Project description

Imagine you are a new software engineer at Interactive Intelligence on Matt Etchison's team. Matt is infamous for giving his new hires a single task on their first day: build and deploy Interactive's entire system end-to-end. Matt sets you up with a computer with none of the software you need and tells you to "get to it." You've heard stories that the system takes people an entire month to learn. Being a learned Rose software engineer, you figure that there must be a faster way to learn how the system works.

For this term's project, you will build a tool to help onboard new software engineers to complex Java projects. The tool will automatically produce an assessment of the design of arbitrary Java code. The tool will display the UML of the code and annotate it with detected design patterns. The tool will also "grade" the design overall. This way, new engineers can quickly get a broad overview of how the system is designed and what areas stand to improve.

Pair programming

This project is difficult and requires exploring APIs with which you are unfamiliar. Students are encouraged to pair program with their group to optimize their problem-solving abilities. However, all students must contribute to writing code. To ensure all students get credit, students need to rotate who is typing approximately every 20 minutes. At the switch-up, the student typing should commit his or her work to GitLab under his or her Rose-Hulman user ID.

Weekly meetings

Each week, you will meet with your instructor concerning the project. Thus, in a two-week milestone, there will be two meetings:

1. **Design review**, at the end of the first week. Halfway through each milestone, you will meet with your instructor to review your design. This means at the minimum, you must have a complete design to review. Come prepared for the design review and steer it toward the aspects you want feedback on. You may also find it useful to finish coding the project in time for the design review. Last year, students were able to code their milestones in only one week without issue, so you should be able to as well.
2. **Milestone demo**, at the end of the second week. Toward the end of the milestone, you will meet with your instructor to demo your code. Your instructor will use your demo as part of determining your grade.

After you've formed your team, your instructor will direct you to a link to schedule a time to meet.

Retries

In CSSE371, you got retries on your milestones. In CSSE374, your design review **is** your retry. There is no retry at the end of the milestone. You are still expected to correct any mistakes for the next milestone.

Resources

- ASM Resources
- GraphViz Resources
- Sample project

- ASM user guide (in particular, we are using the Tree API, not the Core API) http://download.forge.objectweb.org/asm/asm4-guide.pdf

Milestone 1: basic class diagram

Lay the foundation for this term's design assessment tool.

Design and build a simple tool to generate the most essential elements of the UML class diagram for arbitrary Java code: **classes, methods, instance variables**, and **inheritance arrows**. While there are lots of tools that will generate this level of UML for you, you will later be augmenting the graph with more sophisticated analysis of the design than standard tools provide. For this reason, you must use **good design principles**.

What you know will change

As you design the tool, anticipate that *later* milestones will ask you to:

- Add additional command line arguments.
- Invoke the UML generation from another program using a documented Java API that you provide.
- Perform more detailed analysis of the code.
- Build a more detailed graph to display the new analysis, for instance with different kinds of arrows based on some criteria.
- Tweak the formatting of the GraphViz UML, for instance, by changing the color of a class's UML frame based on some criteria.
- Analyze bigger codebases.
- Show the generated image in a JavaFX UI.

Grading

C

- Complete hand-made UML describing the design of the project.
    - To be presented at the time of the design review. **If you don't bring your completed design to your design review, you cannot earn a grade on this milestone.**
    - **Design encapsulates what changes**. The instructor may prompt students to demonstrate how their design facilitates a mystery specification change.
- README.MD documenting how to run the program and describing each team member's contributions to milestone.
- Each student demonstrated his or her individual contributions to the code by **committing** under his or her GitLab login.
- Code meets basic specifications:
    - Input: takes a list of fully-qualified class names to render as command-line arguments.
    - Behavior: Uses the **ASM Tree API** to generate correct GraphViz UML from Java code.
        - Ancestors appear at the top of the diagram.
        - Class names**,** italicized for abstract classes or interfaces.
        - Methods, indicating the method signature and whether they are public, private, or protected.
        - Instance variables, indicating the type and whether they are public, private, or protected.
        - Inheritance and implements arrows.

    - Restrictions
        - **Does not require copy/pasting** the code to be analyzed into your Eclipse project. This practice is dead to you.
        - **Does not extend** any of the following ASM classes: ClassVisitor, MethodVisitor, FieldVisitor, or their derivatives. These classes are dead to you.
        - **Does not invoke or override** any of the ASM visit* methods. These methods are dead to you.
- Demo
    - Students have separate Run Configurations for each demo.
    - Render the diagram for java.lang.String and its interfaces.
    - Render the diagram for a team member's code for Lab 1-1 Strategy.

B

- Students demonstrate appropriate use of basic design principles:
    - Favor composition over inheritance.
    - Program to interfaces, not implementations.
- Code meets intermediate specifications:
    - Takes an additional command line parameter indicating whether to recursively parse superclasses and interfaces that are not passed as arguments to the program, but instead are discovered during parsing.
- Demo
    - Recursively render the diagram for javax.swing.JComponent using your new command-line argument to automatically detect and render all of its ancestors.

A

- Students demonstrate appropriate use of advanced design principles:
    - Strive for loosely coupled designs between objects that interact.
    - Classes should be open for extension, but closed for modifications.
- Code meets advanced specifications:
    - Takes an additional command line parameter indicating the access level to render at:
        - Private: render all classes, methods, and fields.
        - Protected: render only protected or public classes, fields, and methods.
        - Public: render only public classes, fields and methods.
- Demo
    - Render this project's public API.