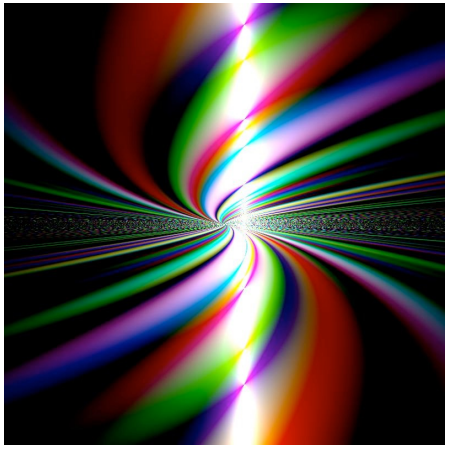
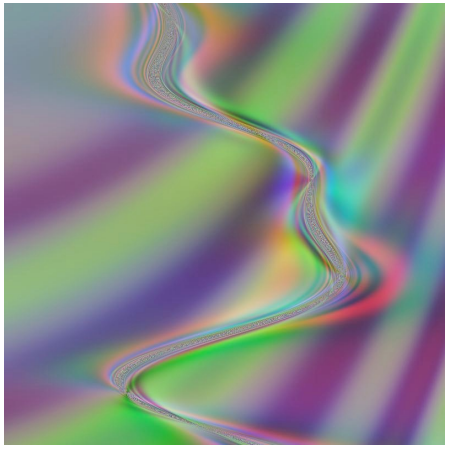
Objectives

- Analysis and Design
- Interpreting programming languages
- Final Project: Picasso

Final Project: Picasso Specification

- User can enter expressions
  - Interactively or from file
  - Language is defined in specification
- Many possible extensions


Picasso: Final Project

- Today: focus on the *requirements* of the project and bigger picture code organization

- Email with team info will come out before Wed's class
  - 3 teams of 5, 1 team of 4

Project Deliverables Timeline

Deliverable	Who	Weight	Due Date
Preparation Analysis	Individual	10%	Fri, Nov 17
Preliminary Implementation	Team	15%	Fri, Dec 1
Intermediate Implementation	Team	15%	Fri, Dec 8
Final Implementation	Team	45%	Team decides →latest 12/14
Analysis	Individual	15%	Fri, Dec 15

Before class

Week 1: Understand code base, analyze/plan project
Week 2: Implement preliminary functionality
Week 3: Implement intermediate functionality
Week 4: Implement final version of application

Teams

CodeCatalysts	AJ	Ben	Janeet	Reese	Tyler
Dynamos	Alexandra	Ashton	Bianca	Ciel	John
Visionaries	Chelsea	Elle	James	Michael	Will
Wizards	Lakpa	Linh	Liz	Trey	

Teams, alphabetically by first name

Teams

GameChangers	Aiden	Chaz	Han	Jenna	Lydia
HotShots	Alexander	Connor	Desire	Ford	Nabil
Invincibles	Barrett	Mark	Nick	Wonjun	Zach
Phenoms	Colin	Jack	Renan	Stephen	

Teams, alphabetically by first name

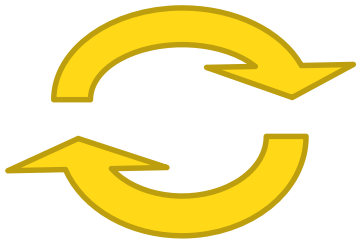
ANALYSIS & DESIGN: FORMALIZED

Analysis Phase

- Create an abstract model in client's vocabulary
- Strategy:
  1. Identify classes that model (shape) system as set of abstractions
  2. Determine each class's purpose or main responsibility
     - API
     - State
  3. Determine helper classes for each
     - Help complete responsibilities

Analysis Phase Discussion

- Expect to **iterate**
  - Won't find all classes at first
    - Especially helpers
  - Won't know all responsibilities



- Uncertainty in problem statement
  - May be concerns that need to be settled
  - Try to understand requested software system at level of those requesting software
- Rarely one true correct best design

Identification of Classes

- Potentially model the system
- Usually **nouns** from problem description or from domain knowledge
- Model real world/problem domain whenever possible
  - More understandable software
  - Helps during maintenance when someone unfamiliar with system must update/fix code

Identifying Responsibilities

- Responsibilities convey purpose of class, its role in system
- Questions to Ask:
  - What are the other responsibilities needed to model the solution?
    - Which class should take on this particular responsibility?
  - What classes help another class fulfill its responsibility?

Have You Modeled Everything?

- Strategy: Role playing
- Act as different classes: can you do everything you want in various scenarios?
  - Fill in missing classes, responsibilities
  - Methods: parameters, what returned
  - Restructure as necessary
    - No code yet so not actually refactoring
- Example **use cases**/scenarios:
  - A student tries to register for a class with no open seats
  - A professor looks at students' interim grades

Definition of Use Case

- Description of steps or actions between a user and a software system towards some goal

- What else can use cases be used for?
  - Test Cases!

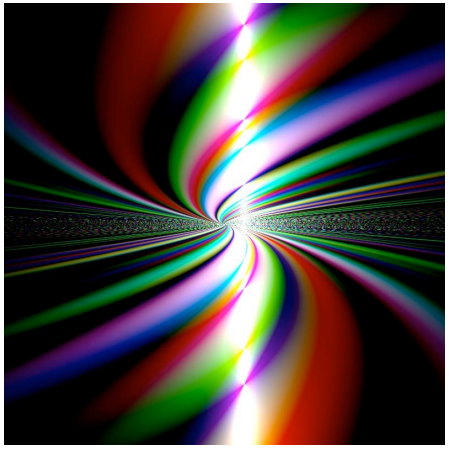
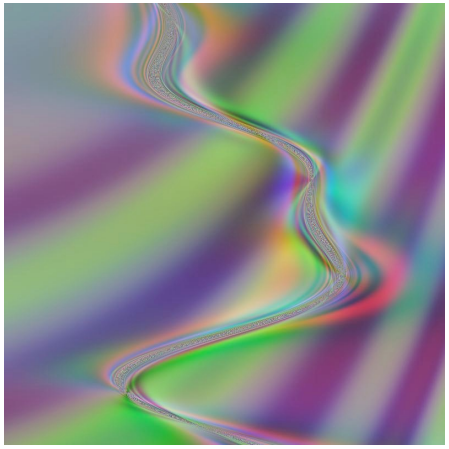
TEAM FINAL PROJECT

Project Metrics

- >1700 lines of code
  - Even more by the time your team is done
- Good for gaining experience
  - Large (for a course) piece of existing code that you need to build on
- Good for job interviews
  - Know the number of lines of code

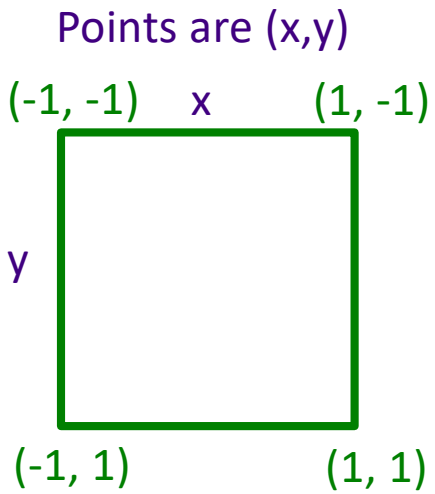
Final Project: Picasso Specification

- User can enter expressions
  - Interactively or from file
  - Language is defined in specification
- Many possible extensions


Picasso Project Overview

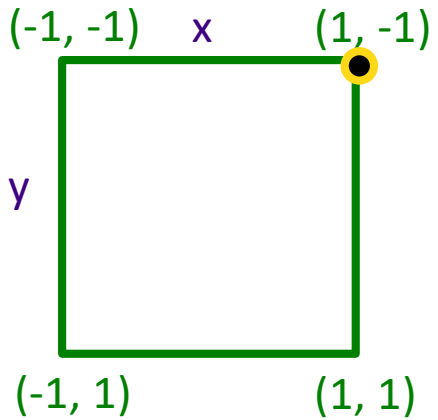
- Goal: Generate images from expressions
- Every pixel at position (x,y) gets assigned a color, computed from its x- and y-coordinate and the given expression
  - Range for x and y is [-1, 1]
- Colors are represented as RGB (red, green, blue) values
  - R, G, B component's range: [-1, 1]
  - Black is [-1,-1,-1]
  - Red is [1,-1,-1]
  - Yellow is [1, 1,-1]



How is white represented?

Generating Images from Expressions

- ***Expressions*** at a specific (x,y) point/pixel evaluate to *RGB colors* [r,g,b]
  - `pixels[x][y] = expression.evaluate(x, y)`
- **x** evaluates to RGB color [x, x, x]
- In top right corner,
  - x evaluates to [1, 1, 1]
  - y evaluates to [-1, -1, -1]



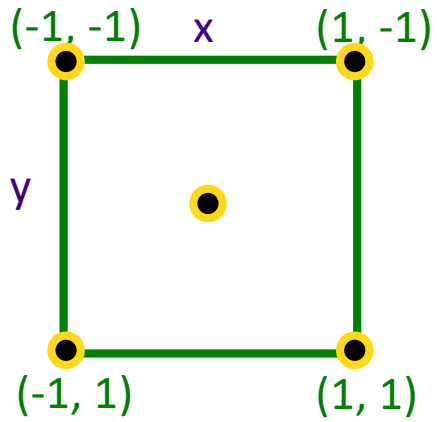
Generating Images from Expressions

```
For all x:
    For all y:
        pixels[x][y] = expression.evaluate(x, y)
```

Consider evaluating expression as

f(x, y) = expression

at various points in the image



Generating Images from Expressions

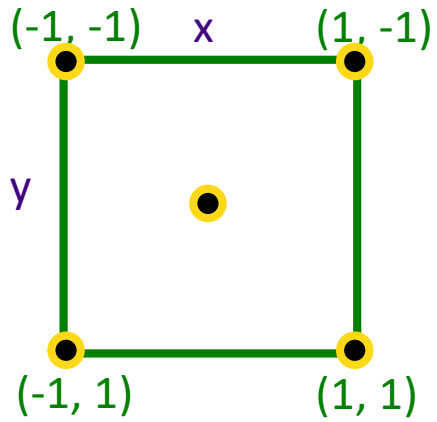
```
For all x:
    For all y:
        pixels[x][y] = expression.evaluate(x, y)
```

Consider evaluating expression as

f(x, y) = expression

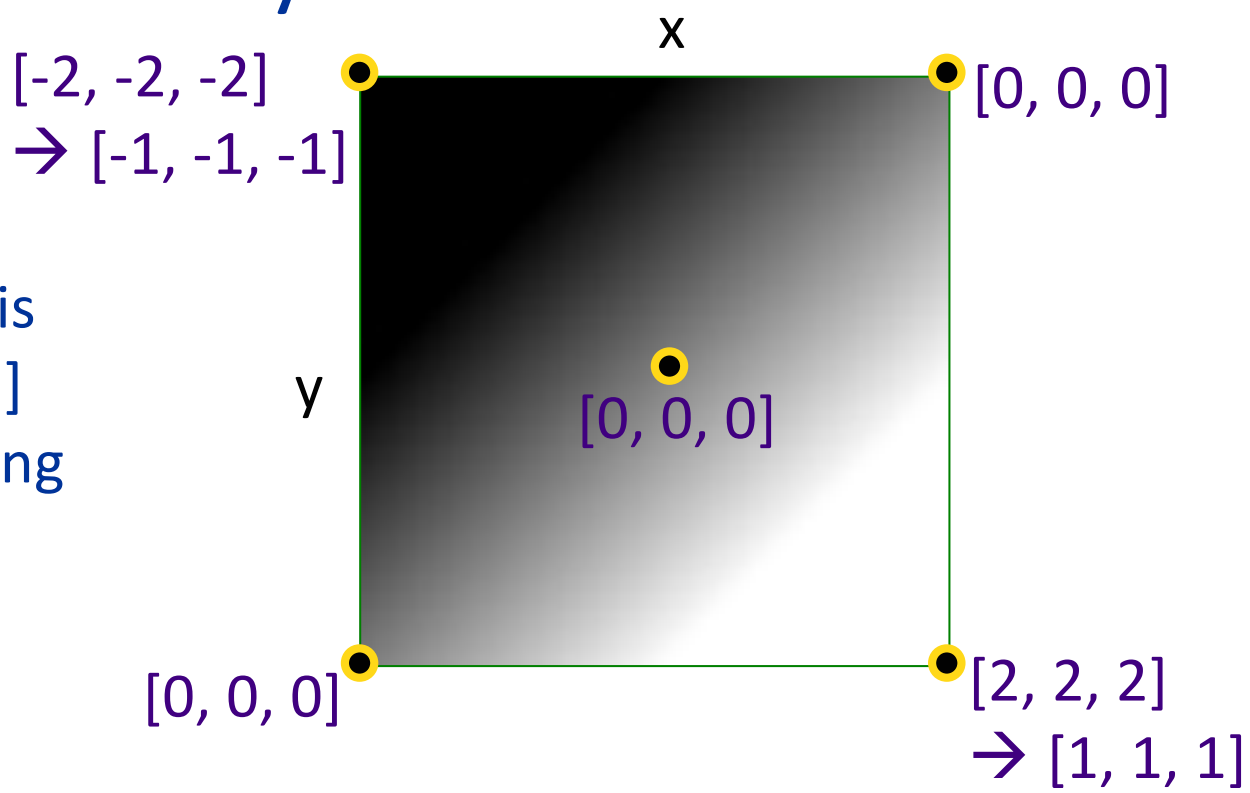
at various points in the image

Example: expression is x+y



Resulting Image for x+y

- Recall that color range is clamped to range [-1, 1]
- Green outline for framing purposes only



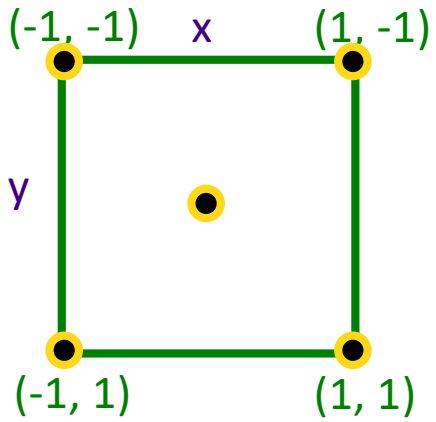
Generating Images from Expressions

```
For all x:
    For all y:
        pixels[x][y] = expression.evaluate(x, y)
```

Consider evaluating expression as

f(x, y) = expression

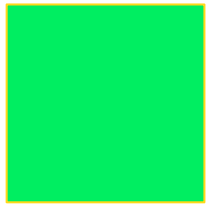
at various points in the image



What is the resulting image if the *expression* is

- `[-1, 1, -1]` ?
- `x` ?
- `x*y` ?

Generated Images from Expressions



[-1, 1, -1]

x

x*y

```
For all x:
        For all y:
                pixels[x][y] = expression.evaluate(x, y)
```

PROCESSING PROGRAMMING LANGUAGES

Programming Language Syntax & Semantics

- What does an assignment statement look like in Java?
  - What can be on the left hand side?
    - What are the rules for an *identifier* in Java?
  - What can be on the right hand side?
- What does a multiplication expression look like?
- How do we evaluate arithmetic expressions?

Programming Language Design

- Must be unambiguous
  - Programming Language defines a ***syntax*** and ***semantics***
- Interpreting programming languages
  1. Parse program into tokens
  2. Verify that tokens are in a valid form
  3. Generate executable code
  4. Execute code

Parsing into Tokens

- Example: x = 4*3; →

```
<id> <assignment> <num> <mult> <num> <endofstmt>
```

- Example: x = * 3 5;

```
<id> <assignment> <mult> <num> <num> <endofstmt>
```

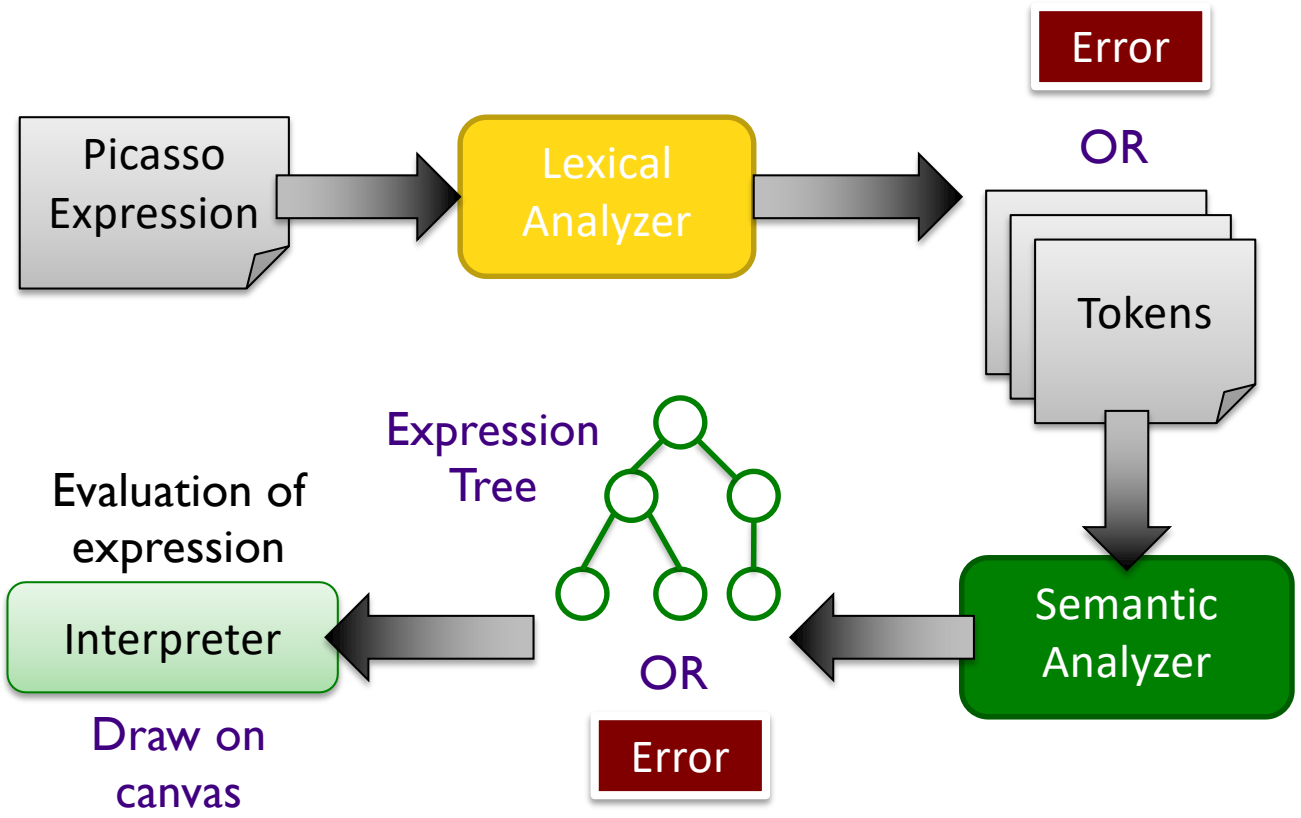
- Tokenizer doesn't care if statement is not valid
  - handled in next step
- Error example: 1x = 4**3;
  - 1x and ** are not valid tokens in Java

Process of Understanding Code: Building Your Mental Model

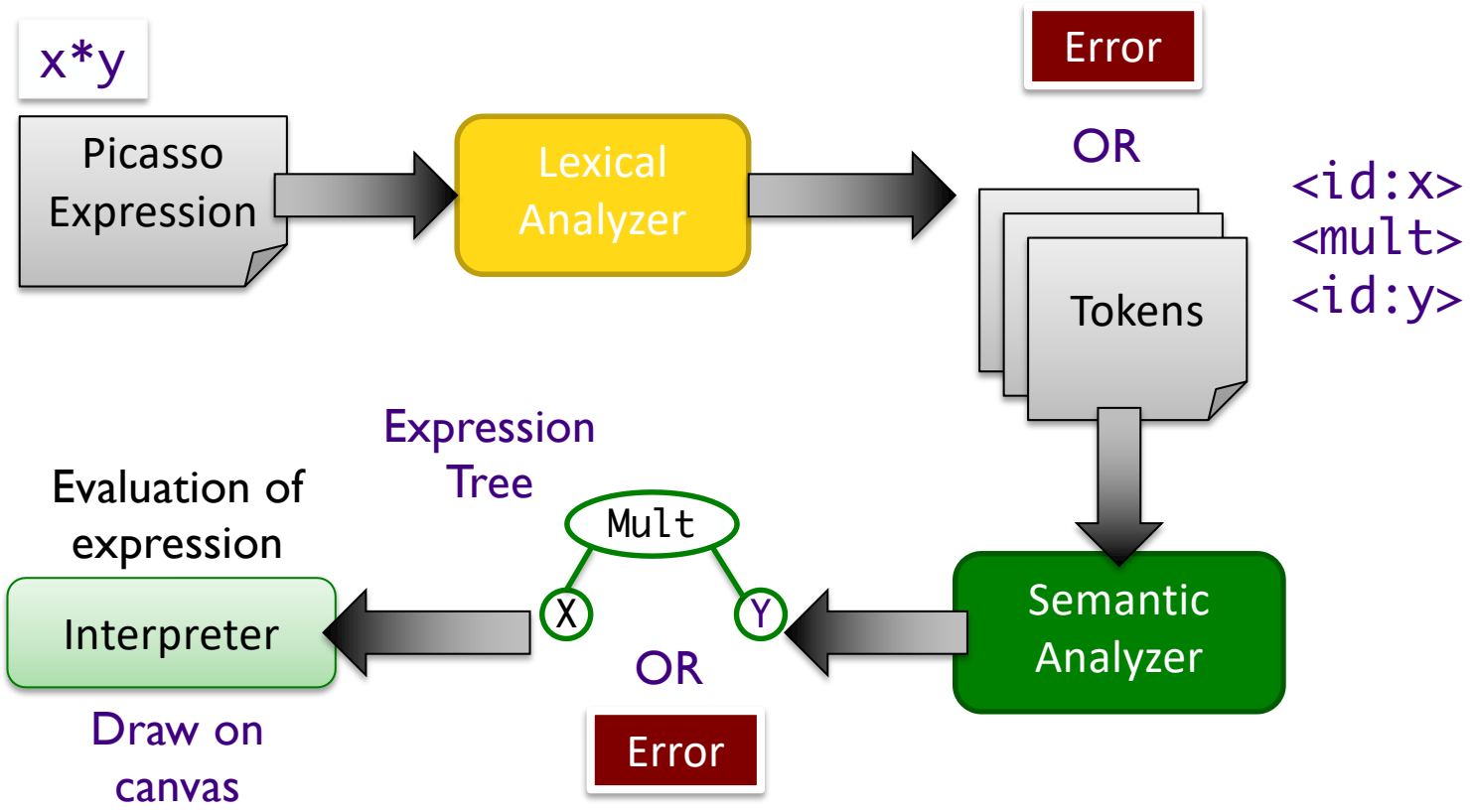
- Apply spiral model to understanding code
- Review problem specification (low-cost effort)
- Explore project at the top-level (low-cost effort)
  - Look at packages, class names
  - Don't take a deep-dive until you have the bigger picture

`https://cs.wlu.edu/~sprenkles/cs209/projects/picasso/doc/`

Interpreting the Picasso Language



Interpreting the Picasso Language



What We Need to Do/Represent

- Lexical Analysis
- Semantic Analysis
- Evaluation

What We Need to Do/Represent

- Lexical Analysis
  - Recognize/create tokens
  - Report errors in creating tokens
- Semantic Analysis
  - Convert infix tokens into postfix
    - Report errors
  - Parse tokens into *expressions* (expression tree)
    - Report errors
- Evaluation
  - Evaluate expressions

Understanding the Code

- How would you run Picasso?

- How does the given code map to lexical analysis, semantic analysis, and evaluation components?
  - Look for packages, classes that map to these steps

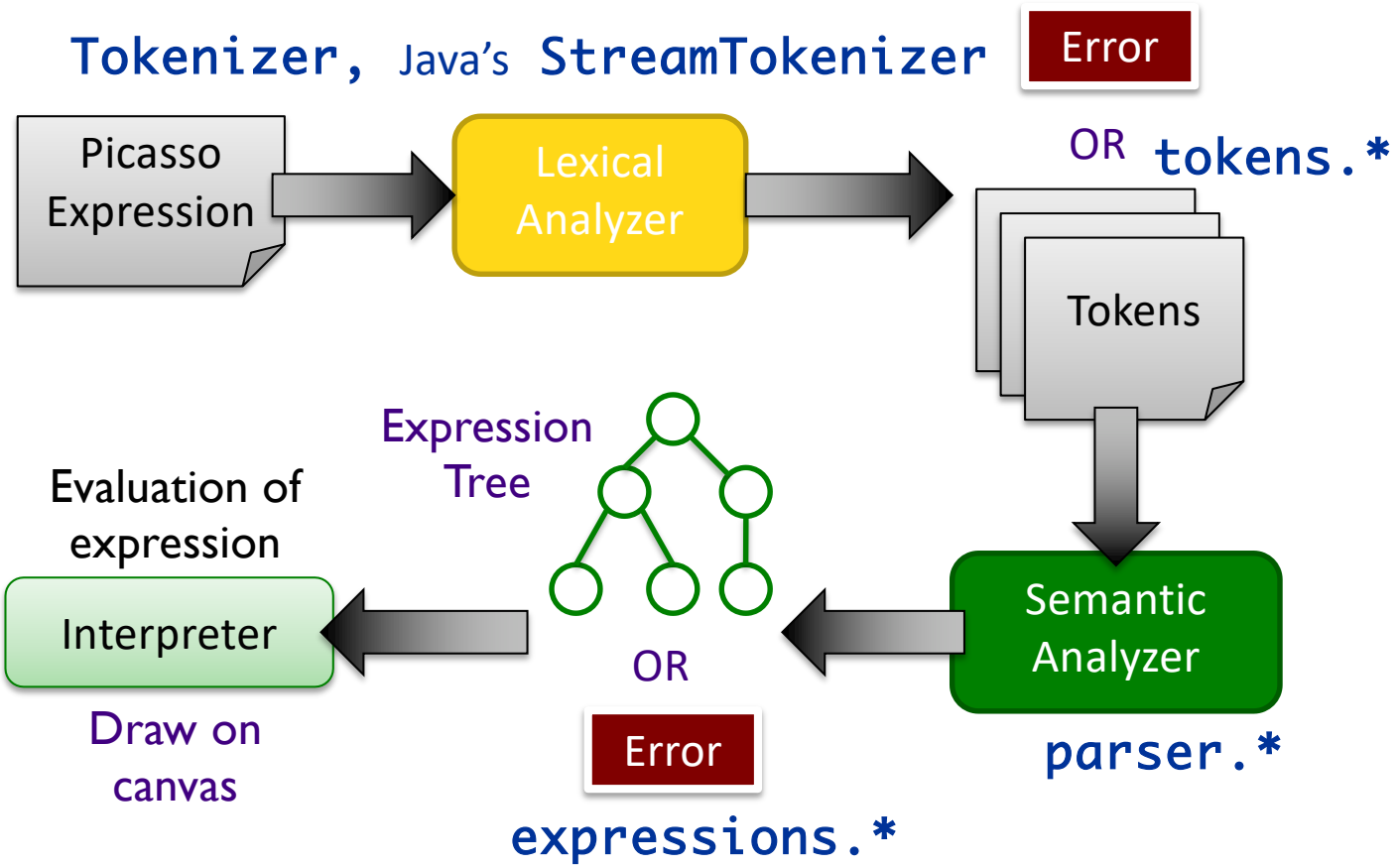
Process of Understanding Code: Building Your Mental Model

- Look for important words/terms from problem domain
- Look for terms from design patterns
- Put code in black boxes or group code together

Process of Understanding Code: Building Your Mental Model

- After you have the big picture, look at most important classes
- Decide: Does this class merit a closer look? Or do I just need the big picture of what it does?
  - Lean towards the latter towards the beginning
  - Look for class hierarchy and focus on parent/base classes
- Iterate!
  - Grow your mental model
  - What a "closer look" means changes over time
    - Early: what public methods does the class have? What does the documentation say they do? What do they return?
    - Later: what do these methods do? How does this class interact with other objects?

Interpreting the Picasso Language



Understanding the Code: Lexical Analysis

- Process
  - `picasso.parser.Tokenizer`
  - `picasso.parser.tokens.TokenFactory`
- Output:
  - `picasso.parser.tokens.*`

`FloorToken`

Understanding the Code: Semantic Analysis

- Process
  - `picasso.parser.ExpressionTreeGenerator`
  - `picasso.parser.SemanticAnalyzer`
  - `picasso.parser.*Analyzer`
- Output
  - `picasso.parser.language.expressions.*`

`FloorAnalyzer`

Understanding the Code: Evaluation

- Process
  - `picasso.parser.language.ExpressionTreeNode`
- Output:
  - `RGBColor`
- Displayed in `PixMap` on `Canvas`

Understanding the Code: Evaluation

- Key Parent class:

`picasso.parser.language.ExpressionTreeNode`

```
public abstract RGBColor evaluate(double x, double y);
```

- "Old" version of expressions:
  - `ReferenceForExpressionEvaluations`

TODO

- Project Analysis due Friday