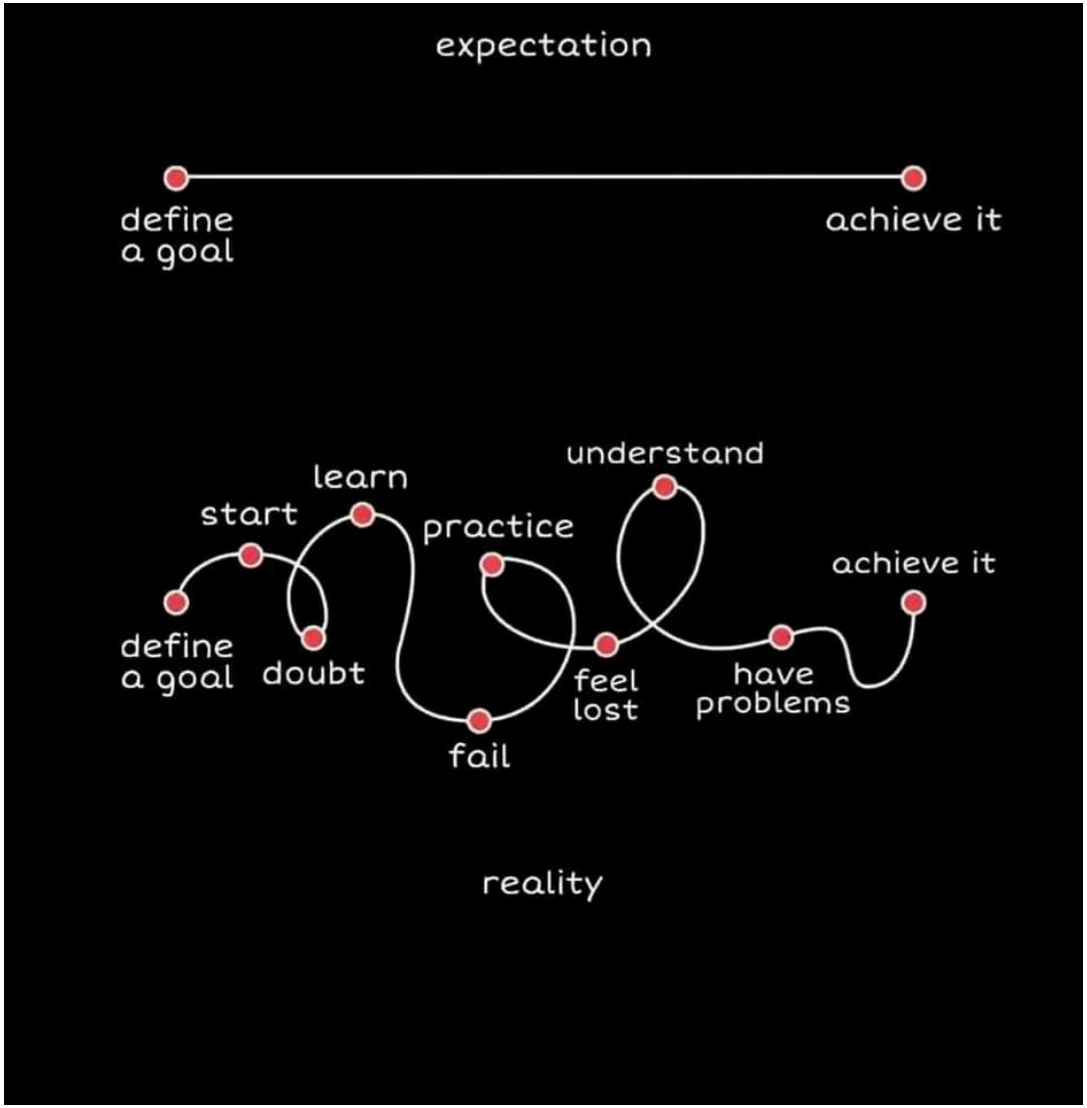
Objectives

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


expectation
define
a goal
achieve it
understand
learn
start
practice
achieve it
define
a goal
doubt
feel
lost
have
problems
fail
reality

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 Picasso function name to `functions.conf`
  - E.g., `cos`
- 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

Merge Conflict

- Occurs when competing changes to the same lines in a file
  - Git doesn't know how to resolve the merge
- Resolving: manually edit the conflicted file to what you want to keep in the merge
  - Stage change, commit and explain your fix
  - Push branch

I expect many of you will see these this week

SINGLETON DESIGN PATTERN

Design Pattern

General reusable solution to a commonly occurring problem in software design

- Not a finished design that can be transformed directly into code
- Description or *template* for how to solve a problem that can be used in many different situations
  - "Experience reuse", rather than code reuse

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)
```

Access to object

Private constructor

When Does Picasso Use the Singleton Design Pattern?

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

```
ExpressionTreeNode paramETN =
        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);
```

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

None of these changes are required; just explaining alternatives

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

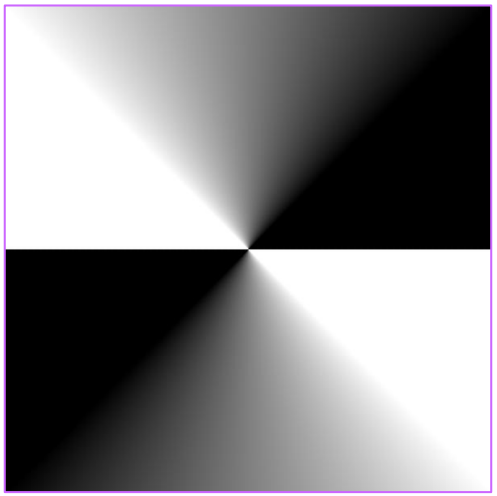
```
public void drawShape( Shape shape ) {
    shape.draw();
}
```

TESTING PICASSO

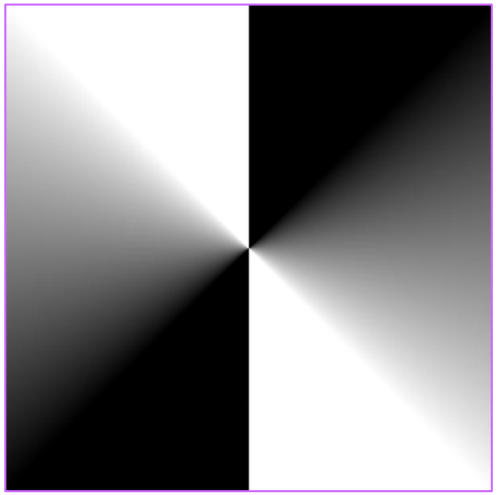
x/y is not the same as y/x

How do you know what should be displayed?

x/y



y/x



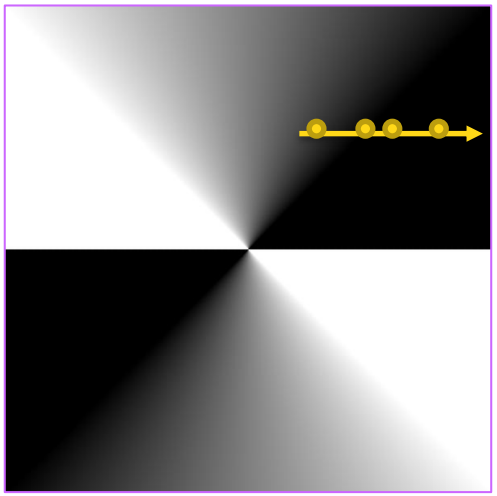
A common implementation mistake is the user enters x/y, but Picasso displays y/x.
Error may also be in x+y, but operation (addition) is commutative.

(placement of points is not exact in illustration)

x/y is not the same as y/x

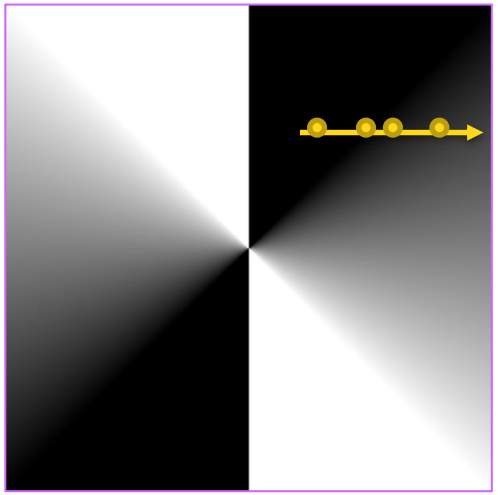
Consider points, holding y steady at -.5

x/y



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

y/x



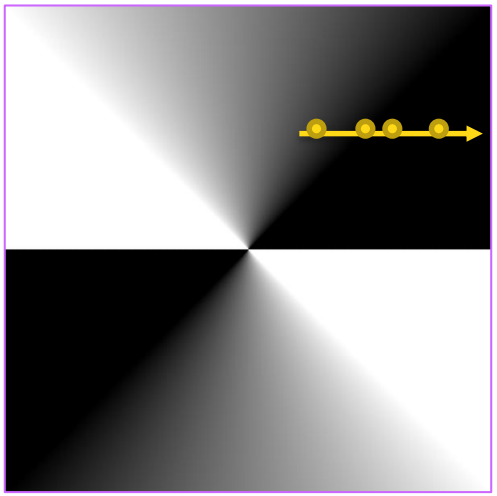
Y	X	.3	.45	.55	.7
Y = -.5					
Color:					

(placement of points is not exact in illustration)

x/y is not the same as y/x

Consider points, holding y steady at -.5

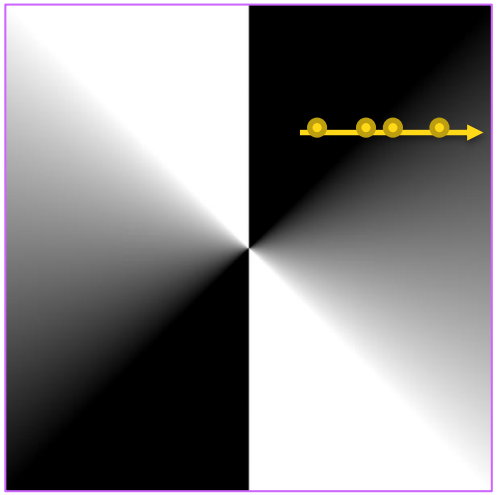
x/y



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

Good tests for the Evaluator test class

y/x



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

Testing Picasso

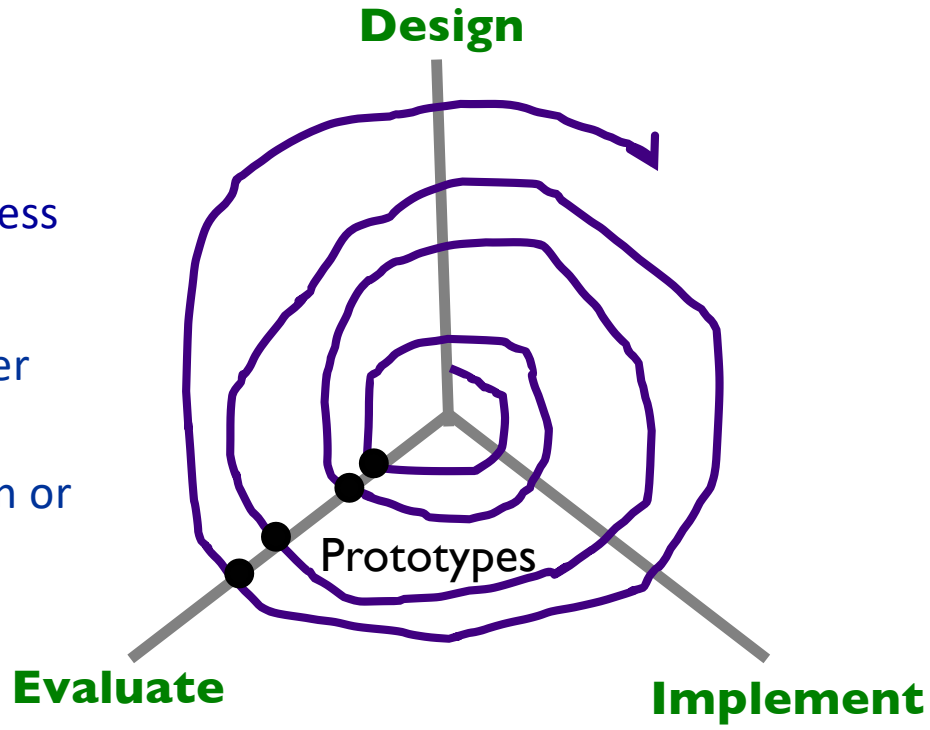
- Automated: JUnit tests
  - Low-cost tests (easy to make, fast to check)
  - Test individual pieces of interpreter
  - Won't catch everything, but catch enough for a low cost
- `ParserTestDriver`
  - Not automated, BUT …
  - Displays the expression tree (using `toString`) that will be generated from 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

Picasso: Your Team's Javadocs

- Automatically generated from `main` branch at 3:58 a.m. every day
- Linked from Documentation section of Picasso project page

Reload the page to see changes/updates

FAQ for Picasso

- Linked from the specification page
- Updated as I get new questions

Reload the page to see changes/updates

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
- 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: 9:15, 9:27, 9:40, 9:55
  - Schedule: 11:35, 11:47, 12:00, 12:15