API\_CHANGES Group 4

Chinmay Patwardhan, Ken McAndrews, Lalita Maraj, Susan Zhang

Model:

* Added saveLibrary and readLibrary to save and load variables and commands
* Added ability to change the interpreter language
* Added the ability to undo and redo turtle commands
* Grant access to a list of active turtle ID’s
  + Have the ability to have multiple turtle ID’s
* Provide access to background color, pen color, pen size, and available shapes and colors
* Allow the view to report when a key has been pressed, the mouse has been clicked, or the mouse has been moved and operates accordingly
* Internally stores access to
  + Interpreter
  + Command cache
  + Variable cache
  + Map of turtle IDs to turtles
  + Command history
  + Instruction factory
* Removed the idea of an instruction queue, instead process parsed instructions right away
* Added path and stamp classes to keep track of turtle paths and stamps

Parser:

* Changed interpreter class to make use of a Parser that parsed in separate words, expressions, and lists
* Words are single string tokens (no white space)
* Expressions are an evaluable sequence of instructions and parameters, for example “sum sum 3 4 5” is an expression that evaluates to 12. But “sum 3 4 sum 3 4” are two separate expressions
* Lists are the content of outermost square brackets, for example “[ 3 4 5 ]” is a list, but “[ [ 3 4 ] [ 4 5 ] ]” is a single list with two interior lists.

Turtle:

* Turtle now keeps track of its own color, shape, pen size, and ID

Instruction Factory

* Keeps track of current built in functions using the given language
* Utilizes a factory design to return the appropriate Instruction object given a string using Java Reflection (forName)
* If a built in instruction is not found, it checks the command cache
* Uses a properties file to keep track of the current language. Uses key value pairs to map command names to Instruction class paths

InstructionLoop class

* Abstract class that extends Instruction and defines a variable name, a starting value, an ending value, and an increment
* It is extended by the three loop types, which change the various parameters of the loop depending on the loop type

InstructionConditional class

* Abstract class that extends Instruction and defines an EPSILON to be used in comparisons for conditional Instructions

ComplexParameterInstruction class

* An interface implemented by commands that read in any parameters that are not simple expressions (TO, FOR, IF, etc.)

InstructionListNode class

* Class that stores a list of Instructions that evaluates each expression in the list and returns the evaluation of the last Instruction

InstructionMultiParameter class

* Stores an instruction that takes in a variable number of parameters as defined by the user with parentheses. For example, “( sum 3 4 5)” is a sum that adds together 3, 4, and 5 to return 12.

The Model and View followed the guidelines laid out in the original API on how they would communicate. To illustrate, the View sends information to the Model by calling public methods and queries the Model for updated data by calling get methods.

**Example Code**

**Changes to Original  ‘Executing User Input from textbox’:**

1. Text is entered into textbox by user
2. When run button is pressed, text from the textbox is sent to Model with ViewController.executeCommand() which calls Model.parseInput(String userInput)
3. All the Updatable components (Canvas, modules, etc.) are updated with myView.notifyUpdatables(String error)

**Changes to Original ‘Primary View Classes/Packages’:**

The view will be in charge of sending user input to the Model to be parsed and retrieving the updated turtle data, currently available commands and variables, and a log of the user input. The View uses the updated turtle data to display the turtle actions.

-  Canvas: will use the updated turtle data such as angle, x/y coordinates and pen status to appropriately display the turtle’s actions.

-  InputPanel: will receive user input via the keyboard or clickable commands from the command log and/or available commands/variables from the sidebar.

- ViewController: is the liaison between the View and Model. The View uses the ViewController to send information to the Model. Additionally, the ViewController is responsible for switching workspaces.

**A note about switching and creating workspaces**

We map a workspace to an instance of a Model. When a new workspace is created a new instance of a Model is instantiated. Therefore, there is no need to re-create the View Class. Instead, we swap the current Model that each View component uses to query information from the Model with the current workspace’s model.