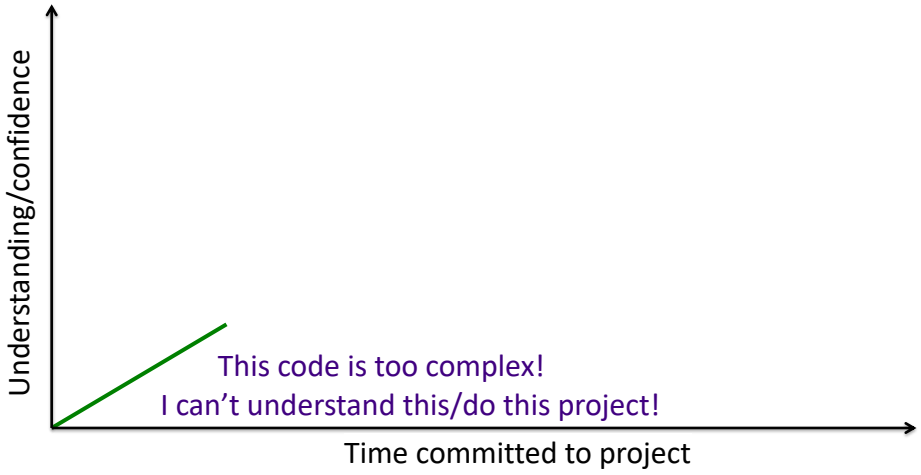
Objectives

- Picasso Design
- Reflection
- GUIs in Java
  - Anonymous inner classes

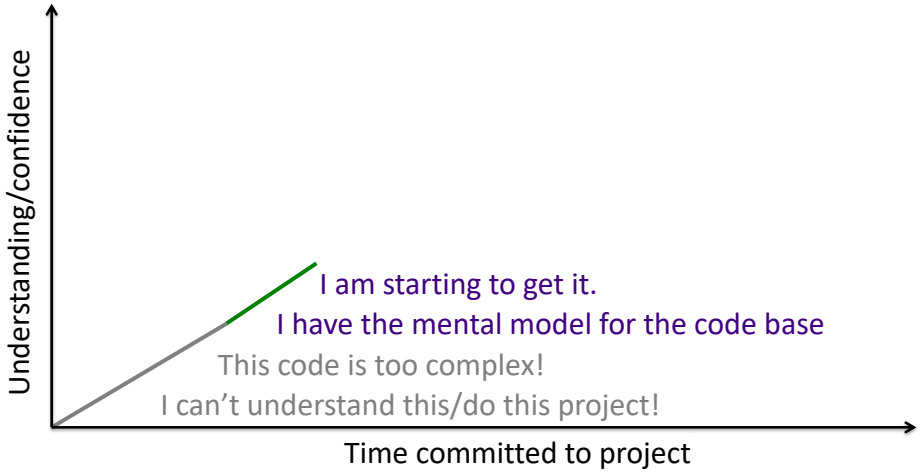
1

Typical Trajectory of Projects



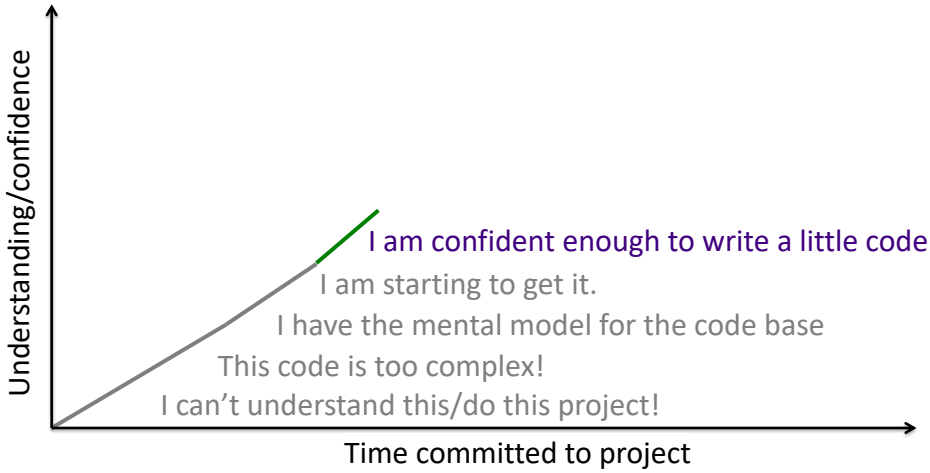
2

Typical Trajectory of Projects



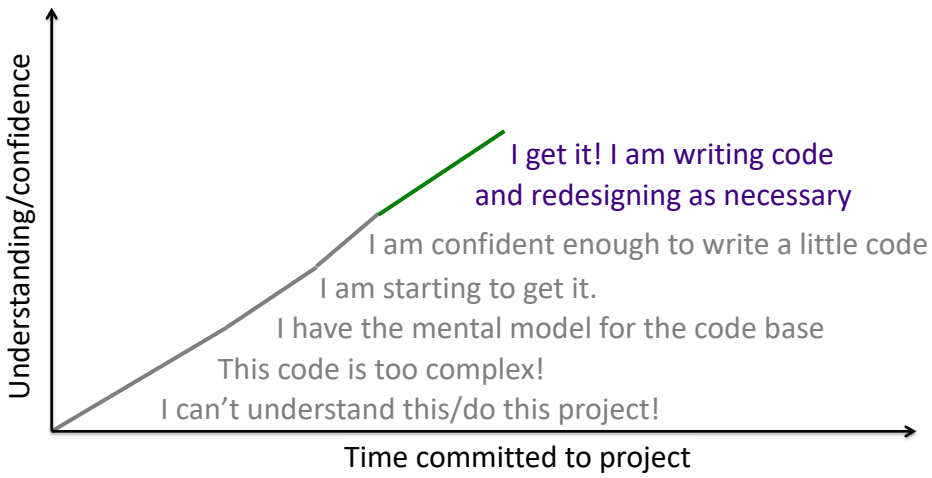
3

Typical Trajectory of Projects



4

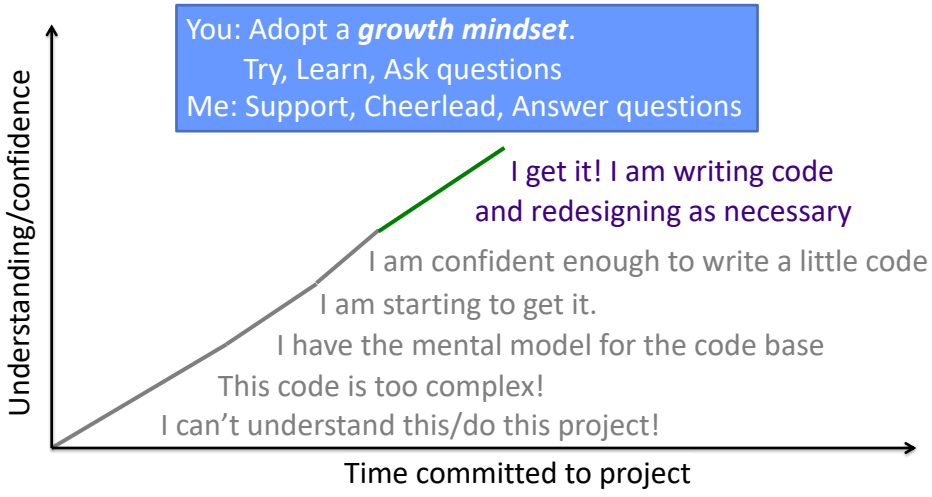
Typical Trajectory of Projects



5

Our Responsibilities

You: Adopt a ***growth mindset***.
Try, Learn, Ask questions
Me: Support, Cheerlead, Answer questions



6

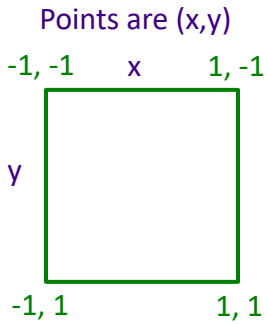
Review

- What is the goal of the Picasso project?
- When you click the Evaluate button in the given version of Picasso, it generates the image for `floor(y)`
  - Explain why the image looks as it does:

- How does an interpreter interpret a programming language?
  - How do those steps map to the Picasso code base?
- What should we think about during design and analysis of a project?
  - What are best practices?

Review: Picasso Project Overview

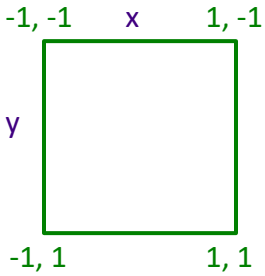
- Goal: Generate images from expressions
- Every pixel at position (x,y) gets assigned a color, computed from its x and y coordinate and the given expression
  - Range for x and y is [-1, 1]
- Colors are represented as RGB [red, green, blue] values
  - Component's range [-1, 1]
  - Black is [-1,-1,-1]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



Review: 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)`

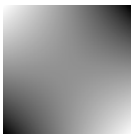
- **x** evaluates to RGB color [x, x, x]
- In top right corner,
  - x evaluates to [1, 1, 1]
  - y evaluates to [-1, -1, -1]



Review: Generated Expressions


[-1, 1, -1]


x


x*y

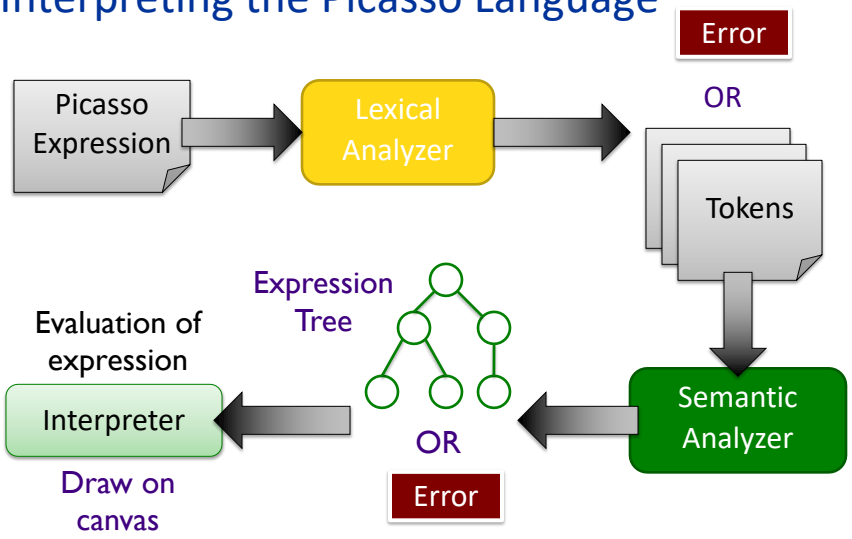
```
For all x:
      For all y:
            pixels[x][y] = expression.evaluate(x, y)
```

Review: 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

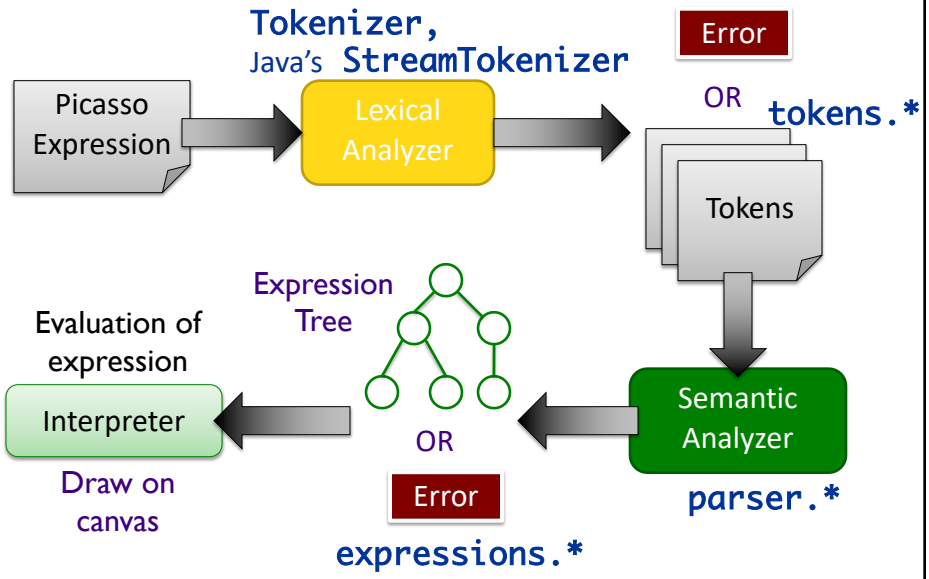
11

Review: Interpreting the Picasso Language



12

Interpreting the Picasso Language

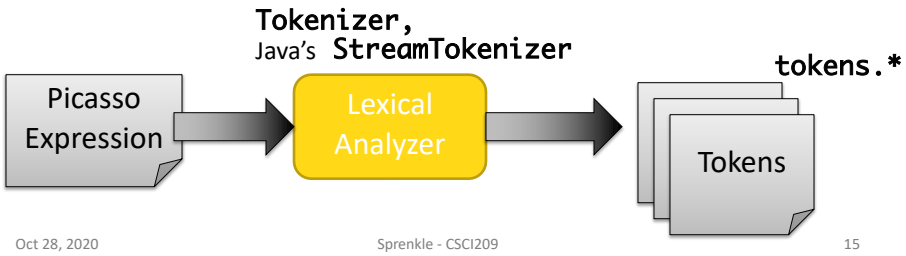


Process of Understanding Code: Building Your Mental Model

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

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
- Example:



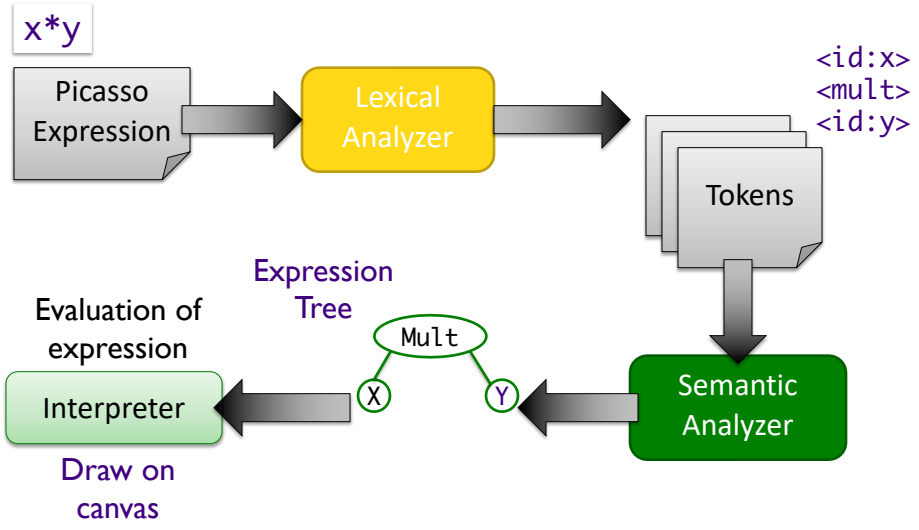
15

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
- Iterate!
  - Grow your mental model
  - What a "closer look" means changes over time
    - Early: what methods does the class have? What classes does this object interact with?
    - Later: what do these methods do? How does this class interact with other objects?

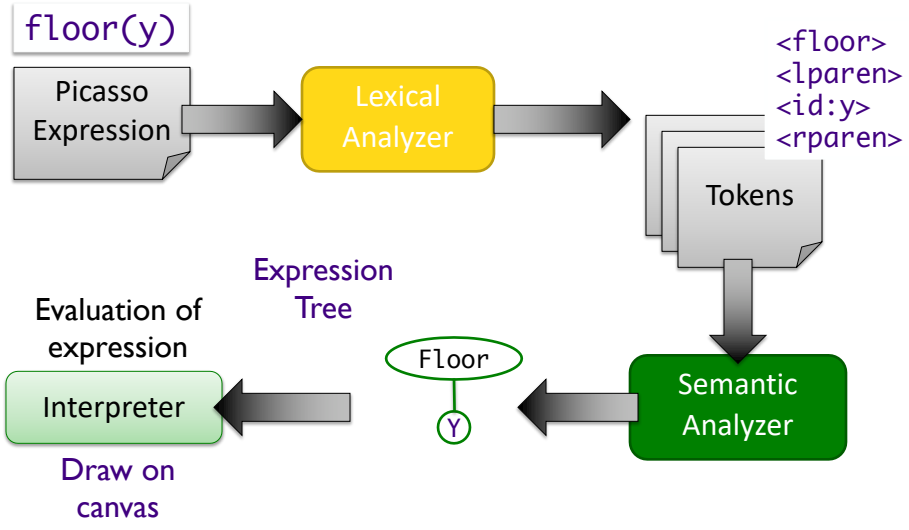
16

Interpreting the Picasso Language



17

Interpreting the Picasso Language



18

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`

`Floor`

Understanding the Code: Evaluation

- Key Parent class: `picasso.parser.language.ExpressionTreeNode`

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

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

Using Reflection in Java

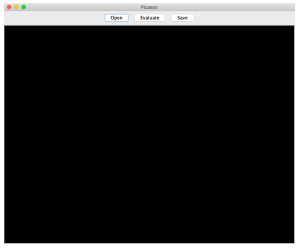
- Can create objects of a class through the *name* of the class
- Example adapted from MutantMaker:

```
public static void initMutantMaker() {
    mutants = new Mutant[numMutants];
    mutants[0] = new Wolverine();
    for (int i = 1; i < numMutants; i++) {
        Class<?> mutantClass;
        try {
            mutantClass = Class.forName("mutants.Mutant"+ i);
            mutants[i] = (Mutant)
                mutantClass.getDeclaredConstructor().newInstance();
        } catch (Exception e) {
                e.printStackTrace();
        }
    }
}
```


Using Reflection in Java

- Can create objects of a class through the *name* of the class
- Used in SemanticAnalyzer
  - Gets list of functions
    - Read from conf/functions.conf
  - Maps a token to the class responsible for parsing that type of token
  - When SemanticAnalyzer sees that token, calls the respective analyzer to parse
  - Example: FloorToken maps to the FloorAnalyzer
    - FloorAnalyzer pops the Floor token off the stack and then parses the (one) parameter for the floor function

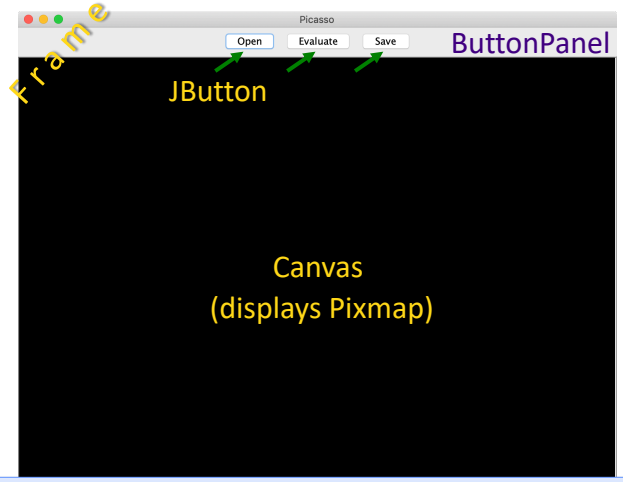
Understanding Code: A Top-Down Approach



- Run program
- Start at Main.java
  - Follow calls to see how GUI is created
    - Breadth or depth-first search
  - What classes make up the GUI?
- GUIs often follow the MVC design pattern
  - Identify the model, view-controller in Picasso

25

Picasso GUI



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

26

Understanding GUI Code

- In ButtonPanel.java, buttons are associated with a command or action

```
private Canvas myView;
…
public void add(String buttonText,
                    final Command<Pixmap> action) {
    JButton button = new JButton(buttonText);
    button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            action.execute(myView.getPixmap());
            myView.refresh();
        }
    });
    add(button);
}
```


Understanding GUI Code

- In ButtonPanel.java, buttons are associated with a command or action

JButton's ActionListener says what to do if button is pressed

```
private Canvas myView;
…
public void add(String buttonText,
                    final Command<Pixmap> action) {
    JButton button = new JButton(buttonText);
    button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            action.execute(myView.getPixmap());
            myView.refresh();
        }
    });
    add(button);
}
```

Understanding GUI Code

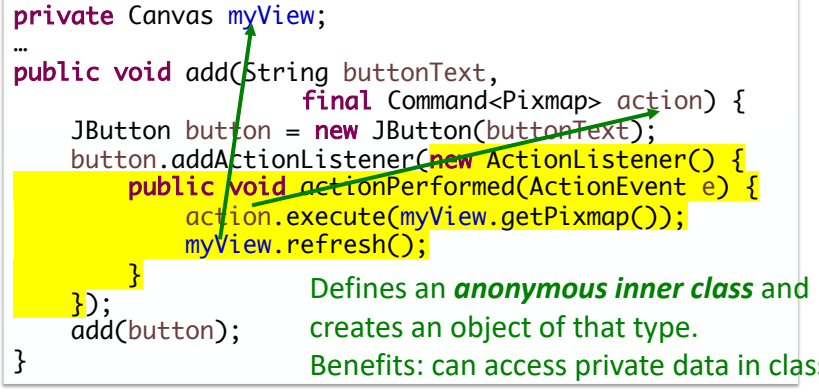
- In ButtonPanel.java, buttons are associated with a command or action

```
private Canvas myView;
…
public void add(String buttonText,
                    final Command<Pixmap> action) {
    JButton button = new JButton(buttonText);
    button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            action.execute(myView.getPixmap());
            myView.refresh();
        }
    });
    add(button);
}
```


Understanding GUI Code

- In ButtonPanel.java, buttons are associated with a command or action

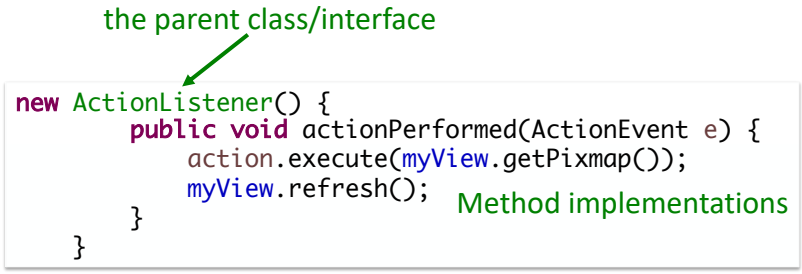
```
private Canvas myView;
…
public void add(String buttonText,
                    final Command<Pixmap> action) {
    JButton button = new JButton(buttonText);
    button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            action.execute(myView.getPixmap());
            myView.refresh();
        }
    });
    add(button);
}
```



Defines an ***anonymous inner class*** and creates an object of that type.
Benefits: can access private data in class

Anonymous Inner Classes

- Common way to write (certain) code
- No classname
  - Class is anonymous
- Extends a parent class or implements an interface



Understanding Picasso Code

- Start in `Evaluator` command's `execute` method

TODO

- Project Analysis due Friday