CSC 217 Lab 00

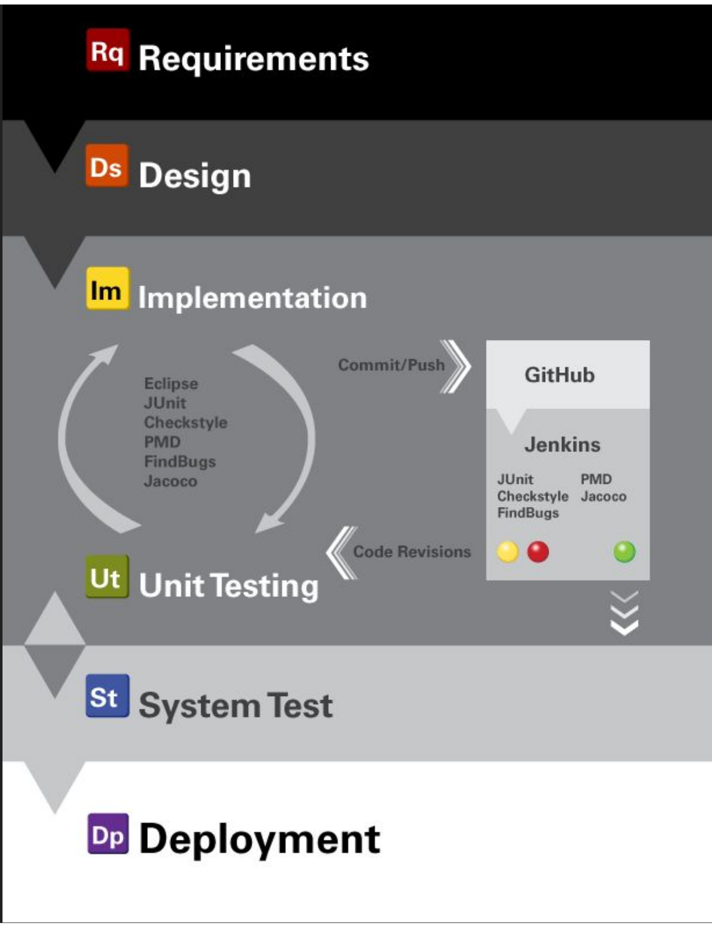
Syllabus, Version Control, and Installation

Welcome to CSC 217!

Goal - apply the concepts you learn in CSC 216 into practice

- Build a stand-alone Java course registration system
- Requirements change and build over each lab
- Each lab is an iteration of our software development process

By the end of the semester, you will have a Java application approaching 10K lines of source and test code!



Attendance & Deadlines

Synchronous Labs

- Engineering Building II, Room 1221 - MASKS REQUIRED!!!
- Attendance is expected - email Dr. Heckman (sarah_heckman@ncsu.edu) to excuse absences (i.e., if you're sick)
- If you are unable to attend lab due to illness, but are well enough to participate, you are expected to connect with your team via Zoom during lab time
- Labs are completed on teams, unless otherwise stated, and unexcused absences/inappropriate participation will result in a zero for the lab
- Deadlines 10 minutes before lab meeting, unless otherwise stated

Asynchronous Labs (Sections 231 and 601)

- Work independently on the labs, can opt to work on a team
- Deadlines are 11:45pm ET on Tuesdays, unless otherwise stated

Course Grade & Policies

Grade = average of all the labs

- The lowest lab grade may be dropped
- If so, the lowest grade dropped will be from Lab 0-10. Lab 11 (recursive lists) will not be dropped!

Academic Integrity - don't cheat on the labs! See syllabus for details.

Continuity

- Don't come to lecture/lab if you're sick/isolated/quarantining - work with your team remotely via Zoom during lab time (if you're well enough to participate)
  - Email Dr. Heckman (sarah_heckman@ncsu.edu) to excuse the absence
- We are planning backup PTFs to keep sync labs in person
- But if needed, we will move sync labs to online sync as the semester evolves
- Bring headphone to connect with remote team members

Course Structure and Resources

This course provides weekly project-based learning in a variety of topics:

- Software Engineering (weeks 1-5)
  - Team process, testing, working with libraries, design, integration
- Design Patterns (weeks 5-6)
  - Model-View-Controller, Singleton, State
- Data Structures (weeks 7+)
  - ArrayList, LinkedList, Stack, Queue, Iterators, Recursion

Resources for help:

- Office Hours via MyDigitalHand - see calendar on Moodle
- Piazza - check out the “How to Use Piazza” pinned note

Lab 00 Overview

- GitHub Intro
- Installation & Environment Checks
- Lab Deadline
- Lab Wrap-Up

GitHub Introduction

Version Control

Version Control systems keep track of source code over the history of a project AND facilitate interactions between teams!

- Allow for multiple users
- Provide mechanisms for checking in conflicting code
- Backup of previous versions

We’re using GitHub!

Version Control Systems

Functions:

- Convenient, secure access by many developers to a shared project
- Easy to backup to a remote server
- Flexible revision to a previous version of your project
- Conflict control among multiple developers of a project
  - (But you should still communicate with your team to minimize conflict!)

Terminology

A **repository** is where the backup (main) copies of all files are stored

- Local repository: on your computer
- Remote repository: shared repository in the cloud where you submit/share your work

A **revision** is a snapshot of a project at one moment in time

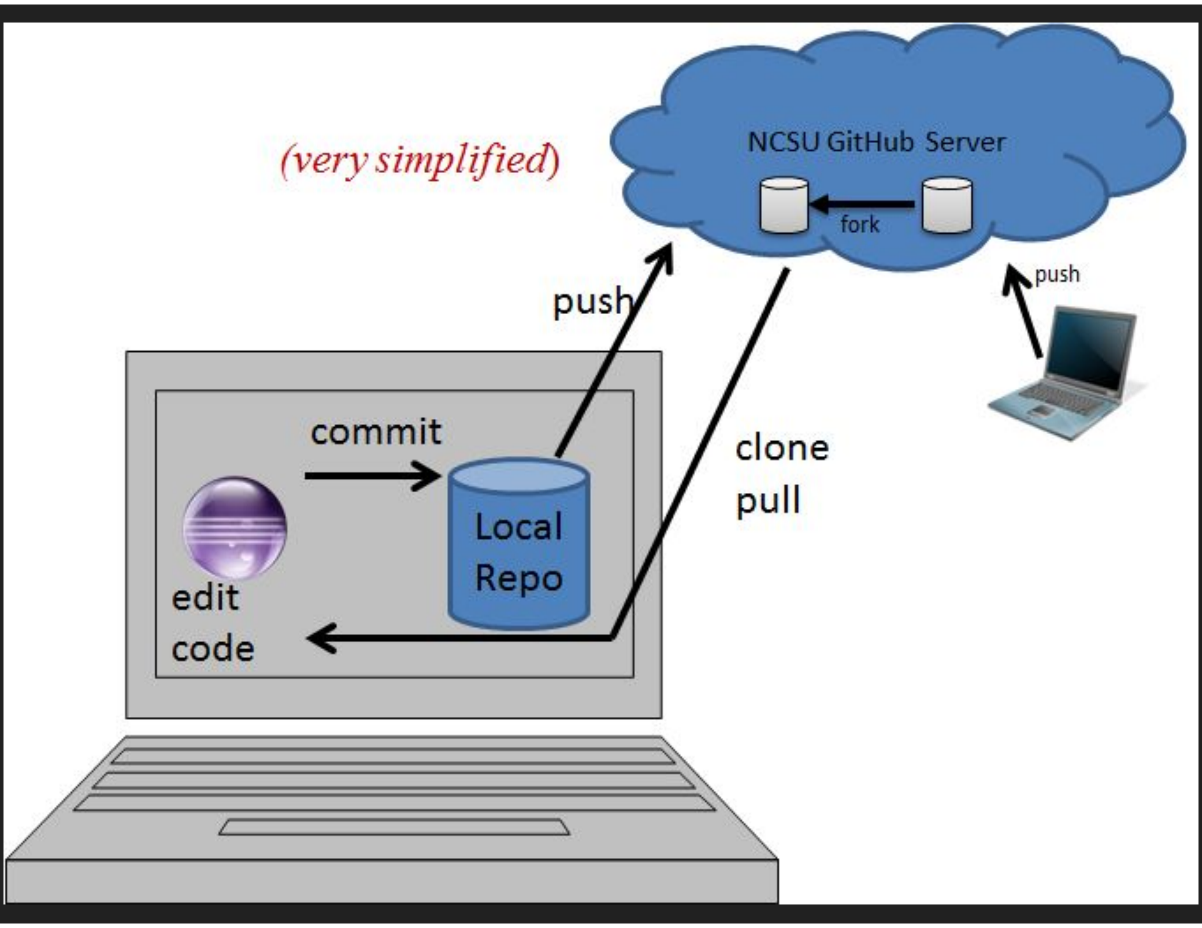
- A **commit** creates a new revision in the local repository
- A **push** copies the revision from the local repository to the remote repository, submitting/sharing your work

Git

Decentralized

All repos are equal!

Code moved between repos by *pulling* and *pushing.*



GitHub

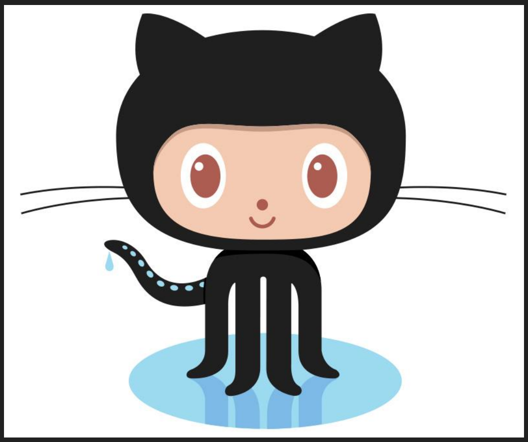
Service that hosts Git repositories in the cloud

Provides additional features

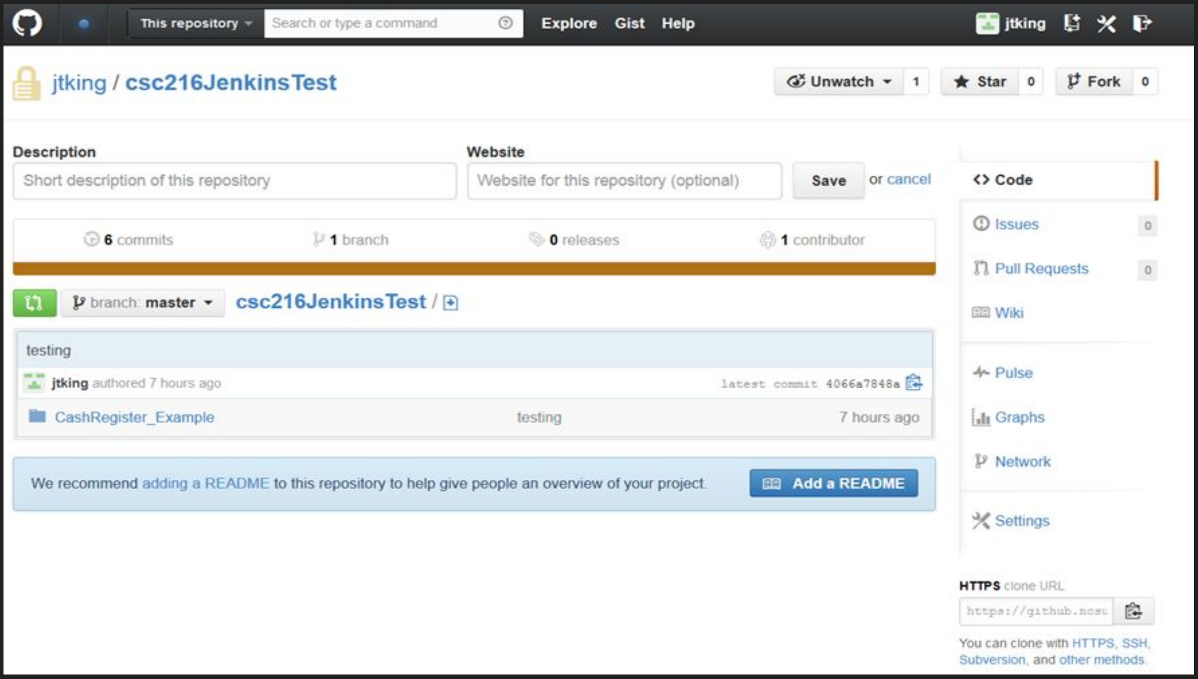
- Wikis
- Bug Tracking
- Inspection Markup

We are using NCSU's Enterprise GitHub

You MUST use the provided repositories for this course to submit your work



Verify All Your Pushes on GitHub!



testing

jtking authored 7 hours ago — latest commit 4066a7848a

CashRegister_Example — testing — 7 hours ago

We recommend adding a README to this repository to help give people an overview of your project.

You can clone with HTTPS, SSH, Subversion, and other methods.

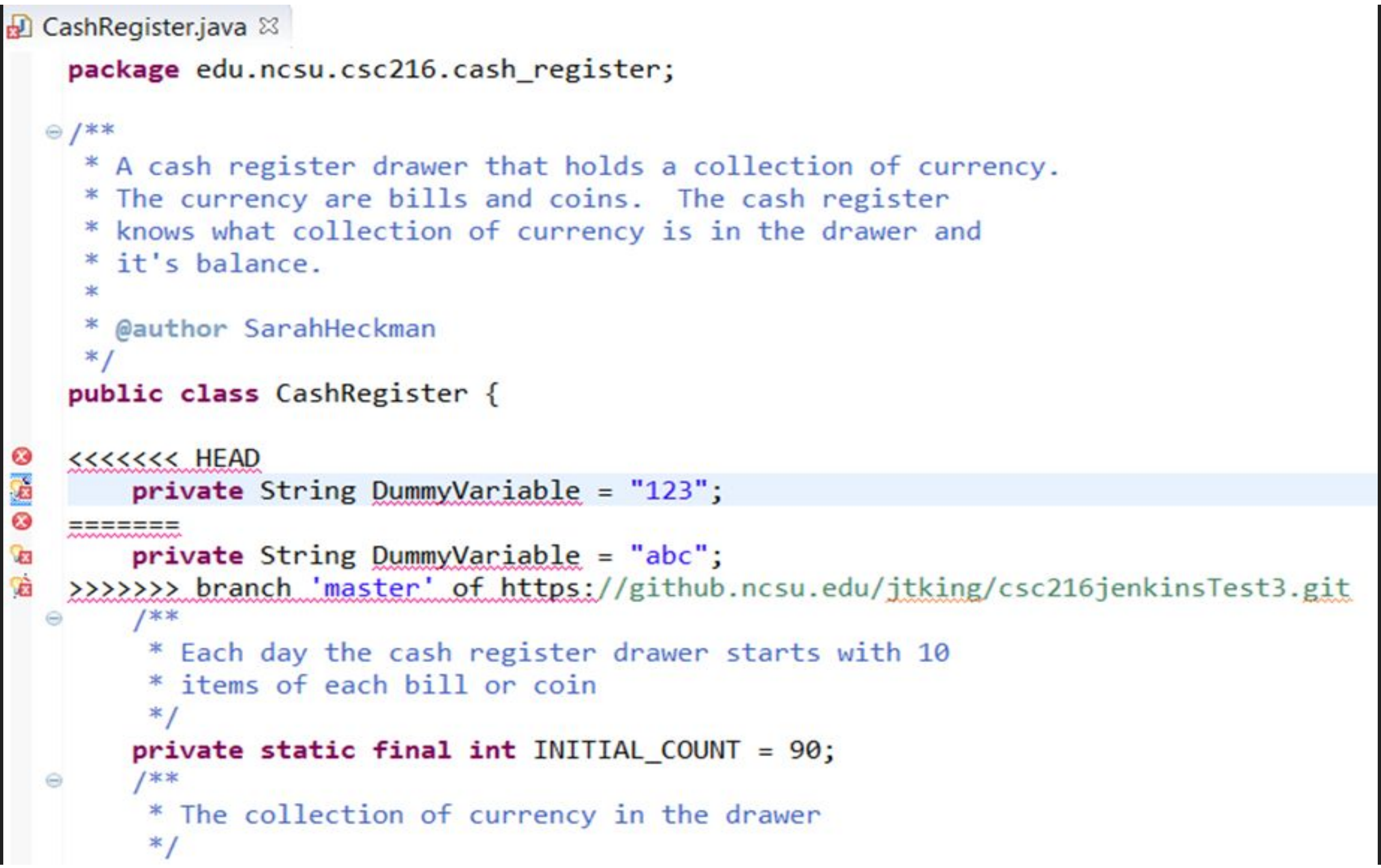
Conflicts

Conflicts may occur when multiple users are working independently on their own local copies of the same file

- Alice and Bob edit file Example.java
- Bob commits & pushes his Example.java to the remote GH repo
- Alice commits her changes and pulls Bob's Example.java from GH
- The incoming Example.java file from Bob conflicts with Alice's changes

Resulting Example.java has embedded conflict markers

Conflicts must be resolved


CashRegister.java

```
package edu.ncsu.csc216.cash_register;

/**
 * A cash register drawer that holds a collection of currency.
 * The currency are bills and coins.  The cash register
 * knows what collection of currency is in the drawer and
 * it's balance.
 *
 * @author SarahHeckman
 */
public class CashRegister {

<<<<<<< HEAD
    private String DummyVariable = "123";
=======
    private String DummyVariable = "abc";
>>>>>>> branch 'master' of https://github.ncsu.edu/jtking/csc216jenkinsTest3.git
    /**
     * Each day the cash register drawer starts with 10
     * items of each bill or coin
     */
    private static final int INITIAL_COUNT = 90;
    /**
     * The collection of currency in the drawer
     */
```

Resolving Conflicts

Resolving Conflicts

- Edit the file to remove conflicts & conflict markers
- Add to index
- Commit & push

If the conflict is in a .project or .classpath file, the conflict will have to be resolved in an OS text editor (Notepad++)

- Use Git Bash to add/commit/push

Communicate with your teammates to reduce potential conflicts in the first place!

Non-Fast-Forward Issue

A non-fast-forward error doesn't always mean a conflict. What it means is:

- You have local changes to a file that haven't been committed to your local repo
- There are changes to the file in the remote that MAY cause conflict

To resolve:

- Commit (don't push) your local changes
- Pull from remote
- Handle any conflicts (git should be able to merge most things without your intervention)
- Add/commit/push

Individual Workflow

[Optional] If working with multiple computers

- Git Bash: git pull //Gets the latest from the remote
- Eclipse: Right click on project > Team > Pull

Edit your files

Add changes to index

- Git Bash: git add .
- Eclipse: Staging view > select files > right click > Add to index

Commit

- Git Bash: git commit -am "Message"
- Eclipse: Enter a message in Staging view (you can commit/push at the same time)

Push

- Git Bash: git push
- Eclipse: Commit/Push

Partner/Team Workflow

Pull partner's changes

- Git Bash: git pull //Gets the latest from the remote
- Eclipse: Right click on project > Team > Pull

Edit your files

Add changes to index

- Git Bash: git add .
- Eclipse: Staging view > select files > right click > Add to index

Commit

- Git Bash: git commit -am "Message"
- Eclipse: Enter a message in Staging view and Commit (not Commit/Push)

Pull any new changes (see red text to the left)

Push

- Git Bash: git push
- Eclipse: Commit/Push

Last Resort: Reclone

If the previous things do not fix the conflict, you can reclone the repo. Assuming Partner 1's code is the one you want to save:

Partner 1 saves their current project somewhere on their computer. MAKE SURE IT'S SAFE.

Right click on project > Delete from Workspace

Partner 1 reclones the repository to a new location and imports the saved project. Then copy in any of the saved files, commit, push.

Partner 2 pulls the changes.

Git Demo

Lab Activity - Development Environment Setup

Set up your Development Environment!

1. Work through the Install Tutorial (linked on Moodle and Lab 00 writeup)
2. Show demo items to one of the Teaching Staff either:
    a. During lab (Raise your Hand in breakout rooms)
    b. In Office Hours (via MyDigitalHand)
    c. Via a submitted video on Moodle

In-person - We don’t bite! Don’t worry about having it perfect

Deadline: January 20th @ 11:45 PM

Development Environment Setup Demo

Step	Demo
Step 1: Launch Eclipse	Eclipse is open
Step 2: Import Demo Project	Project is in Package Explorer and Java 1.11 is associated with project
Step 3: Run CheckStyle	Appropriate notifications are displayed
Step 4: Run SpotBugs	Appropriate notifications are displayed
Step 5: Run PMD	Appropriate notifications are displayed
Step 6: Run Unit Tests for Coverage	Appropriate test results and coverage are displayed
Step 7: Git Configuration in Eclipse	NCSU information is displayed
Step 8: Git Configuration on Command Line [Optional + Extra Credit]	NCSU information is displayed

Wrap-Up

General Wrap-Up

- Deadline Reminder: January 20th at 11:45 PM ET
- Make a plan for finishing up the lab
- In-person demo
    - Stop by office hours to demo to instructor/PTF
    - Don’t worry about having everything perfect - we’ll help!
- If you’d rather demo at your own pace
    - Record a video and submit via the Moodle assignment by the deadline