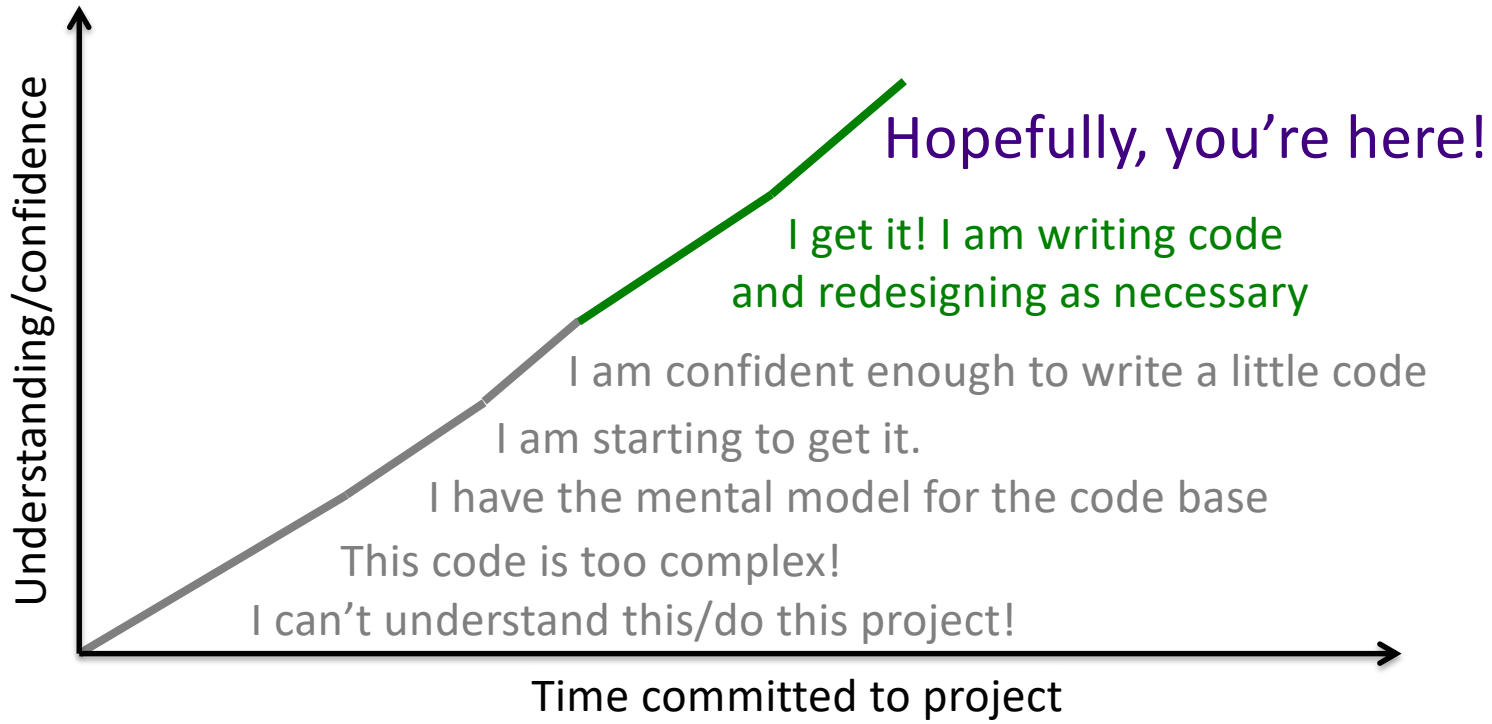
Objectives

- Picasso!

Review: Typical Trajectory of Projects



Preliminary deadline retrospective
(starting early, testing)

Review

1. What is a design pattern?
   - What design patterns have we discussed?
     - What problems do they solve?
   - What design patterns are used in the Picasso project?
     - (This could vary by team)
2. Why do we need to convert the input to postfix?
3. What is our git workflow?
4. What is a merge conflict? How do you resolve it?

Review: Design Pattern

General reusable solution to a commonly occurring problem in software design

- Not a finished design that can be transformed directly into code
- Description or *template* for how to solve a problem that can be used in many different situations
  - "Experience reuse", rather than code reuse

Design Pattern: **Strategy**

- Defines a family of algorithms, encapsulates each one, and makes them interchangeable
- Allows algorithm/behavior to vary independently of clients that use it
  - Allows behavior changes at runtime
- Design Principle:

Favor ***composition*** over inheritance

Merge Conflict

- Occurs when competing changes to the same lines in a file
  - Git doesn't know how to resolve the merge
- Resolving: manually edit the conflicted file to what you want to keep in the merge
  - Stage change, commit and explain your fix
  - Push branch

Draw the Stacks

- x+y
- x+floor(y)*x
- (x+floor(y))*x

PICASSO

Towards Intermediate Deliverable

- Set up to report errors to users
  - Currently: in the printed output but users aren't going to see that
  - Helpful errors → translated for users
- Opening a file that contains an expression
- Handling new operations
  - Order of operations
  - Assignment statement
- Functions with multiple arguments, image names
- Extensions

Hints

- Check out the FAQ
- Create unit tests, when possible/appropriate
  - Run using coverage tool to see what is (and isn't) covered.
- Draw things (e.g., stacks, trees) out on paper
- Trace through the code

Project Goals

- Everyone contributes significantly to the project
  - Has at least one part where they can say “I made this!”
- Everyone understands the code and its design
  - All of it. Well, 90% of it, at least at a high level
- Everyone feels valued as a team member

Contributing to the Team

- Always some concern that your grade is based on lines of code written
  - Not all lines of code are equal
  - Number of lines of code is not a good indicator of work or quality of code
- Variety of opportunities to contribute to the team

Tip: Comparing Binary Operators

- Likely need to implement the equals method in various classes (e.g., Addition, Subtraction, …)
- Stop after you've written two
- Compare the methods
  - Is there a code smell? Refactor!

Tip: Error Handling

- Don’t do too much translation too soon
- Can mask *your programming* errors (that aren’t user error errors)

Final Implementation: Documentation

- You leave, I'm still here, trying to use [grade] your code
- Documentation
  - Extensions aren't always obvious
  - State in README
- Javadocs: Purpose of Java classes
  - Update comments
  - Auto-generated daily
  - Can be seen on the project web site

Deliverables: Tagging

- While given code had compiler errors because of using test-driven development, there should be no compilation errors in deliverables' tagged versions
  - None for final version
  - For others, okay if you have clearly marked test classes for test-driven development

Secondary Goals

- You're going to figure out that your final design isn't perfect—maybe not even good!
  - Fix more critical and/or smaller things
    - Refactoring!
  - Note larger things
    - analysis/post-mortem due at end of finals week

Good judgment comes from experience.
How do you get experience?
Bad judgment works every time.

Final Project: Project Analysis - Individual

- Understand teammates' design/code/parts
  - *At least* at a high level
- Contents: Description, Planning, Status, Code Analysis, Collaboration, Future Work
  - Complete specification online

Project Planning

- Review project specifications
- Make sure you know what tasks are left
  - Intermediate deadline provides some direction, but there are a variety of other tasks that can be implemented.
- Be agile!

Looking Ahead

- Wednesday: Course Retrospective
- Friday: Intermediate Deadline, Demo
- Finals Week
  - Thursday: Final Implementation Deadline
  - Friday - noon: Final Analysis