

Objectives

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

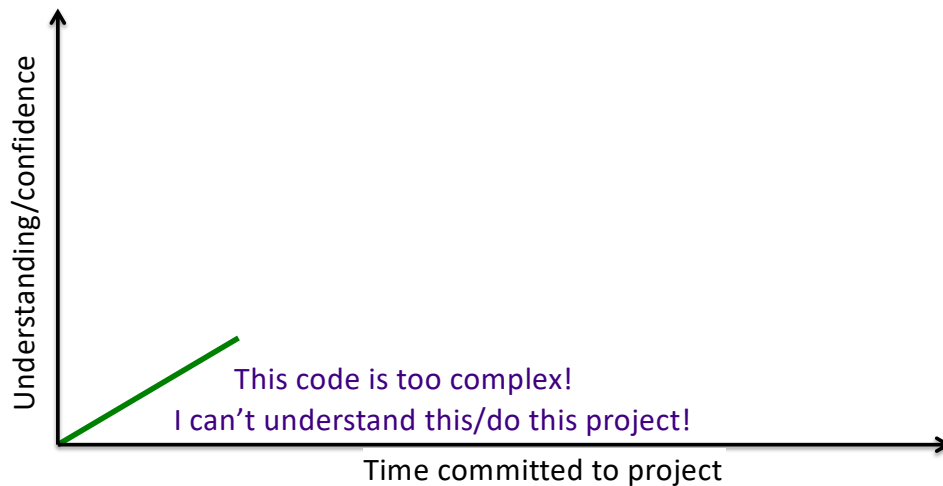
Nov 17, 2021

Sprengle - CSCI209

1

1

Typical Trajectory of Projects



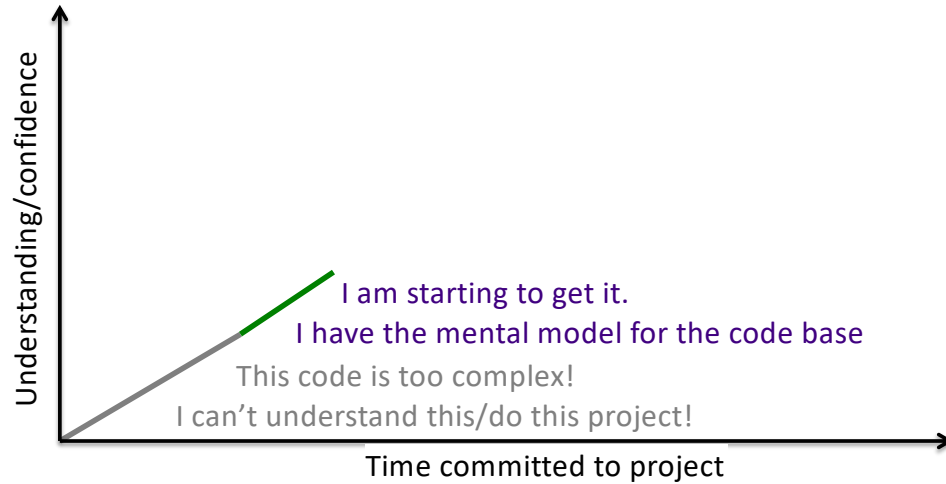
Nov 17, 2021

Sprengle - CSCI209

2

2

Typical Trajectory of Projects



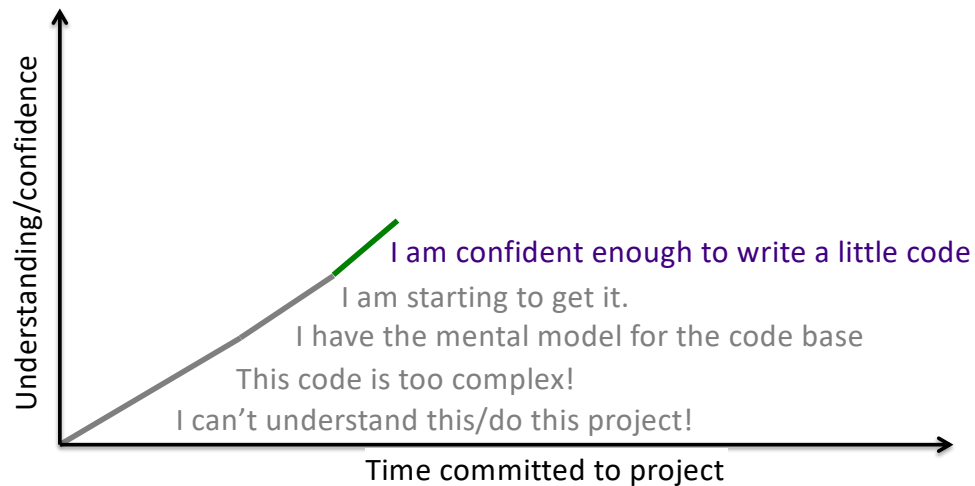
Nov 17, 2021

Sprengle - CSCI209

3

3

Typical Trajectory of Projects



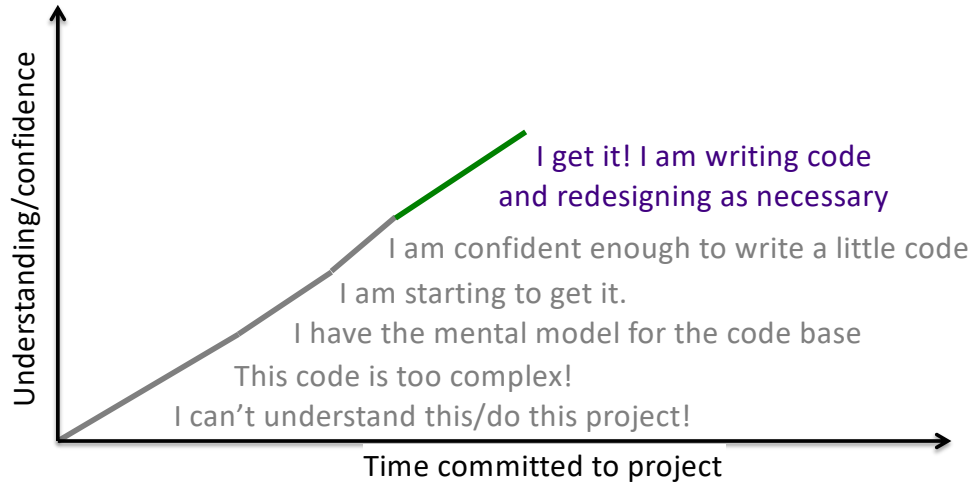
Nov 17, 2021

Sprengle - CSCI209

4

4

Typical Trajectory of Projects



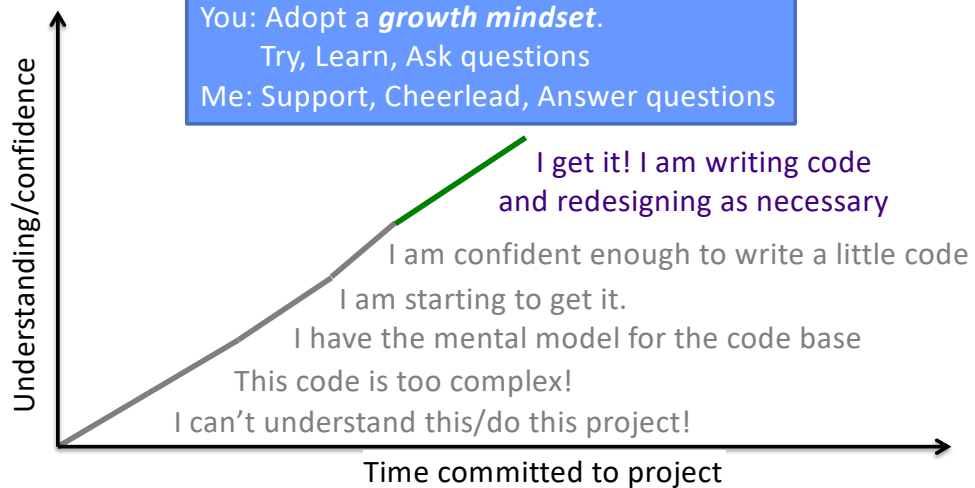
Nov 17, 2021

Sprenkle - CSCI209

5

5

Our Responsibilities



Nov 17, 2021

Sprenkle - CSCI209

6

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 like the image:
 - Include the constraints/rules of Picasso
- 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?
- How should we learn a code base?



Nov 17, 2021

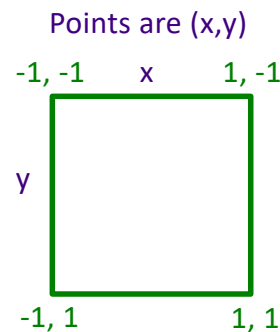
Sprenkle - CSCI209

7

7

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 $[1, 1,-1]$



Nov 17, 2021

Sprenkle - CSCI209

8

8

Review: Generating Images from Expressions

- **Expressions** at a specific (x,y) point/pixel evaluate to *RGB colors* [r,g,b]

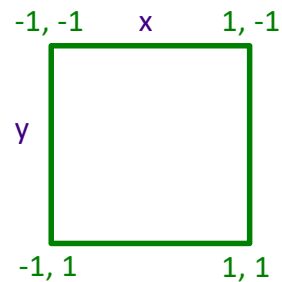
➤ `pixels[x][y] = expression.evaluate(x, y)`

- **x** evaluates to RGB color [x, x, x]

- In top right corner,

- x evaluates to [1, 1, 1]

- y evaluates to [-1, -1, -1]



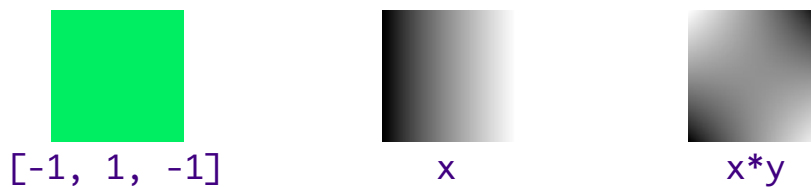
Nov 17, 2021

Sprenkle - CSCI209

9

9

Review: Generated Expressions



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

Nov 17, 2021

Sprenkle - CSCI209

10

10

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

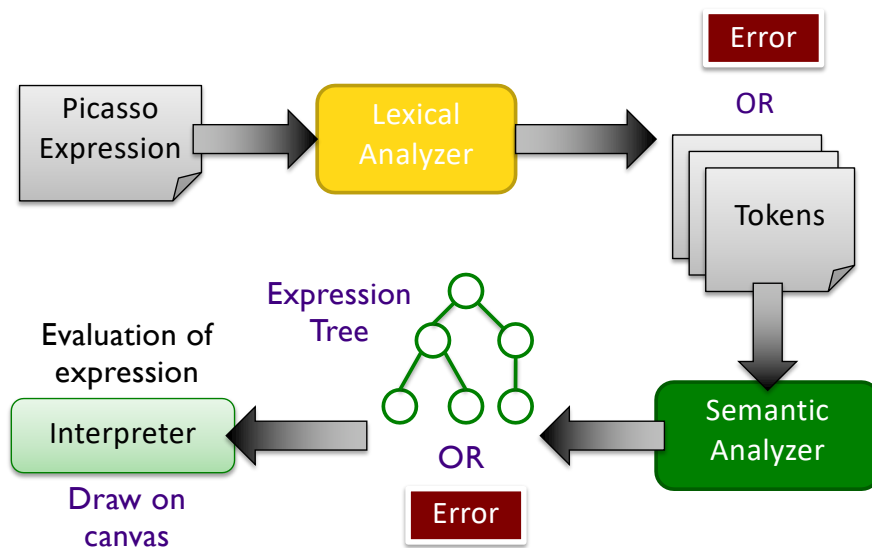
Nov 17, 2021

Sprenkle - CSCI209

11

11

Review: Interpreting the Picasso Language



Nov 17, 2021

Sprenkle - CSCI209

12

12

Understanding the Code

- How does the given code map to lexical analysis, semantic analysis, and evaluation components?
 - Look for packages, classes that map to these steps
- Suggestions:
 - Look for important words/terms from problem domain
 - Look for terms from design patterns
 - Put code in black boxes or group code together

Nov 15, 2021

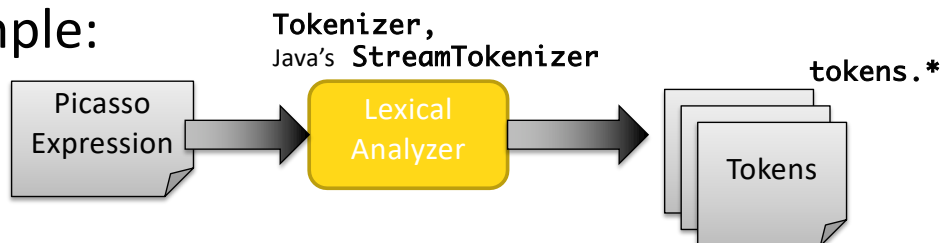
Sprenkle - CSCI209

13

13

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:



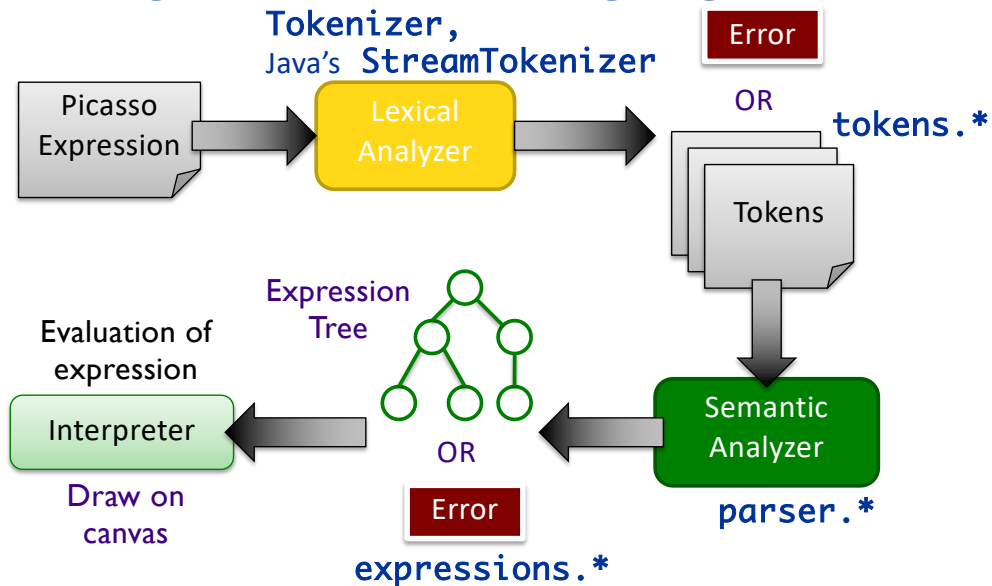
Nov 17, 2021

Sprenkle - CSCI209

14

14

Interpreting the Picasso Language



Nov 17, 2021

Sprenkle - CSCI209

15

15

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

Nov 17, 2021

Sprenkle - CSCI209

16

16

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?

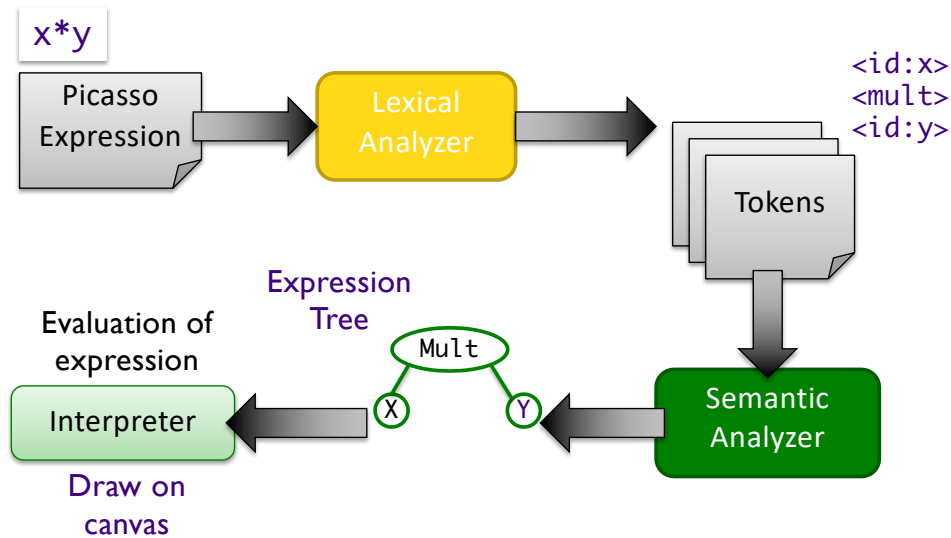
Nov 17, 2021

Sprenkle - CSCI209

17

17

Interpreting the Picasso Language



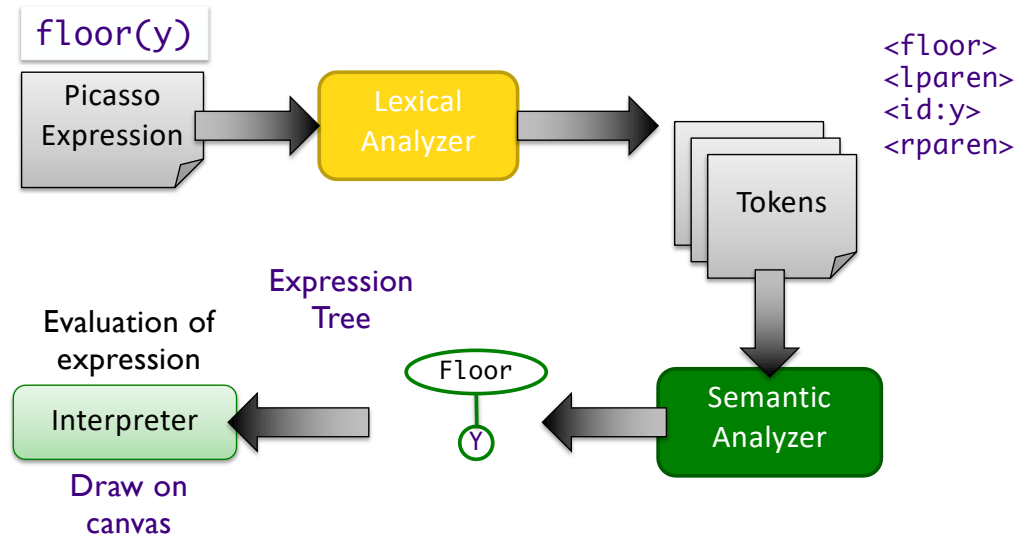
Nov 17, 2021

Sprenkle - CSCI209

18

18

Interpreting the Picasso Language



Nov 17, 2021

Sprenkle - CSCI209

19

19

Understanding the Code: Lexical Analysis

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

Nov 17, 2021

Sprenkle - CSCI209

FloorToken

20

20

Understanding the Code: Semantic Analysis

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

Understanding the Code: Evaluation

- Process
 - `picasso.parser.language.ExpressionTreeNode`
- Output:
 - `picasso.parser.language.expressions.RGBColor`
- Displayed in QPixmap 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

Nov 17, 2021

Sprenkle - CSCI209

23

23

Using Reflection in Java

- *Reflection* allows us to create objects of a class using 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();
        }
    }
}
```

Nov 17, 2021

24

24

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

Nov 17, 2021

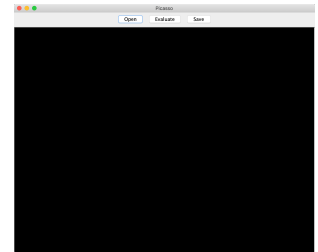
Sprenkle - CSCI209

25

25

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



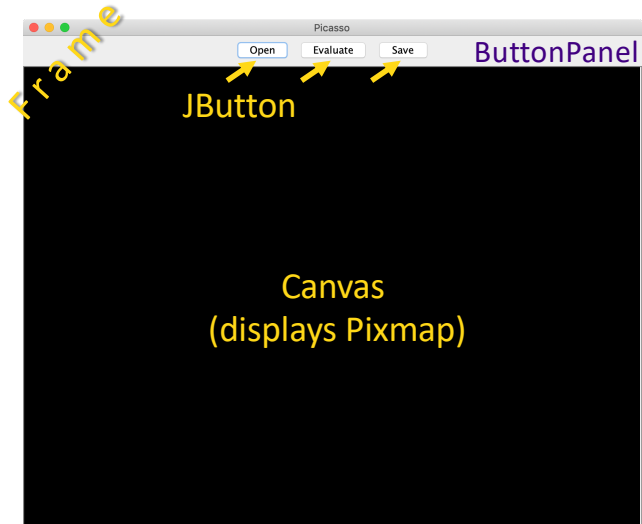
Nov 17, 2021

Sprenkle - CSCI209

26

26

Picasso GUI



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

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

Nov 17, 2021

Sprenkle - CSCI209

27

27

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);
}
```

Nov 17, 2021

Sprenkle - CSCI209

28

28

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);
}
```

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

Nov 17, 2021

Sprenkle - CSCI209

29

29

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);
}
```

Nov 17, 2021

Sprenkle - CSCI209

30

30

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

Nov 17, 2021

Sprenkle - CSCI209

31

31

Anonymous Inner Classes

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

the parent class/interface

```
new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        action.execute(myView.getPixmap());
        myView.refresh();
    }
}
```

Method implementations

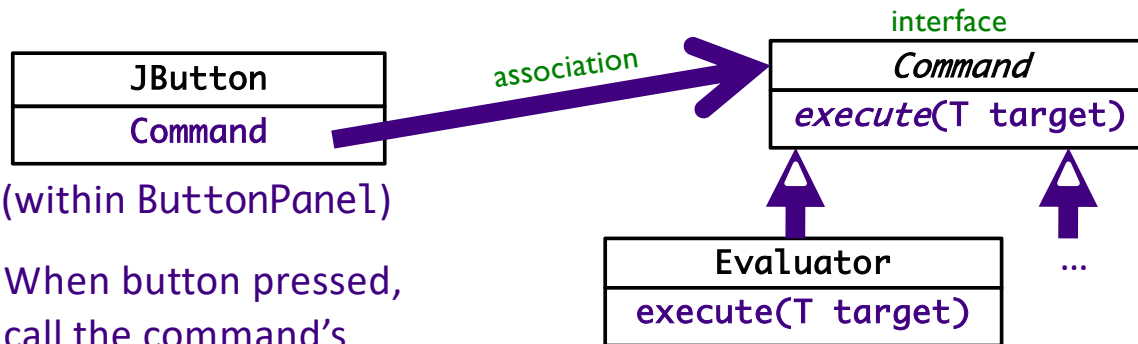
Nov 17, 2021

Sprenkle - CSCI209

32

32

Picasso GUI: ButtonPanel



Nov 10, 2021

33

33

Understanding Picasso Code

- Start in Evaluator command's execute method

Nov 17, 2021

Sprenkle - CSCI209

34

34

TODO

- Project Analysis due Friday before class