

Objectives

- Version Control
- Parsing commands

Nov 18, 2011

Sprenkle - CSCI209

1

Review

- Who are your teammates for the final project?
- What is the final project?
- How does an interpreter execute a programming language?

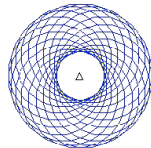
Nov 18, 2011

Sprenkle - CSCI209

2

SLogo Project Overview

- Goal: Create an IDE for simplified version of Logo
- Logo: programming language designed to teach children to program
 - Low floor, high ceiling

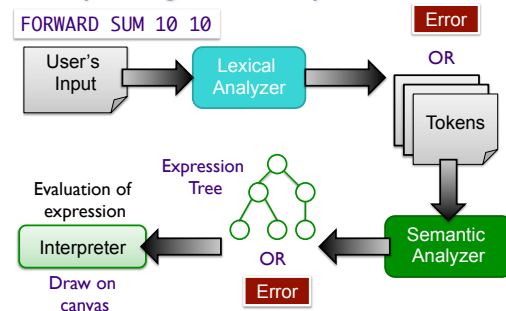


Nov 18, 2011

Sprenkle - CSCI209

3

Interpreting User's Input



Nov 18, 2011

Sprenkle - CSCI209

4

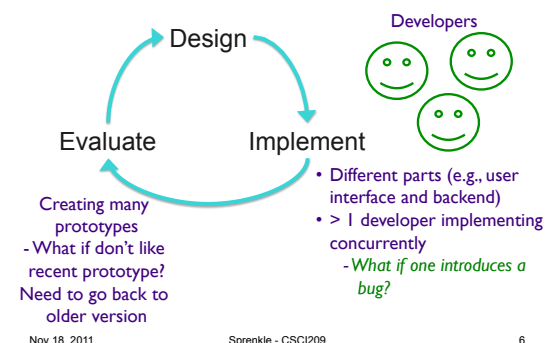
VERSION CONTROL

Nov 18, 2011

Sprenkle - CSCI209

5

Problems in Collaborating on Code



Nov 18, 2011

Sprenkle - CSCI209

6

Version Control Features

- Synchronization
 - Lets people share files
 - Stay up-to-date with the latest version
- Backup and Restore
 - Files are saved as they are edited
 - Revert to a specific version/checkpoint
- 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

Nov 18, 2011

Sprenkle - CSCI209

7

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

Nov 18, 2011

Sprenkle - CSCI209

8

Version Control Systems

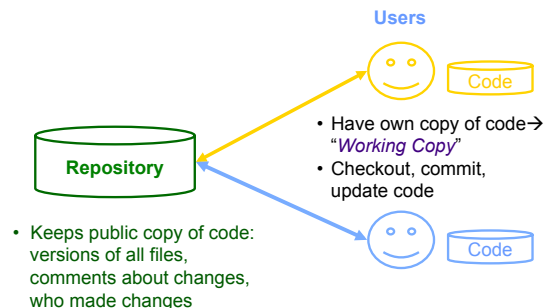
- Popular Version Control Systems
 - CVS, Subversion, Git, ...
- Terms used are common for most version control systems
- We will use Subversion with Subclipse
 - Mark Phippard, a W&L grad works on both
 - Director of the Subversion engineering team at CollabNet, the company that founded Subversion

Nov 18, 2011

Sprenkle - CSCI209

9

Using Version Control



Nov 18, 2011

Sprenkle - CSCI209

10

Using Version Control: checkout

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



Nov 18, 2011

Sprenkle - CSCI209

11

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

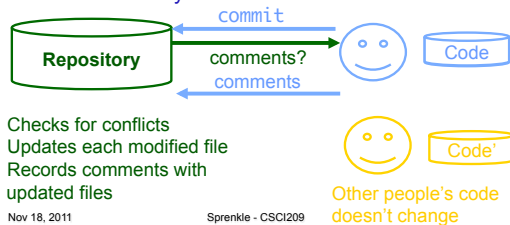
Nov 18, 2011

Sprenkle - CSCI209

12

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



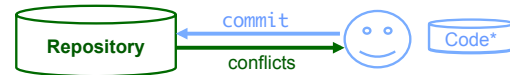
- Checks for conflicts
- Updates each modified file
- Records comments with updated files

Nov 18, 2011

Sprenkle - CSCI209

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

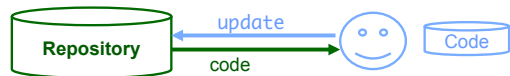
Nov 18, 2011

Sprenkle - CSCI209

14

Using Version Control: **update**

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



Nov 18, 2011

Sprenkle - CSCI209

15

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

Nov 18, 2011

Sprenkle - CSCI209

16

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

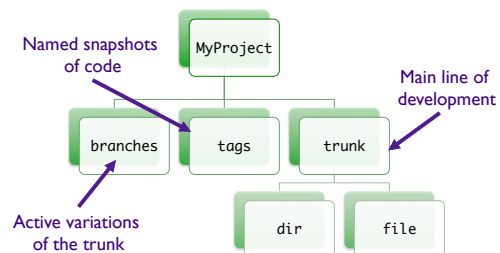
Nov 18, 2011

Sprenkle - CSCI209

17

Code Organization

- Organize code into appropriate structure



Nov 18, 2011

Sprenkle - CSCI209

18

SUBCLIPSE

Nov 18, 2011

Sprenkle - CSCI209

19

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

Nov 18, 2011

Sprenkle - CSCI209

20

Checking Out Code in Eclipse

- In SVN Repository view, create a new SVN repository:
 - File → New → Other → SVN
 - Repository:
 - file:///home/courses/cs209/shared/svn_repo/
 - If you want to connect from your home computer:
 - svn+ssh://knuth.cs.wlu.edu/home/courses/cs209/shared/svn_repo/
- Checkout **SLogo<gradyear>/trunk** from repository
 - As a new Java project (Wizard)
 - Java project, named **SLogo**

No angle brackets

Nov 18, 2011

Sprenkle - CSCI209

21

Checking Out Code in Eclipse

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

Nov 18, 2011

Sprenkle - CSCI209

22

Practice with Subclipse

- Create file named with 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?

Nov 18, 2011

Sprenkle - CSCI209

23

UNDERSTANDING A CODE BASE

Nov 18, 2011

Sprenkle - CSCI209

24

Code Review

- What questions do you have about the code?
- What do you want to find out?

Nov 18, 2011

Sprenkle - CSCI209

25

Understanding the Code

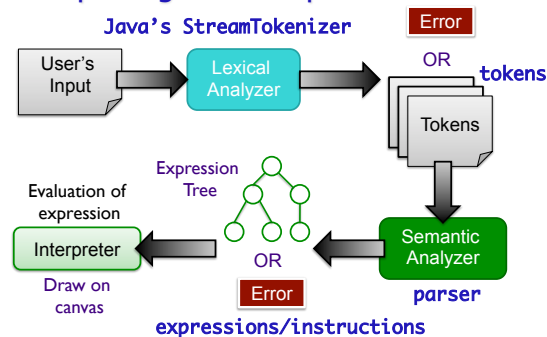
- How does the given code map to lexical analysis, semantic analysis, and evaluation components?

Nov 18, 2011

Sprenkle - CSCI209

26

Interpreting User's Input



Nov 18, 2011

Sprenkle - CSCI209

27

Understanding the Code: Lexical Analysis

- Important classes
 - `slogo.jelan.parser.ElanInterpreter`
 - `slogo.jelan.parser.tokens.TokenFactory`
- Output: `slogo.jelan.parser.tokens.*`

PrintToken
EqualToken
AssignmentToken

Nov 18, 2011

Sprenkle - CSCI209

28

Understanding the Code: Semantic Analysis

- Important Classes
 - Common interface: `slogo.jelan.parser.Parser`
 - `slogo.jelan.parser.*Parser`
 - Ex: `slogo.jelan.parser.ExpressionParser`
 - `slogo.jelan.parser.InstructionParser`
 - Decides which instruction parser to call
- Output: `slogo.jelan.expressions.*` or `slogo.jelan.instructions.*`

PrintParser, AssignmentParser

Nov 18, 2011

Sprenkle - CSCI209

29

Understanding the Code: Evaluation

- Important Classes
 - Base class: `slogo.jelan.GrammarElement`
 - Subclasses:
 - `slogo.jelan.instructions.Instruction`
 - `slogo.jelan.expression.Expression`
- Output: Object

Print
Assignment

Nov 18, 2011

Sprenkle - CSCI209

30

Bringing it together

- `slogo.jelan.*`
 - Breaks classes into appropriate packages: Tokens, Expressions, Instructions, Parsers
- `slogo.jelan.parser`
 - Parse Tokens to create Instructions
- `slogo.jelan.instructions`
 - Represent instructions
 - `evaluate` method

Nov 18, 2011

Sprenkle - CSCI209

31

Bringing it together

- Mapping between Token, Instruction, Parser
 - Knows which Parser to call based on `instructions.prop` and mapping from Token to Parser
- Run `ElanInterpreter` with tests/`assign_repeat`

Nov 18, 2011

Sprenkle - CSCI209

32

Practice Adding Instructions

1. Create a token for instruction
 - Likely a subclass of `token.ReservedToken`
 - Same prefix as new instruction, e.g., `IfToken.java`
2. Create a parser for the instruction with same prefix as instruction, e.g., `IfParser.java`
 - Parsing class (presumably implementing `Parser`) returns an instance of parsed `Instruction`
3. Create an instruction with prefix name, e.g., `If.java`
4. Add instruction name to file `instructions.prop`, e.g., add a single line to file containing string `If`

Nov 18, 2011

Sprenkle - CSCI209

33

Brainstorming

- What do you need to do to complete the project?
- What do you “see” for the final project?
- What's going to *change*?
- Where do you think you'll run into problems?
- To focus your thinking, consider this use case: "The user starts the program, types 'fd 50' in the command window, and sees the turtle move in the display window leaving a trail."
 - What are other use cases?

Nov 18, 2011

Sprenkle - CSCI209

34

Preparation Analysis

Due Monday
after Thanksgiving

- What are the main parts/steps that need to be completed to complete the project?
 - How much work does each part require?
 - Approximate in terms of time or relative to the other steps.
 - How many people should work on each part?
- How will your program handle the following use case: "The user starts the program, types 'fd 50' in the command window, and sees the turtle move in the display window leaving a trail."?
 - From your description, it should be clear which classes/objects are responsible for completing each part of the task.
- What 3 extensions would you like to have in the final application?
- A plan for how you would tackle implementing the project
 - What parts can be completed independently of the other parts?
 - What parts need to be completed before other parts?
- The parts of the project you're most interested in working on, in ranked order.
- Any questions about the given specification.

Nov 18, 2011

Sprenkle - CSCI209

35

TODO

- Project Analysis due Monday after Thanksgiving

Nov 18, 2011

Sprenkle - CSCI209

36