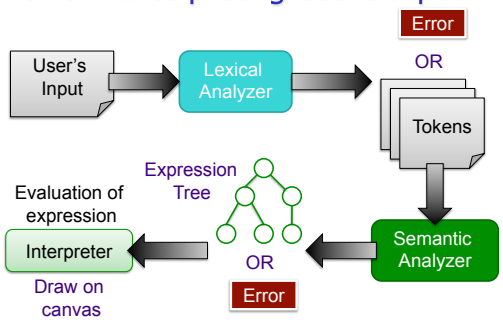
Objectives

- Picasso Design
  - Finish parsing commands
- Collaborating with Subversion
- Discussion of Preparation Analyses

Review

- How is a programming language processed?
  - What are the different phases?
- Start up Eclipse

Review: Interpreting User's Input



Practice Adding Functions

1. Create a **token** for the sine function
   - Same prefix as new function, e.g., `SinToken.java`
   - `sin` needs to be added to `functions.conf`
2. Create a **semantic analyzer** for the function with same prefix as function, e.g., `SinAnalyzer.java`
   - Implements `SemanticAnalyzerInterface`, `generateExpressionTree` returns an instance of `ExpressionTreeNode`
3. Create an `ExpressionTreeNode` for function `Sine.java`
   - How should the "evaluate" method be implemented?

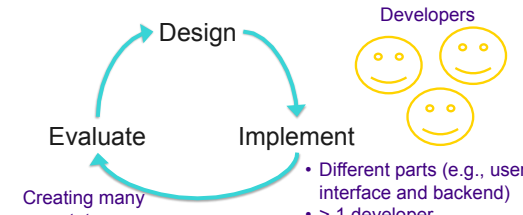
Expanding Expressions

- How would you handle `perlinColor (expr,expr)` ?
- How would you handle addition?

Goal: easily updated, extendible code

VERSION CONTROL

Problems in Collaborating on Code



Creating many prototypes
- What if don't like recent prototype? Need to go back to older version

- Different parts (e.g., user interface and backend)
- > 1 developer implementing concurrently
  - *What if one introduces a bug?*

Version Control Features

- Backup and Restore
  - Files are saved as they are edited
  - Revert to a specific version/checkpoint
- Synchronization
  - Lets people share files
  - Stay up-to-date with the latest version
- Track changes to code
  - Save comments explaining why change happened
  - Stored in the VCS, not the file
  - Track how, why a file evolves over time
- Track Ownership
  - Tags every change with the name of the person who made it

Version Control Features

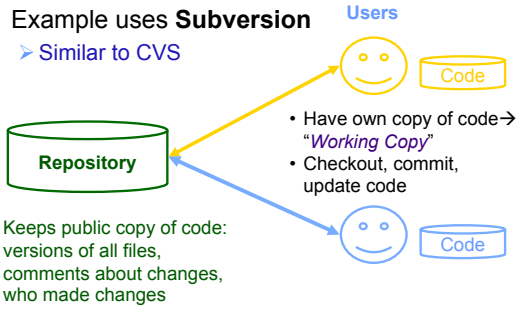
- Short-term undo
  - Messed up a file? Go back to the last good version
- Long-term undo
  - Created a bug a year ago? Jump back to see change you made.
- Sandboxing
  - Making a big change? Make temporary changes in isolated area, test, work out kinks before "checking in" your changes
- Branching and merging
  - Branch a copy of your code into a separate area, modify it in isolation (tracking changes separately)
  - Later, merge work into common area

CVS and Subversion

- Popular Version Control Systems
- Subversion is newer, more flexible
- Terms used are common for most version control systems

Using Version Control

- Example uses **Subversion**
  - Similar to CVS

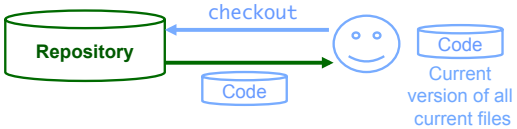


- Have own copy of code→ "*Working Copy*"
- Checkout, commit, update code

- Keeps public copy of code: versions of all files, comments about changes, who made changes

Using Version Control: `checkout`

- To start, need to **`checkout`** your working copy of the code



Using Version Control: commit

- After you make changes that you *want others to see*, **commit** your version
  - Include comments about what changes you made and why

- Checks for conflicts
- Updates each modified file
- Records comments with updated files

Using Version Control: commit

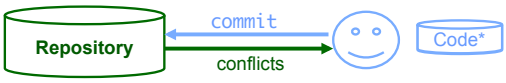
- After you make changes that you *want others to see*, **commit** your version
  - Include comments about what changes you made and why

- Checks for conflicts
- Updates each modified file
- Records comments with updated files

Other people's code doesn't change

Using Version Control: commit

- After you make changes that you *want others to see*, **commit** your version



- Checks for conflicts -- code conflicts with recent changes in the public copy

- Update code, fix conflicts
- Try commit again

Using Version Control: update

- To see the *current* version of the code, **update** your repository
  - Resolve conflicts

Using Version Control: add, delete

- You need to **add** and **delete** files and directories to the repository, then **commit**

- Create new records for added files
- Close records for deleted files
  - Files not deleted from repository

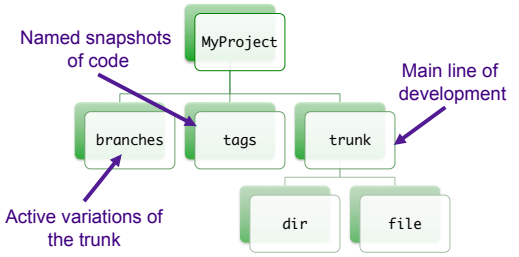
- Add, delete files and directories
- Commit

Version Control Advice

- Does not eliminate need for communication
  - Process becomes much more difficult if developers do not communicate
- Before picking up again, **update** your working copy
- **Commit** only after you've tested code and you're fairly sure it works
  - Write descriptive comments so your team members know what you did and why

Code Organization

- Organize code into appropriate structure



SUBCLIPSE

Subclipse

- Plugin for Eclipse
- Installation:
  - Help → Install New Software
  - Create remote site:
    - Name: Subclipse
    - `http://subclipse.tigris.org/update_1.6.x`
      - Select all those packages

Checking Out Code in Eclipse

- Create a new SVN Repository:
  - File → New → Other → SVN
  - Repository:

    `file:///home/courses/cs209/shared/svn/Picasso/trunk`
  - If you want to connect from your home computer:

    `svn+ssh://knuth.cs.wlu.edu/home/courses/cs209/shared/svn/Picasso/trunk`
- Checkout from repository
  - As a new project (Wizard)
  - Java project, named Picasso

Checking Out Code in Eclipse

- If many compiler errors in `src/tests`, may need to add JUnit to classpath
  - Configure Build Path
  - Libraries, Add Library
    - Junit 4

Practice with Subclipse

- Named: your name
- Put some text into it
- **Add** the file to the Repository:
  - Right-click on the file you created → Team → Add
- **Commit** your file *(Save for group to see)*
  - Right-click on top-level directory/project → Team → Commit
  - Add an appropriate comment
- **Update** your repository *(Get latest working version)*
  - Right-click on top-level directory/project → Team →Update
  - Do you have any one else's files?

PLANNING THE PROJECT

What Steps Need To Be Completed?

What Steps Need to be Completed?

- GUI
  - Command interface
  - More options/buttons (optional)
  - Listeners
  - Improve GUI
- **TESTING!**
- Parsing Picasso language
  - Handle functions, arithmetic operators
  - Handling **image** functions
  - Handling assignment statements
  - **Handle errors** appropriately
- Evaluating expressions

Dependencies

Dependencies

- Parsing classes (tokens, analyzer, expression) are very dependent on each other
- Need to hook GUI to Parser
- Can test without other pieces but easier and more satisfying to see results displayed

Effect of Extensions

- Extensions could affect your code design
  - Where could change --> abstraction
- Decision?
  - May change your minds after start working on the code
  - Top vote getters

Plan

- Tasks/Steps
  - Testing
  - Think about iterative development
    - Monday deadline: basic functionality of all parts
- Division of tasks
  - # of people per part
- Deadlines

Goals

- Implement a version of application
  - Iterative development
- Don't go too far in depth, more breadth
  - See design issues sooner
    - "We need method/functionality X in class Y"

Secondary Goals

- You're going to figure out that your design isn't perfect--maybe not even good!
  - Could be partially fault of given code
  - Fix smaller things
    - Refactoring!
  - Note larger things (analysis/post-mortem due at end of finals week)

Good judgment comes from experience.
How do you get experience?
Bad judgment works every time.

Picasso Timeline

- Monday, Dec 7: demo application (group)

What should be the goal for Monday?

- ??: final implementation due (group)
- Fri, Dec 18: "Post-mortem" (individual)

Friday: Guest Speaker Emily Hill

- On Sakai forum for class, answer the following questions about the talk (also on the forum):
  - What is the main problem she is trying to solve?
  - What is her approach(es) to the problem?
  - How did she evaluate her approach(es)?
  - Are there any limitations to her approach? (i.e., will her approach always work?)
  - Do you have any ideas for future work, e.g., how to extend her approach or what other questions you'd like to know the answers to?
  - What did you find most interesting about the work?
  - How did the talk/research relate to the class, if at all?
- Due on Monday