

# Objectives

- Picasso Discussion

- Best development practices
- Singleton Design Pattern
- Code Smell
- Testing Picasso
- Preliminary implementation deliverable and demo

# Extra Credit Opportunity: Nate Tsang

**AMA (Ask Me Anything)**  
November 30  
11:45-12:45 & 4:45-5:45  
P404 (CS Seminar Room)  
sign up with QR Code on Box

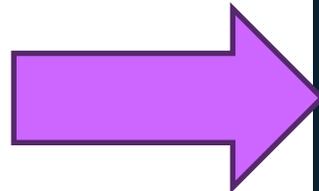


**NATE TSANG**  
AI RESEARCH SCIENTIST



**"AI AND GAMING"**  
December 1 5-6pm  
Northen Auditorium

Nate will delve into the cutting-edge applications of generative AI and reinforcement learning to video games, revealing how AI is changing the rules and creating entirely new kinds of play.



# Review

1. What are the Picasso project components?
2. What are the steps to add a new unary function into the Picasso language in the current implementation?
  - How much code needs to *change* to add the function?
  - How would you write this code without using reflection?
3. What can you do to help your team succeed?
4. What is our work flow with Git?
5. What is the spiral model of development?

# Review: Process of Adding Cosine Function to the Picasso Language

(in given code)

- Add function name to `functions.conf`
- Create a *token* for the cosine function
  - Same prefix as new function, e.g., `CosToken.java`
- 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`

Name/prefix must match for all but ETN

## Review: 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?

# Review: Git WorkFlow

1. Pull to get the most recent updates to the repository
2. Create a new branch from `main` for your work
  - Commit periodically
  - Write descriptive comments so your team members know what you did and why
3. Push your branch
4. 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
5. Merge pull request into `main` branch
6. In Eclipse, pull `main`
  - Merge into your branch or create a new branch from `main`

# SINGLETON DESIGN PATTERN

# Problem: Too Many Objects!

- Sometimes, we only want one object to *ever* be created for a class
  - Often because there is some state that needs to be coordinated across the application

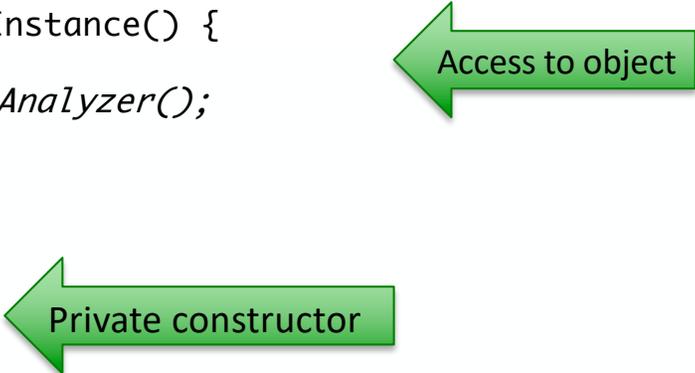
## Solution: Singleton Design Pattern

- Make the constructor private
- Make a public method for accessing the one and only instance

# Solution: Singleton Design Pattern

- Make the constructor private
- Make a public method for accessing the one and only instance (a static variable)

```
public class SemanticAnalyzer implements SemanticAnalyzerInterface {  
    private static SemanticAnalyzer ourInstance;  
  
    public static SemanticAnalyzer getInstance() {  
        if (ourInstance == null) {  
            ourInstance = new SemanticAnalyzer();  
        }  
        return ourInstance;  
    }  
  
    private SemanticAnalyzer() {  
        ...  
    }  
  
    public ExpressionTreeNode generateExpressionTree(Stack<Token> tokens)
```



## When Does Picasso Use the Singleton Design Pattern?

- Specialized analyzers need to refer to ***the*** `SemanticAnalyzer` to parse its parameters/operators

```
return new Floor(  
    SemanticAnalyzer.getInstance().  
    generateExpressionTree(tokens) );
```

- Need to call methods on that one-and-only object

In Picasso:

## Is the Singleton Design Pattern the Best Design?

- Is this the best design? <shrug/>
- Alternative 1: pass in the SemanticAnalyzer as another parameter:

```
public ExpressionTreeNode  
generateExpressionTree(Stack<Token> tokens,  
    SemanticAnalyzer semAnalyzer);
```

- Alternative 2: make SemanticAnalyzer's methods be static
  - Requires making state static too

# CODE SMELL CASE STUDY

# Code Smell: Using `instanceof`

```
public void drawShape( Shape shape ) {  
    if ( shape instanceof Square ) {  
        drawSquare(shape);  
    }  
    else if( shape instanceof Circle ) {  
        drawCircle(shape);  
    }  
}
```

- Why is using `instanceof` a code smell?
  - Always consider: how is this code likely to change?
- How could we write this in a better way?

## Code Smell: Using `instanceof`

- Previous example: had to know all of the Shape classes
  - Update whenever a Shape is added or removed
- Better code: ***Polymorphic!***
  - There was a draw method specific to each Shape
  - Refactor those methods into Shape child classes

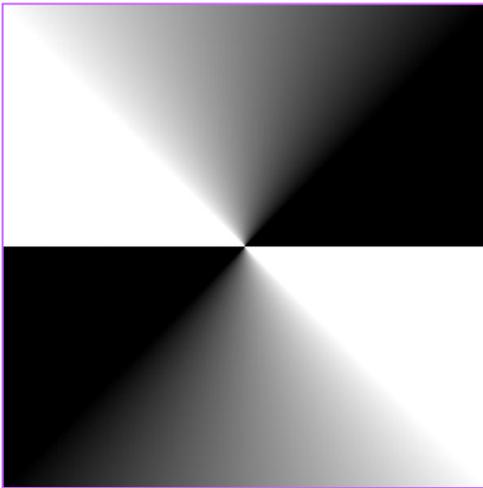
```
public void drawShape( Shape shape ) {  
    shape.draw();  
}
```

# TESTING PICASSO

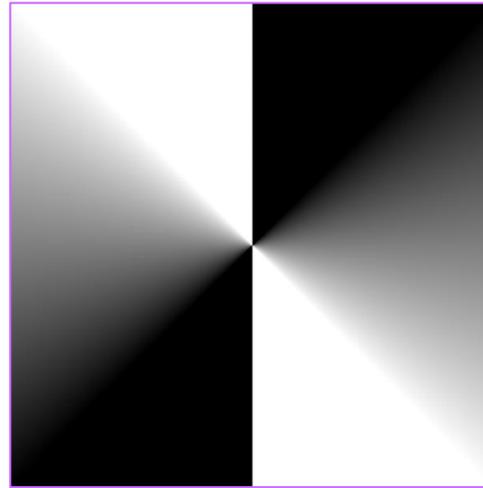
$x/y$  is not the same as  $y/x$

How do you know what should be displayed?

$x/y$



$y/x$



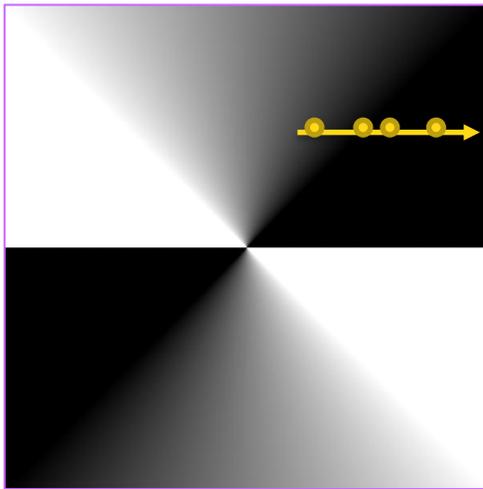
A common implementation mistake is the user enters  $x/y$ , but Picasso displays  $y/x$ . Error may also be in  $x+y$ , but operation (addition) is commutative.

(placement of points is not exact in illustration)

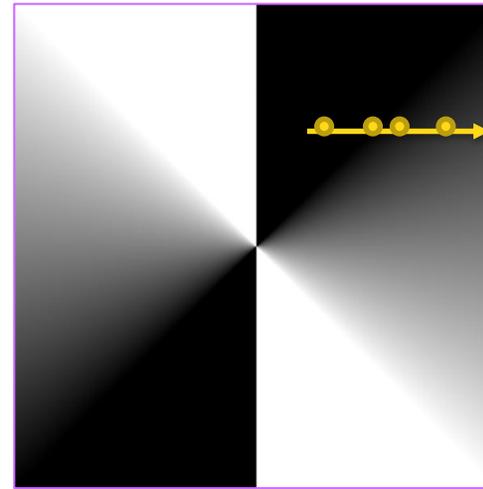
# $x/y$ is not the same as $y/x$

Consider points, holding  $y$  steady at  $-.5$

$x/y$



$y/x$



Y	X	.3	.45	.55	.7
Y = -.5					
Color:					

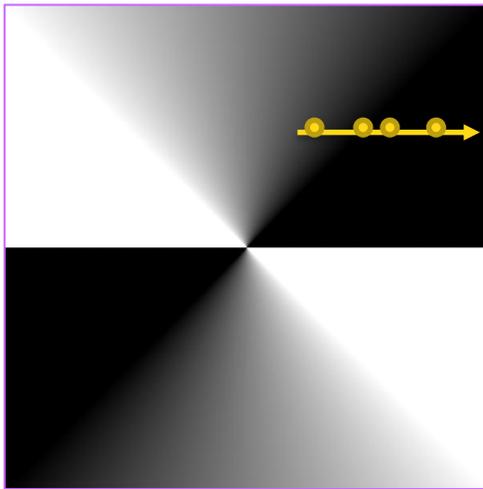
Y	X	.3	.45	.55	.7
Y = -.5					
Color:					

(placement of points is not exact in illustration)

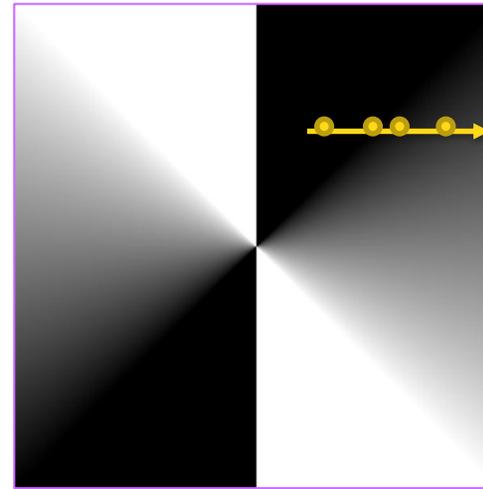
# $x/y$ is not the same as $y/x$

Consider points, holding  $y$  steady at  $-.5$

$x/y$



$y/x$



Good tests  
for the  
Evaluator  
test class

Y	X	.3	.45	.55	.7
Y = -.5		-6	-9	-1.1	-1.4
Color:		Mid-gray	Dark gray	Black	Black

Y	X	.3	.45	.55	.7
Y = -.5		-1.67	-1.11	-.91	-.71
Color:		Black	Black	Dark gray	Mid dark gray

# Testing Picasso

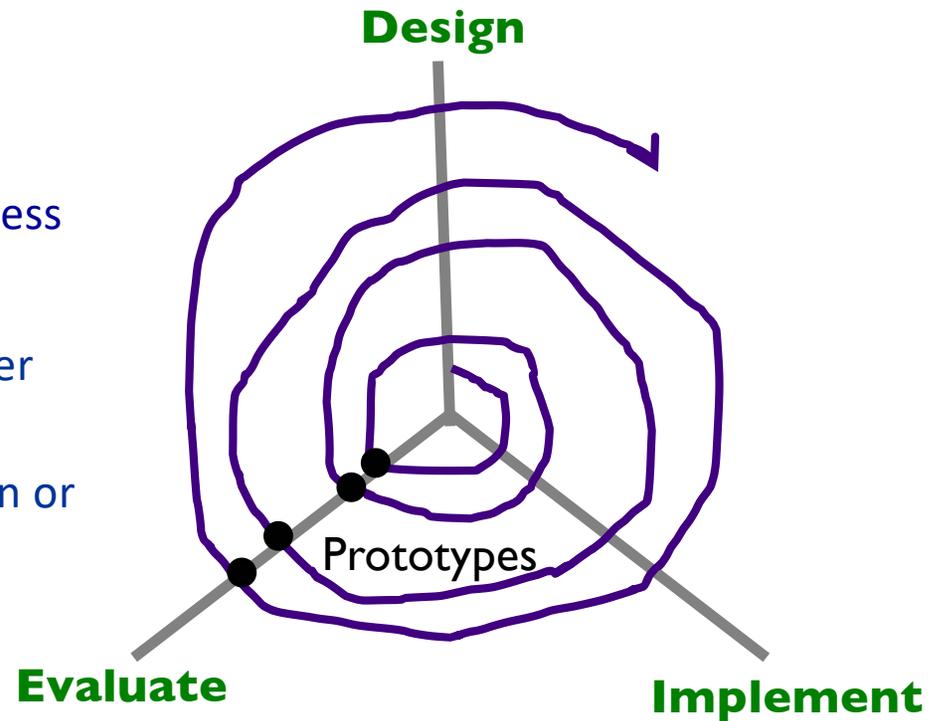
- Automated: JUnit tests
  - Low-cost tests (easy to make, fast to check)
  - Test individual pieces of interpreter
  - Won't catch everything, but catch enough for a low cost
- `ParserTestDriver`
  - Not automated, BUT ...
  - Displays the expression tree (using `toString`) that will be generated from a `String` expression
- GUI/Displayed images
  - <https://cs.wlu.edu/~sprenkles/cs209/projects/picasso/intrinsics/>
  - Visual check – big picture check; low precision

## How good is your testing?

- Use EclEmma, a plugin for Eclipse that comes with the Enterprise Edition we're using 
- What can you cover using unit tests? With other testing?

# Review: Spiral Development Model

- Idea: smaller prototypes to test/fix/throw away
  - Finding problems early costs less
- In general...
  - Break functionality into smaller pieces
  - Implement most depended-on or highest-priority features first



[Boehm 86]

Radial dimension: cost

# What Kind of Prototypes for Deliverables?

- Both for given code and for preliminary implementation
- High fidelity with respect to the GUI
- Vertical prototype/Depth
  - From GUI → Backend → GUI
  - But limited implementation of GUI features and Picasso language

# Preliminary Implementation

- Goals
  - Get your team working together
  - Find kinks in design
    - Rework now instead of later
- Tag your version
- Can keep working after that
  - Return to the tagged version for Friday's demo

# Friday Demos: Preliminary Implementation

- Demo to me (only) in teams in Parmly 404
- Choose one person to demo the code
- Demo content:
  - Show what you have done for the preliminary implementation
  - Discuss design decisions
  - Tell me what you're thinking for extensions
- Order of teams will be randomly generated on Friday
  - Schedule: 8:35, 8:47, 9:00, 9:15
  - Schedule: 11:05, 11:17, 11:30, 11:45

# Looking Ahead

- Friday: Preliminary Deadline and Demos
- Order of teams will be randomly generated on Friday
  - Schedule: 8:35, 8:47, 9:00, 9:15
  - Schedule: 11:05, 11:17, 11:30, 11:45