github.

Open a terminal window or PowerShell and **navigate to your lab12-partner folder using the cd command** (it's probably in the git folder in your home directory). Now run the following commands.

1. `git pull`
 - a. Make sure you repository is up to date, per git rule number 3
2. `git commit -a -m "updated TreeNode class"`
 - a. If this gives you a branch up to date method try...
 - b. `git add .`
 - c. `git commit -m "updated TreeNode class"`
3. `git push`
4. **What happens?**
5. Your group mate needs to add you as a collaborator on their project. Have them go to Settings->Collaborators and add your GitHub username.
6. Now run `git push` again what happens.
7. Make sure each of your group mates has successfully pushed their code.
8. Now go into your original lab12 folder from the terminal or PowerShell and run the command `git pull`. **What happens?**
9. Go check your original lab12 Eclipse project.

Hopefully, you are starting to see how useful git and GitHub can be to back up your code and collaborate with others. Usually, you will just have one shared repository for the project.

STOP

Now it's time for your exit interview

(Optional) Bonus Part 5 (2pt)

If you still have some time, you can explore additional features of git and github including [github desktop](#), github and eclipse ([link](#), [link](#)), etc.

If you want more practice with Trees, try the following:

- Make `TreeNode` generic. Change every occurrence of “`TreeNode`” to “`TreeNode<T>`”. In `TreeNode.java`, change the type of “`name`” from `String` to `T`. Now, since the node’s data can be of any class, “`name`” isn’t a good variable name; change it to “`data`”. To do this the cool way, double-click on the declaration of “`name`” to highlight it. Then right-click on it, and in the popup menu, select “`Refactor->Rename...`”. When prompted, replace “`name`” with “`data`”. Notice how the name changes in the declaration line and also everywhere else in the code. Now think of a better name for the `getName()` method and use `Refactor->Rename` to assign your better name. The compiler will now complain about lines that don’t use generic nodes; fix those lines.
- Add lines with creative Hobbit names to `BagginsFamilyTree.txt`. Run `FamilyTree` as an app and see if the printout of the tree matches your expectations.
- Create and process a different family tree. Use your own family tree, or choose anything that interests you. Try googling any of these:
 - The Pandavas
 - The Pharaohs of Egypt
 - King Alfred the Great of Wessex and his descendants
 - The Zhou dynasty of China
 - The Indo-European languages
 - Carl Woese’s tree of life (an early triumph of bioinformatics!)
- To make a family tree file, right-click on the “`data`” folder in the Eclipse Package Explorer. In the popup menu, choose “`New->File`” and create a file whose name ends with “`.txt`”. This will create an empty text file. Double-click on the filename in Package Explorer to open an editor on that file. Type in lines with the correct format. For example, if you’re modeling the Pandavas, Pandu was the father of Yudhishthira, Bhima, Arjuna, Nakula, and Sahadeva. This would be represented in the file by:


```
Pandu:Yudhishthira,Bhima,Arjuna,Nakula,Sahadeva (with no spaces)
```
- Before you test your code, delete the lines from `main` that find the most recent common ancestor of Bilbo and Frodo.

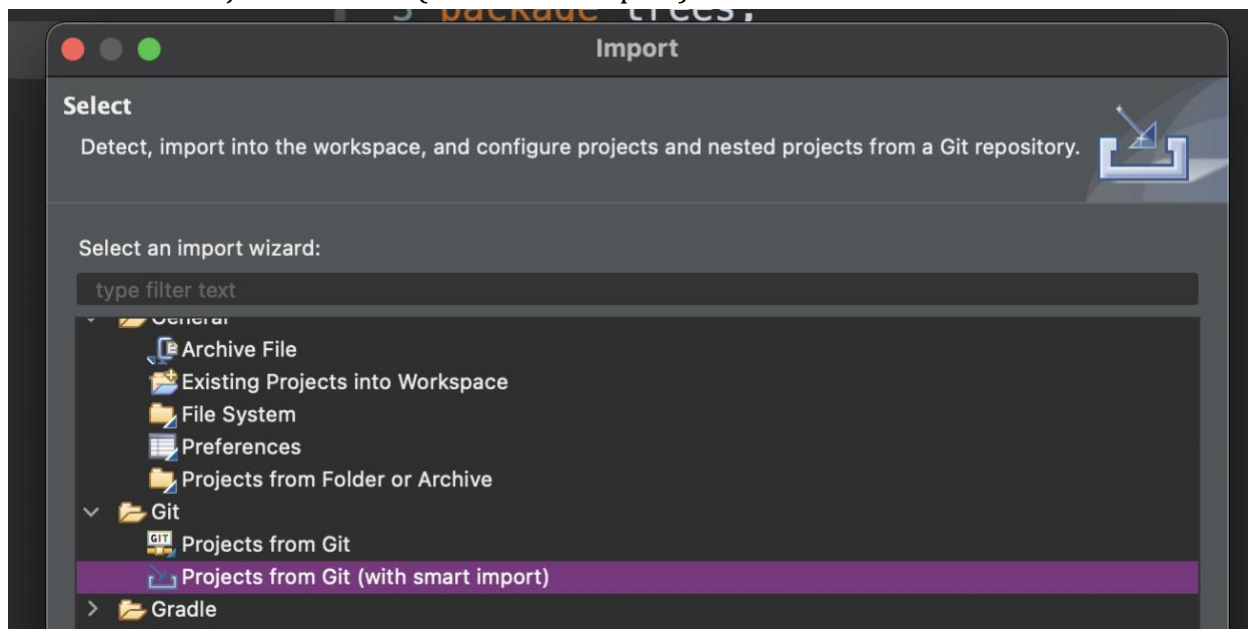
Saving your work

It can be a good idea to back up your work in case you ever accidentally delete your eclipse-workspace. If you ever actually read this far down in the lab document, you may remember that previous versions said that: *you will learn better ways to backup your work using version control and github*. Now you know a much easier way to do it. Instead of cluttering up your Desktop with old jar files, you can just store all your projects on Github. Next week will continue working with version control as we collaboratively complete the lab using one shared repository. It can be a good idea to keep your class projects private especially if they are homework related.

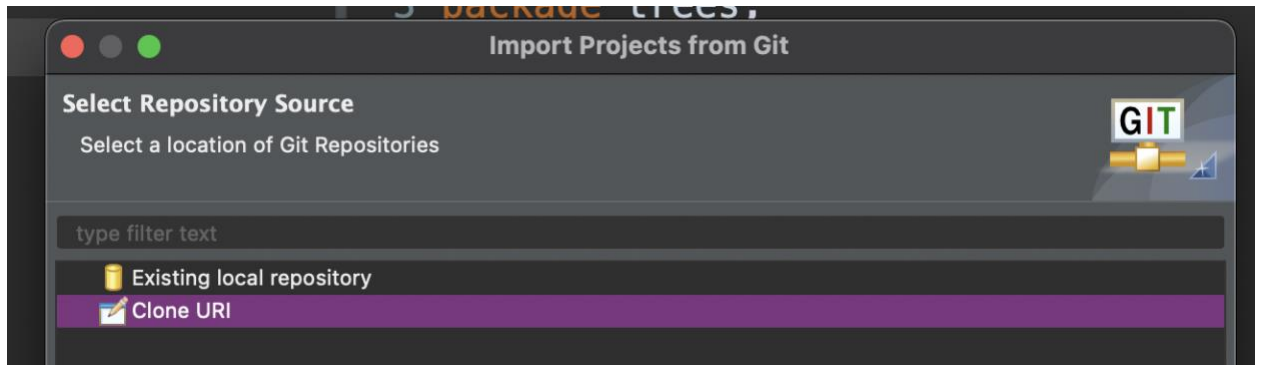
Appendix A

Follow these steps to add your group mates Github repository to Eclipse.

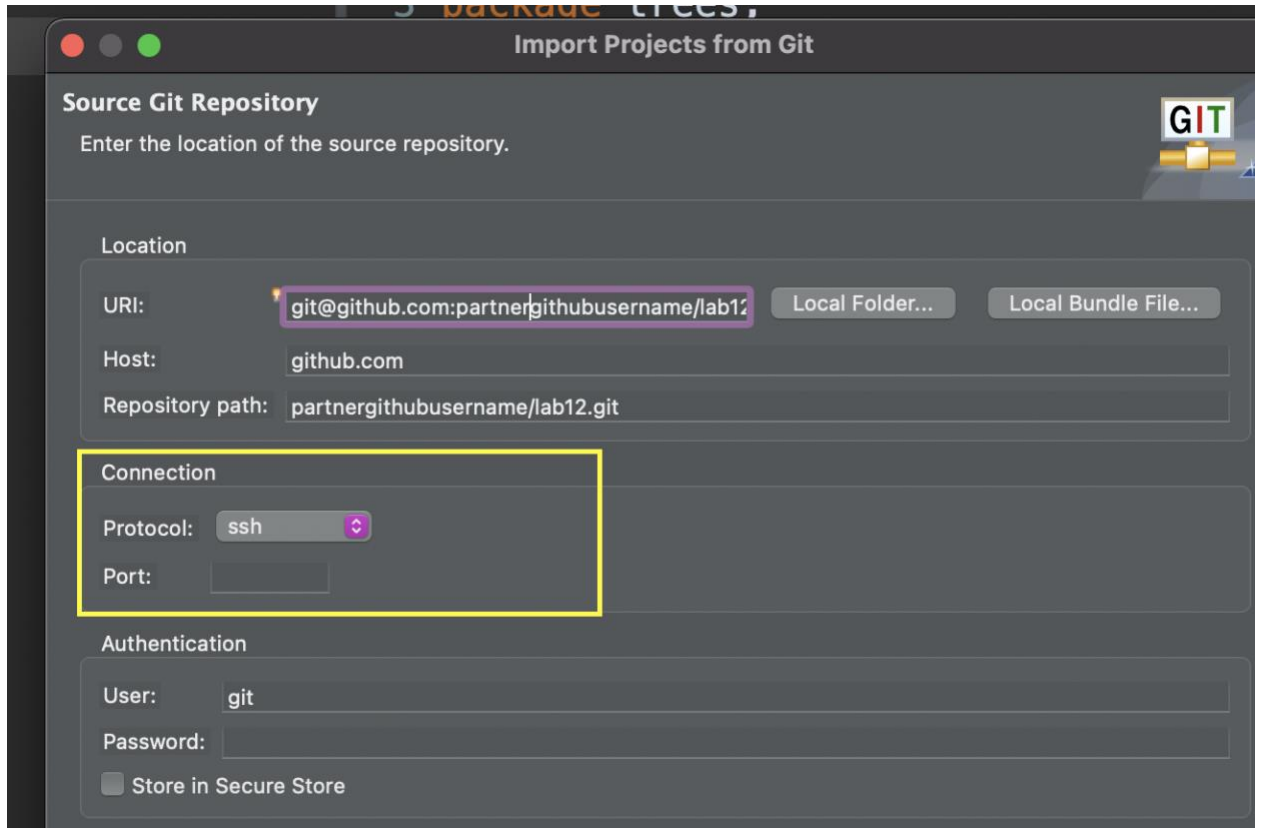
1. In Eclipse, create a new workspace or switch to a workspace that doesn't contain your lab12 project. Click File -> Import
2. Select Git -> Projects from Git (with smart import)



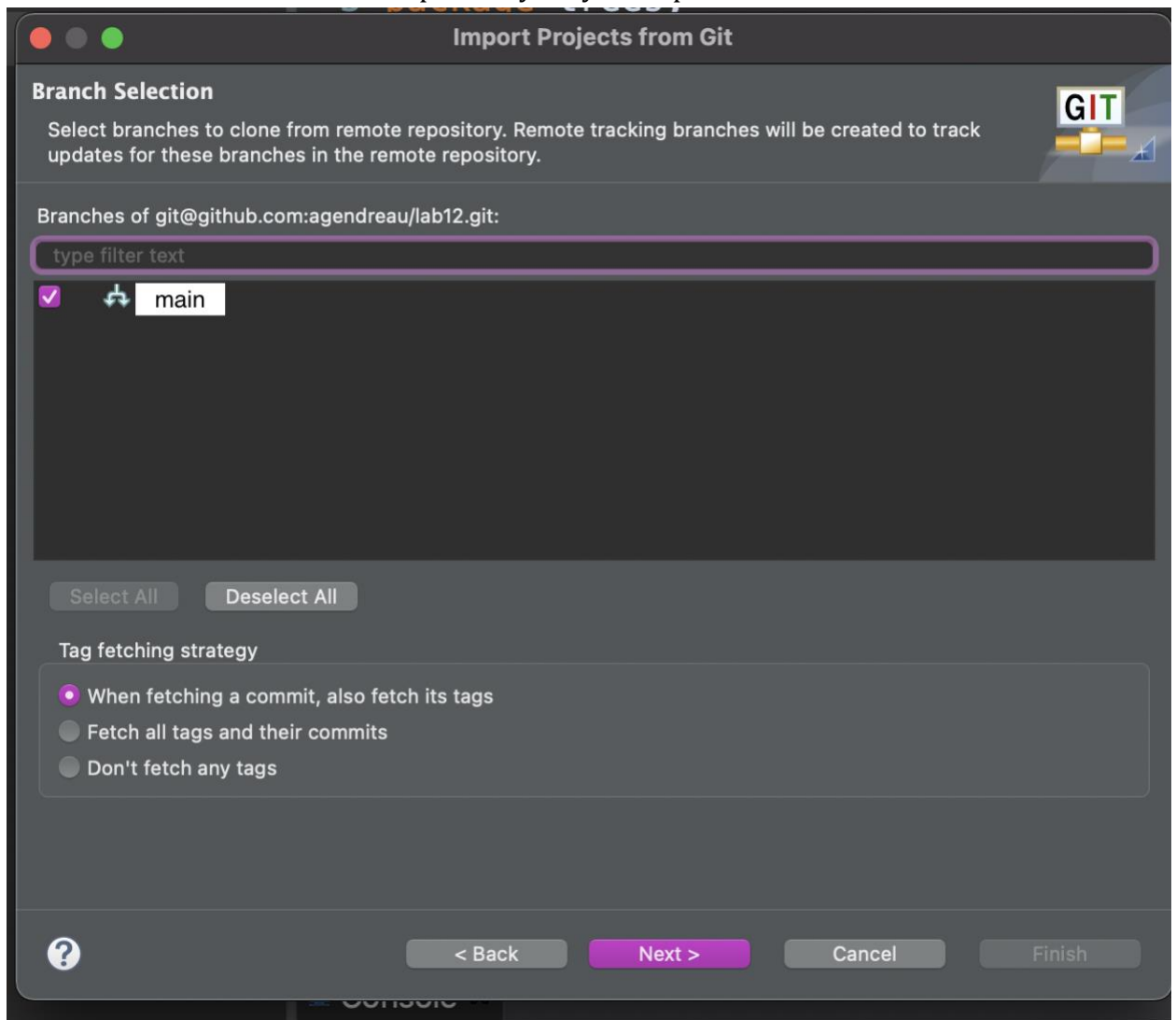
3. Select Clone URI



4. Enter the information for your partner's Github repository. Make sure the connection protocol is ssh. If you encounter a problem in this step, cancel this import and go to Appendix B.

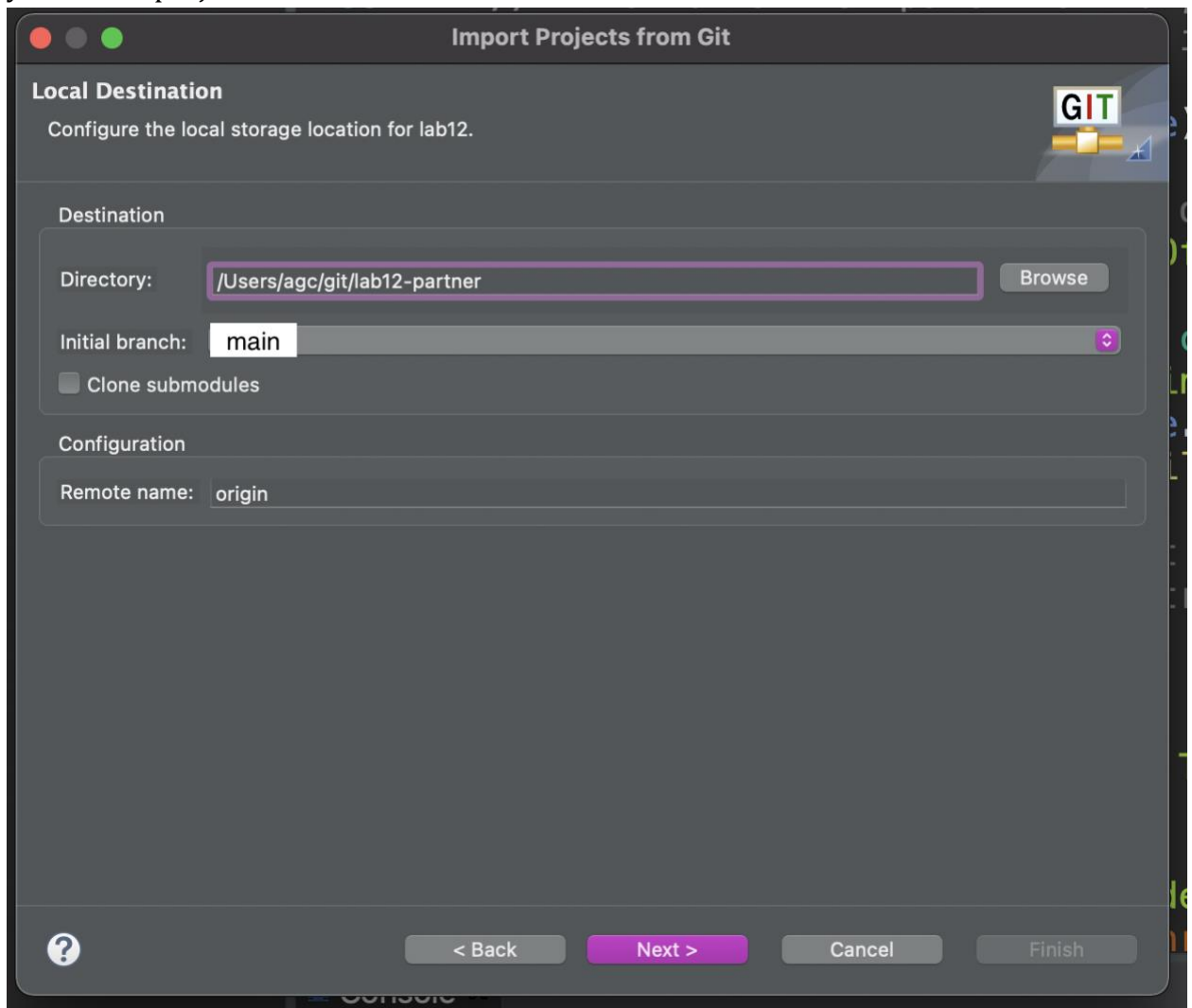


5. Select the main branch. There is probably only one option.



6. You probably already have a project called lab12 stored in your git folder on your machine, call this one lab12-partner or something else that differentiates it from

your lab12 project.



The image shows a macOS-style dialog box titled "Import Projects from Git". It has a dark gray background and standard macOS window controls (red, yellow, green buttons) in the top-left corner. The dialog is divided into two main sections: "Local Destination" and "Configuration".

Local Destination

Configure the local storage location for lab12.

Destination

Directory:

Initial branch:

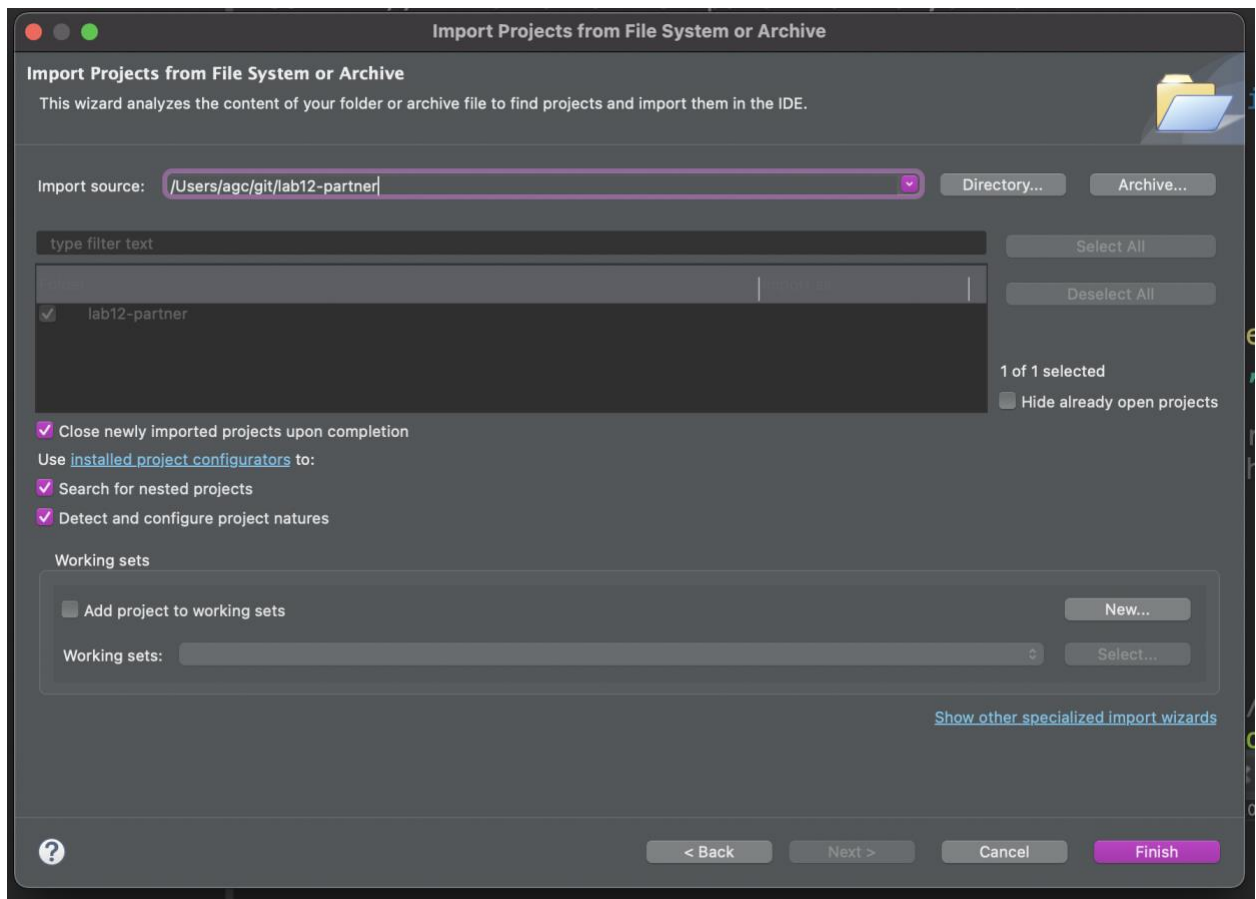
☐ Clone submodules

Configuration

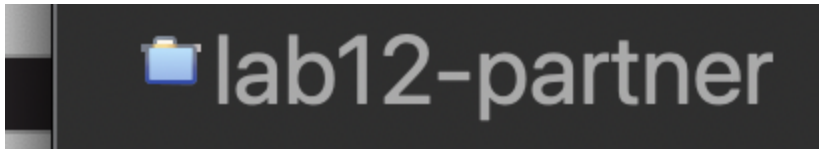
Remote name:

At the bottom of the dialog, there is a row of four buttons: a help button (question mark icon), "< Back", "Next >" (highlighted in purple), and "Cancel". A "Finish" button is also present at the bottom right.

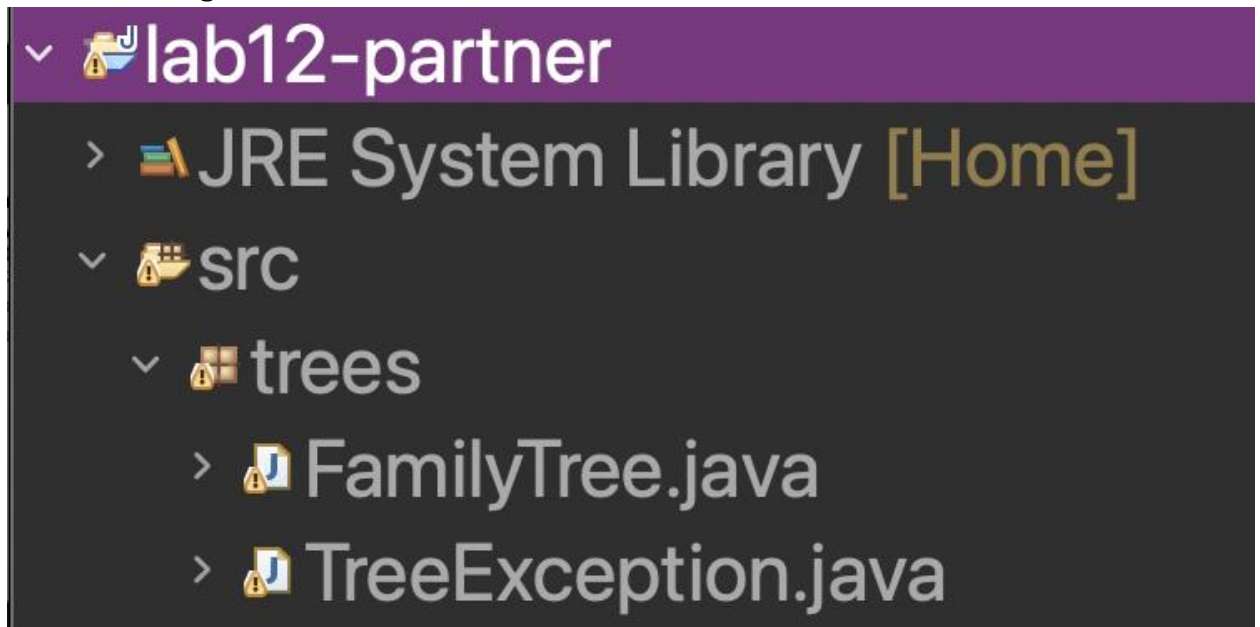
7. Click Finish.



8. You should see something like this in your project explorer in Eclipse.



9. Double click on the lab12-partner folder to expand it. It should look something like the below image.



Now return to Part 3.

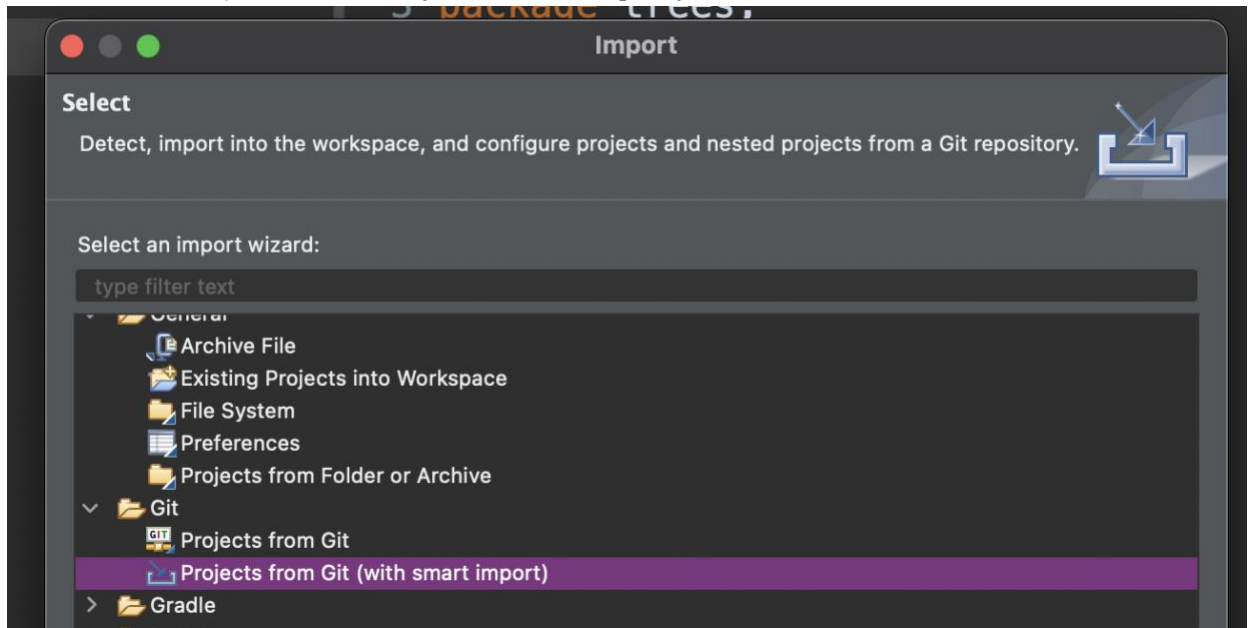
Appendix B

If you could not clone your group mate's repository through Eclipse, you can do it from the command line. Open your terminal or powershell and navigate to whatever folder you want (e.g., Documents, Desktop, etc.). Run the following command

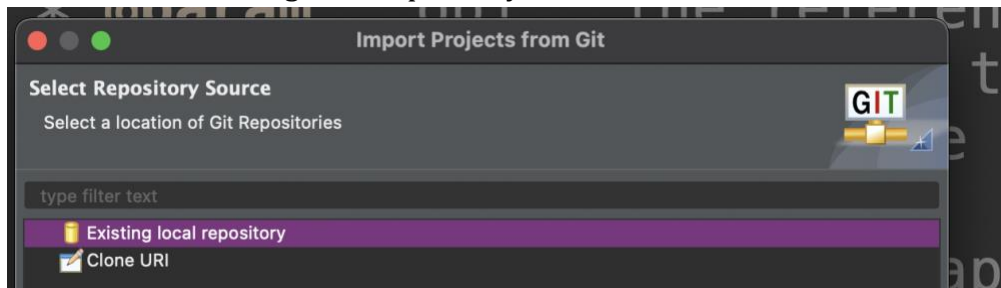
```
git clone git@github.com:partnerusername/lab12.git
```

1. In Eclipse in the same workspace you tried to import your partner's repository before, Click File -> Import

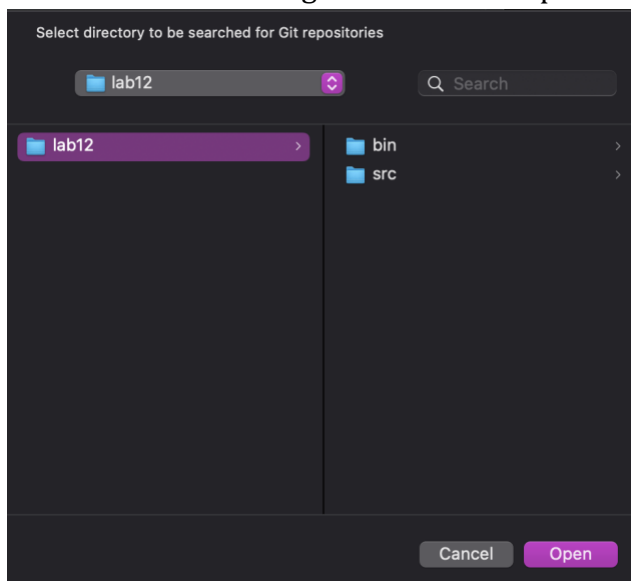
2. Select Git -> Projects from Git (with smart import)



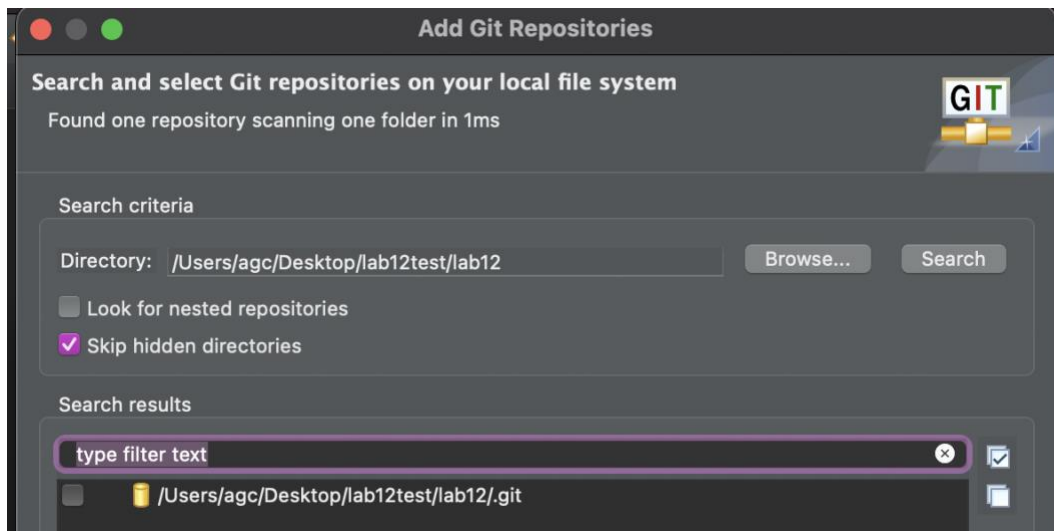
3. Select Existing local repository.



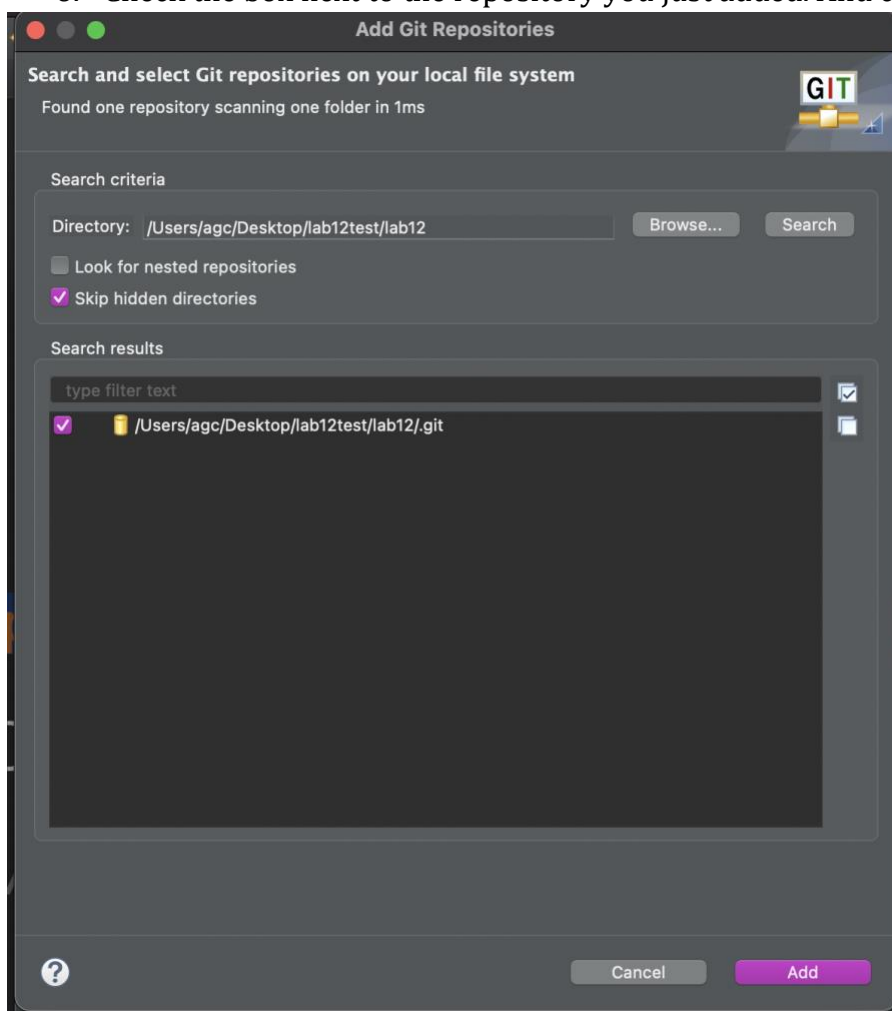
4. Now click Add and navigate to the lab12 directory that you cloned in Step 1. It should look something like this. Click Open.



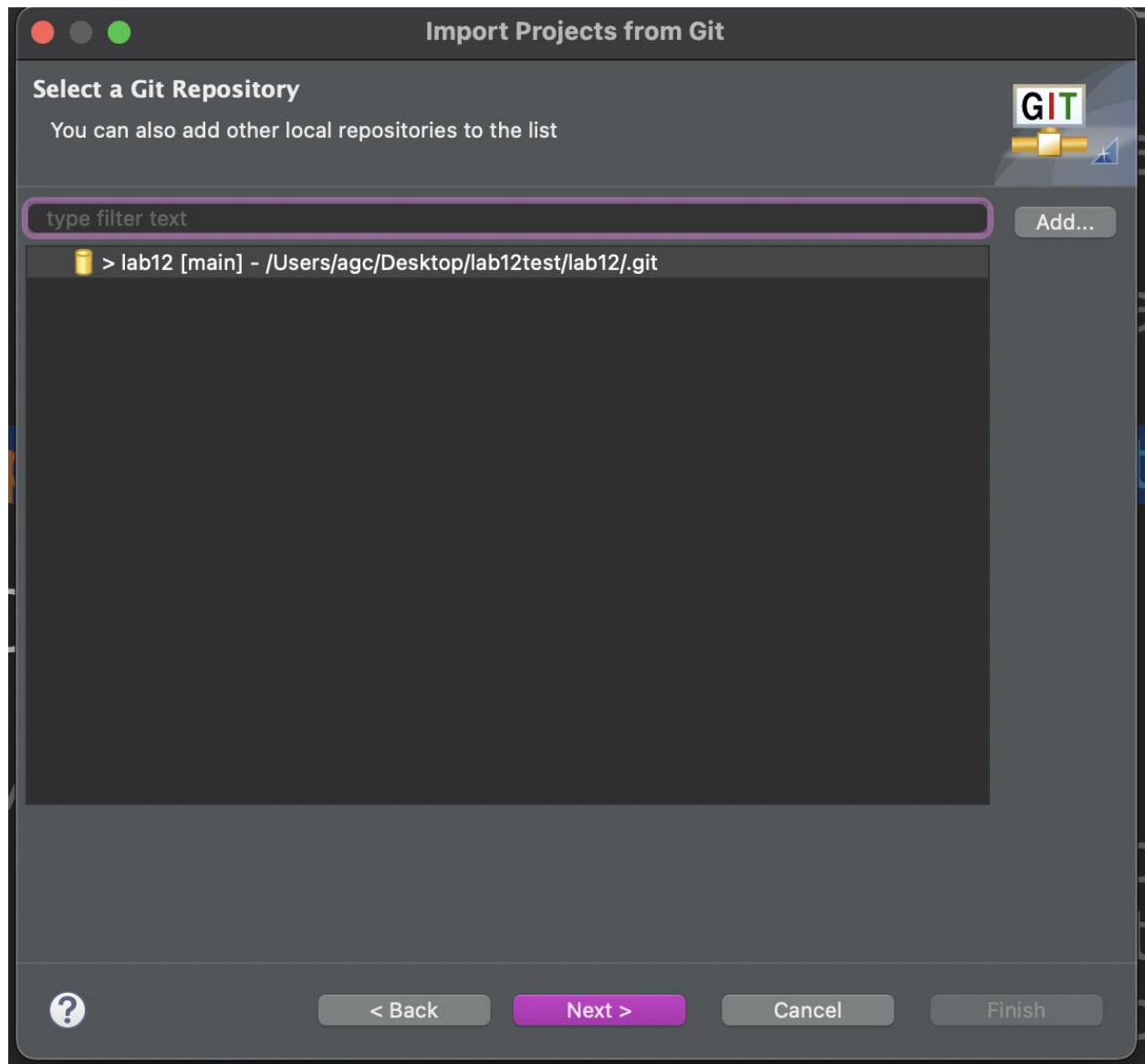
5. Now your import wizard should look like this.



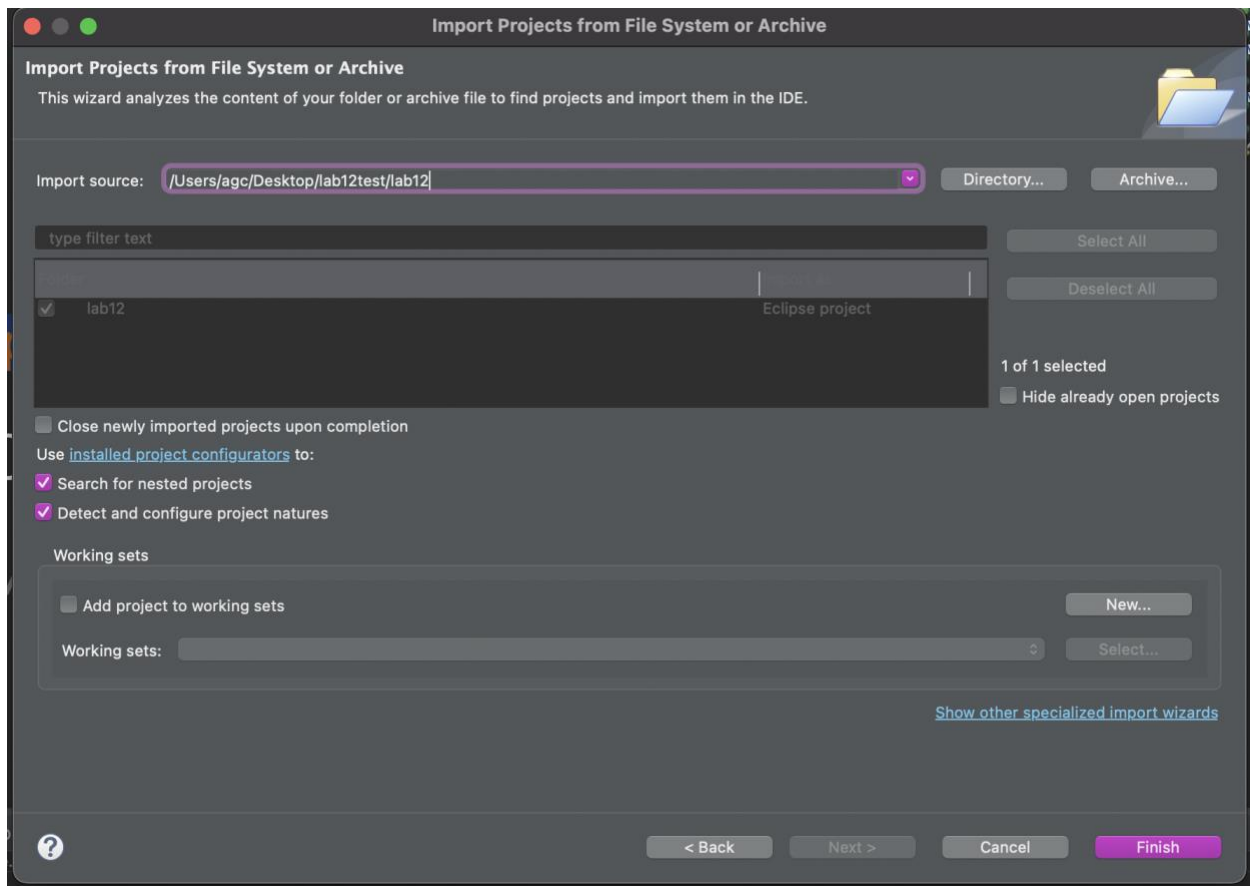
6. Check the box next to the repository you just added. And click Add.



7. Now Click Next



8. And Finally Click Finish



9. You should now see something that looks like this



10. Double click on the lab12 folder to expand it (this also compiles the code). It should look something like the below image.



Now return to Part 3.