Git and GitHub: Managing your code and team-based projects

Prof. Dionne Aleman

MIE250: Fundamentals of object-oriented programming
University of Toronto

What is Git?

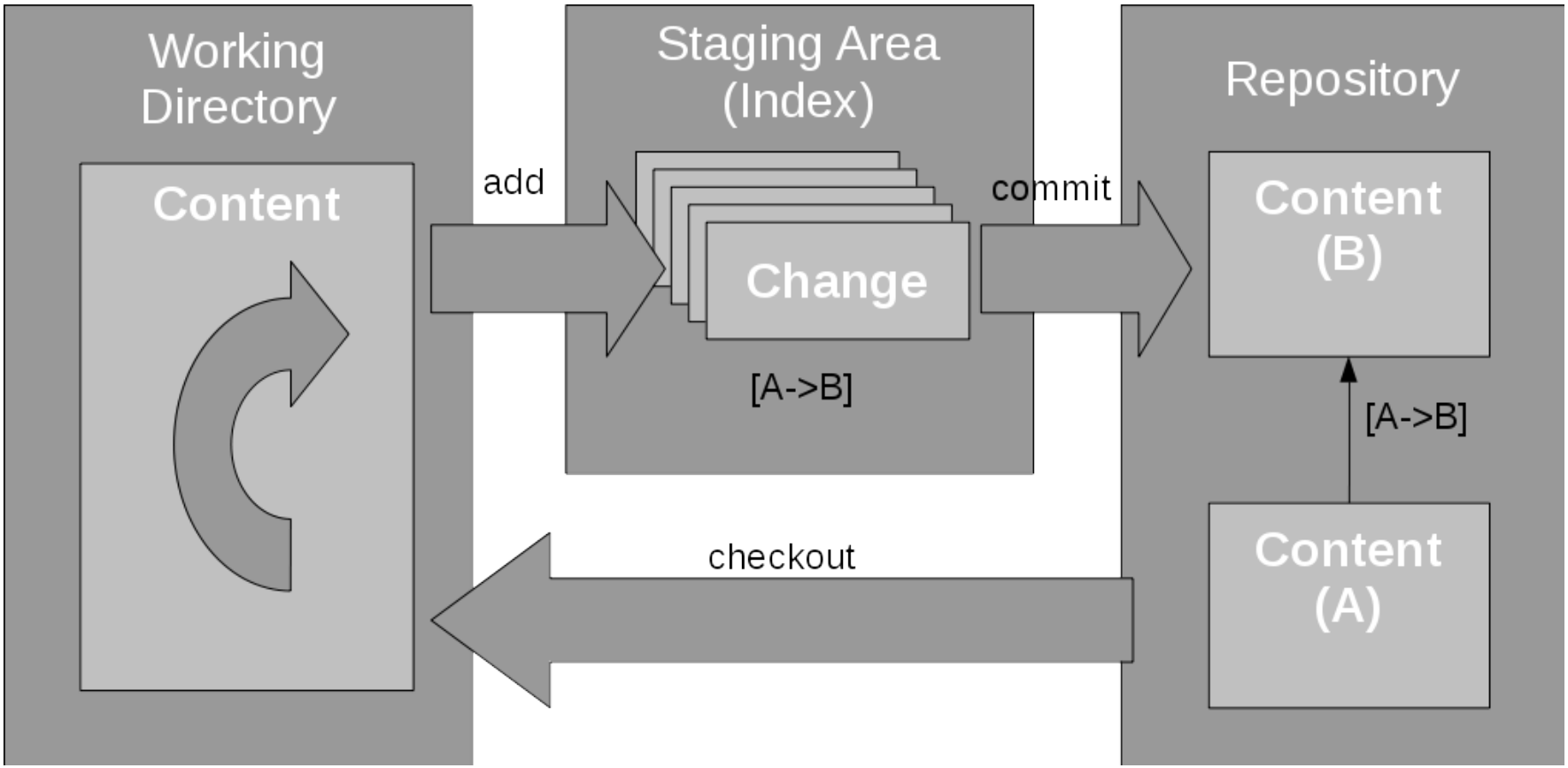
- Version-control system
- Open-source, Linux-based (but can run on any OS)
- Creates a repository (repo) of all your project code and versions
- Allows multiple people to work on different parts of project simultaneously
- Allows different branches to exist, which can be merged when ready
- Provides efficiency, flexibility, accountability, and stability

What is GitHub?

https://www.github.com

- Most popular online Git management system
- Visual interface
- Integrated with many popular IDEs (including Eclipse, Visual Studio)
- Free to use (pay for more advanced features)

How does Git work?



All managed through a bunch of hidden files in the repo directory! What's in those files? Hashes ... lots of hashes.

From IBM's Git tutorial: https://developer.ibm.com/tutorials/d-learn-workings-git/

Goal of this lecture content

- Not to make you a Git expert.
- Teach you the basics of Git and GitHub so that you can implement most of what you would ever need to do, and be able to look up and understand anything beyond that on your own.
- Feel comfortable enough to use GitHub for your course projects (now and in future courses).

Installing Git

GitHub's Git installation guide:

https://github.com/git-guides/install-git

- Mac users: If you think there will be a lot of coding in your future, get familiar with Homebrew (Mac version of `apt-get`).
- Linux users: You should already be familiar with `apt-get`.

Essential Git commands: `git <command>`

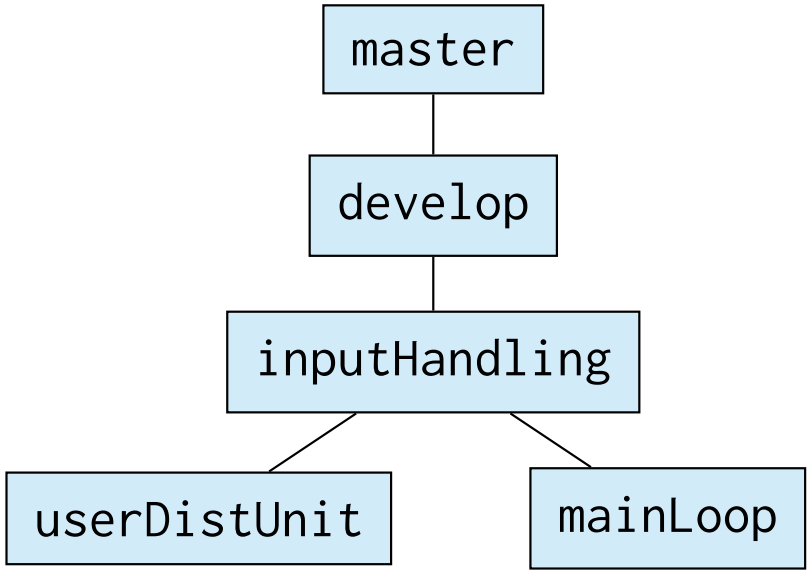
Command	Meaning
`init`	Create (initialize) a new repository
`status`	Check the status of your local code (what branch, current with central repo, etc.)
`checkout`	Switch to a branch (checkout the branch from the repo)
`add`	Add files to the staging area to be pushed to the repo
`commit`	Commit the added files to the repo
`merge`	Merge one branch into another
`reset`	Re-sync with the central repo, discarding local changes

Git example: Distance calculator program

- Imagine you are building this program from scratch, maybe with a team.
- How would you start coding?
  - Include all features all at once?
  - Create basic working structure, then add in features?
- Each team member can work on a particular feature independently (or, you can work on each feature yourself without worrying about messing up the main working program).

Branches[1]

- `master` (or `main`, `primary`): Just bare bones for now
- `develop`: All features, pretty sure things are working but not yet ready for public release
- `inputHandling`: Error handling on time, speed inputs
- `userDistUnit`: User selects unit; if miles (km), also show km (miles)
- `mainLoop`: Loop until user exits



[1] You won't always know all the branches in advance, especially if branches are created to solve complex bugs, but the branches are all laid out here just so you know where we are headed.

A note on branch names

- Usually best to use a systematic naming scheme for branches other than `master`, `develop`
- `iss001`, `iss002`, etc. is common, and comments can be left to describe nature of each "issue"

Getting started

1. Create directory for the repo.
   - I like to have a directory for the project in general, with a repo directory inside. That way, everything related to the project goes in the main directory, and code itself goes in the repo directory.
2. Initialize the new repo.
3. Put your code in the new repo.
4. Add, commit, and push code.

The terminal commands

```
cd /Users/dionne/courses/MIE250/_lectureNotes/src/gitlocal/
mkdir -p DistanceCalculator/repo
cd DistanceCalculator/repo/
git init
git status
git add -A *.java
git status
git commit -m "Initial commit"
```

src/git/init.txt

Initial commit sample code is in src/git/DistanceCalculator/initialCommit/.

The terminal commands

Go to directory

```
cd /Users/dionne/courses/MIE250/_lectureNotes/src/gitlocal/
mkdir -p DistanceCalculator/repo
cd DistanceCalculator/repo/
git init
git status
git add -A *.java
git status
git commit -m "Initial commit"
```

src/git/init.txt

Initial commit sample code is in src/git/DistanceCalculator/initialCommit/.

The terminal commands

```
cd /Users/dionne/courses/MIE250/_lectureNotes/
mkdir -p DistanceCalculator/repo
cd DistanceCalculator/repo/
git init
git status
git add -A *.java
git status
git commit -m "Initial commit"
```

src/git/init.txt

Make (recursive) directories for project and repo

Initial commit sample code is in src/git/DistanceCalculator/initialCommit/.

The terminal commands

```
1 cd /Users/dionne/courses/MIE250/_lectureNotes/
2 mkdir -p DistanceCalculator/repo
3 cd DistanceCalculator/repo/
4 git init
5 git status
6 git add -A *.java
7 git status
8 git commit -m "Initial commit"
```

Go to repo directory

src/git/init.txt

Initial commit sample code is in src/git/DistanceCalculator/initialCommit/.

The terminal commands

```
cd /Users/dionne/courses/MIE250/_lectureNotes/src/gitlocal/
mkdir -p DistanceCalculator/repo
cd DistanceCalculator/repo/
git init
git status
git add -A *.java
git status
git commit -m "Initial commit"
```

Initialize repo

src/git/init.txt

Initial commit sample code is in src/git/DistanceCalculator/initialCommit/.

The terminal commands

```
cd /Users/dionne/courses/MIE250/_lectureNotes/src/gitlocal/
mkdir -p DistanceCalculator/repo
cd DistanceCalculator/repo/
git init
git status
git add -A *.java
git status
git commit -m "Initial commit"
```

Check git status

src/git/init.txt

Initial commit sample code is in src/git/DistanceCalculator/initialCommit/.

The terminal commands

```
cd /Users/dionne/courses/MIE250/_lectureNotes/src/gitlocal/
mkdir -p DistanceCalculator/repo
cd DistanceCalculator/repo/
git init
git status
git add -A *.java
git status
git commit -m "Initial commit"
```

Add all changed .java files

src/git/init.txt

Add/create code in the repo first.

Initial commit sample code is in src/git/DistanceCalculator/initialCommit/.

The terminal commands

```
cd /Users/dionne/courses/MIE250/_lectureNotes/src/gitlocal/
mkdir -p DistanceCalculator/repo
cd DistanceCalculator/repo/
git init
git status
git add -A *.java
git status
git commit -m "Initial commit"
```

src/git/init.txt

Initial commit sample code is in src/git/DistanceCalculator/initialCommit/.

The terminal commands

```
cd /Users/dionne/courses/MIE250/_lectureNotes/src/gitlocal/
mkdir -p DistanceCalculator/repo
cd DistanceCalculator/repo/
git init
git status
git add -A *.java
git status
git commit -m "Initial commit"
```

src/git/init.txt

Commit changed files with message

Initial commit sample code is in src/git/DistanceCalculator/initialCommit/.

The terminal commands

```
cd /Users/dionne/courses/MIE250/_lectureNotes/src/gitlocal/
mkdir -p DistanceCalculator/repo
cd DistanceCalculator/repo/
git init
git status
git add -A *.java
git status
git commit -m "Initial commit"
```

src/git/init.txt

Commit changed files with message

It's important to leave USEFUL (but brief) comments!

Initial commit sample code is in src/git/DistanceCalculator/initialCommit/.

Add develop, inputHandling branches

- Create a branch by checking it out.
- When you create a branch, it is initialized with files from the current branch.

```
1 git checkout -b develop
2 git checkout -b inputHandling
3 git status
4 git branch
```

src/git/newBranches1.txt

```
DionneMBP:repo dionne$ git branch
  develop
* inputHandling
  master
```

Update inputHandling branch

- Update the inputHandling branch to catch user input errors.
- Commit changes in chunks or when you are completely finished.

```
git checkout inputHandling
git status
git add -A *.java
git status
git commit -m "Non-num, non-neg error handling"
git status
```

src/git/inputHandlingBranch.txt

Sample input error handling code is in src/git/DistanceCalculator/inputHandling/.

Update inputHandling branch

- Update the inputHandling branch to catch user input errors.
- Commit changes in chunks or when you are completely finished.

```
git checkout inputHandling
git status
git add -A *.java
git status
git commit -m "Non-num, non-neg error handling"
git status
```

src/git/inputHandlingBranch.txt

Sample input error handling code is in src/git/DistanceCalculator/inputHandling/.

Update inputHandling branch

- Update the inputHandling branch to catch user input errors.
- Commit changes in chunks or when you are completely finished.

```
1 git checkout inputHandling
2 git status
3 git add -A *.java
4 git status
5 git commit -m "Non-num, non-neg error handling"
6 git status
```

src/git/inputHandlingBranch.txt

```
DionneMBP:repo dionne$ git status
On branch inputHandling
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   DistanceCalculator.java
        modified:   DistanceV5.java

no changes added to commit (use "git add" and/or "git commit -a")
```

Sample input error handling code is in src/git/DistanceCalculator/inputHandling/.

Update inputHandling branch

- Update the inputHandling branch to catch user input errors.
- Commit changes in chunks or when you are completely finished.

```
1 git checkout inputHandling
2 git status
3 git add -A *.java
4 git status
5 git commit -m "Non-num, non-neg error handling"
6 git status
```

src/git/inputHandlingBranch.txt

Sample input error handling code is in src/git/DistanceCalculator/inputHandling/.

Update inputHandling branch

- Update the inputHandling branch to catch user input errors.
- Commit changes in chunks or when you are completely finished.

```
git checkout inputHandling
git status
git add -A *.java
git status
git commit -m "Non-num, non-neg error handling"
git status
```

src/git/inputHandlingBranch.txt

```
DionneMBP:repo dionne$ git status
On branch inputHandling
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   DistanceCalculator.java
        modified:   DistanceV5.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        DistanceCalculator.class
        DistanceV5.class
```

Sample input error handling code is in src/git/DistanceCalculator/inputHandling/.

Update inputHandling branch

- Update the inputHandling branch to catch user input errors.
- Commit changes in chunks or when you are completely finished.

```
git checkout inputHandling
git status
git add -A *.java
git status
git commit -m "Non-num, non-neg error handling"
git status
```

src/git/inputHandlingBranch.txt

Sample input error handling code is in src/git/DistanceCalculator/inputHandling/.

Update inputHandling branch

- Update the inputHandling branch to catch user input errors.
- Commit changes in chunks or when you are completely finished.

```
git checkout inputHandling
git status
git add -A *.java
git status
git commit -m "Non-num, non-neg error handling"
git status
```

src/git/inputHandlingBranch.txt

```
DionneMBP:repo dionne$ git status
On branch inputHandling
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        DistanceCalculator.class
        DistanceV5.class

nothing added to commit but untracked files present (use "git add" to track)
```

Sample input error handling code is in src/git/DistanceCalculator/inputHandling/.

Merge inputHandling branch into develop branch

```
git checkout develop
git merge inputHandling
git status
git branch -d inputHandling
```

src/git/inputHandlingBranch_merge.txt

Merge inputHandling branch into develop branch

```
1 git checkout develop
2 git merge inputHandling
3 git status
4 git branch -d inputHandling
```

Switch to target branch

src/git/inputHandlingBranch_merge.txt

Merge inputHandling branch into develop branch

```
1 git checkout develop
2 git merge inputHandling
3 git status
4 git branch -d inputHandling
```

Merge!

src/git/inputHandlingBranch_merge.txt

Merge inputHandling branch into develop branch

```
git checkout develop
git merge inputHandling
git status
git branch -d inputHandling
```

src/git/inputHandlingBranch_merge.txt

```
DionneMBP:repo dionne$ git merge inputHandling
Updating 483f83b..8a3cc92
Fast-forward
 DistanceCalculator.java | 18 ++++--------------
 DistanceV5.java         |  4 +---
 2 files changed, 5 insertions(+), 17 deletions(-)
```

Merge inputHandling branch into develop branch

```
1 git checkout develop
2 git merge inputHandling
3 git status
4 git branch -d inputHandling
```

src/git/inputHandlingBranch_mer

Delete finished branch if desired

Update userDistUnit and mainLoop branches

```
git checkout develop
git checkout -b userDistUnit
git add -A *.java
git commit -m "User selects dist unit, other dist unit also printed"
```

src/git/userDistUnitBranch.txt

```
git checkout develop
git checkout -b mainLoop
git add -A *.java
git commit -m "Program loops until user exits"
```

src/git/mainLoopBranch.txt

Sample branch code is in is in src/git/DistanceCalculator/userDistUnit/, src/git/DistanceCalculator/mainLoop/.

Update userDistUnit and mainLoop branches

```
1 git checkout develop
2 git checkout -b userDistUnit
3 git add -A *.java
4 git commit -m "User selects dist unit, other dist unit also printed"
```

Start from develop

src/git/userDistUnitBranch.txt

```
1 git checkout develop
2 git checkout -b mainLoop
3 git add -A *.java
4 git commit -m "Program loops until user exits"
```

Start from develop

src/git/mainLoopBranch.txt

Sample branch code is in is in src/git/DistanceCalculator/userDistUnit/, src/git/DistanceCalculator/mainLoop/.

Update userDistUnit and mainLoop branches

```
1 git checkout develop
2 git checkout -b userDistUnit
3 git add -A *.java
4 git commit -m "User selects dist unit, other dist unit also printed"
```

src/git/userDistUnitBranch.txt

```
1 git checkout develop
2 git checkout -b mainLoop
3 git add -A *.java
4 git commit -m "Program loops until user exits"
```

src/git/mainLoopBranch.txt

Sample branch code is in is in src/git/DistanceCalculator/userDistUnit/, src/git/DistanceCalculator/mainLoop/.

Update userDistUnit and mainLoop branches

```
1 git checkout develop
2 git checkout -b userDistUnit
3 git add -A *.java
4 git commit -m "User selects dist unit, other dist unit also printed"
```

src/git/userDistUnitBranch.txt

```
1 git checkout develop
2 git checkout -b mainLoop
3 git add -A *.java
4 git commit -m "Program loops until user exits"
```

src/git/mainLoopBranch.txt

Sample branch code is in is in src/git/DistanceCalculator/userDistUnit/, src/git/DistanceCalculator/mainLoop/.

Update userDistUnit and mainLoop branches

```
1 git checkout develop
2 git checkout -b userDistUnit
3 git add -A *.java
4 git commit -m "User selects dist unit, other dist unit also printed"
```

src/git/userDistUnitBranch.txt

```
1 git checkout develop
2 git checkout -b mainLoop
3 git add -A *.java
4 git commit -m "Program loops until user exits"
```

src/git/mainLoopBranch.txt

Sample branch code is in is in src/git/DistanceCalculator/userDistUnit/, src/git/DistanceCalculator/mainLoop/.

Merge everything up to develop

```
1 git checkout develop
2 git merge userDistUnit
3 git merge mainLoop
```

src/git/mergeTwoBranches.txt

Merge everything up to develop

```
1 git checkout develop
2 git merge userDistUnit
3 git merge mainLoop
```

src/git/mergeTwoBranches.txt

Success! But what does the program look like now?

Merge everything up to develop

```
1 git checkout develop
2 git merge userDistUnit
3 git merge mainLoop
```

src/git/mergeTwoBranches.txt

Success! But what does the program look like now?

```
------------------------------------------------
Distance calculator (enter 0 for both values to quit)
------------------------------------------------
Enter a distance unit ("mi" or "km"): km
Enter a speed (kph): 0
Enter a time traveled (minutes): 0

Goodbye!
```

Merge everything up to develop

```
git checkout develop
git merge userDistUnit
git merge mainLoop
```

src/git/mergeTwoBranches.txt

Success! But what does the program look like now?

```
-------------------------------------------------
Distance calculator (enter 0 for both values to quit)
-------------------------------------------------
Enter a distance unit ("mi" or "km"): km
Enter a speed (kph): 0
Enter a time traveled (minutes): 0

Goodbye!
```

Kind of awkward to require a unit, when the instructions say enter 0 for both values to quit. Good thing we have a `develop` branch before committing to the `master`!

Messed up your merge, want to start over?

Your new best friends:

```
git merge --abort
git reset --hard
```

Difference between Git and GitHub

- With GitHub, the repo is stored on GitHub's website instead of locally.
- It may be easier to configure IDEs to work with GitHub than a local repo.
- You can use all the same command line commands with GitHub, other than initialization to connect to GitHub.
- You still need to install Git on your computer to use GitHub.

Essential GitHub commands: `git <command>`

Command	Meaning
`clone`	Clone code from and set up Git connection to the remote repo
`pull`	Pull code from the remote repo
`push`	Push the commit to the remote repo

Quick note about local v. remote repo

- The remote repo has "origin" before all the branch names.

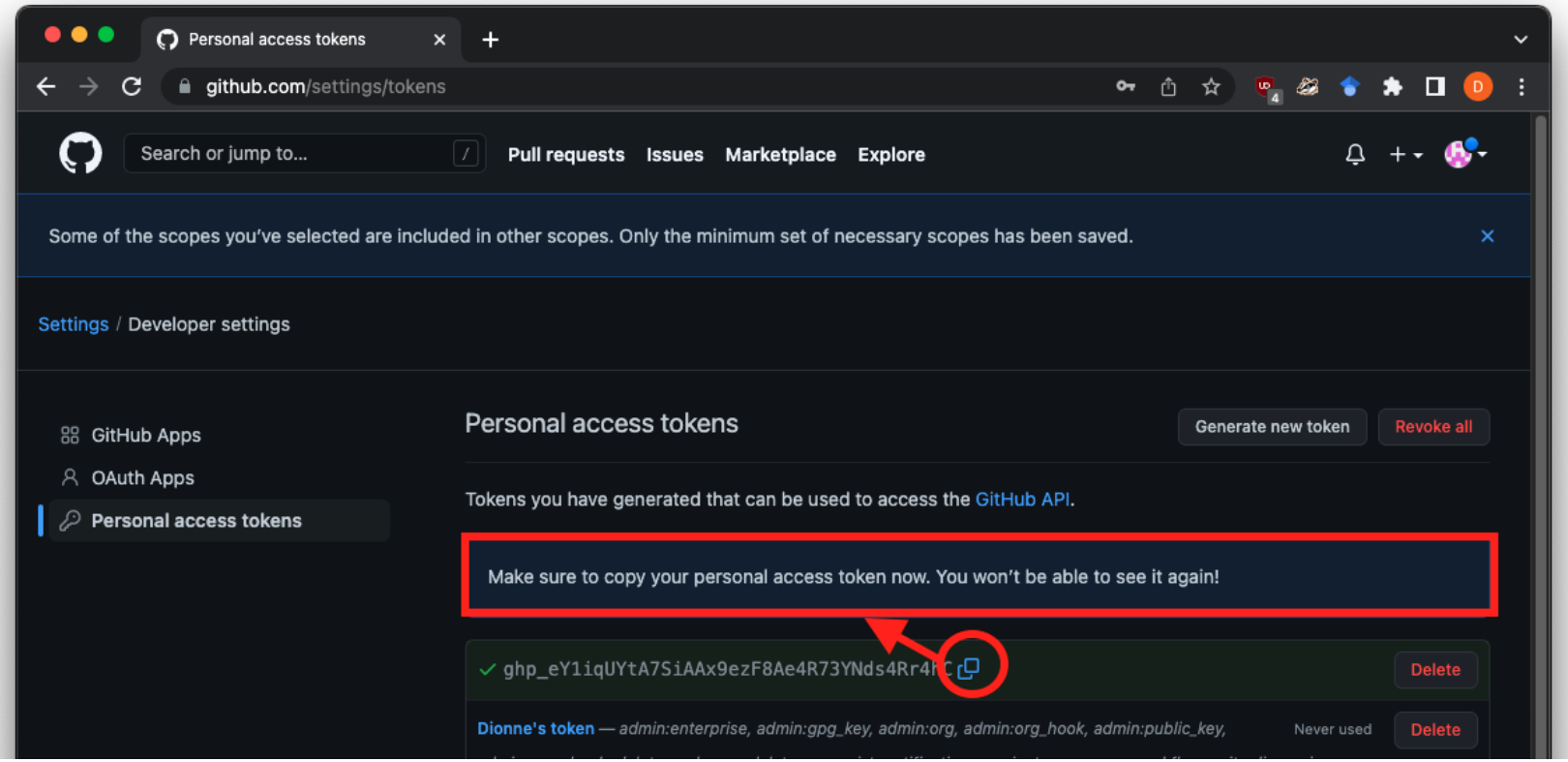
- For example:
  - Local branch: `develop`
  - GitHub branch online: `origin/develop`

Before getting started

1. Go to https://www.github.com and create an account.
2. Generate a personal access token (PAT) in GitHub.
3. Create an SSH key on your computer.
4. Connect your SSH key to your GitHub profile for authentication.

Generate a personal access token (PAT) in GitHub

Go to your profile → Settings → Developer Settings → Personal access tokens and follow the prompts to generate a new token. Make sure to save the token somewhere, since you will never be able to see it again.



Not my actual PAT, just a temporary demo for you to see.

Create an SSH key (if you don't already have one) and add it to GitHub

- Create an SSH key using ECDSA encryption by command line:

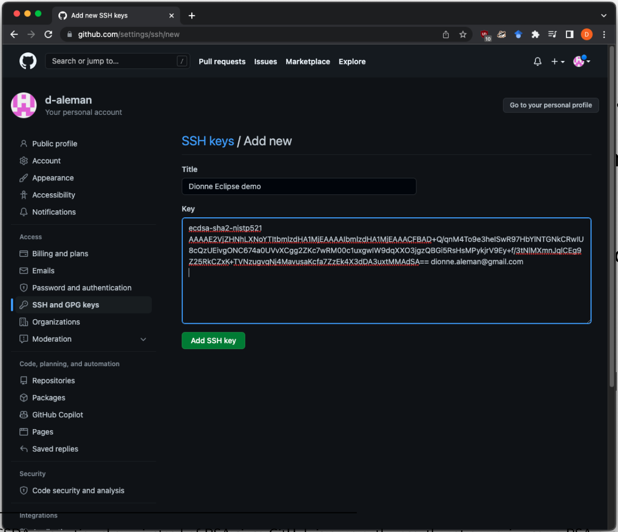
  ```
  ssh-keygen -t ecdsa -b 521 -C "your_email@example.com"
  ```

- Put the key in the default file location (your `.ssh` directory). Change the file name if you already have an `id_ecdsa.pub` file.

- Paste contents of `id_ecdsa.pub` into GitHub → Profile → SSH and GPG keys → Add new SSH key.

ECSDA encryption shown instead of RSA since GitHub is apparently recently not approving some RSA-encrypted keys.

Create an SSH key (if you don't already have one) and add it to GitHub



- ...and line:

  ...mple.com"

- ...ctory). Change the

- ...e → SSH and

ECDSA encryption shown instead of RSA since GitHub is apparently recently not approving some RSA-encrypted keys.

Getting started

1. Create repo on GitHub's website.
   - Is this repo public or private?
2. Clone repo to your local computer.
3. Perform initial commit.
4. Create branches that you know you will need.
5. Give access to collaborators.

Getting started

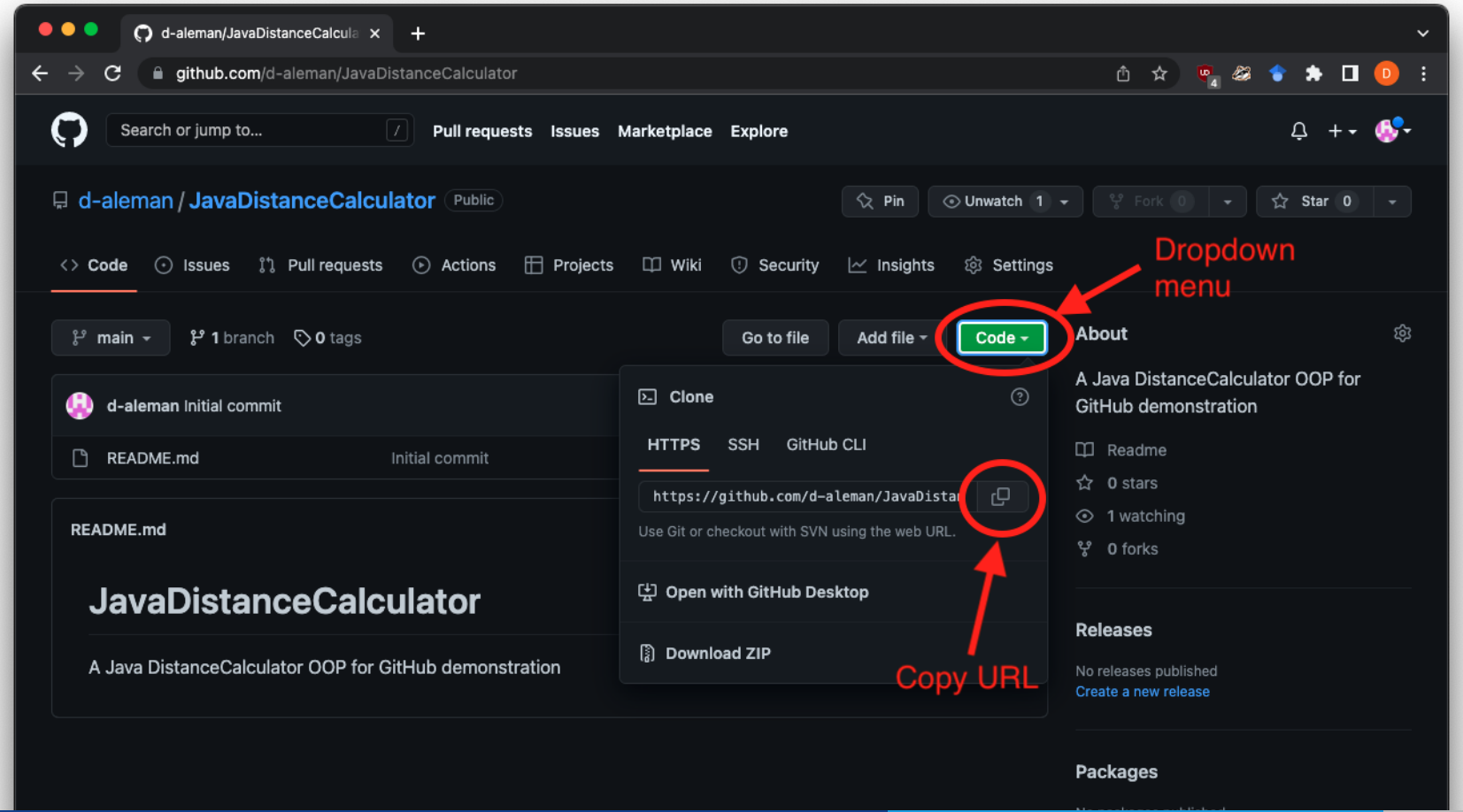
1. Create repo on GitHub's website.
   - Is this repo public or private?
   - **DO NOT MAKE COURSE CODE PUBLIC!** Academic offense danger!
2. Clone repo to your local computer.
3. Perform initial commit.
4. Create branches that you know you will need.
5. Give access to collaborators.

1. Create repo on GitHub's website

- Log in, then click Repositories → New.
- Give the repository a short name.
- Select public or private.
  - Best practice is private unless you have a compelling reason for public.
  - You can always share sensitive projects with potential employers by adding them to your repo.
- Choose to add a default README or `.gitignore` file or not
  - README: Basic instructions, in `.md` (Markdown) format.
  - `.gitignore`: Files that will be ignored by Git
- Pick a licensing type, if you expect others will use your code. Best to investigate those options carefully for what suits your needs and intellectual property concerns.

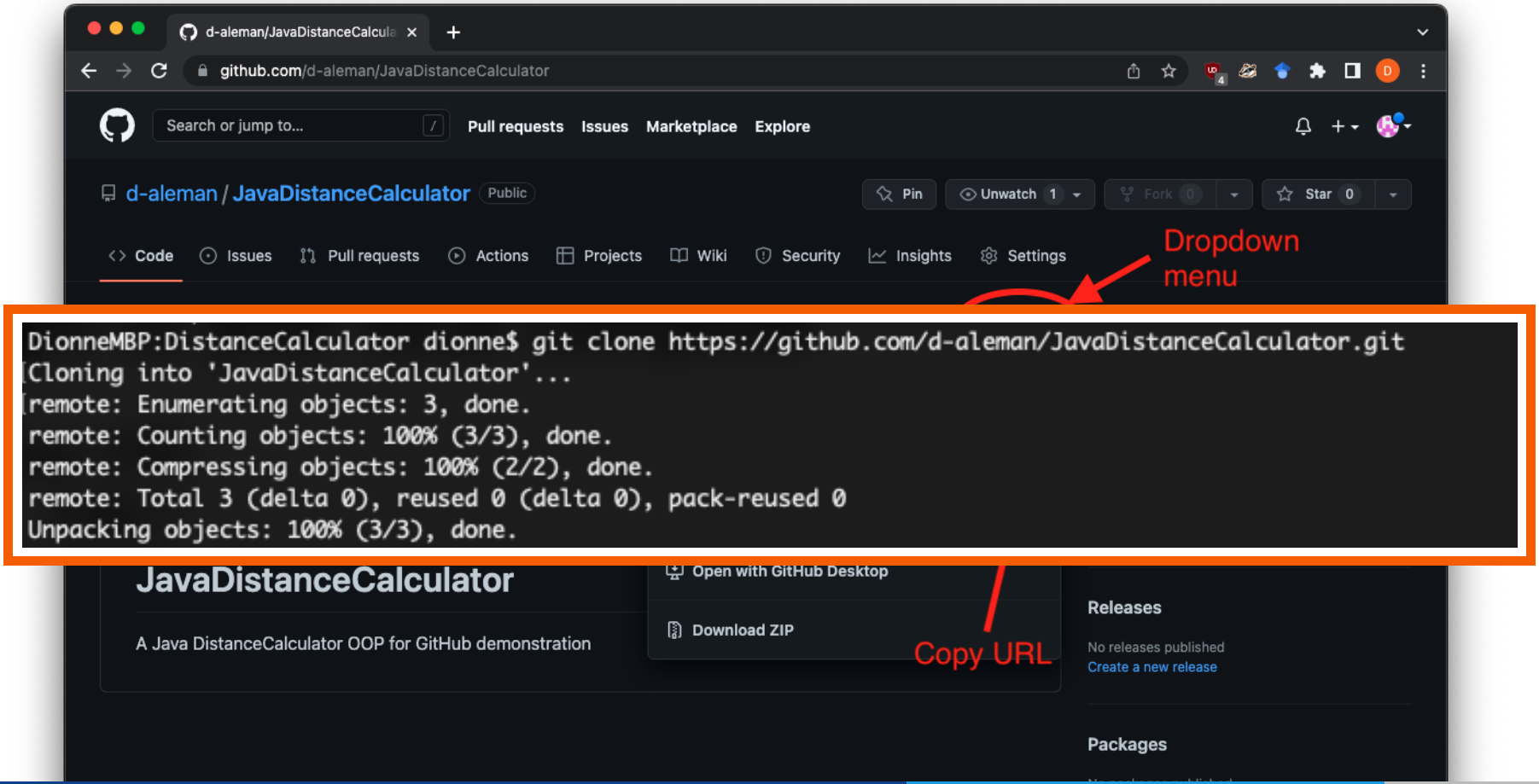
2. Clone repo to your local computer

- Code → copy URL
- `git clone <URL>` (creates a folder inside current folder with the repo with GitHub repo name)



2. Clone repo to your local computer

- Code → copy URL
- `git clone <URL>` (creates a folder inside current folder with the repo with GitHub repo name)



2. Perform initial commit

```
git add -A *.java
git commit -m "Initial commit"
git push
```

src/git/init_github.txt

2. Perform initial commit

```
git add -A *.java
git commit -m "Initial commit"
git push
```

src/git/init_github.txt

2. Perform initial commit

```
git add -A *.java
git commit -m "Initial commit"
git push
```

src/git/init_github.txt

2. Perform initial commit

```
1 git add -A *.java
2 git commit -m "Initial commit"
3 git push
```

src/git/init_github.txt

Push changes to GitHub repo

2. Perform initial commit

```
git add -A *.java
git commit -m "Initial commit"
git push
```

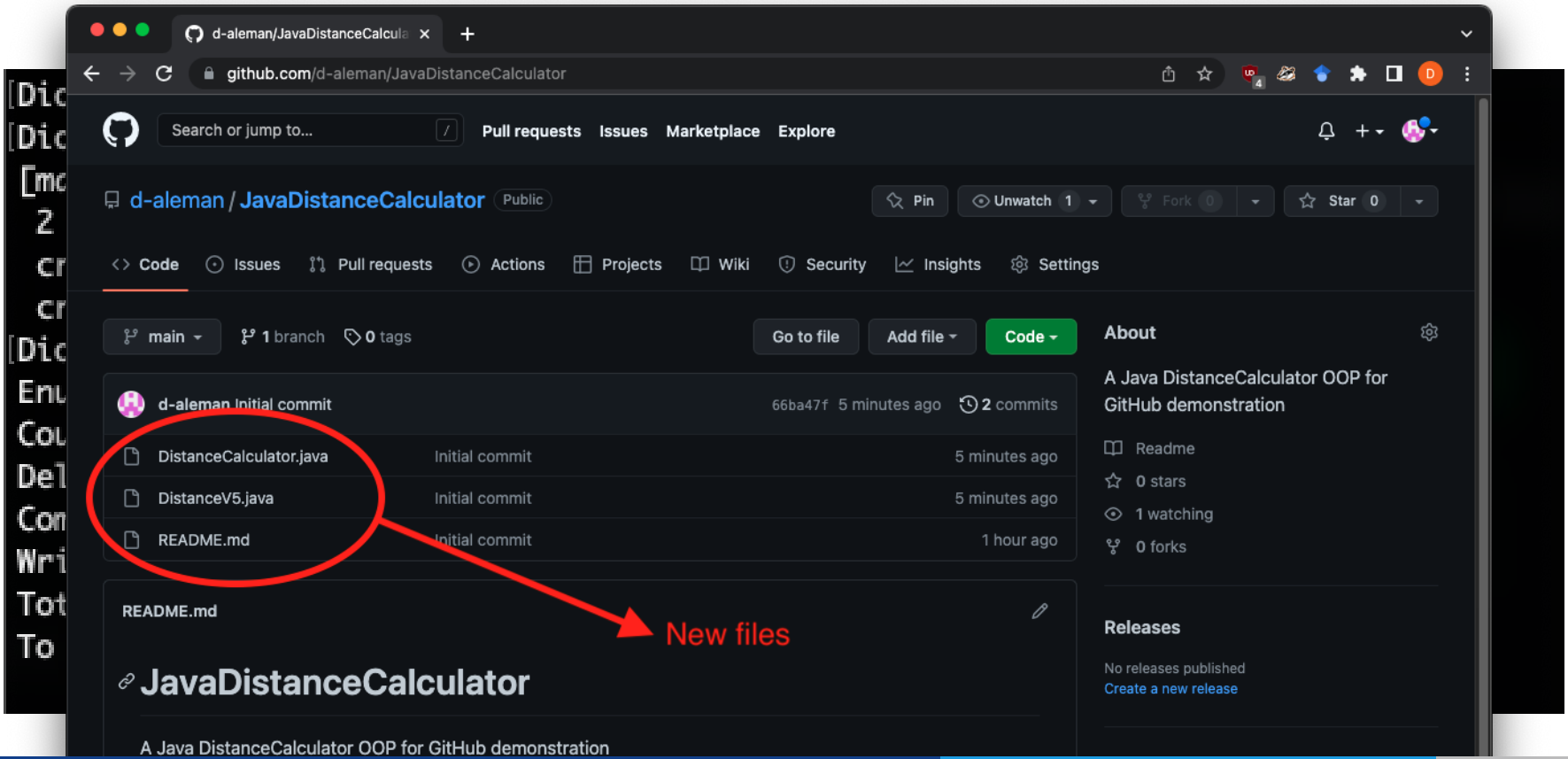
src/git/init_github.txt

```
DionneMBP:JavaDistanceCalculator dionne$ git add -A *.java
DionneMBP:JavaDistanceCalculator dionne$ git commit -m "Initial commit"
[main 66ba47f] Initial commit
 2 files changed, 76 insertions(+)
 create mode 100755 DistanceCalculator.java
 create mode 100755 DistanceV5.java
DionneMBP:JavaDistanceCalculator dionne$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.14 KiB | 1.14 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/d-aleman/JavaDistanceCalculator.git
   e35e598..66ba47f  main -> main
```

2. Perform initial commit

```
git add -A *.java
git commit -m "Initial commit"
git push
```

src/git/init_github.txt



4. Give access to collaborators

- From repo page, go to the `Settings` page → Collaborators.
- Add people by their GitHub usernames.

4. Give access to collaborators

- From repo page, go to the `Settings` page → Collaborators.
- Add people by their GitHub usernames.

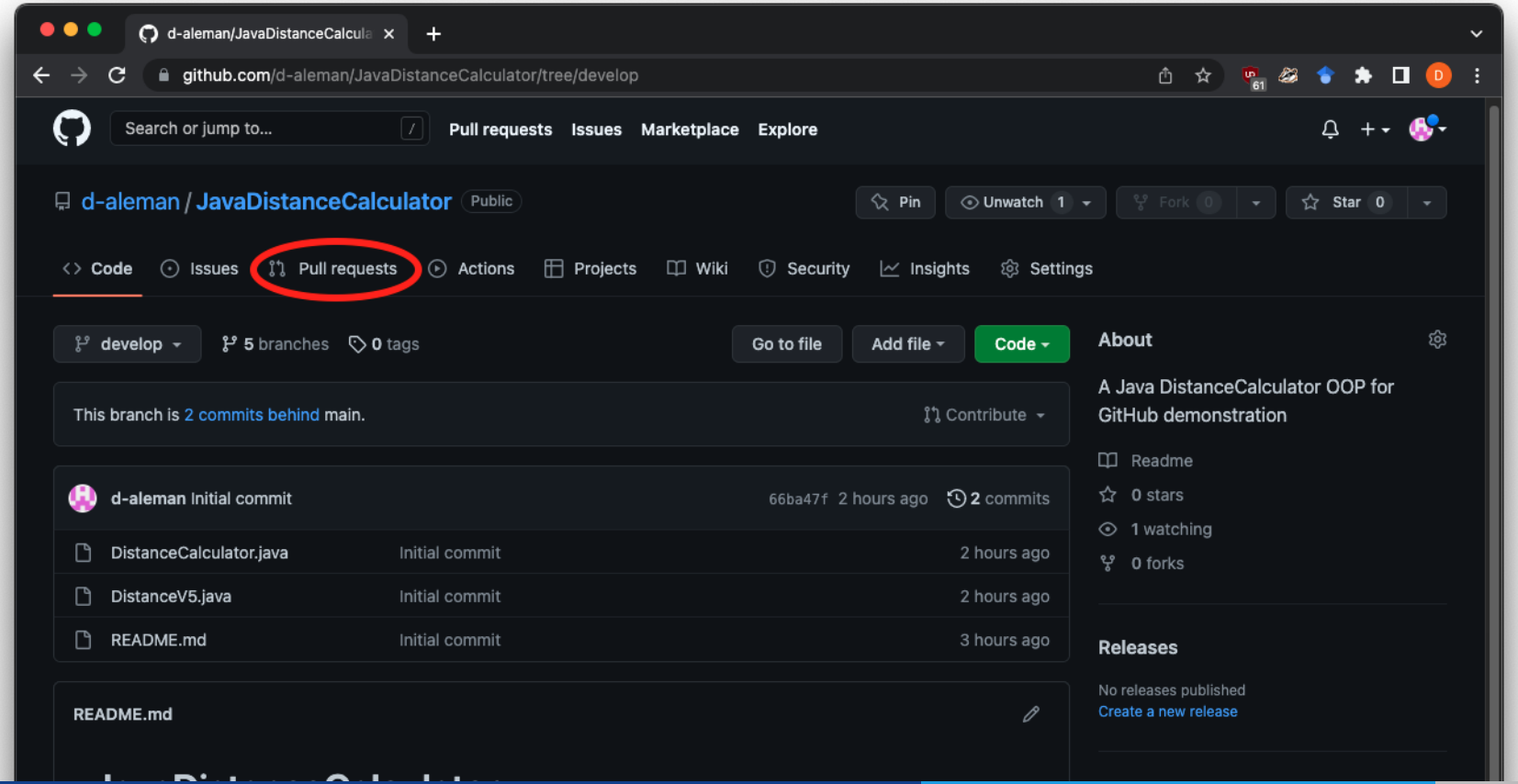
Since other people will see your GitHub username, choose a username wisely. Your username is also in the URL of all your repos.

Merging in GitHub: Best to do it from the website

- Let's merge `inputHandling` into `develop`.
- Go to the Pull Request page and select the branches you want to merge.
- Review changes and then select "Create pull request".
- Leave a useful comment, then follow the prompts to approve the merge.

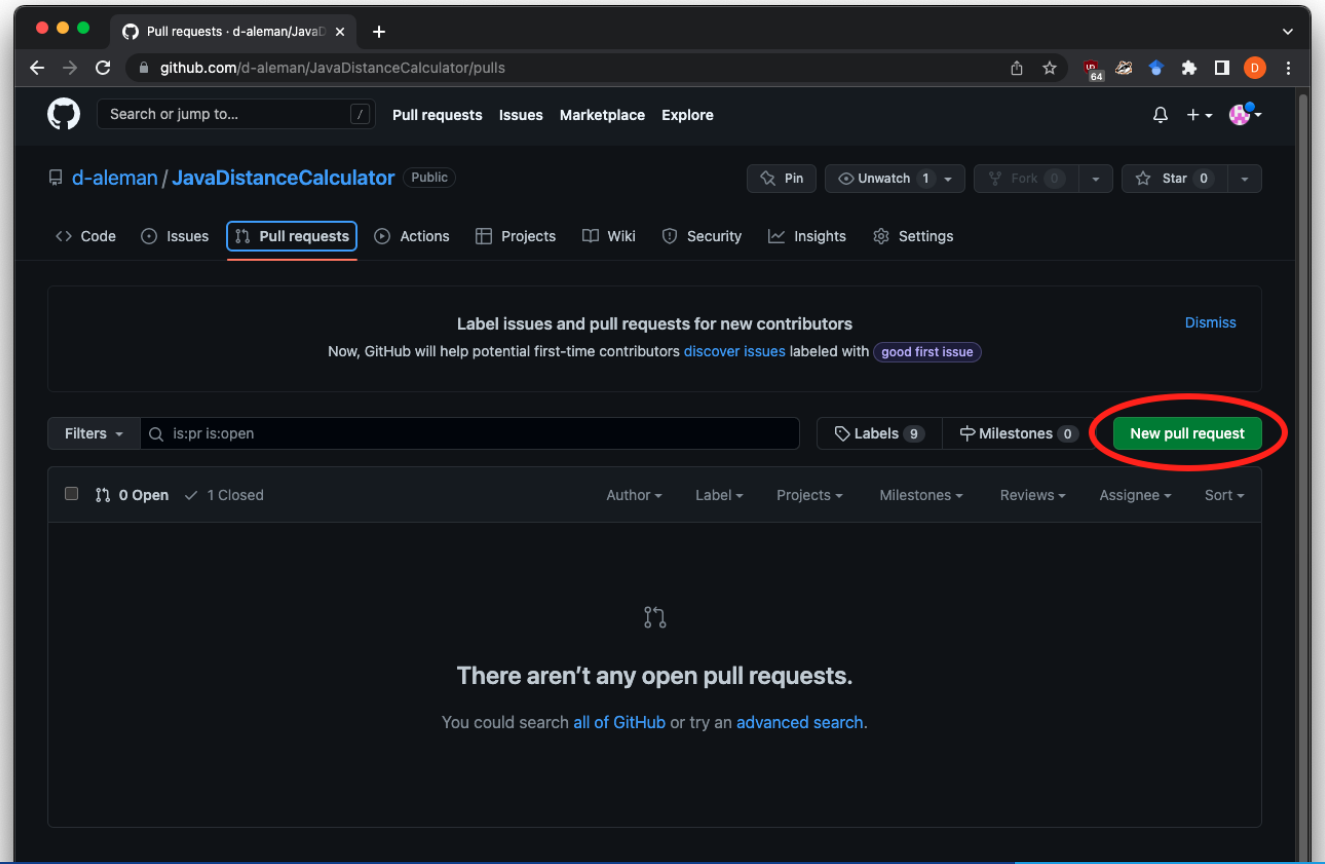
Merging in GitHub: Best to do it from the website

- Let's merge `inputHandling` into `develop`.
- Go to the Pull Request page and select the branches you want to merge.
- Review changes and then select "Create pull request".
- Leave a useful comment, then follow the prompts to approve the merge.



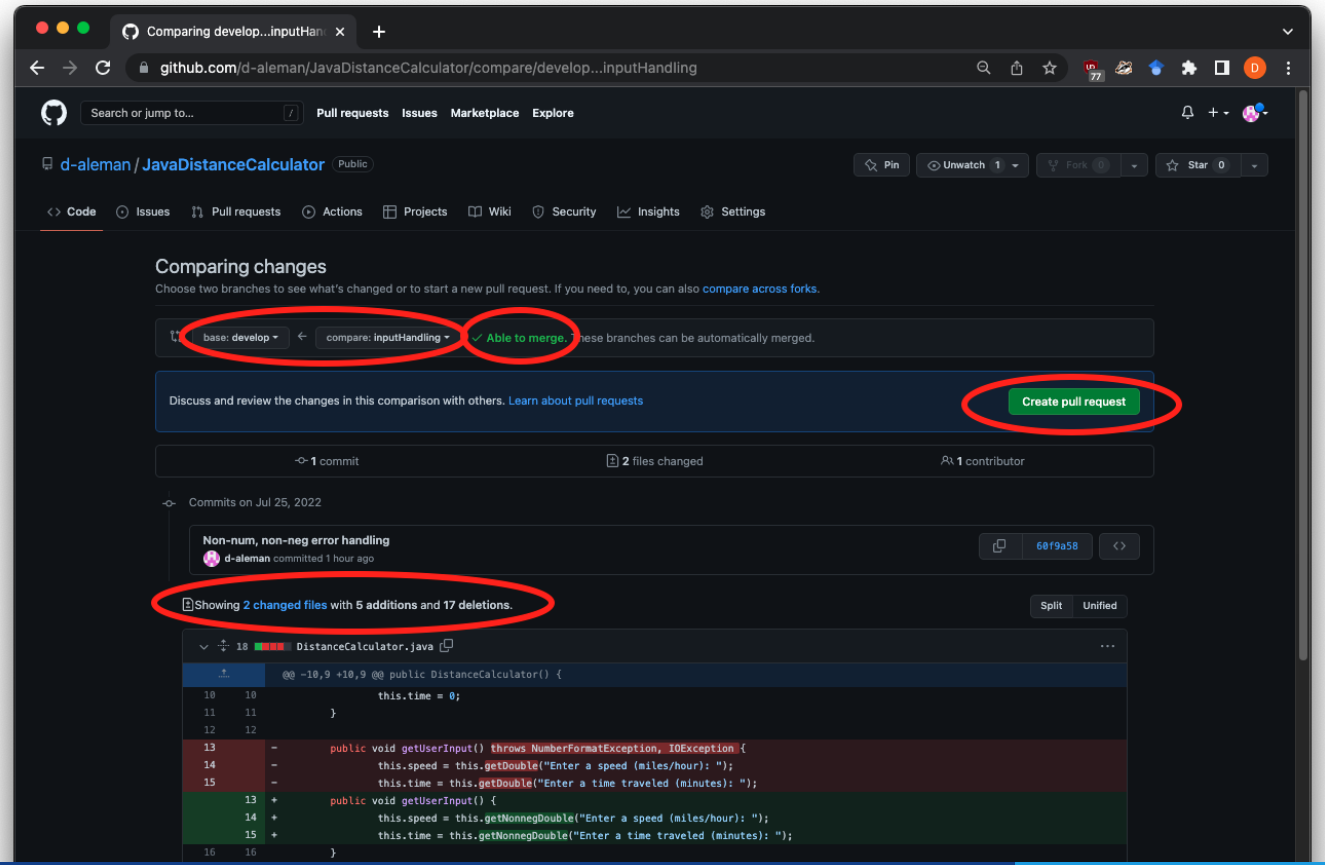
Merging in GitHub: Best to do it from the website

- Let's merge `inputHandling` into `develop`.
- Go to the Pull Request page and select the branches you want to merge.
- Review changes and then select "Create pull request".
- Leave a useful comment, then follow the prompts to approve the merge.



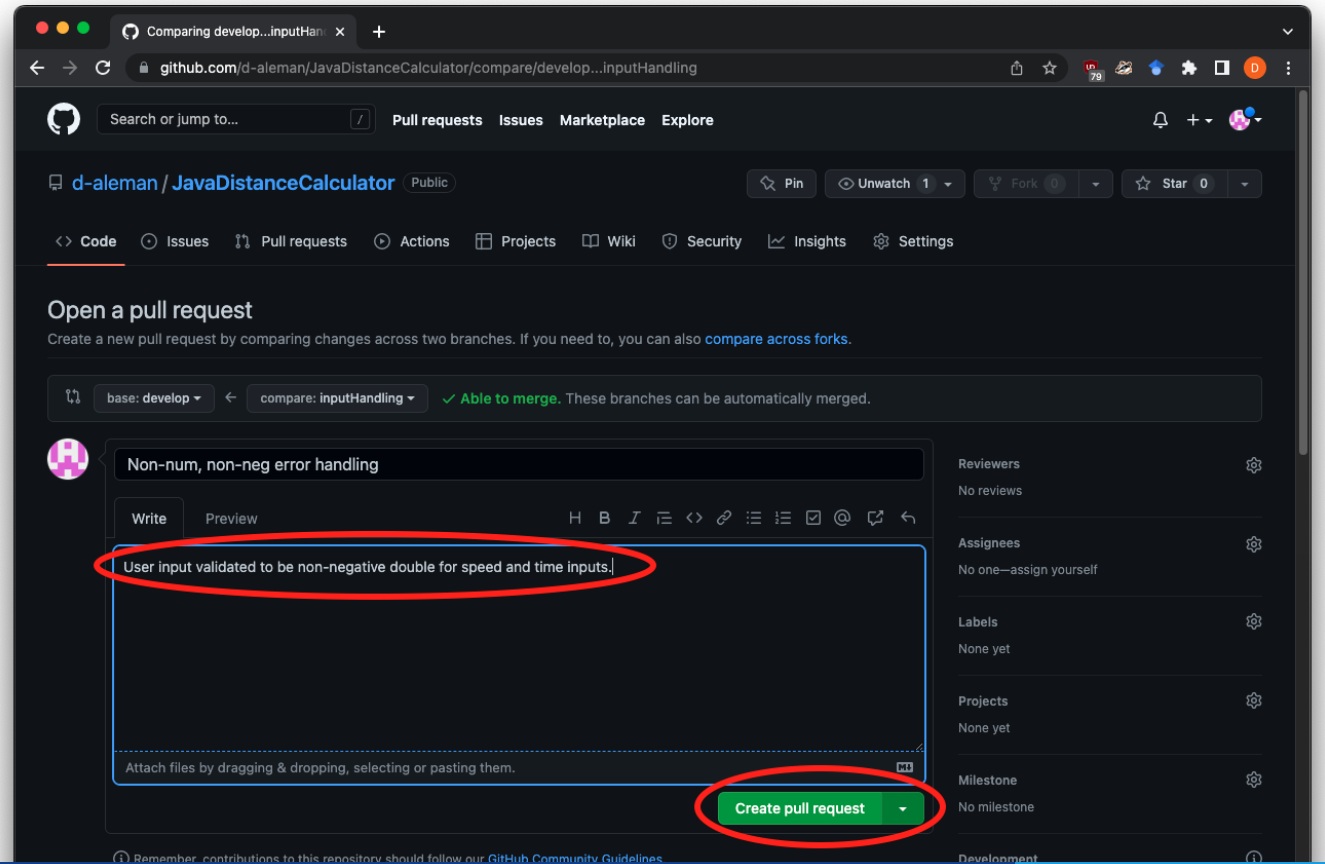
Merging in GitHub: Best to do it from the website

- Let's merge `inputHandling` into `develop`.
- Go to the Pull Request page and select the branches you want to merge.
- Review changes and then select "Create pull request".
- Leave a useful comment, then follow the prompts to approve the merge.



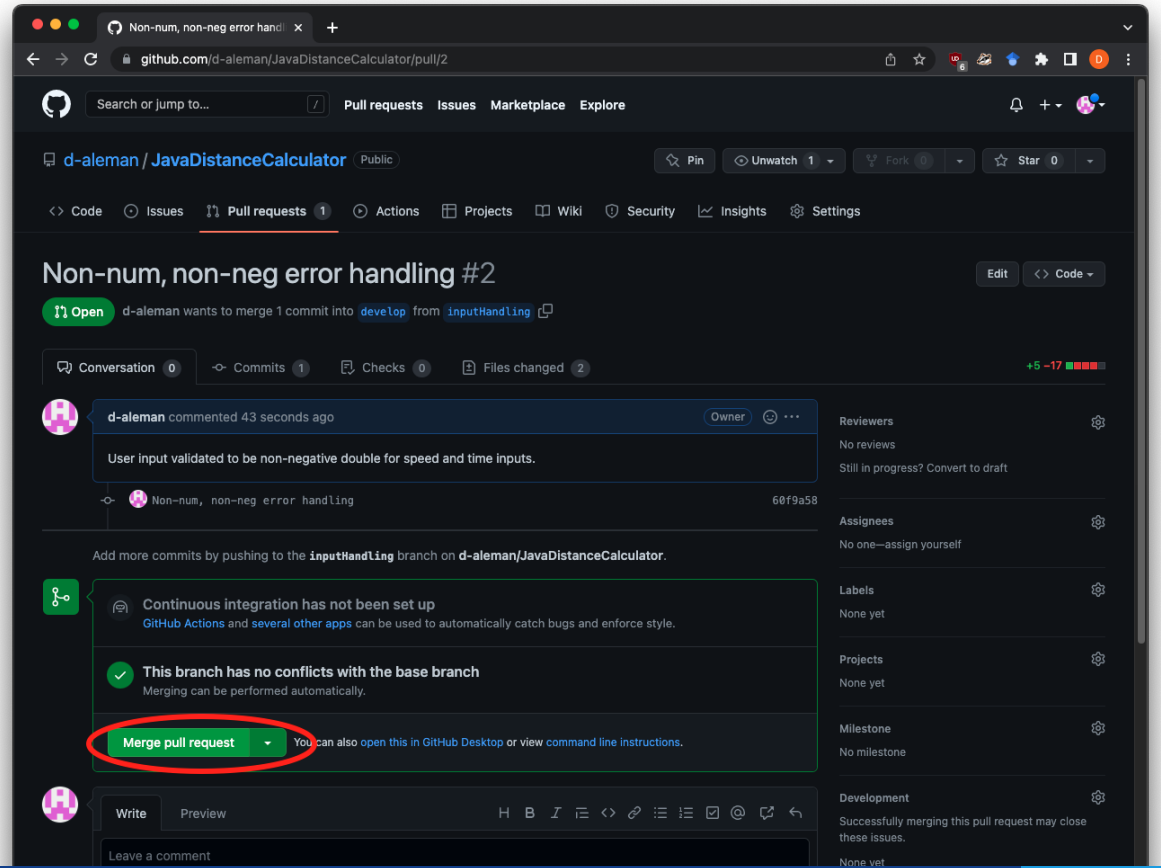
Merging in GitHub: Best to do it from the website

- Let's merge `inputHandling` into `develop`.
- Go to the Pull Request page and select the branches you want to merge.
- Review changes and then select "Create pull request".
- Leave a useful comment, then follow the prompts to approve the merge.



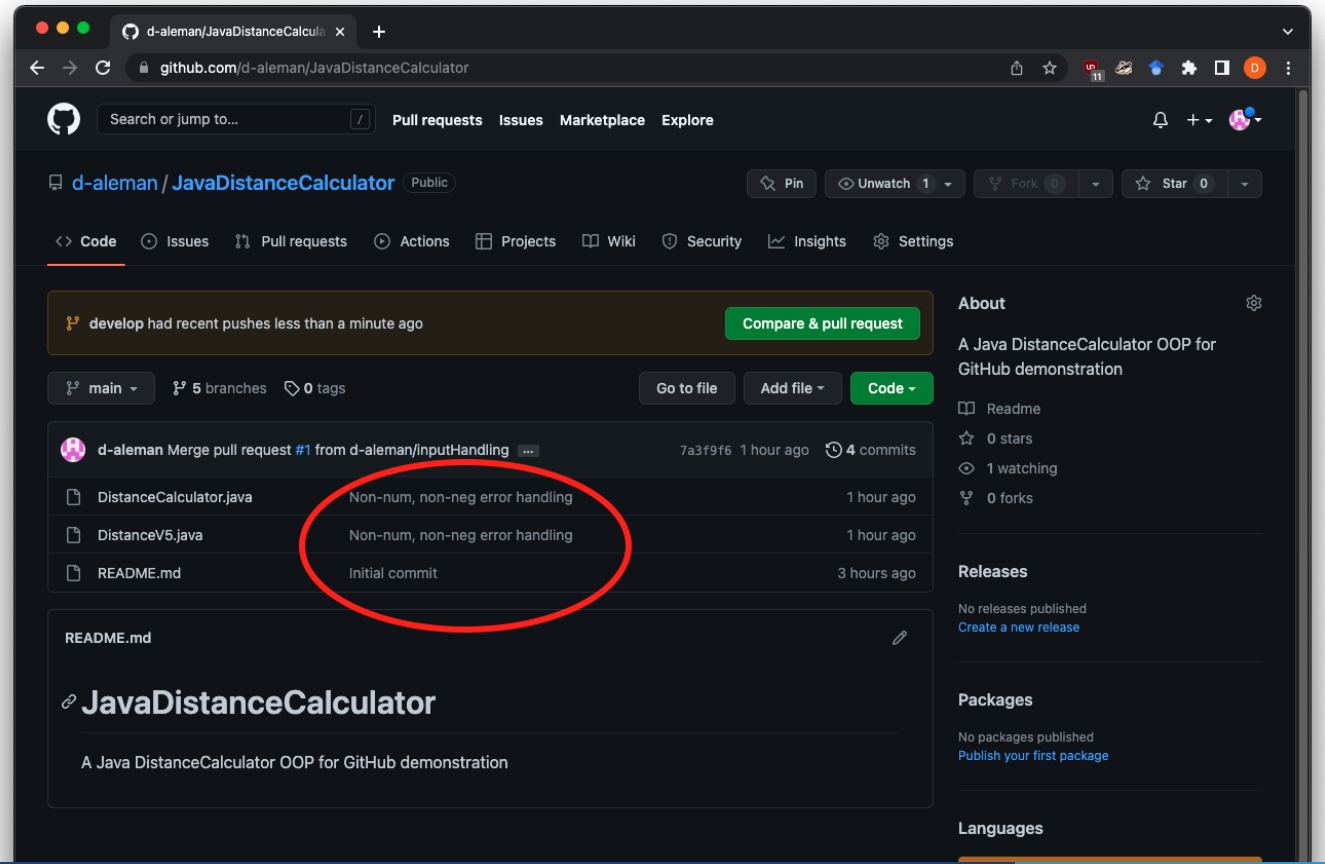
Merging in GitHub: Best to do it from the website

- Let's merge `inputHandling` into `develop`.
- Go to the Pull Request page and select the branches you want to merge.
- Review changes and then select "Create pull request".
- Leave a useful comment, then follow the prompts to approve the merge.



Merging in GitHub: Best to do it from the website

- Let's merge `inputHandling` into `develop`.
- Go to the Pull Request page and select the branches you want to merge.
- Review changes and then select "Create pull request".
- Leave a useful comment, then follow the prompts to approve the merge.



Merge conflicts

- Merges sometimes can't be resolved automatically. Maybe Git can't tell which version is the most up-to-date, or maybe there is a conflict in the code itself.
- Git will tell you which files in the branches are in conflict, and then it is up to you to resolve them.
- A little difficult to do by command line; easier in IDE or GitHub directly.

An example merge conflict

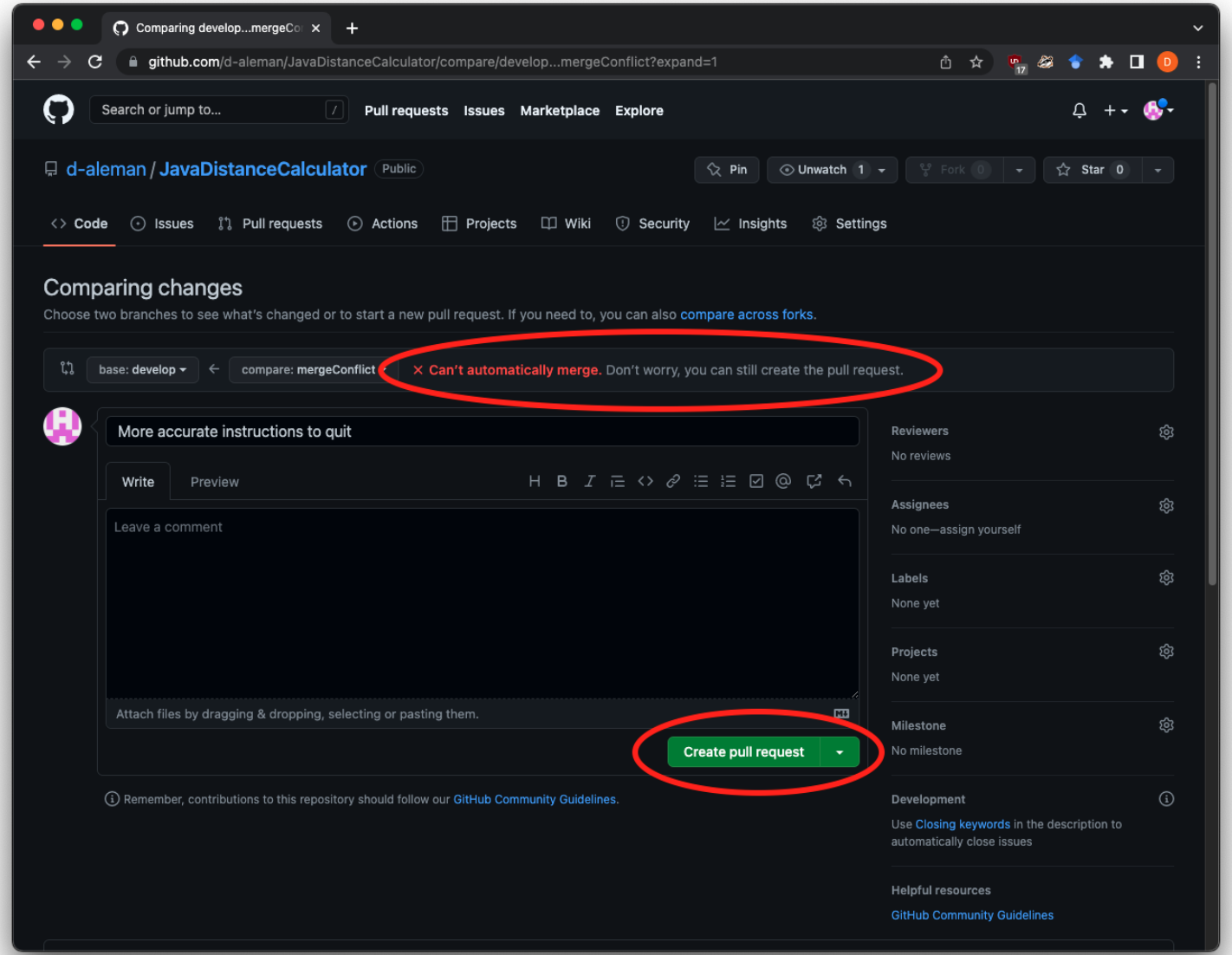
- Create a new branch called `mergeConflict` from `develop`, then update and commit improved instructions to quit on the line that has the instructions.
- In `develop`, update the quitting instructions on the same line to something else.
- Try merging `mergeConflict` into `develop`.

An example merge conflict

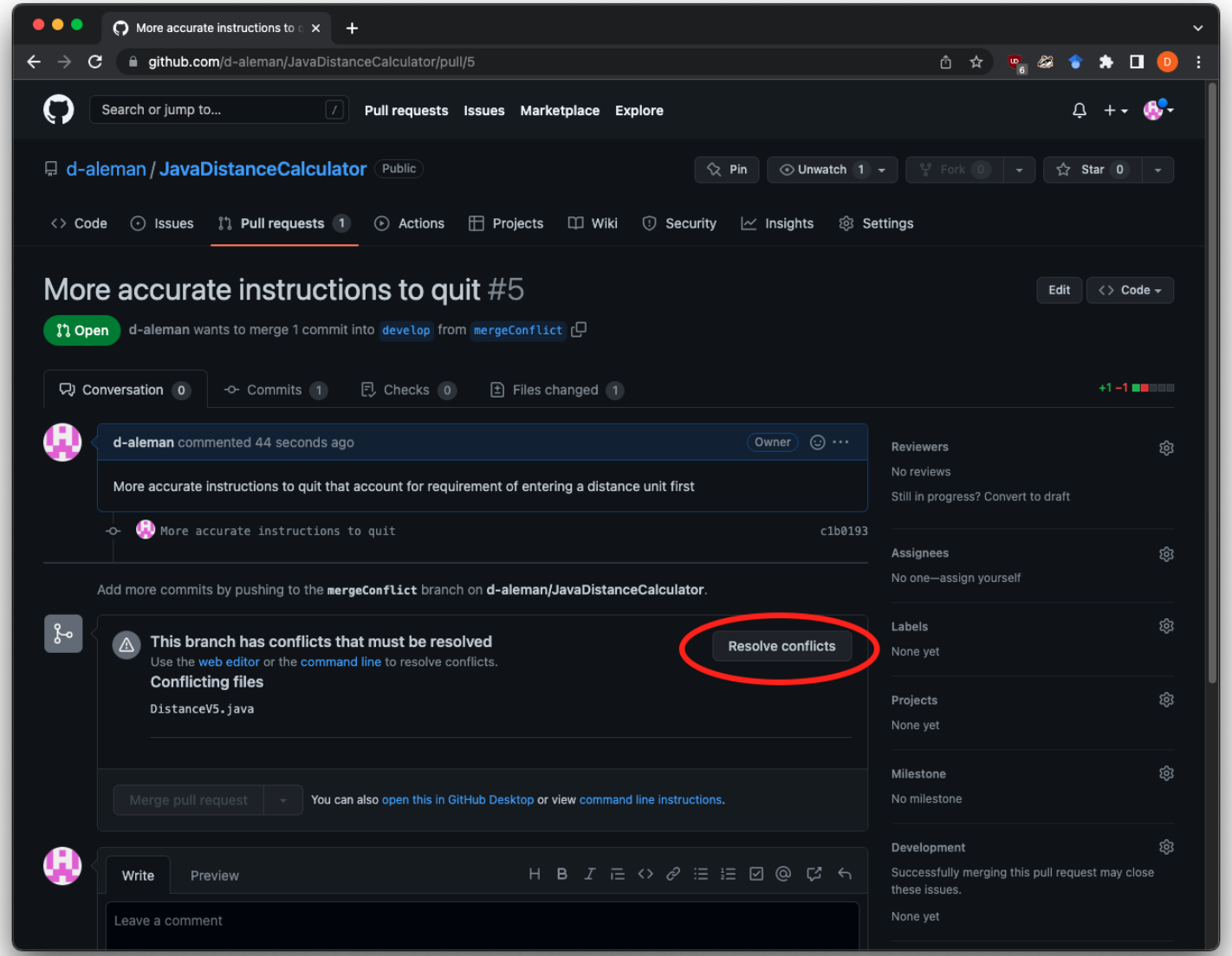
- Create a new branch called `mergeConflict` from `develop`, then update and commit improved instructions to quit on the line that has the instructions.
- In `develop`, update the quitting instructions on the same line to something else.
- Try merging `mergeConflict` into `develop`.

```
DionneMBP:repo dionne$ git checkout develop
Switched to branch 'develop'
DionneMBP:repo dionne$ git merge mergeConflict
Auto-merging DistanceV5.java
CONFLICT (content): Merge conflict in DistanceV5.java
Automatic merge failed; fix conflicts and then commit the result.
```

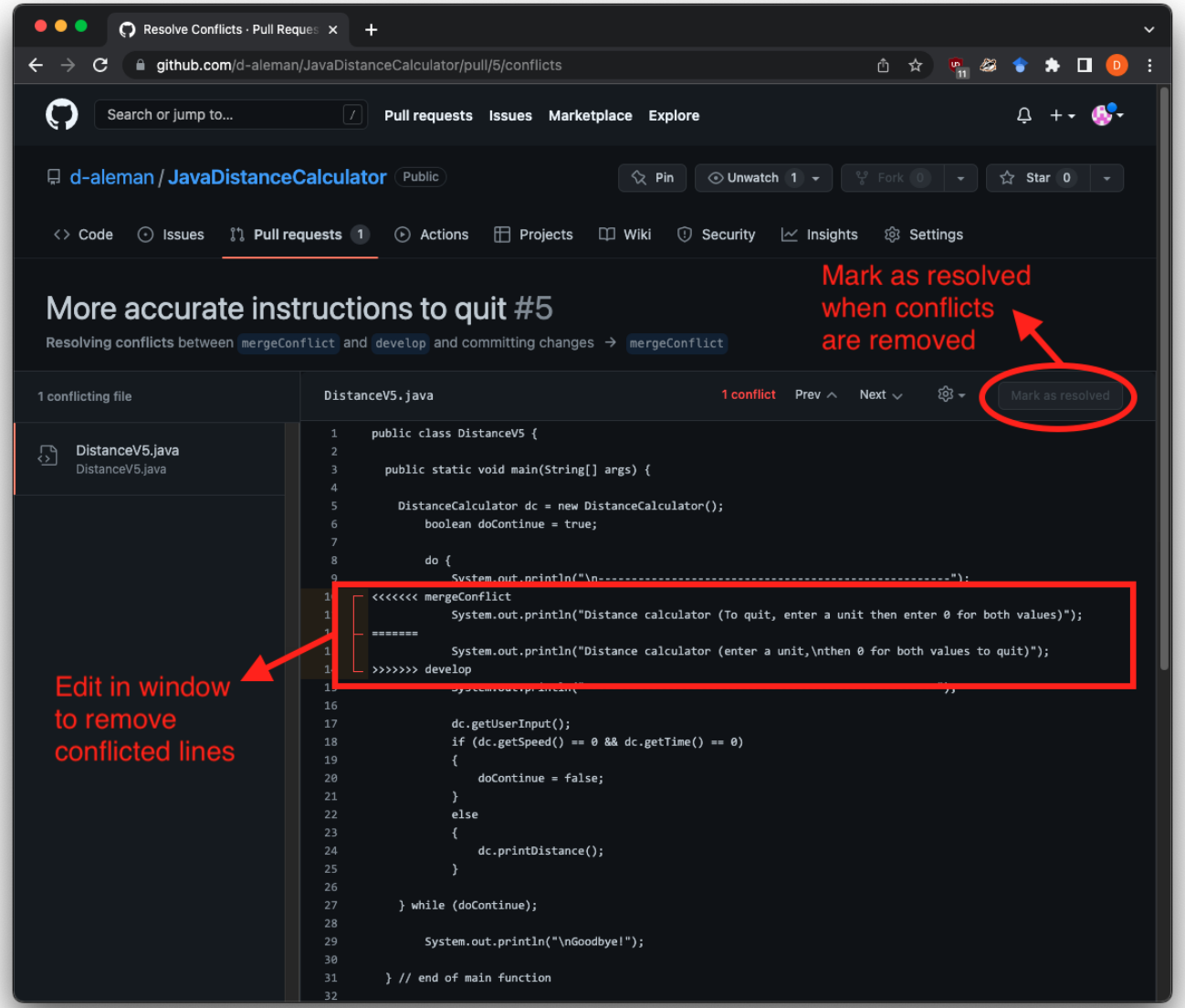
An example merge conflict: In GitHub



An example merge conflict: In GitHub

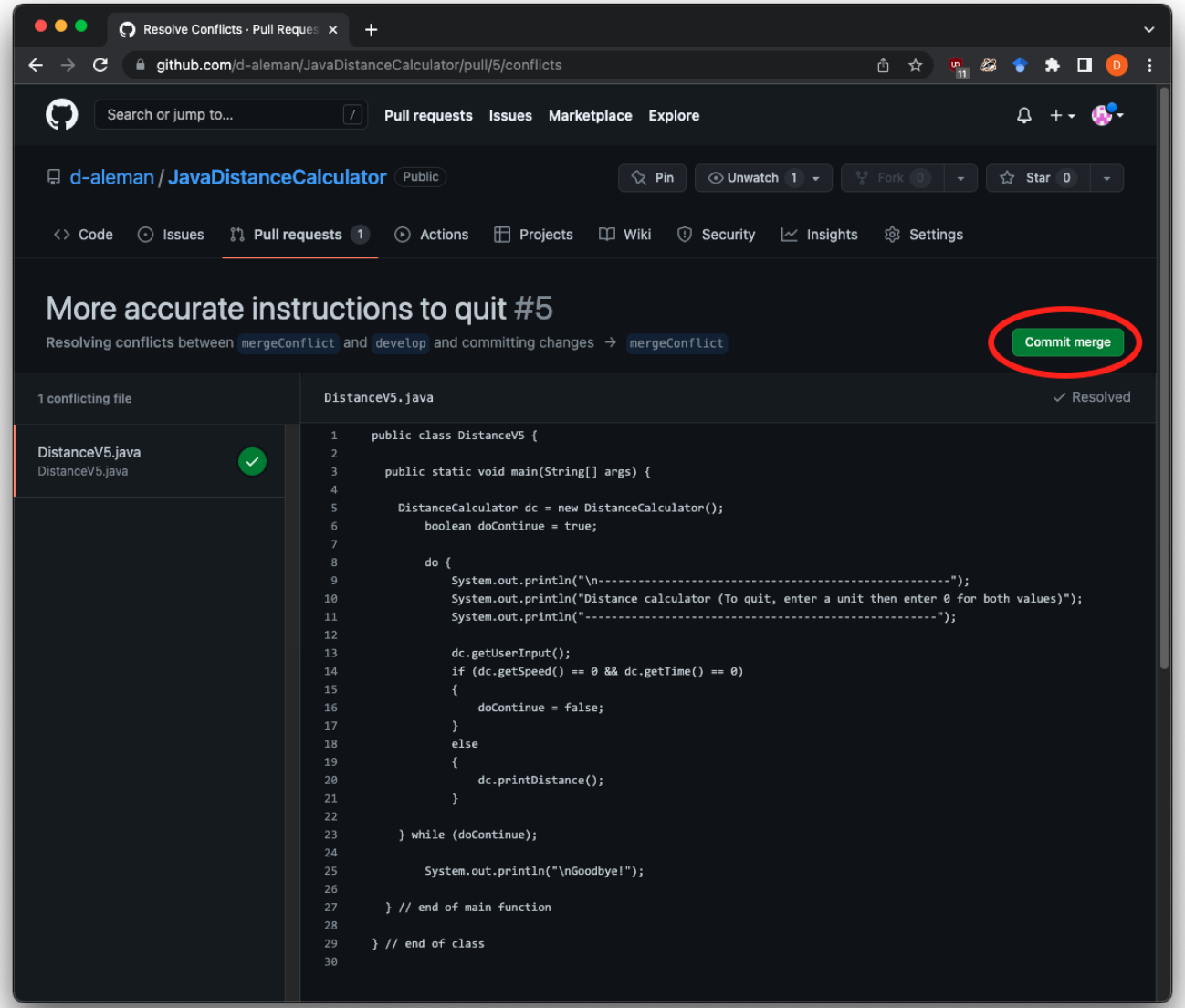


An example merge conflict: In GitHub




```
public class DistanceV5 {

  public static void main(String[] args) {

    DistanceCalculator dc = new DistanceCalculator();
        boolean doContinue = true;

        do {
            System.out.println("\n-----------------------------------------------------");
            dc.getUserInput();
            if (dc.getSpeed() == 0 && dc.getTime() == 0)
            {
                doContinue = false;
            }
            else
            {
                dc.printDistance();
            }

    } while (doContinue);

        System.out.println("\nGoodbye!");

  } // end of main function
```

An example merge conflict: In GitHub



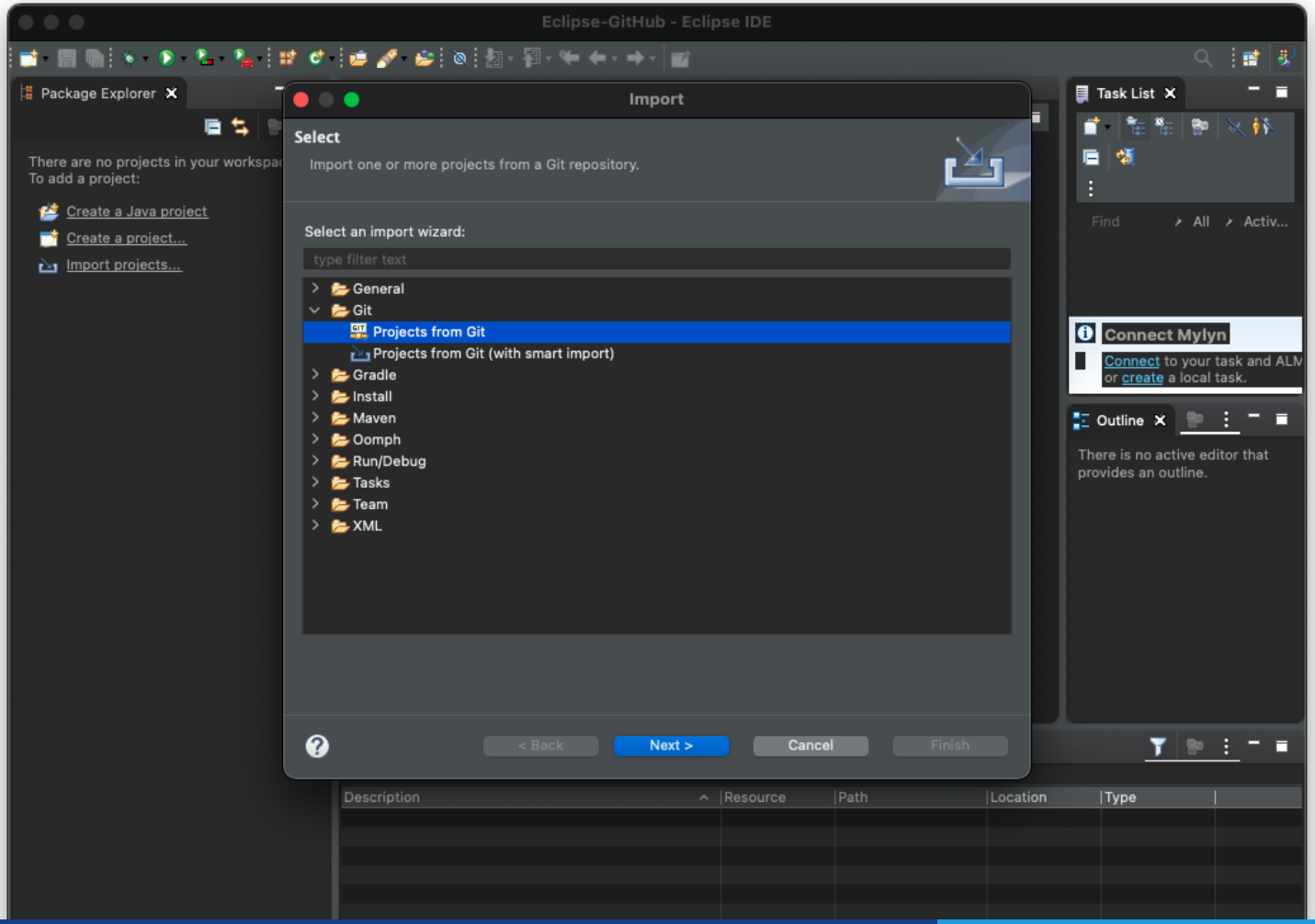
Using GitHub from Eclipse

- Modern IDEs have built-in Git/GitHub integration.
- Can be a massive headache to set up due to GitHub's authentication process[2], but once set up, it is super convenient, and generally allows for easier management of staging commits and pushes and managing merges than exclusive command-line operations.
- The demo:
  1. Create project in GitHub.
  2. Connect Eclipse project to the GitHub project.
  3. Add in keys to authenticate your computer with GitHub.

[2] GitHub's SSH key-based authentication replaces username/password authentication. The new system was launched in late 2021, so don't read any help files or instructions older than late 2021.

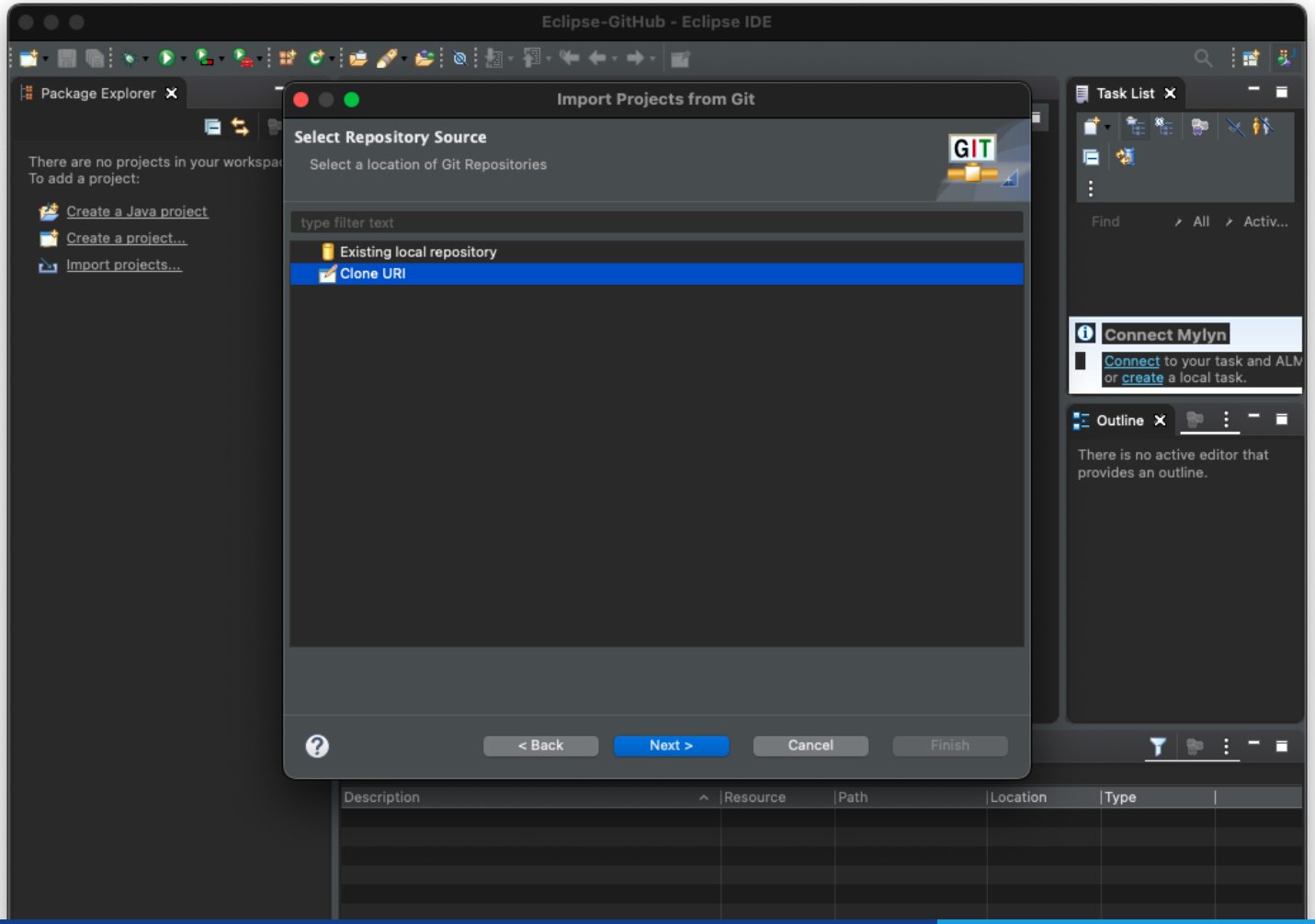
2. Connect Eclipse project to the GitHub project

Go to File → Import... and follow the prompts. Make sure the name of your Eclipse project is the same as the GitHub repo name.



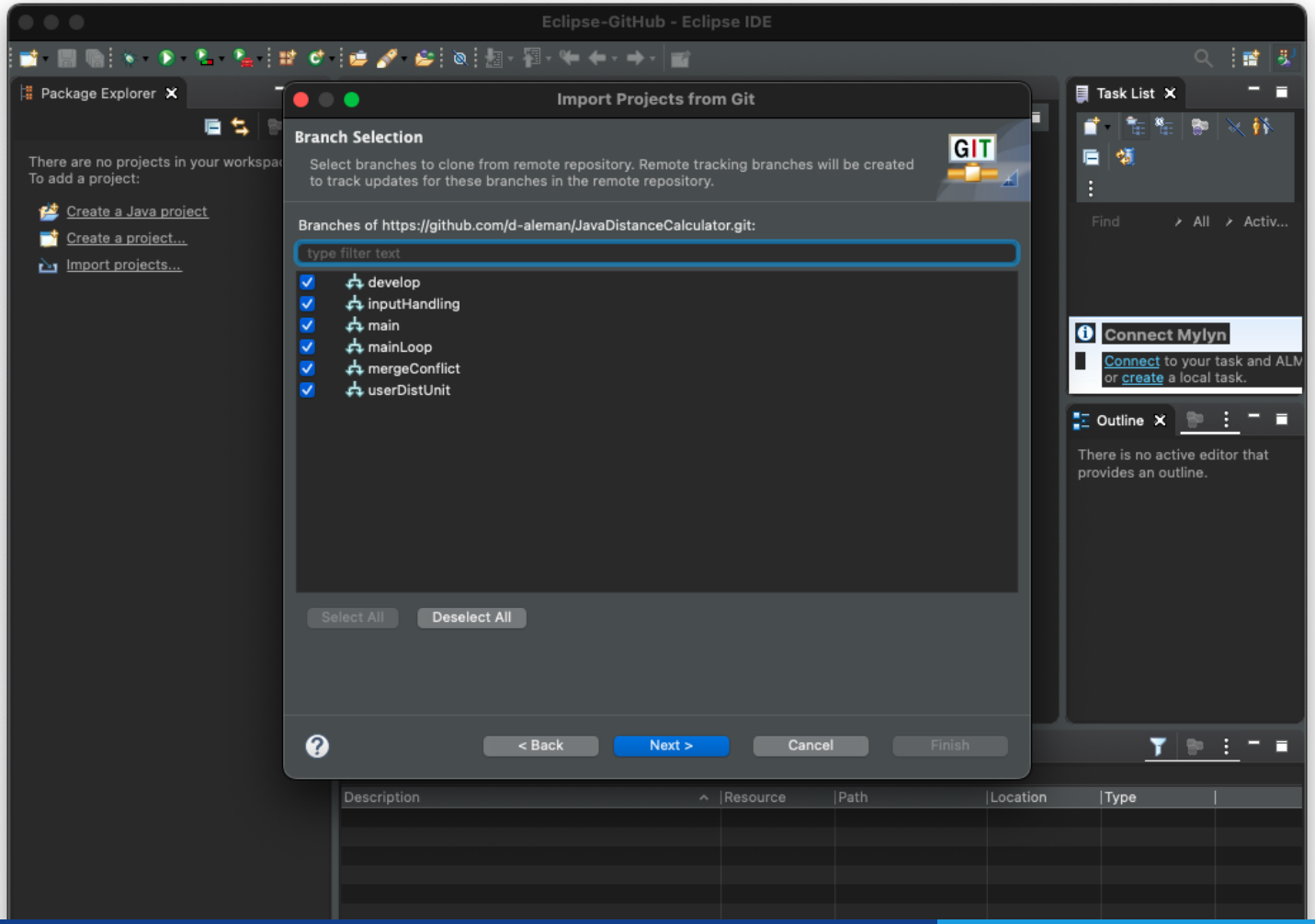
2. Connect Eclipse project to the GitHub project

Go to File → Import... and follow the prompts. Make sure the name of your Eclipse project is the same as the GitHub repo name.



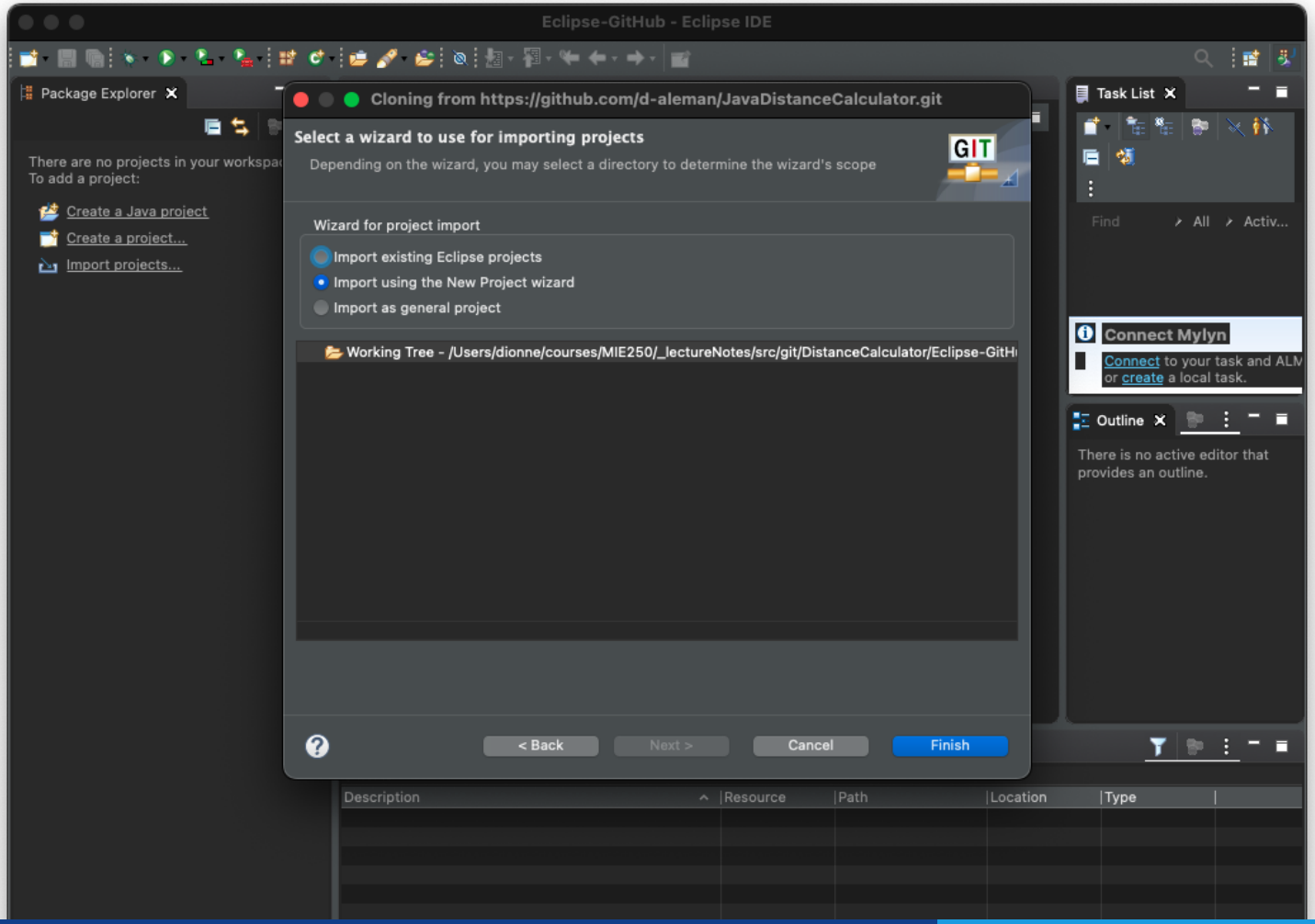
2. Connect Eclipse project to the GitHub project

Go to File → Import... and follow the prompts. Make sure the name of your Eclipse project is the same as the GitHub repo name.



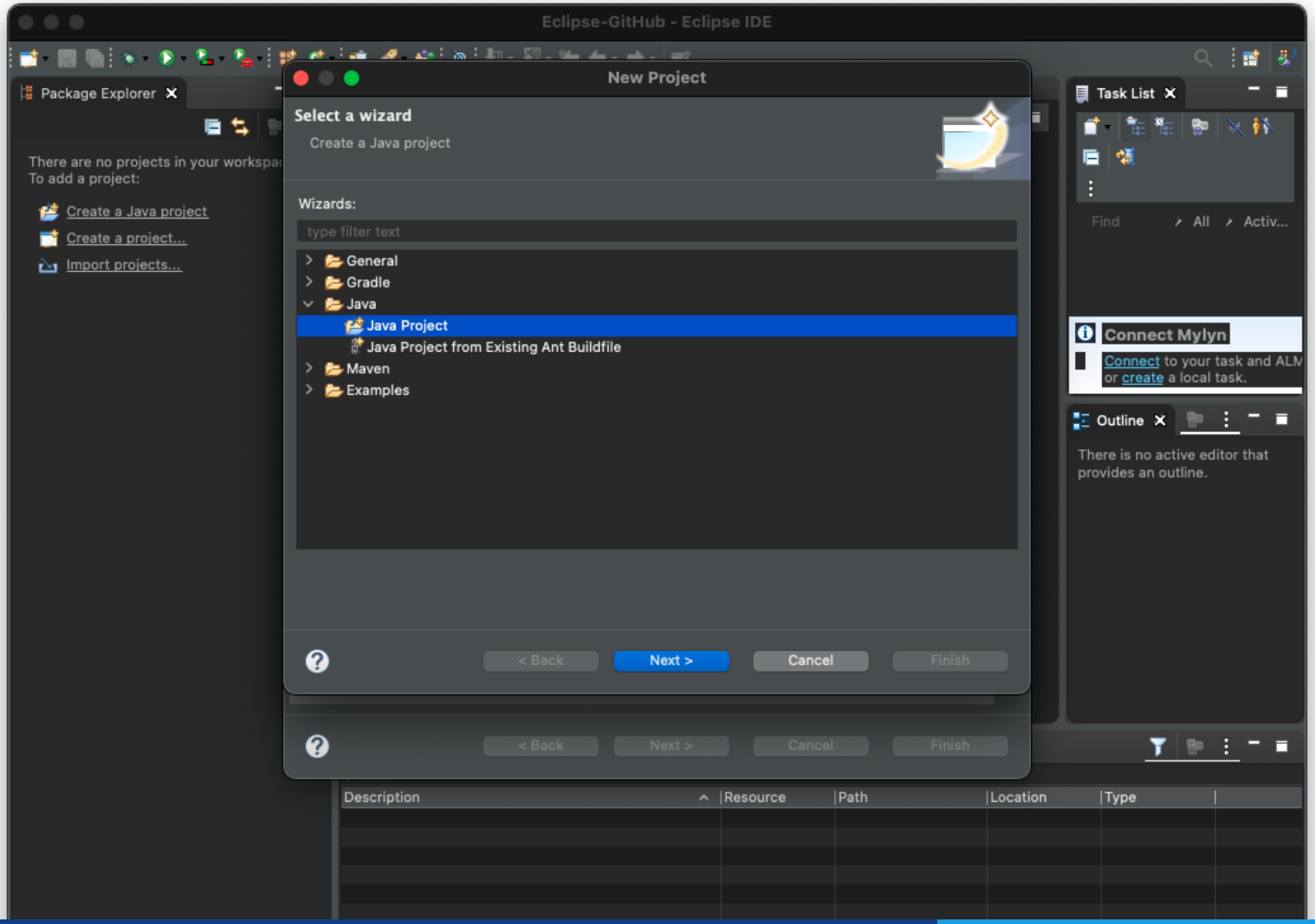
2. Connect Eclipse project to the GitHub project

Go to File → Import... and follow the prompts. Make sure the name of your Eclipse project is the same as the GitHub repo name.



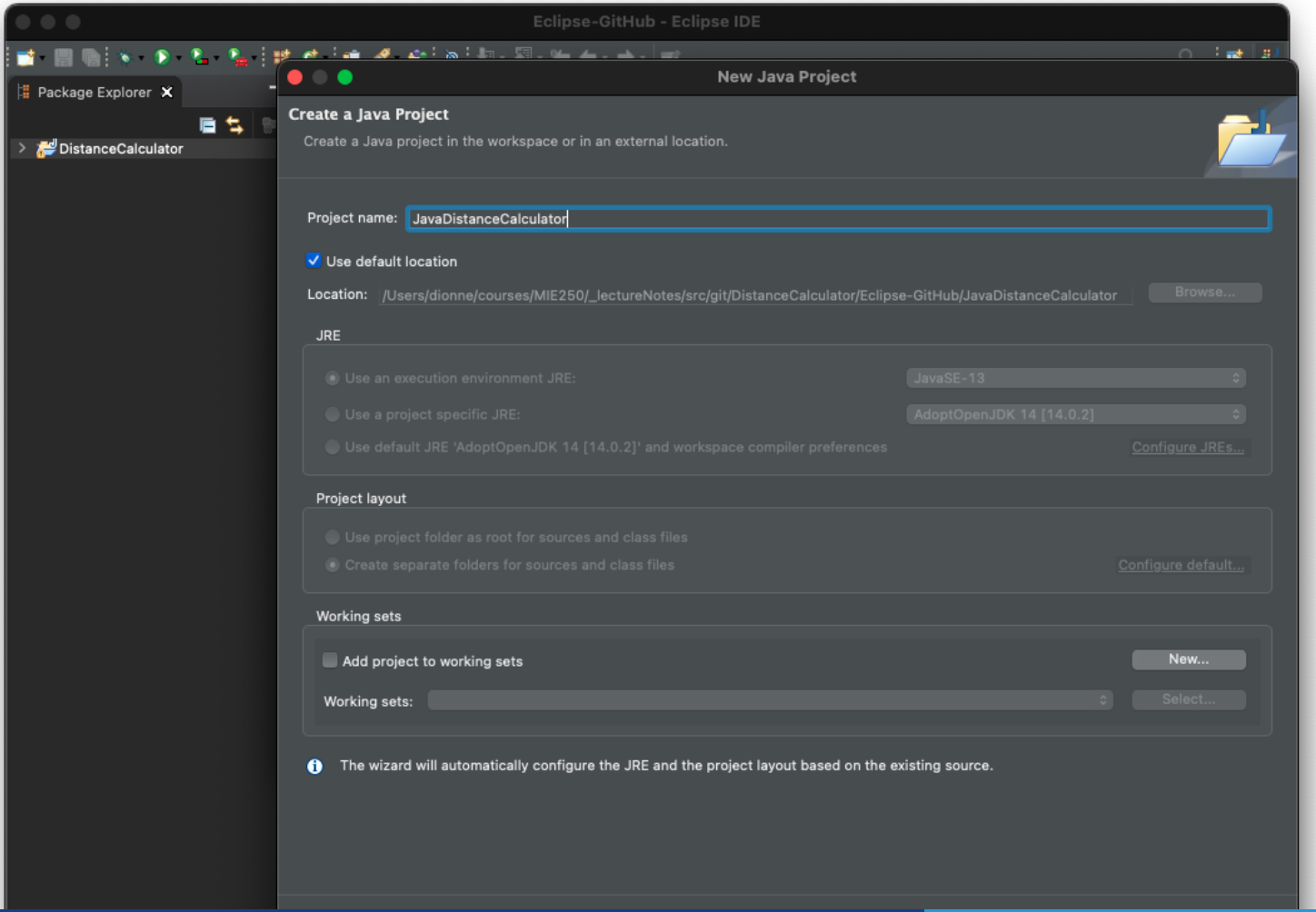
2. Connect Eclipse project to the GitHub project

Go to File → Import... and follow the prompts. Make sure the name of your Eclipse project is the same as the GitHub repo name.



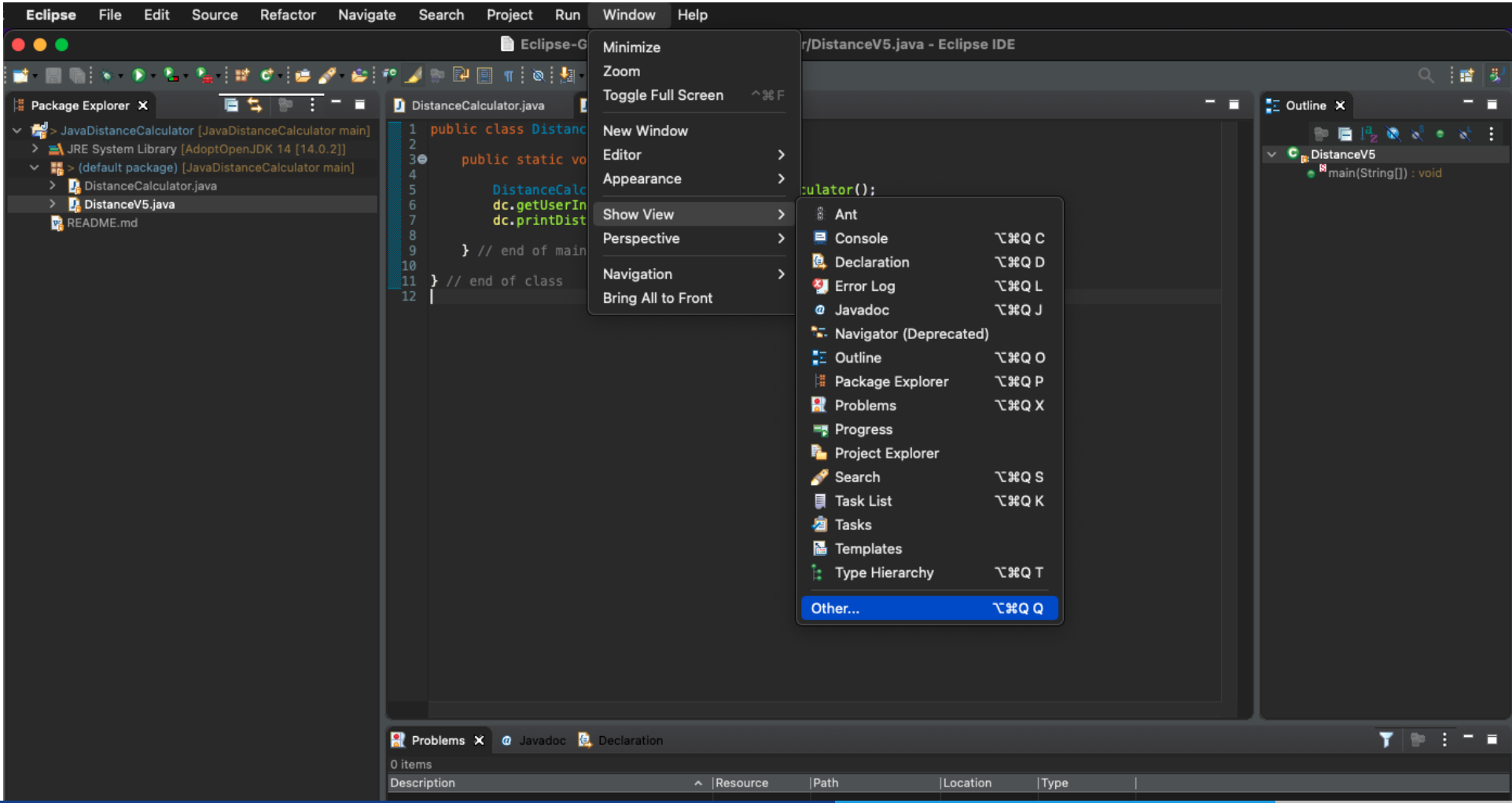
2. Connect Eclipse project to the GitHub project

Go to File → Import... and follow the prompts. Make sure the name of your Eclipse project is the same as the GitHub repo name.



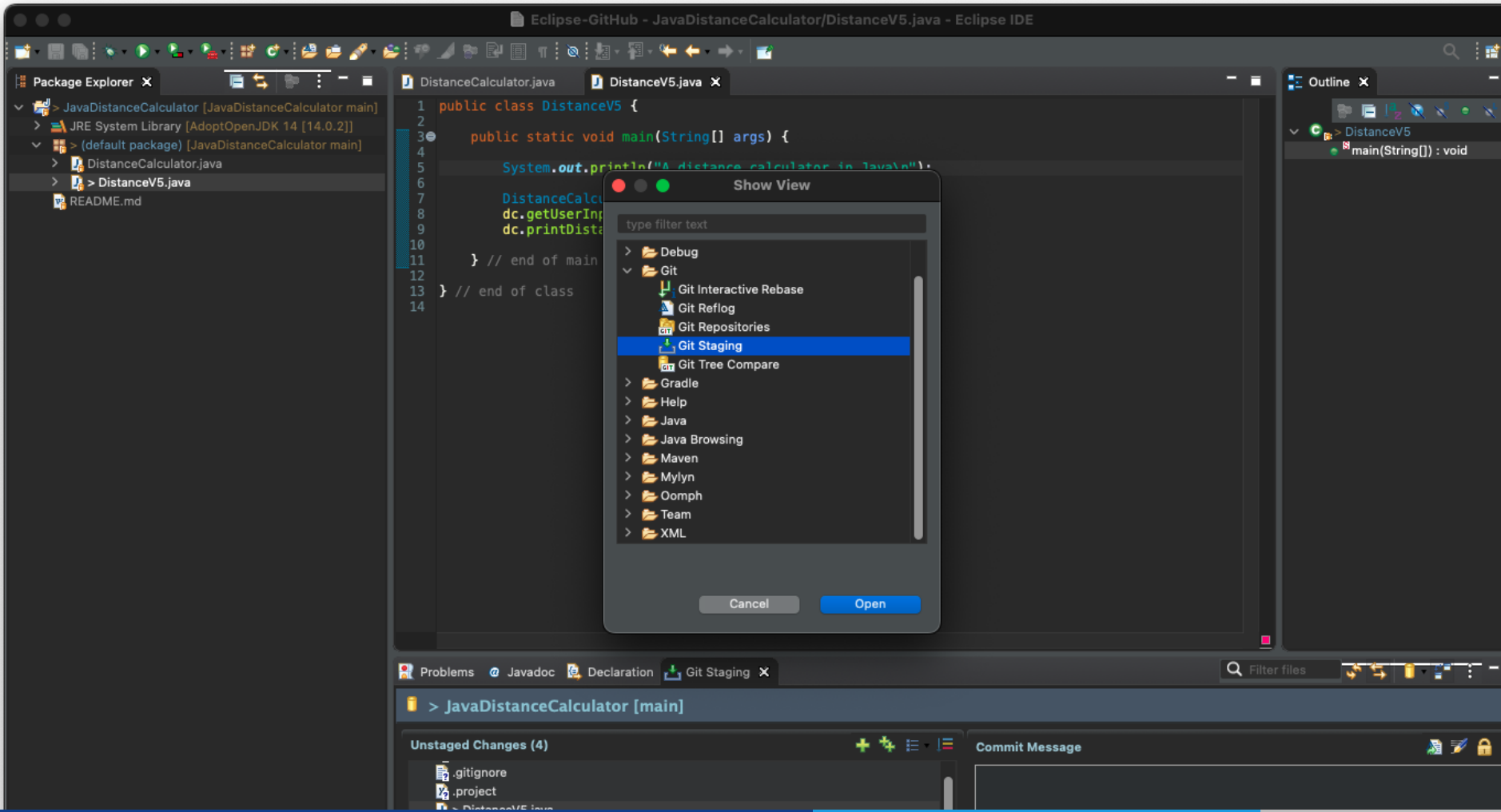
2a. Show Git options in Eclipse

Go to Window → Show view → Other... and select Git Staging and any other desired views.



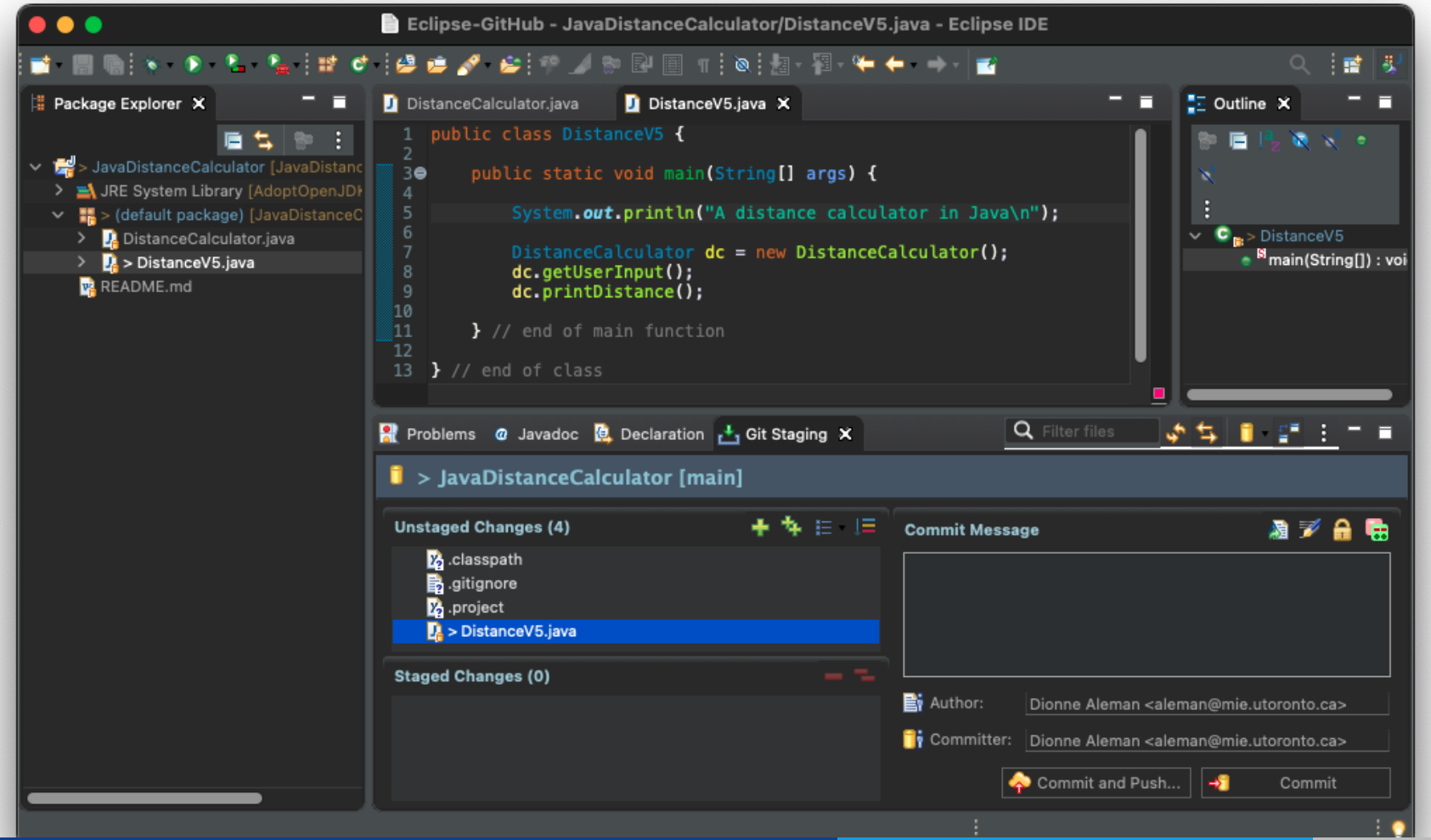
2a. Show Git options in Eclipse

Go to Window → Show view → Other... and select Git Staging and any other desired views.



2a. Show Git options in Eclipse

Go to Window → Show view → Other... and select Git Staging and any other desired views.



Before making any changes to the code ...

- At this moment, if you try to push any changes, Eclipse will ask for your GitHub username and password. But, GitHub will reject them.
- You must use GitHub's key-based authentication instead of username/password[3]:
  - Generate personal access token (PAT) in GitHub.
  - Create SSH key on your computer[4].
  - Connect your key to your GitHub profile.
- Then, re-configure Eclipse to use the keys.

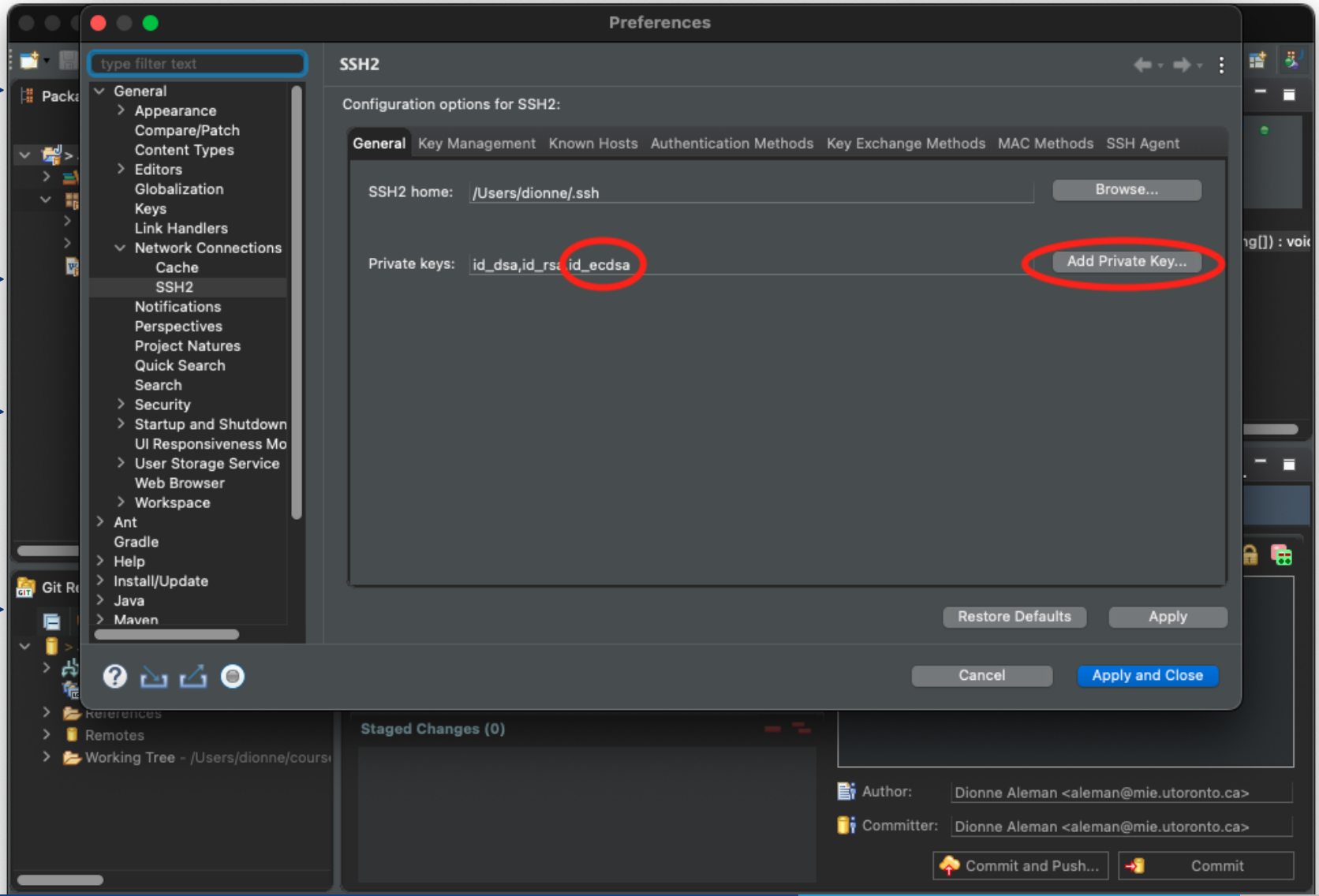
[3] See earlier instructions in these notes.

[4] You CANNOT use Eclipse's built-in key generator since something changed in GitHub recently that no longer allows Eclipse-generated RSA keys. This hiccup may change in the future, but if you encounter SHA-1 key errors when you try to push changes, use the command-line method shown earlier.

Re-configure Eclipse to use the GitHub keys

- Add `id_ecdsa.pub` to Eclipse: Preferences → Network Connections → SSH2 → General tab → private keys list.
- Enable the Git repositories view (Window → Show view → Other...).
- Configure repo to use the PAT as the password (username blank).
  - Right-click remote origin branch → Configure Push.
- Update repo URL to `git@github.com:<username>/<repo>.git`
  - Right-click main repo name → Properties.

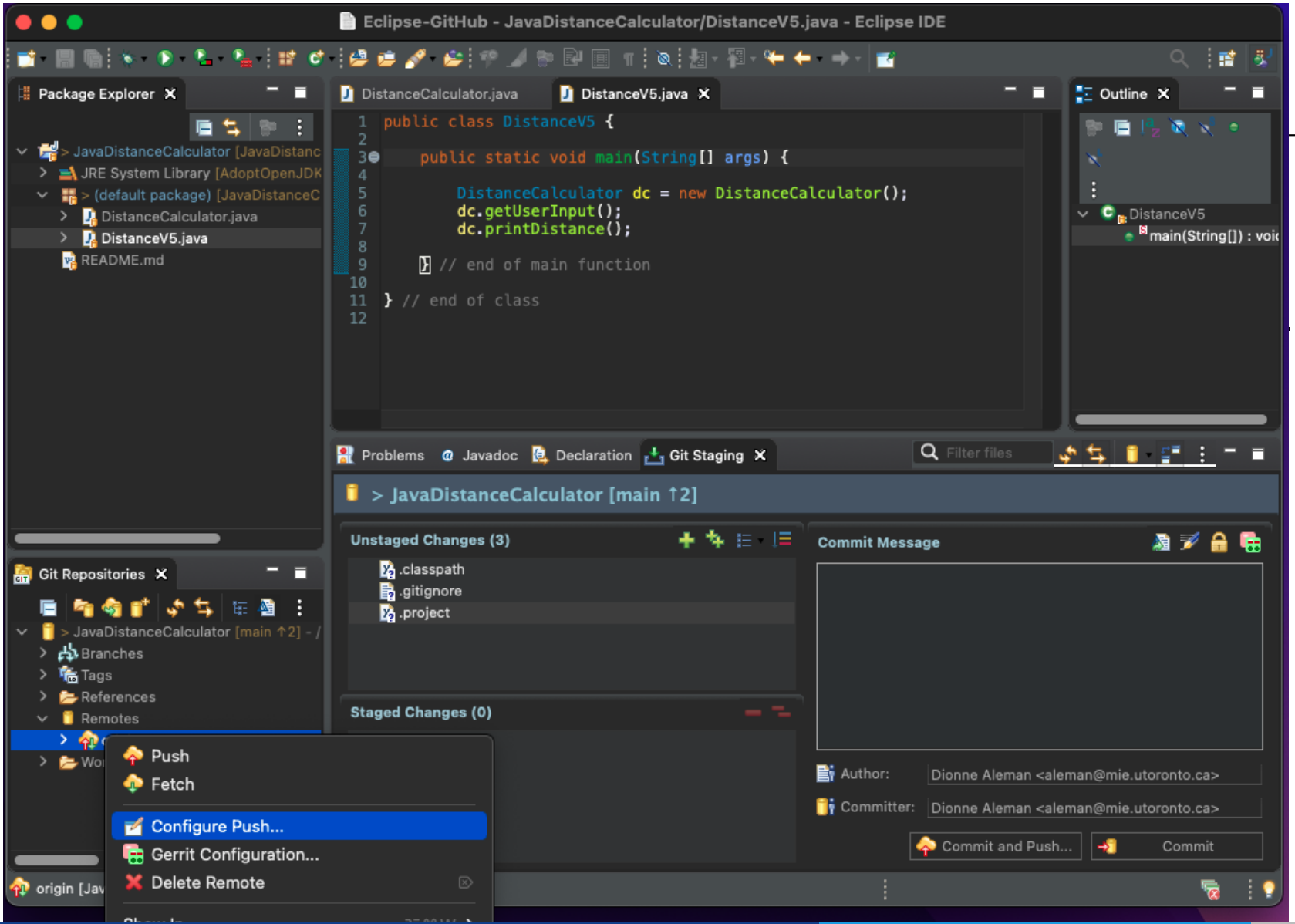
Re-configure Eclipse to use the GitHub keys



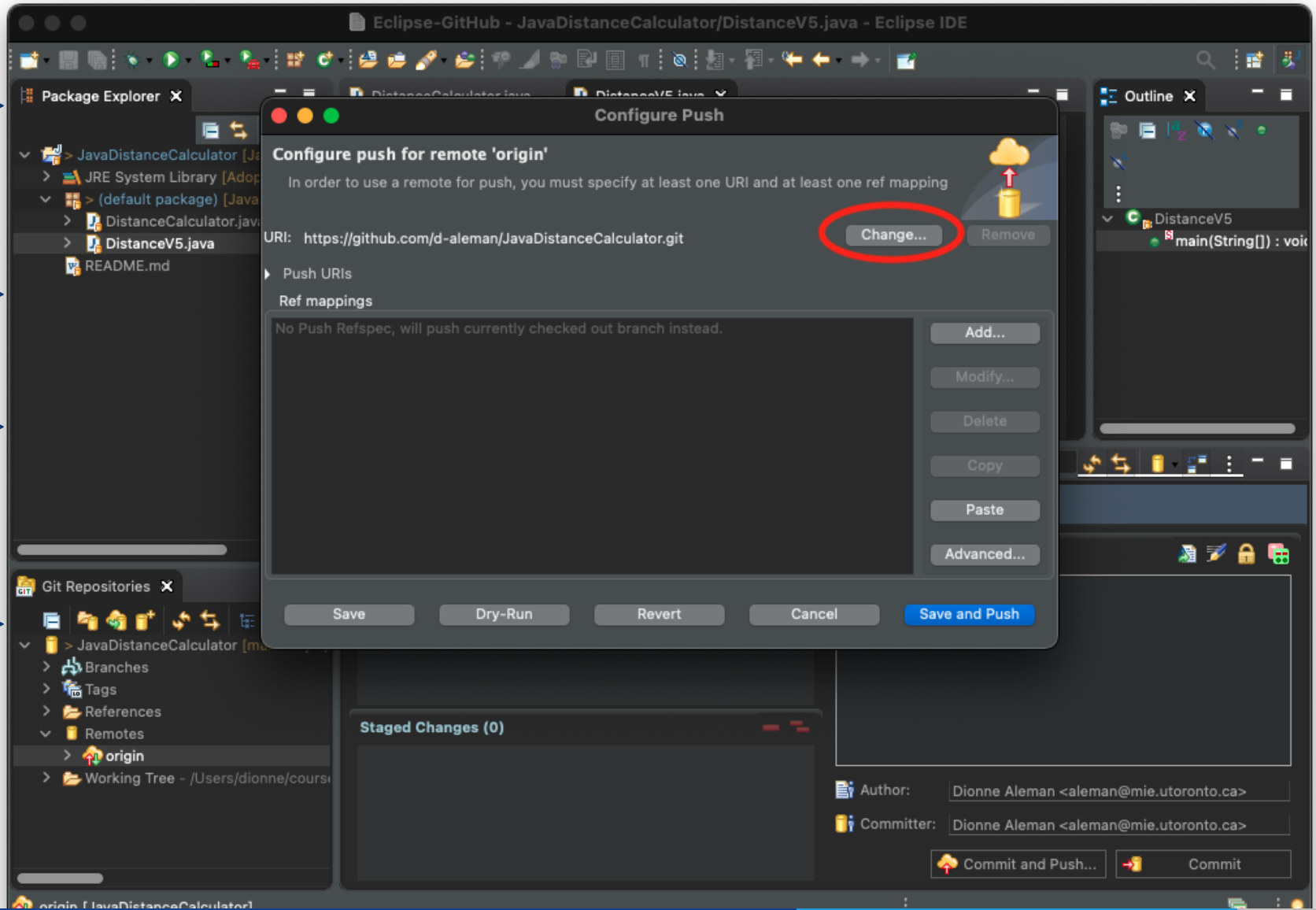
Re-configure Eclipse to use the GitHub keys

- Add `id_ecdsa.pub` to Eclipse: Preferences → Network Connections → SSH2 → General tab → private keys list.
- Enable the Git repositories view (Window → Show view → Other...).
- Configure repo to use the PAT as the password (username blank).
  - Right-click remote origin branch → Configure Push.
- Update repo URL to `git@github.com:<username>/<repo>.git`
  - Right-click main repo name → Properties.

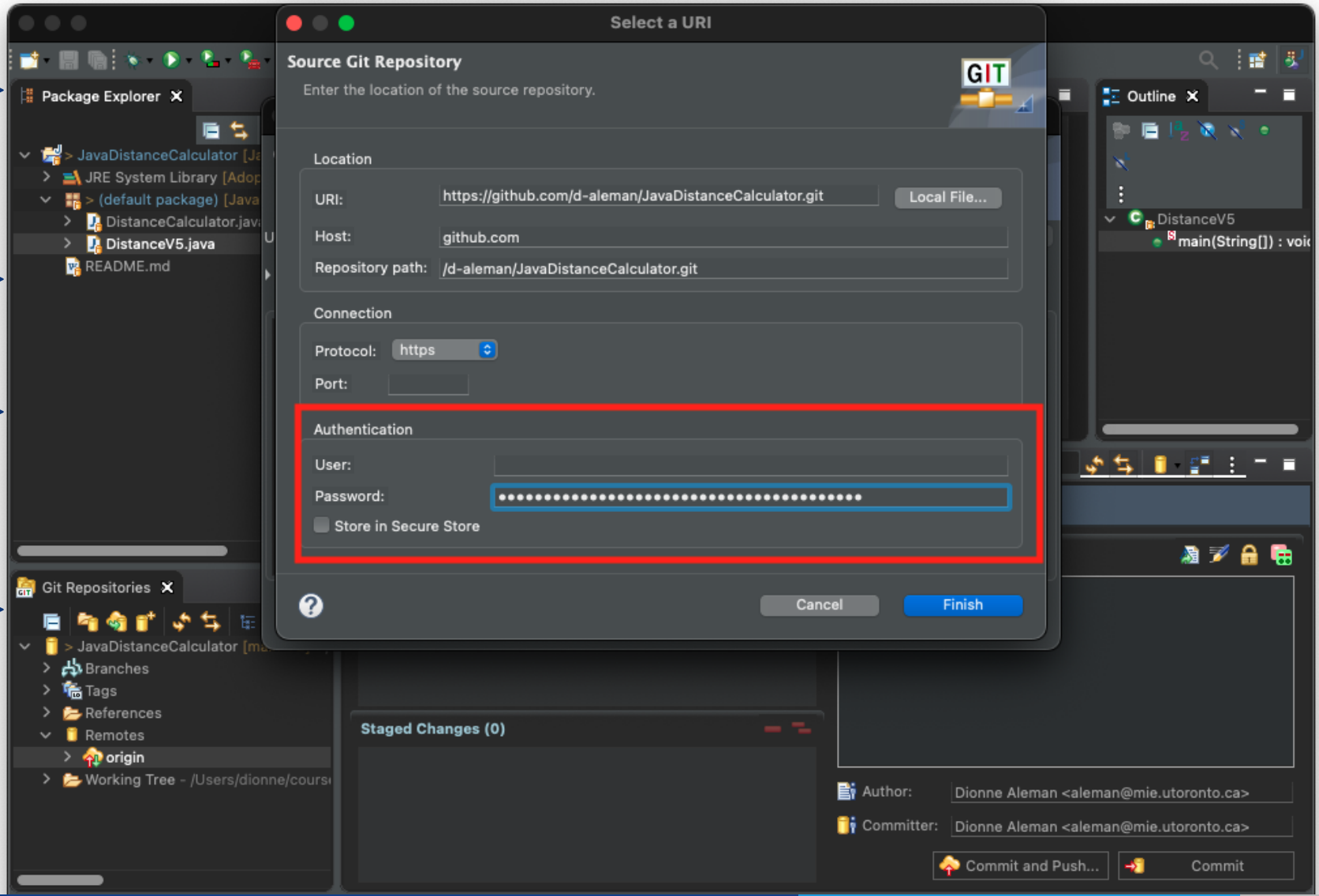
Re-configure Eclipse to use the GitHub keys



Re-configure Eclipse to use the GitHub keys



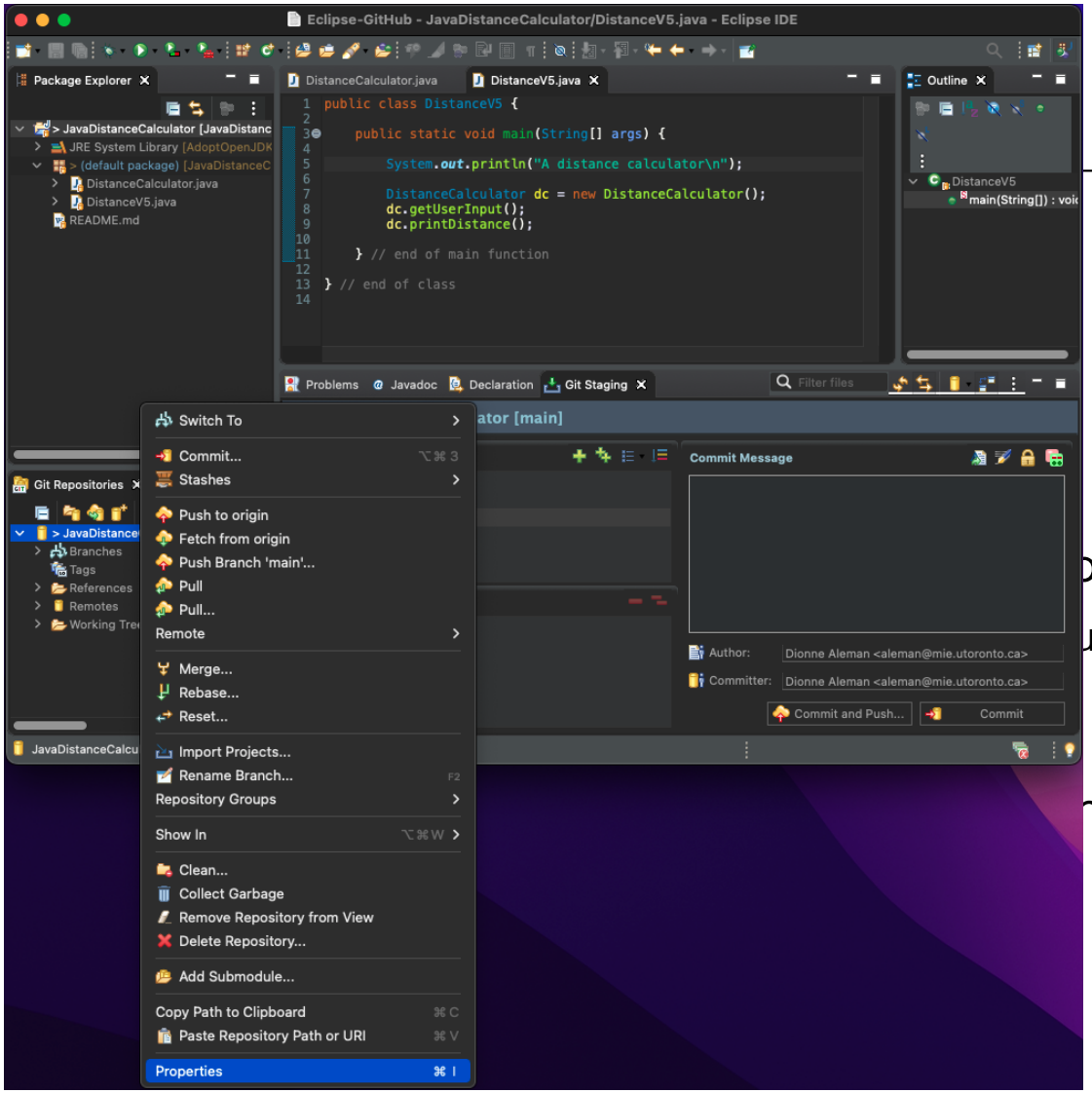
Re-configure Eclipse to use the GitHub keys



Re-configure Eclipse to use the GitHub keys

- Add `id_ecdsa.pub` to Eclipse: Preferences → Network Connections → SSH2 → General tab → private keys list.
- Enable the Git repositories view (Window → Show view → Other...).
- Configure repo to use the PAT as the password (username blank).
  - Right-click remote origin branch → Configure Push.
- Update repo URL to `git@github.com:<username>/<repo>.git`
  - Right-click main repo name → Properties.

Re-configure Eclipse to use the GitHub keys



→ Network Connections →

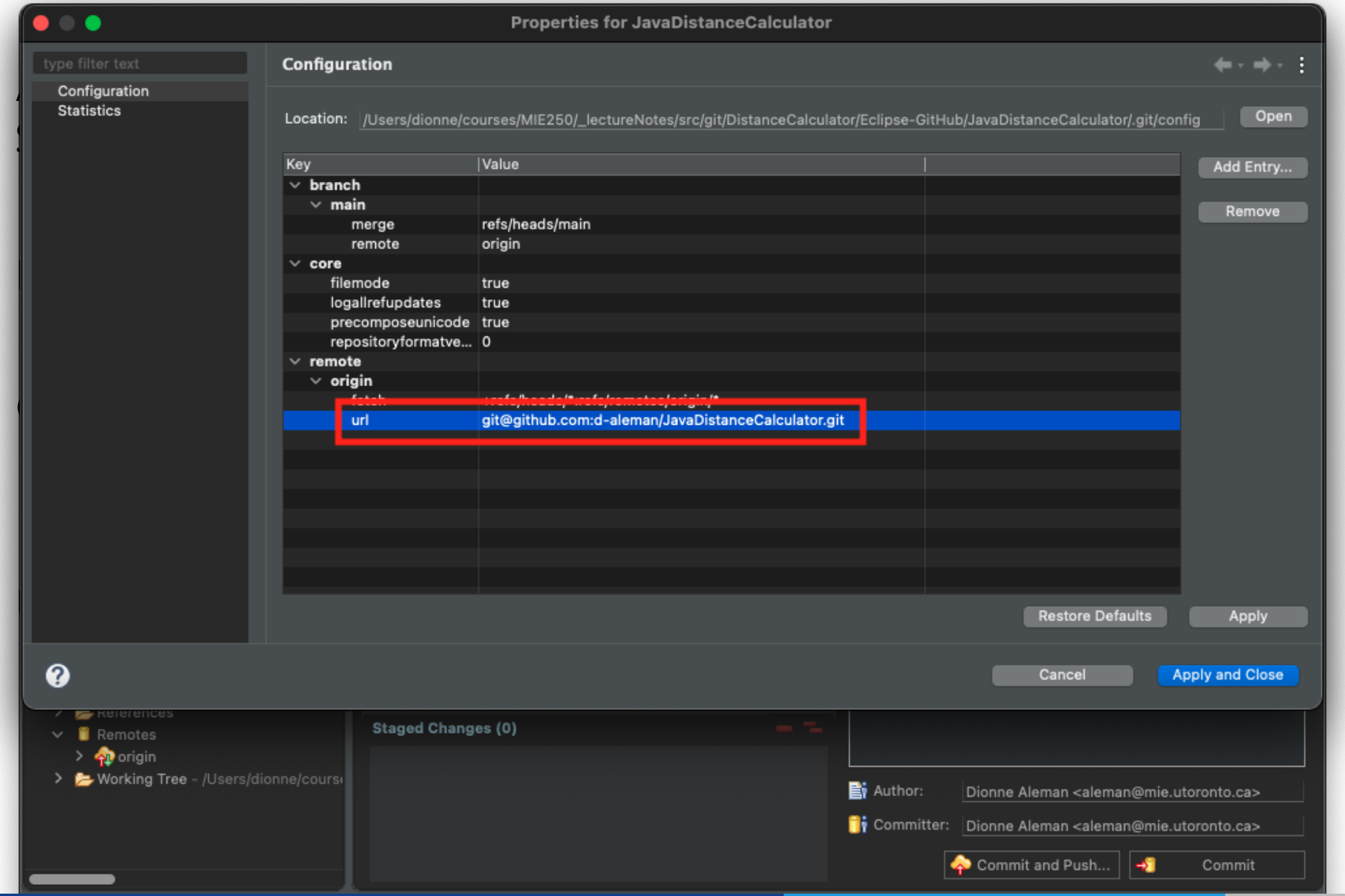
Show view → Other...).

ord (username blank).

ure Push.

rname>/<repo>.git

Re-configure Eclipse to use the GitHub keys

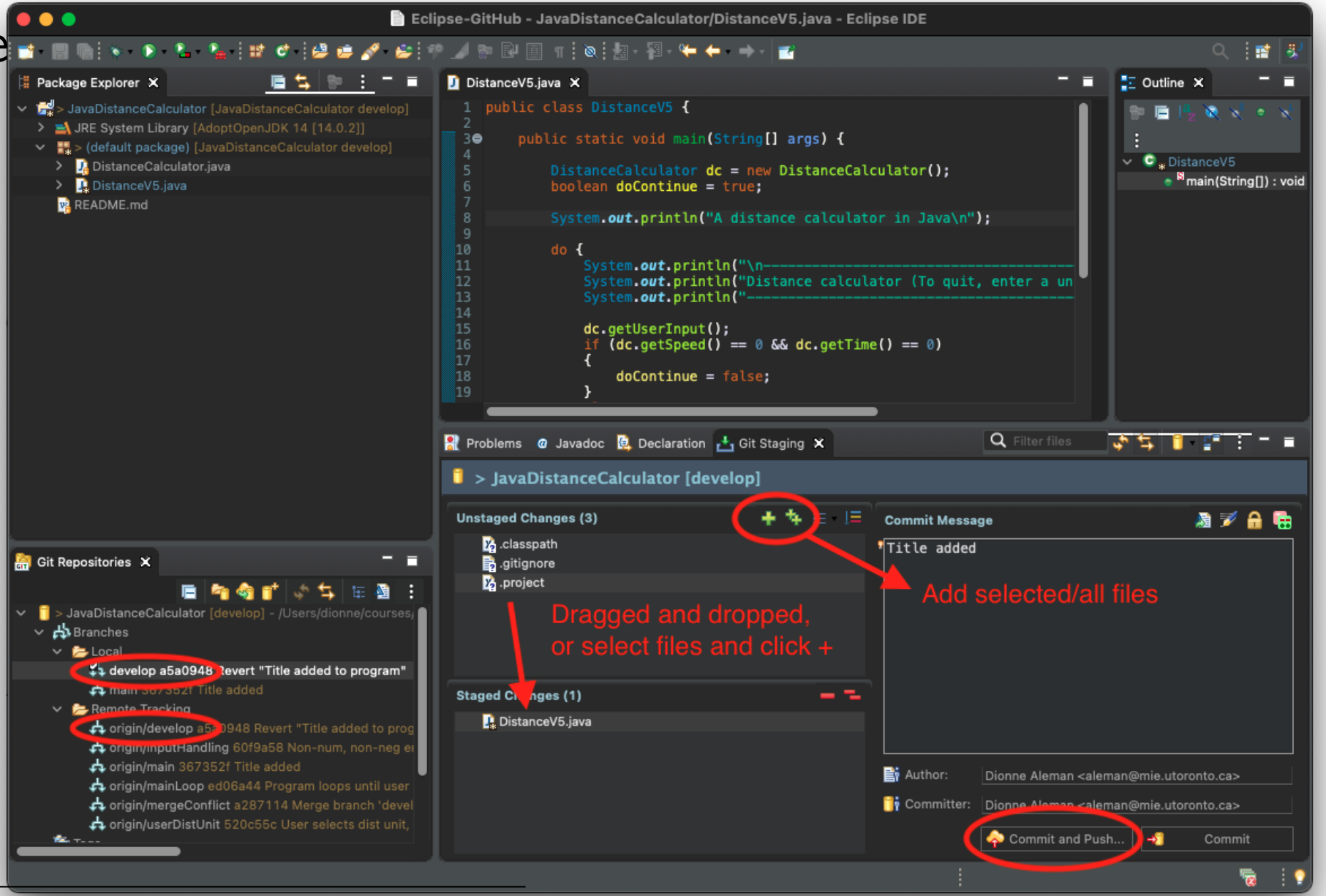


Checking out branches and making changes

- Double-click on a remote (`origin`) branch to copy it to your local computer.
- Double-click on the local copy and make your changes.
- Add changed files to staging area, then commit and push.

WARNING: The branch name shown in the bottom left of the Eclipse window is just the currently highlighted branch, and not the branch that is actually checked out.

Che



WARNING: The branch name shown in the bottom left of the Eclipse window is just the currently highlighted branch, and not the branch that is actually checked out.

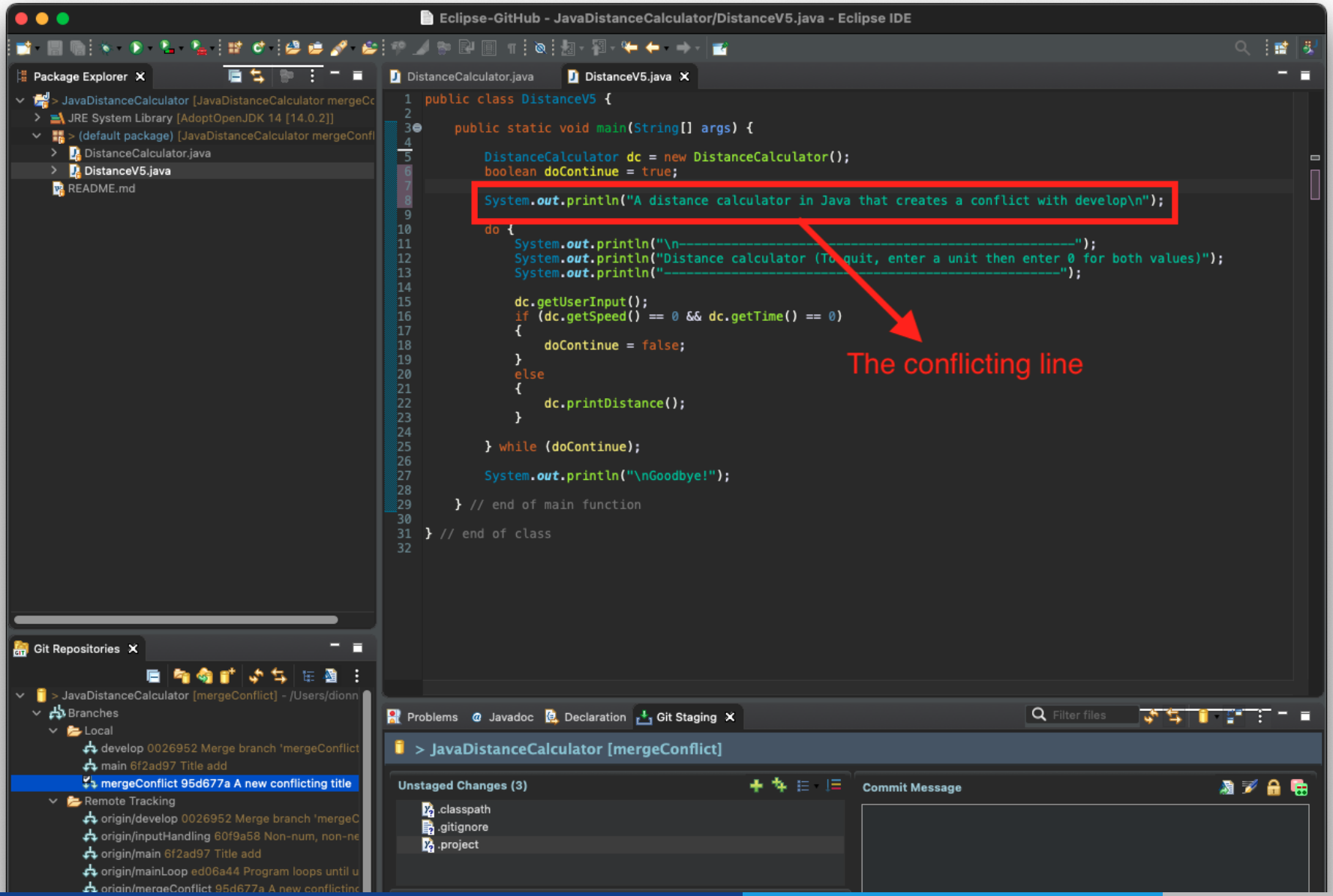
Merging in Eclipse

- Checkout a branch (double-click) to have changes merged into.
- Right-click on the branch and select Merge.
- Select incoming branch.
- Commit and push changes.

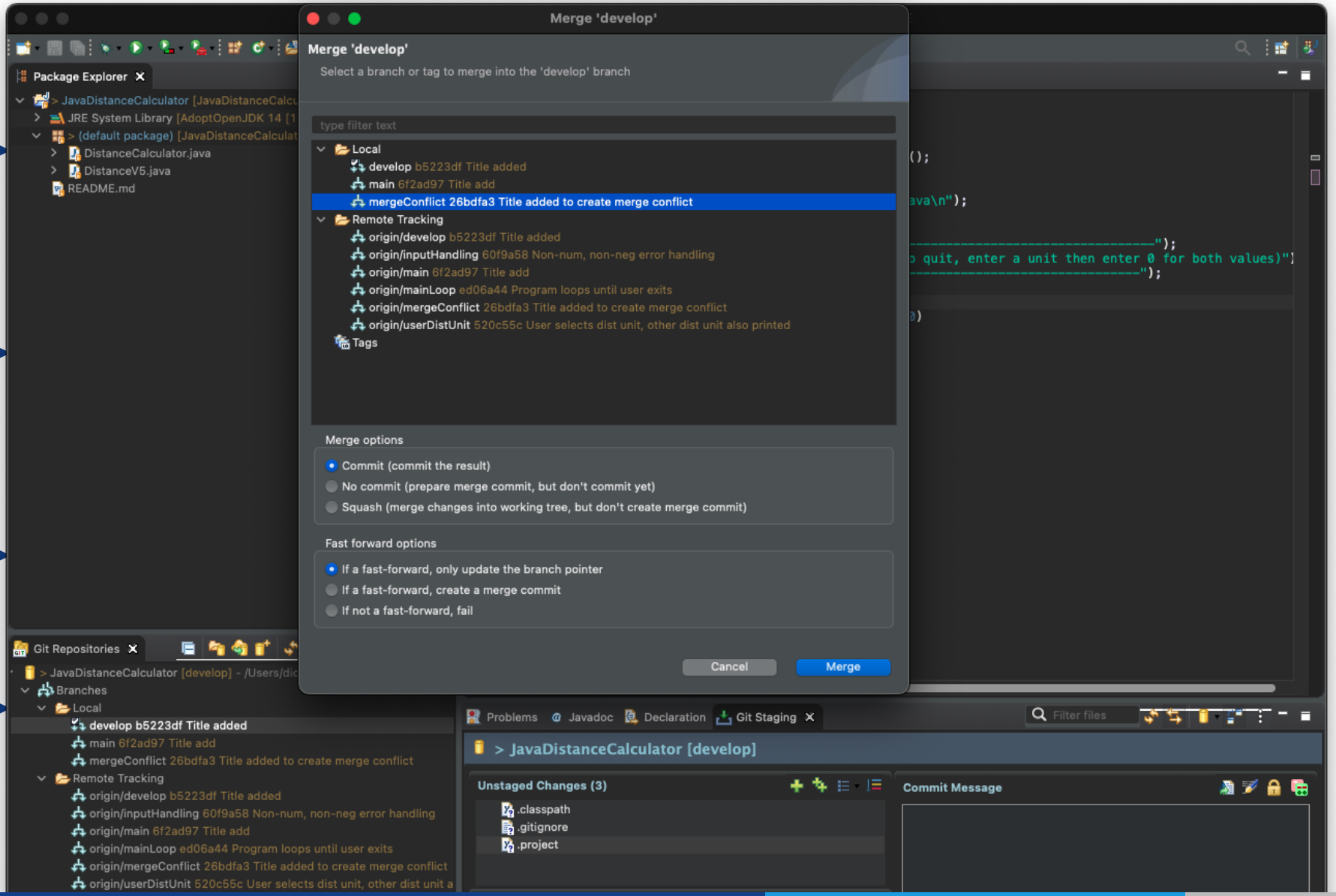
Merge conflicts in Eclipse

- Like before, let's use the `mergeConflict` branch to create a conflict (in a print statement).
- Use the Merge Tool to view conflicting file in both branches and pick and choose which code lines you want to keep. Save the finished file.
- Add the resolved file to the Index.
- Finish the merge.

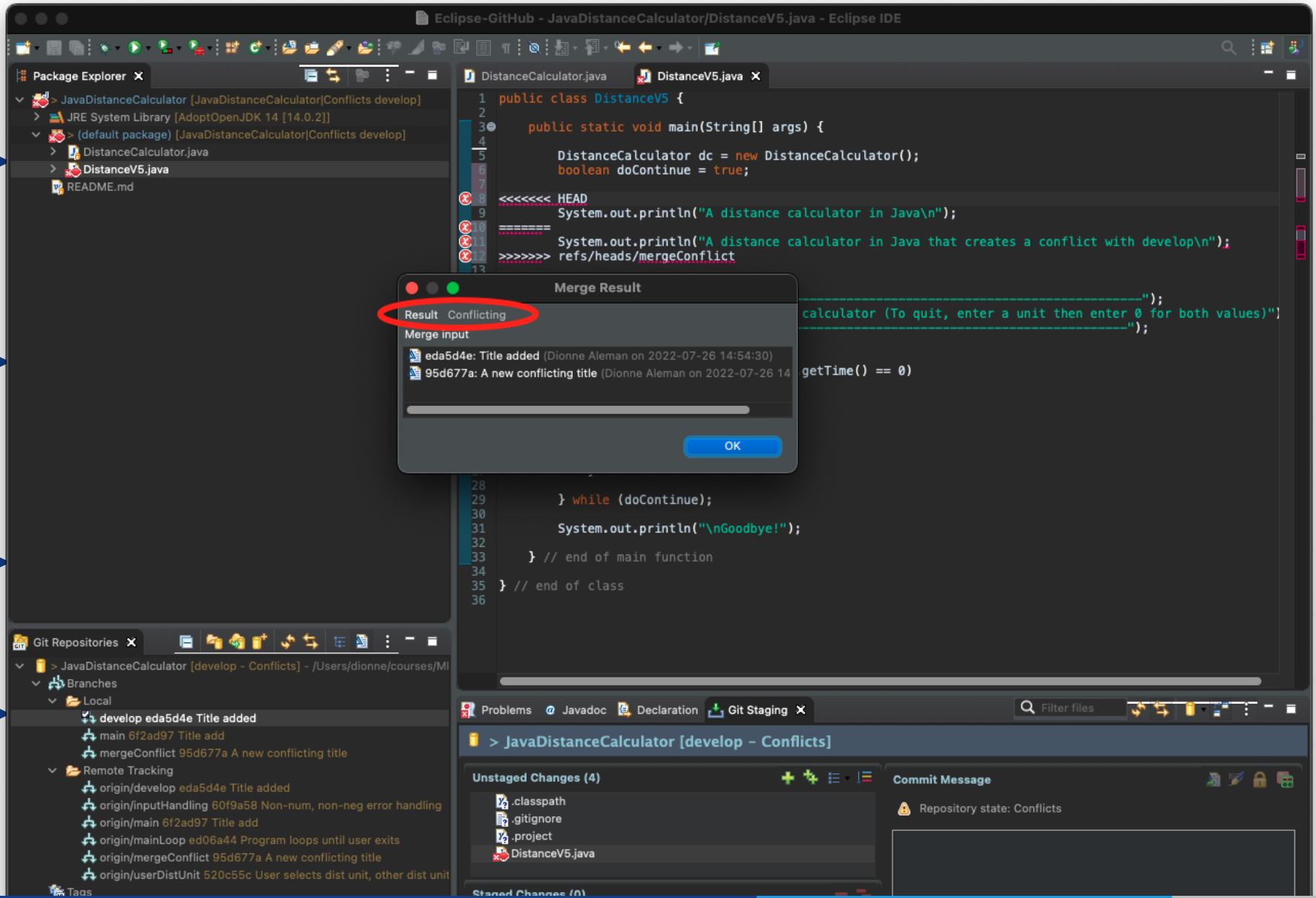
Merge conflicts in Eclipse



Merge conflicts in Eclipse



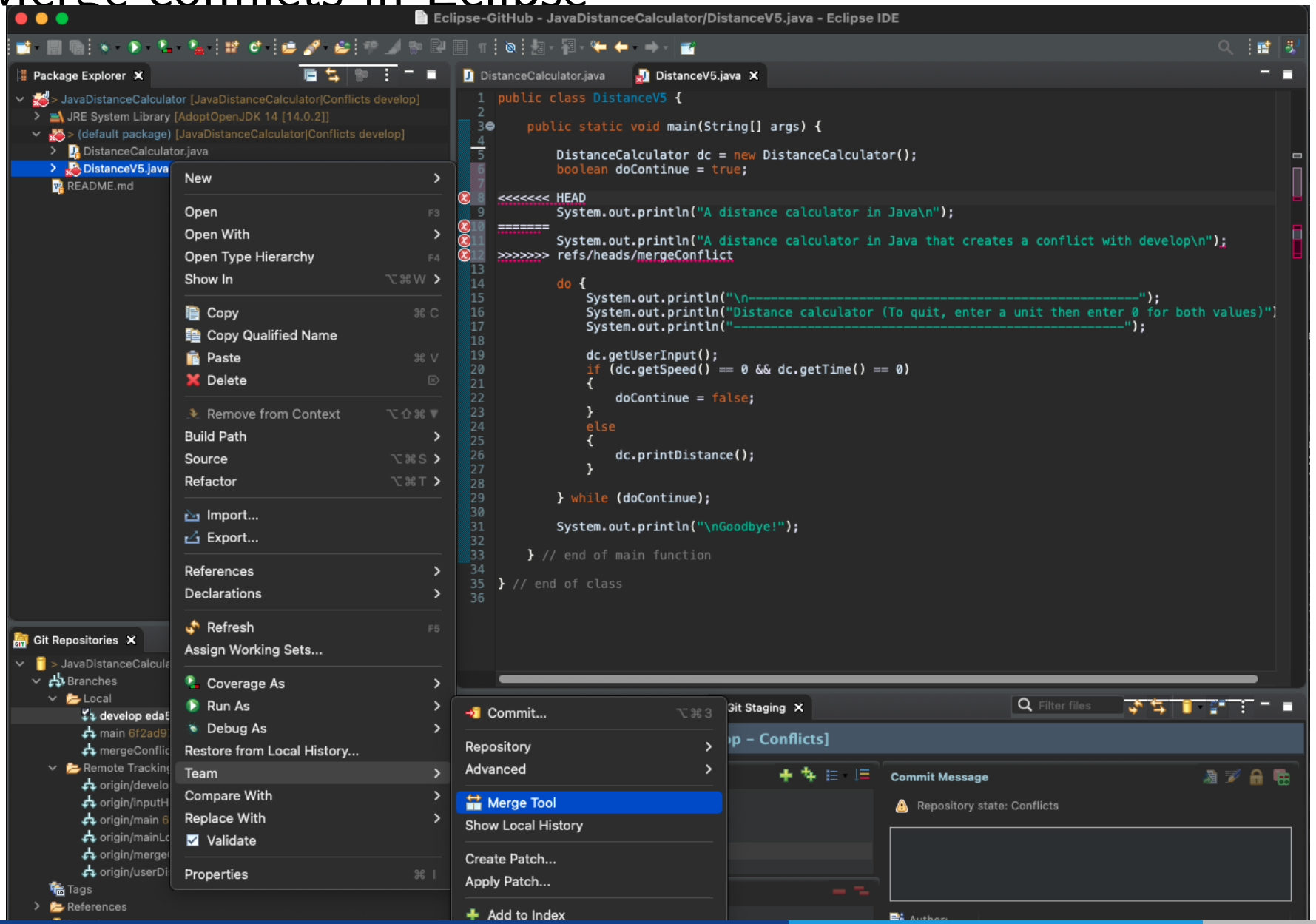
Merge conflicts in Eclipse



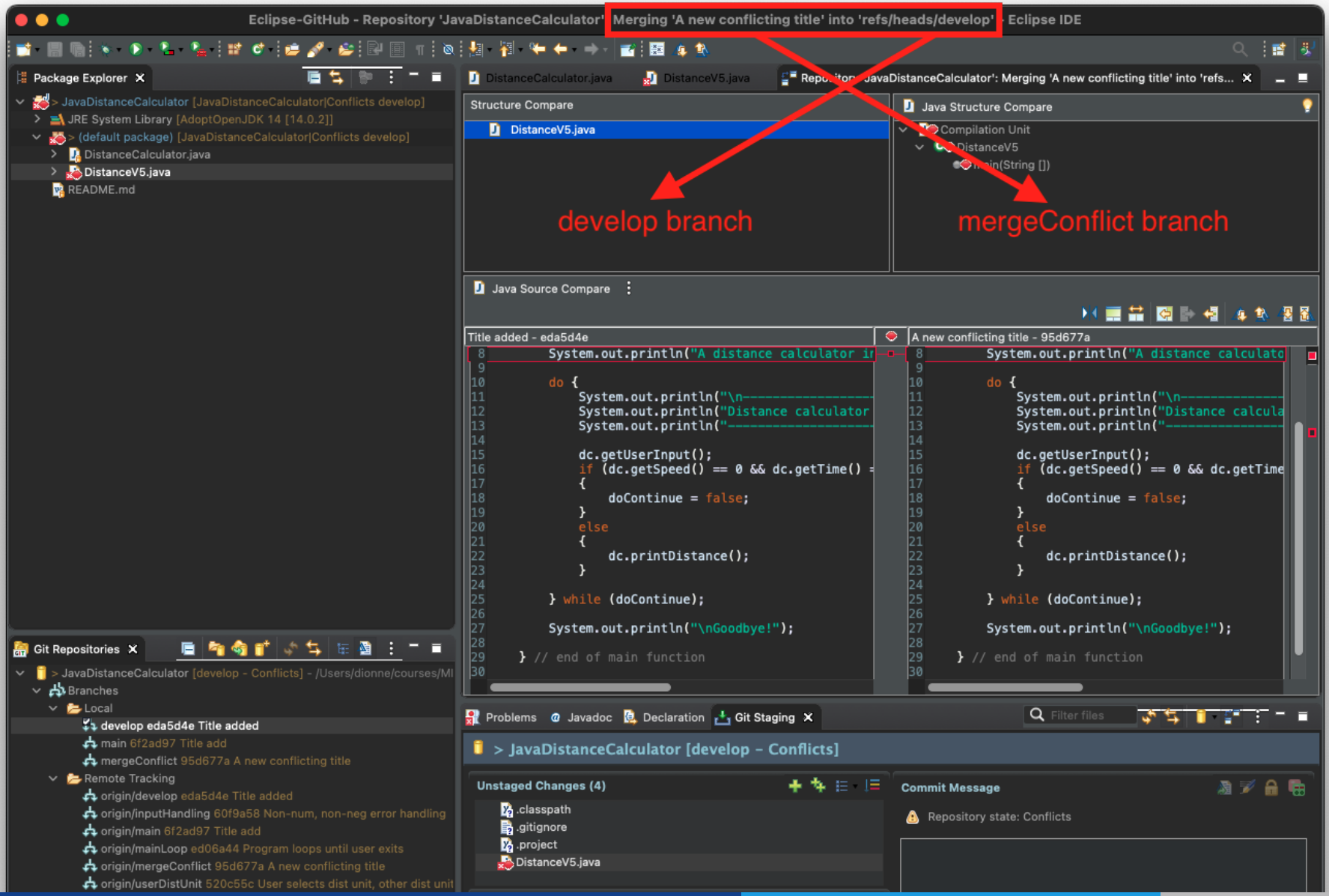
Merge conflicts in Eclipse

- Like before, let's use the `mergeConflict` branch to create a conflict (in a print statement).
- Use the Merge Tool to view conflicting file in both branches and pick and choose which code lines you want to keep. Save the finished file.
- Add the resolved file to the Index.
- Finish the merge.

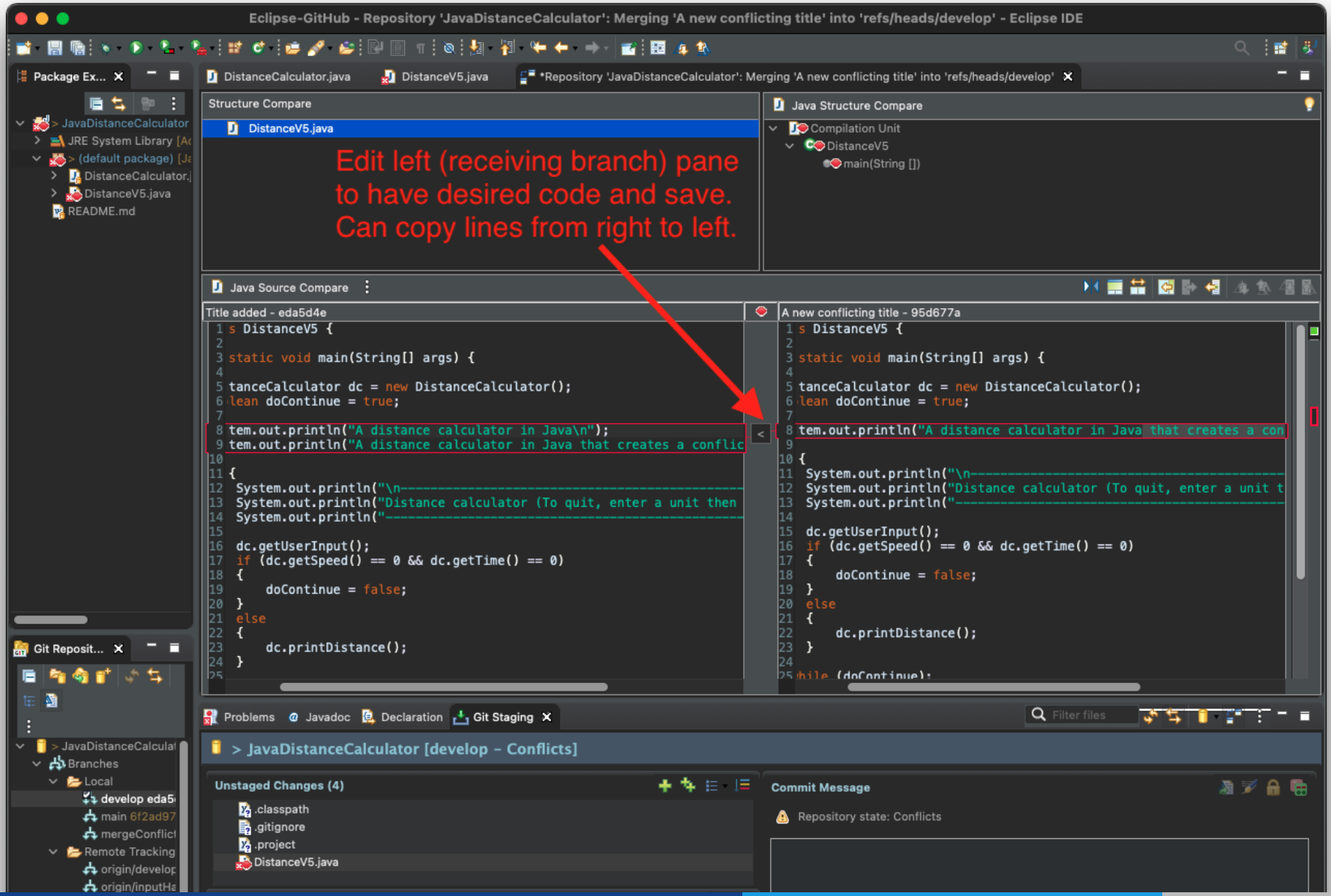
Merge conflicts in Eclipse



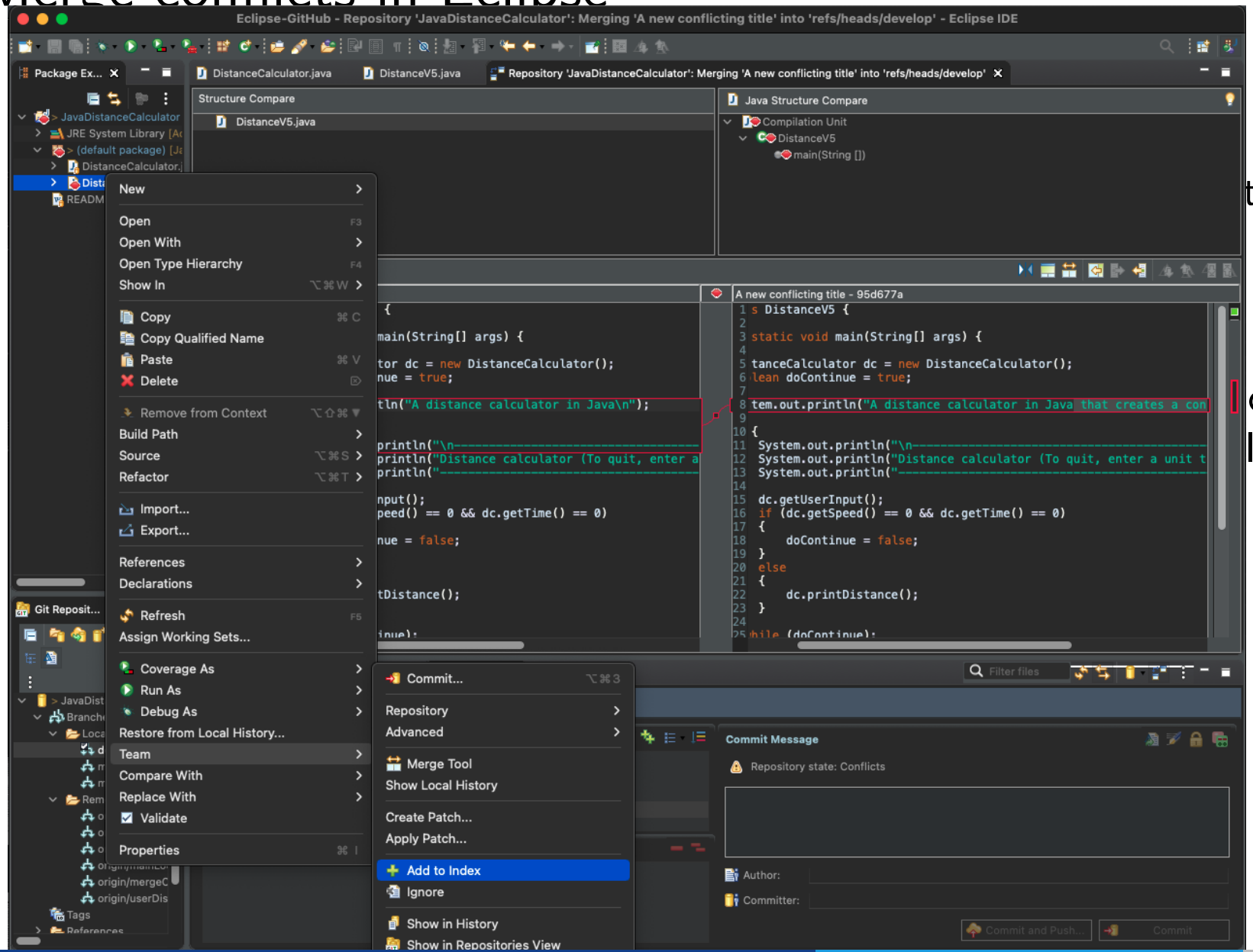
Merge conflicts in Eclipse



Merge conflicts in Eclipse



Merge conflicts in Eclipse



t (in a

ck

le.

Merge conflicts in Eclipse

- Like before, let's use the `mergeConflict` branch to create a conflict (in a print statement).
- Use the Merge Tool to view conflicting file in both branches and pick and choose which code lines you want to keep. Save the finished file.
- Add the resolved file to the Index.
- Finish the merge.

Eclipse merging advice

- Be REALLY sure which branch is merging into which before proceeding.
  - First, double-click the receiving branch to check it out.
  - Then, right-click the branch and select Merge.
- Eclipse can make things a bit confusing due to poor UI/UX.
  - The branch name at the bottom left of the Eclipse window is what branch is highlighted in the tree, not what branch is actually checked out.
  - The Merge Tool doesn't clearly indicate which branch is which in the left/right panes.
- If you are doing a lot of programming, you may want to investigate other IDEs (e.g., Microsoft Visual Studio Code).
- I prefer just doing merges directly in GitHub, even with VS Code.