Objectives

- Planning
- Team Work

1

Review

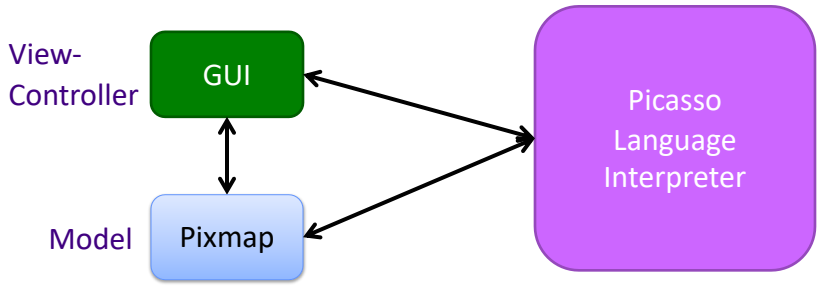
- What is the Picasso project?
  - Are you scared?
- What are the major components of the existing Picasso code base?
- What parts of project need to be completed?
- (Rhetorical) Who are your teammates?

2

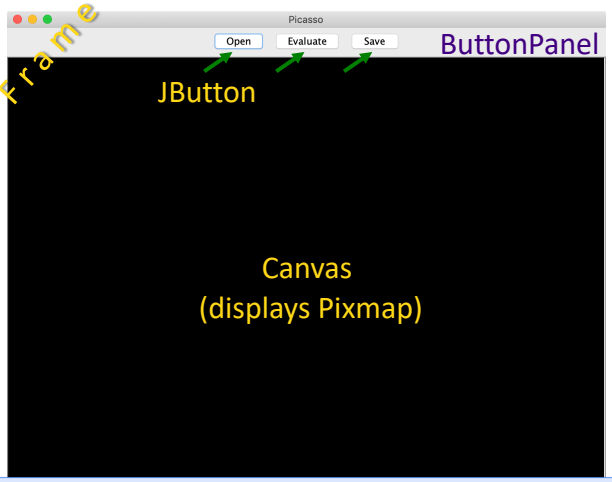
Review: Picasso

- It's okay to be a little scared. It's Halloween!
- Let that motivate you
- But *believe* that you can tackle the project

Picasso Architecture



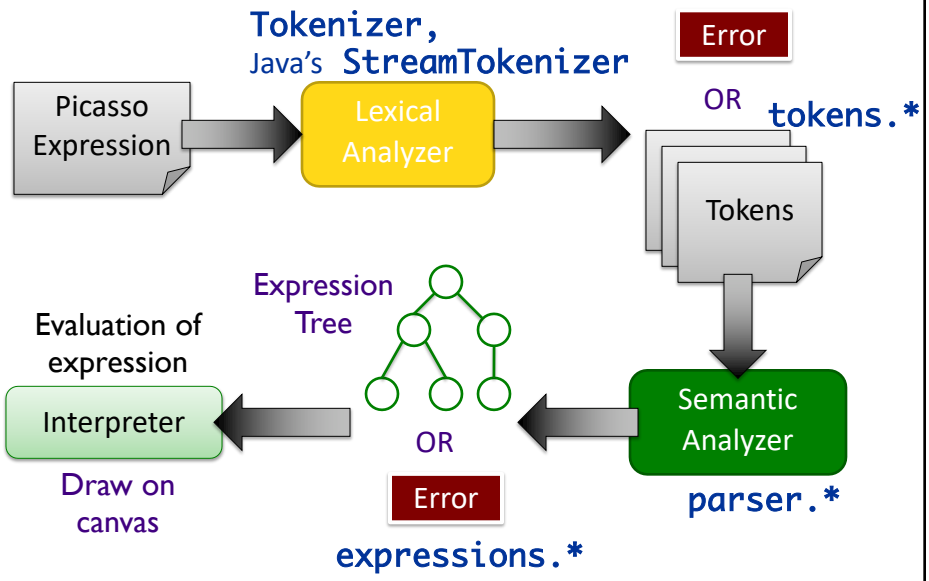
Picasso GUI



Picasso's GUI uses classes from two main Java packages:
- Abstract Windowing Toolkit: java.awt
- Swing: javax.swing

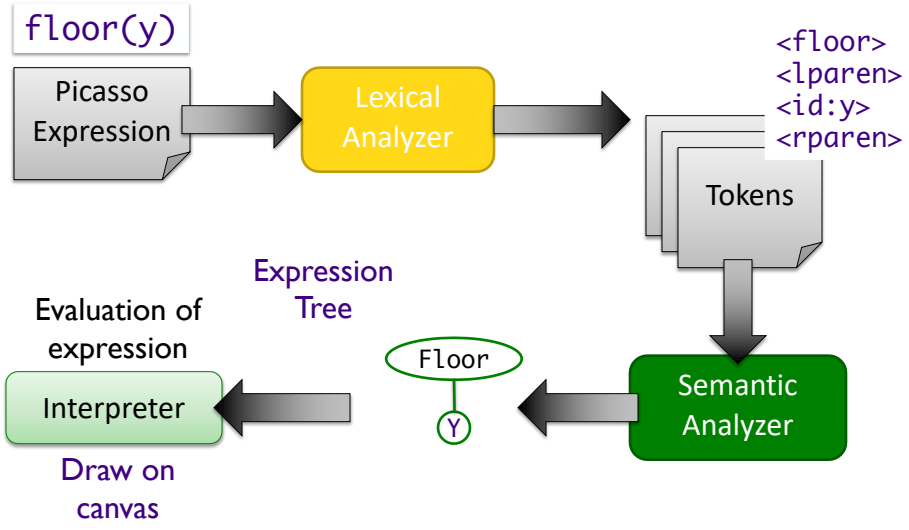
5

Interpreting the Picasso Language



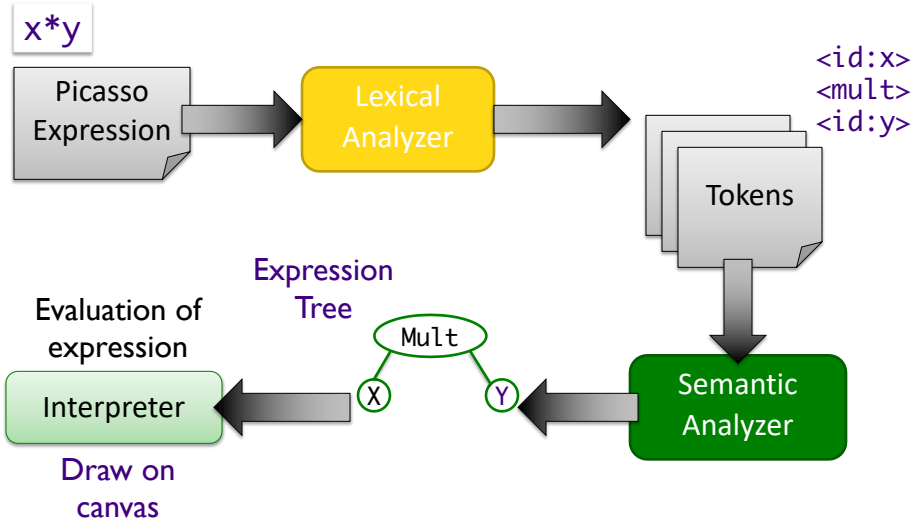
6

Interpreting the Picasso Language



7

Interpreting the Picasso Language



8

Process of Adding Cosine Function to the Picasso Language (in given code)

- Create a *token* for the cosine function
  - Same prefix as new function, e.g., `CosToken.java`
  - Needs to be added to `functions.conf`
- Create a *semantic analyzer* for the function with same prefix as function, e.g., `CosAnalyzer.java`
  - `Analyzer` class implements `SemanticAnalyzerInterface`, returns an instance of `ExpressionTreeNode`
- Create an `ExpressionTreeNode` for function: `Cosine.java`

Why is the naming important for the token and analyzer?

Process of Adding Cosine Function to the Picasso Language (in given code)

- Create a *token* for the cosine function
  - Same prefix as new function, e.g., `CosToken.java`
  - Needs to be added to `functions.conf`
- Create a *semantic analyzer* for the function with same prefix as function, e.g., `CosAnalyzer.java`
  - `Analyzer` class implements `SemanticAnalyzerInterface`, returns an instance of `ExpressionTreeNode`
- Create an `ExpressionTreeNode` for function: `Cosine.java`

Using Java *reflection* to map tokens to analyzers. (How would we do this otherwise?)

What Steps Need To Be Completed?

- Model: Images
  - API
  - State
- GUI
  - Expression user interface (interactive)
  - Open expression files (batch)
  - Talk to Picasso interpreter
- **TESTING!**
- Picasso interpreter
  - Parse expressions (functions, operations, variables, …)
    - **Handle errors** appropriately
  - Evaluate expressions
    - Manipulate canvas appropriately
- Extensions

Dependencies?

Dependencies

- Interpreter classes (tokens, analyzer, expression) are very dependent on each other
- Need to hook GUI to Interpreter
- Need to hook Image/Canvas to GUI and Interpreter
- Can test without other pieces but easier and more satisfying to see results displayed

Extensions

- Extensions could affect your code design
  - Where could change → abstraction
- When does your team need to decide?
  - Technically, not until the final implementation deadline
    - But, see above

Planning for Preliminary Implementation

- Goal is to have you do enough that you'll see issues with an initial design you create and adjust
- Implementation requirement:
  - Input an expression interactively that includes at least one binary operator and display an image from the resulting expression
  - Tag the version in Git
- Requirement involves a lot of different pieces
  - Don't go too far in breadth, more depth
    - See design issues sooner
      - "We need method/functionality X in class Y"
- Don't stop if you have more time
  - Keep going to find issues earlier

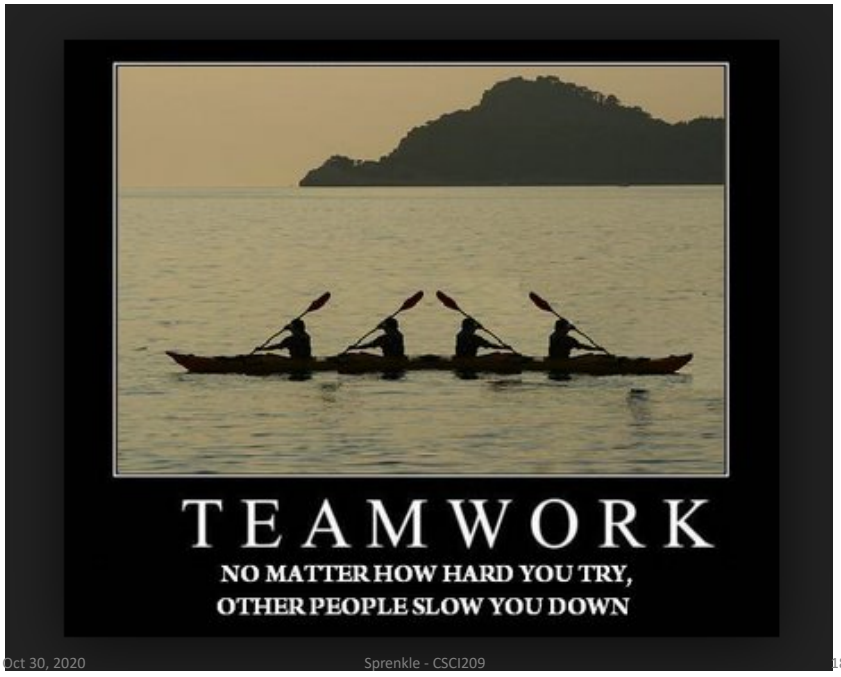
Planning: Tasks/Steps

- Testing
- Think about iterative development
  - Not recommended: write all the tokens/parsers/expressions first
    - Recall in Roulette: started creating 3 bets, realized there was a design problem, refactored, tested those 3 bets, maybe realized there was a problem and adjusted, then implemented other bets
  - What is an appropriate process for this project?
- Decide on APIs where there are dependencies
  - Parameters and what is returned

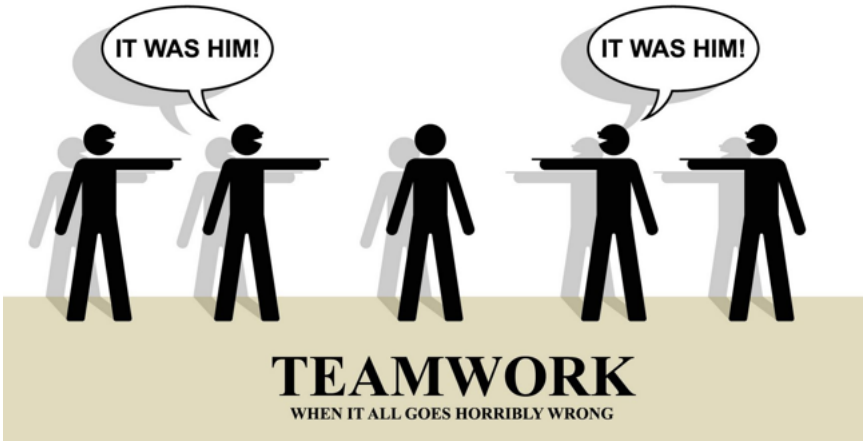
Planning: Division of Tasks

- Work in subgroups?
- Consider how not to step on each other's toes
  - Reminder: Use git branches!
- Consider best # of people per part
  - Likely will keep changing as work gets done and you learn your design
- Not recommended: Person X does all the testing
  - Perhaps pair people up to write tests for each other

17



18



19

Teams Work Best When They are **Interdependent**

- In code terms, we want *loose coupling*
  - Depend on each other but don't depend on their details
- Consider
  - Are you allowing your team to truly be interdependent?
  - Who might be you be ignoring?
  - Who might be allowing themselves to feel inadequate?
  - How do you show appreciation for each other and yourself?

20

Questions

- Any code we shouldn't change?
  - There is likely code that you won't change but depends on your extensions
- What if our design isn't perfect?
  - It won't be
  - BUT try to get it to pretty good, especially before the preliminary deadline

Implementation/Code Questions?

Looking Ahead

- In two Mondays, preliminary implementation deadline
  - Demo in class