

Objectives

- Planning
- Team Work

Nov 19, 2021

Sprenkle - CSCI209

1

1

Review: Picasso

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

Nov 19, 2021

Sprenkle - CSCI209

2

2

Review

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

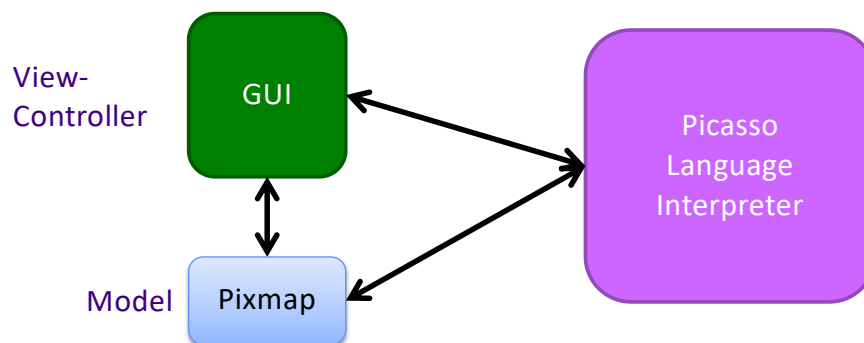
Nov 19, 2021

Sprenkle - CSCI209

3

3

Picasso Architecture



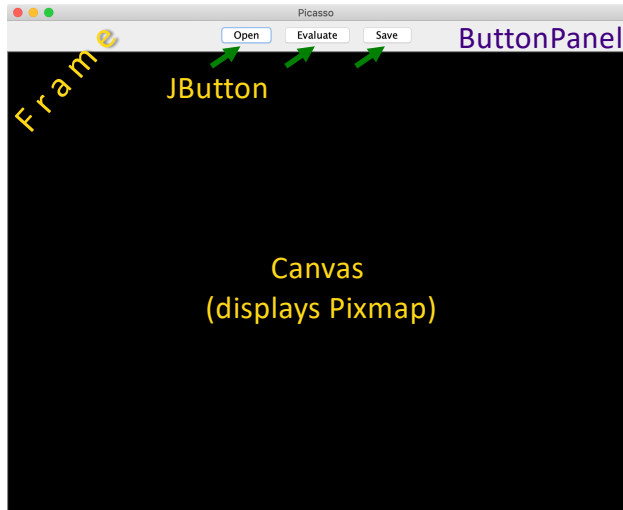
Nov 19, 2021

Sprenkle - CSCI209

4

4

Picasso GUI



Picasso's GUI uses classes from two main Java packages:

- Abstract Windowing Toolkit: java.awt
- Swing: javax.swing

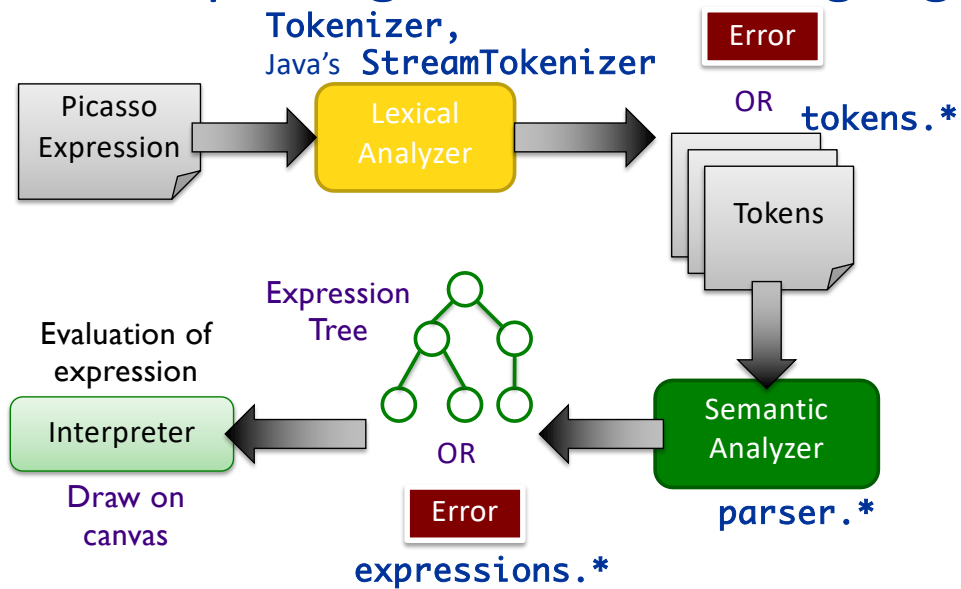
Nov 19, 2021

Sprenkle - CSCI209

5

5

Review: Interpreting the Picasso Language



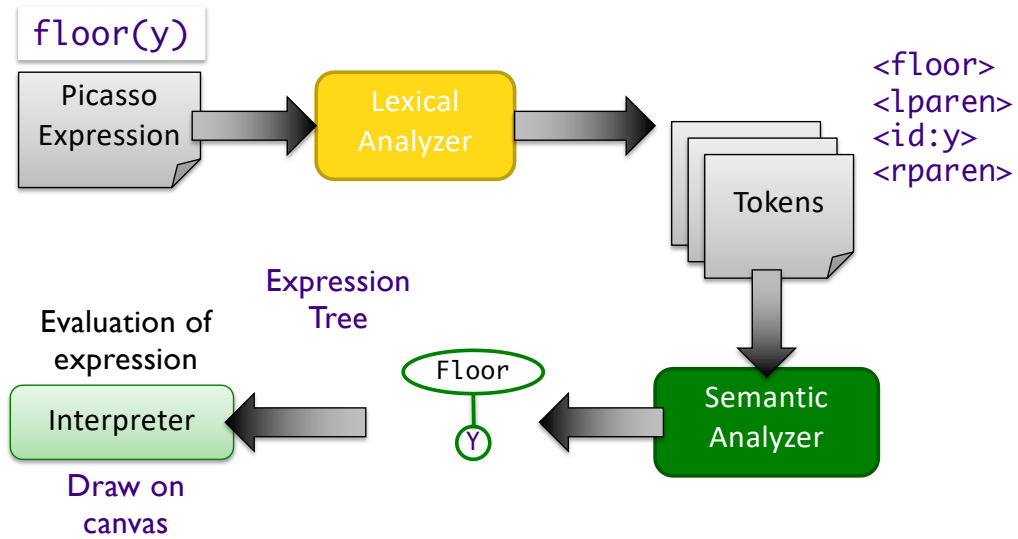
Nov 19, 2021

Sprenkle - CSCI209

6

6

Review: Interpreting the Picasso Language



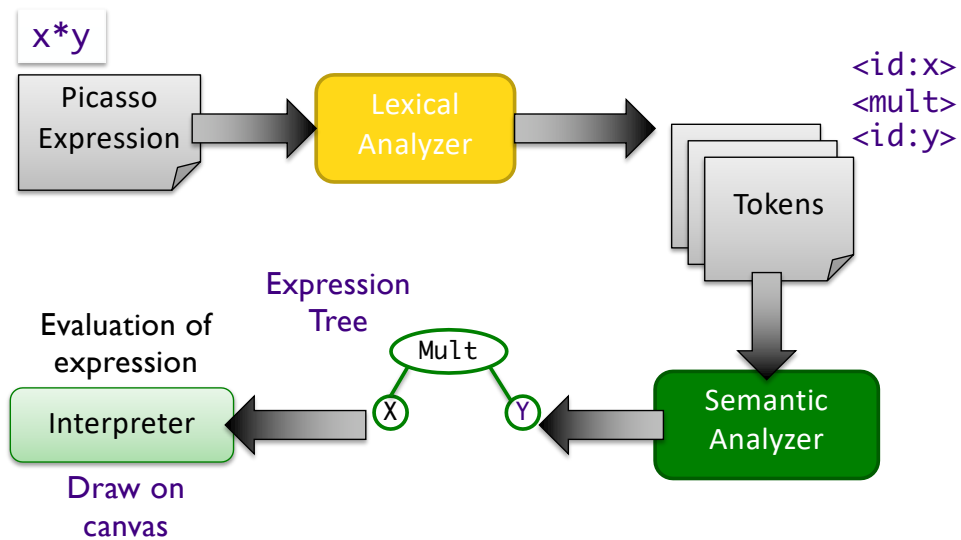
Nov 19, 2021

Sprenkle - CSCI209

7

7

Interpreting the Picasso Language



Nov 19, 2021

Sprenkle - CSCI209

8

8

What Steps Need To Be Completed?

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

Nov 19, 2021

Sprenkle - CSCI209

9

9

Dependencies?

Nov 19, 2021

Sprenkle - CSCI209

10

10

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

Nov 19, 2021

Sprenkle - CSCI209

11

11

How is the floor function parsed?

(in given code)

- What classes are needed?
- How would you add another function to the language?

Nov 19, 2021

Sprenkle - CSCI209

12

12

How is the floor function parsed?

(in given code)

- Has a *token* to represent the *floor* function
 - Same prefix as function, e.g., `FloorToken.java`
 - `floor` is listed in `functions.conf`
- `FloorAnalyzer` is the *semantic analyzer* for the function
 - Note has same prefix as function: `FloorAnalyzer.java`
 - `Analyzer` class implements `SemanticAnalyzerInterface`, returns an instance of `ExpressionTreeNode`
 - Specifically: `Floor` object

Why is the naming important for the token and analyzer?

Nov 19, 2021

Sprenkle - CSC

13

13

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`

Nov 19, 2021

Sprenkle - CSC1209

14

14

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 `ExpressionTree`
- Create an `ExpressionTreeNode` for function: `Cosine.java`

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

Nov 19, 2021

Sprenkle - CSCI209

15

15

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

Nov 19, 2021

Sprenkle - CSCI209

16

16

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 (see project description page for more)
 - 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

Nov 19, 2021

Sprenkle - CSCI209

17

17

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

Nov 19, 2021

Sprenkle - CSCI209

18

18

Planning: Division of Tasks

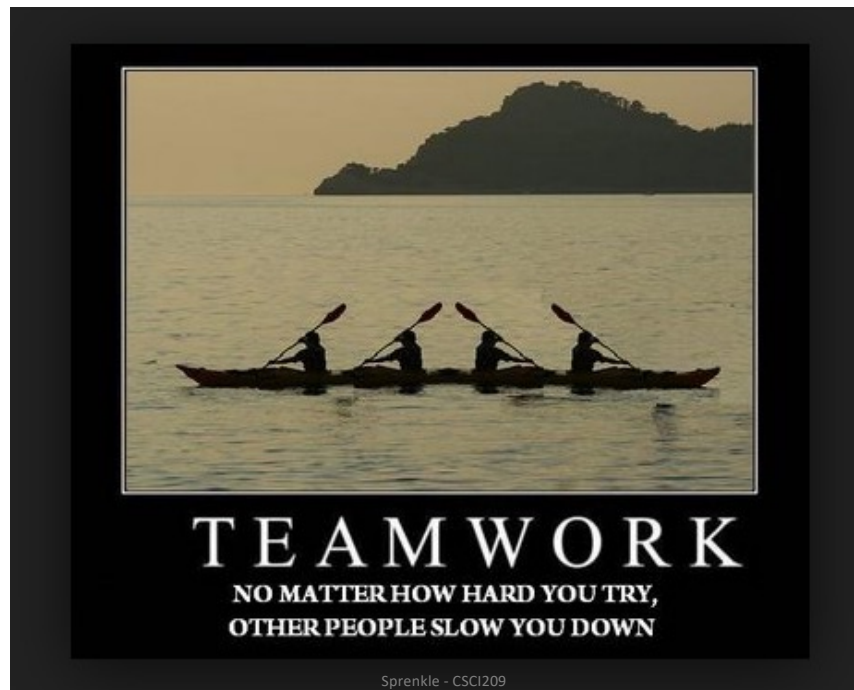
- Work in subgroups?
- Consider how not to step on each other's toes
 - Reminder: Use git branches!
- Consider best # of people per part
 - Likely will keep changing as work gets done and you learn your design
- Not recommended: Person X does all the testing
 - Perhaps pair people up to write tests for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

Nov 19, 2021

Sprenkle - CSCI209

19

19



Nov 19, 2021

Sprenkle - CSCI209

20

20

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?

Nov 19, 2021

Sprenkle - CSCI209

21

21

Review: Collaboration

- What is our workflow in Git when collaborating?
- What did you like about how your team worked together on previous projects?
 - What didn't you like?

Nov 19, 2021

Sprenkle - CSCI209

22

22

Review: Collaboration: Workflow – Seeking Feedback

1. Create a branch from `main` for your work
 - Commit periodically
 - Write descriptive comments so your team members know what you did and why
2. Push your branch
3. On GitHub, open a **Pull Request** on your branch
 - Discuss and review potential changes – can still update
 - You can tag your teammates to let them know that you've completed your work
4. Merge pull request into `main` branch
5. In Eclipse, pull `main`

Nov 19, 2021

Sprenkle - CSCI209

23

23

Collaboration Models

Good

- Team physically works all together or in subteams
- Division of labor is clear
 - Keep track of tasks, what has been completed in a document
 - Agree on team deadlines
- Good, frequent communication
 - Be a sounding board for your teammate even if you don't understand everything they are working on

Bad

- Multiple people are trying to do the same task
 - Overwriting each other's code
- Everyone is working in the main branch
- Make a plan as a team, then someone goes rogue
- Asking for help too late

Nov 19, 2021

Sprenkle - CSCI209

24

24

Student 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 intermediate deadline

Nov 19, 2021

Sprenkle - CSCI209

25

25

Implementation/Code Questions?

Nov 19, 2021

Sprenkle - CSCI209

26

26

Looking Ahead

- Next Friday, preliminary implementation deadline
 - Demo in class