SLogo Design - Turtle
http://www.cs.duke.edu/courses/compsci308/spring15/assign/03_slogo/index.php
http://www.cs.duke.edu/courses/compsci308/spring15/assign/03_slogo/part1.php
http://www.cs.duke.edu/courses/compsci308/spring15/assign/03_slogo/part2.php

Turtle:
Instance variables:
- Point2D - old and new state
- Angle - old and new double
- Double distanceTraveled
- Boolean - Hiding or not
- Int totalDistanceTraveled
- Inner Class:-
  - Pen
    - State (Up Or Down) - Boolean

Methods:
- getters OR immutable map
- move(double distance, double angle)
  - -updates current point 2D, old point 2D, adds to total distance
- rotate(double angle)
  - -add the angle give to the current angle of the turtle

Controller:
Instance Variables:
- Model
- View

Methods:
- clearScreen()
- execute(String code) - gets string from
- changeLanguage(String language)

Model:
Instances:
- List of turtles

Constructor:
- Model(View)

Public Methods:
- changeLanguage(String language)
- processCommand(String code)

Internal API:

EdgeChecker
- Checks for whether it is toroidal/infinite and gives the updated result to the model.

View

- -ArrayList<ViewTurtle> turtles
- **Constructor(myController, Stage)**
- drawTurtle(Point2D new, Point2D old, ID)
- moveTurtle(Point2D new, Point2D old, ID)
- rotate(double angle, double Angle)
- showTurtle()
- clearScreen()
- printToConsole(String message)
- addTurtle(ID, Point2D)

View Private API-
Contains XML parser to read button layout.
Contains seperate classes for each panel
-Grid pane for the texts
-Grid pane which contains the text box and enter command
-a root of a blank Window in the middle which contain nothing. This is where the simulation is ran from ? Could use window perhaps?
- Grid pane to have drop-down menu for the change in language
-Grid pane to organize all the stuff.
- create error pop up box

add Listeners to turtle to set default turtle

LineDrawer Method
clearTerminal Method
Remove All ViewTurtles
initializeViewTurtle
sendCommand internal
printConsole Method

ViewTurtle

- -Color Turtle
- -Shape Turtle
- -Color Pen
- -Turtle ID
- --Method:-
  - HidingShowing
  - Move
  - Listeners :)
    - change each parameter except id
  - setID

rotate

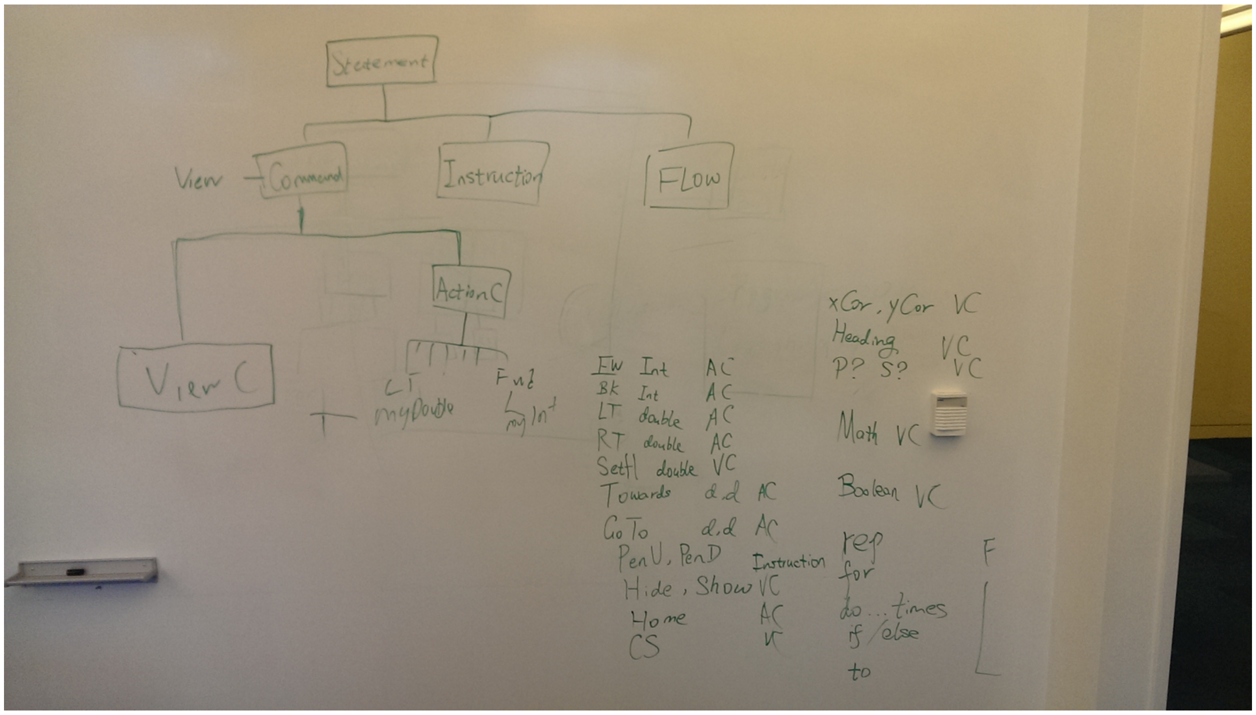
Do we do color change in code or in hui??

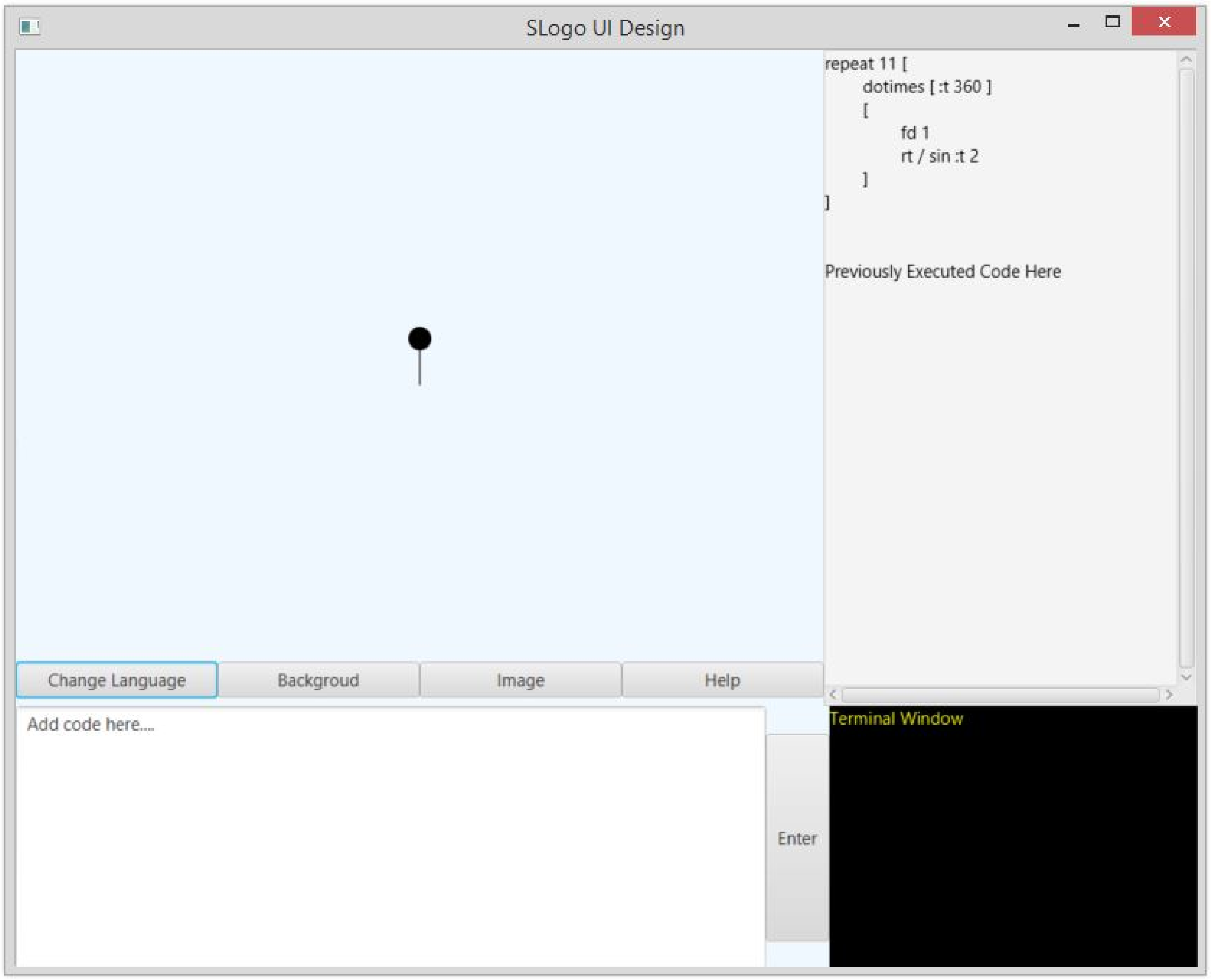
Design Document
-Introduction - Sierra/Yancheng
-UI - Siva
-Design - Together
-Overview - together
-View API - External - Jangsoon
             - Internal - Siva
-Model API -External - Sierra
               -Internal - Yancheng
Design Considerations - Together
Team Responsibilities
    Jangsoon/Siva - View
    Sierra/Yancheng - Model
    Unknown - Controller

Exceptions
-no if statements

View Exception - Try and catch - Image not found, wrong color

Model Exception - Dummy Exception class, print new messages because of error that might occur


Statement
View
Command
Instruction
FLOW
ActionC
View C
LT
myDouble
Fwd
myInt
Fw Int AC
Bk Int AC
LT double AC
RT double AC
SetH double VC
Towards d,d AC
GoTo d,d AC
PenU, PenD Instruction
Hide, Show VC
Home AC
CS V
xCor, yCor VC
Heading VC
P? S? VC
Math VC
Boolean VC
rep
for
do...times
if/else
to
F



```
repeat 11 [
     dotimes [ :t 360 ]
     [
          fd 1
          rt / sin :t 2
     ]
]
```

Previously Executed Code Here

User Interface

The User interface is represented by the figure above. The user interface has several components which are described below

* Turtle Playground

This is where the turtle acts upon the code given. This is represented by the aliceBlue colored location in the figure above.

* Previously Used Commands

The scroll pane on the right side of the scene represents a list of previously used commands and also suggested commands. This allows the user to click on it and reload the previously used commands into the codeField.

* Terminal

The black window on the lower right side represents the output window. This window displays the results from the user’s print statement or displays error messages which were caught in the model.

* Button Pane
    * Change Language allows the user to switch between the avaliable language to code in. At final product, this will be represented by a scroll down instead of a button.
    * Background allows the user to customise the background based on an image. When clicked, it will open a file chooser for the user to pick their color.
    * Image allows the user to change either the color or the image of the turtle. When clicked it will open a popup box asking for the new color or to open a file chooser to select the new shape.
    * Help allows the user to open up a HTML webpage where there will be a [help guide](http://www.cs.duke.edu/courses/compsci308/spring15/assign/03_slogo/commands.php) for the user to see before being able to code in the program.
    * Enter allows the user to submit their code. When enter is clicked, the code inside the codeBox is sent to the controller as a string to be parsed in the model.

* CodeBox

This is represented by the white text area in the lower left of the scene. This is where the user can input their code.

External View API

View class' external APIs are the public methods called by the Model and call the internal APIs where detailed functionalities are defined. The methods and their descriptions are as follows:

public void drawTurtle(Point2D new, String ID)
Moves the location of given turtles from the original location (obtained from the ViewTurtle instance with corresponding ID) and calls lineDrawer() to draw the line (color of which is obtained from the ViewTurtle instance) between the original point and the new point.

public void clearScreen( )
Removes all the turtles by calling removeViewTurtle() for every ViewTurtle in the ArrayList and clears the terminal by calling clearTerminal(), and initializes

Flexibility
- add more commands (factory)
- adding more turtles
- adding buttons in the view
- changing language
- changing area of turtle playground
- able parse through complex commands such as "lt (fd (product (Sum (Random 3) (Random 50)) (Random 10))))"

Architecture
- open: commands open to extension, user interaction with turtles (#turtles, functionality), adding language preferences, appearance of view
- closed: parsing, the statement hierarchy, structure, internal management of turtles

Introduction
========
In this project we are creating a simplified SLogo parser and visualization program. This program is often used to teach young children basic programming skills. In addition, through writing this program, we hope to enhance our design capabilities through the good use of hierarchies, factories, model-view-controller, and other design patterns.

We designed our program with the intention of having it be very flexible in the following areas: adding more commands (via use of a factory), adding turtles to the "turtle playground," adding buttons and functionality to the view, changing the language of the program, changing the dimensions of the turtle playground, and being able to parse complicated commands/user programs.

Regarding architecture, the parts of our program that are closed include the parser, the "statement" (command) hierarchy (described below), the structure of the program, and internal management of the turtles. Specifically, the parser is generic in that it functions the same way no matter what commands are available to the user or how they are given to the parser. The hierarchy we will create for the commands available to the user is also closed, as the parent classes and functionality of the current classes won't change no matter what commands are added. The model-view-controller and control flow within the program will also remain the same for any extensions made, as well as how the internal API updates or changes the turtles.

The parts of our program that remain open are the user interaction with turtles, such as number of turtles on the screen and their functionality, as well as possible language preferences and the appearance of the view. Our program would be able to support more turtles, as well as more functionality of each turtle. The command hierarchy is also open to extension.

Overview
=======
<img
src="https://raw.githubusercontent.com/duke-compsci308-spring2015/slogo_team02/master/DESIGN/MVC.png?token=AIax0AN6tXyNIGaHa7ndMzPhFw_Xm3Pjks5U6nkEwA%3D%3D">
<img
src="https://raw.githubusercontent.com/duke-compsci308-spring2015/slogo_team02/master/DESIGN/hierachyCommands.JPG?token=AIax0FZW9NL8pnUaaTPDJeyWckE3Ixkaks5U6m9hwA%3D%3D">

Our project is divided into two major sections, view and model. The view is the front-end of the IDE that deals with the visualization of the scene and the user interface; the model is the back-end the processes the user commands and manages the states and behaviors of the turtles. The two sections are separated from each other and server their goals as public APIs. As we designed the APIs in a way that each API is independent of use of the calling party, both APIs are compatible with platforms other than the ones specified in the project. The model class acts on the turtles (or the designated objects) in response to the user commands passed from the view class and updates the current states of the turtles. By calling the public methods of the view class, the model class is then able to display the state of each turtle on the scene. To facilitate the communication between the model class and the view class, we have a controller class designed to manage the flow of the whole command interpreting process and passes data from the view to the model class.

The model class serves as the external API of the back-end. It has a processCommand() method and a changeLanguage() method. The processCommand() method is the main method that parses the string and applies the command to the turtles. The changeLanguage() method changes the default language of the parser. The parser class serves as the internal API that interprets and executes the commands.

The view class serves as the external API of the front-end. It has specific methods that display the effects of different commands and presents the current state of the turtle to the user. In more details, it has drawTurtle() and moveTurtle() method to visualize forward and backward movement, and rotate() method to visualize rotation. In addition, it provides a user interface including dropdown menus and other setting buttons for the user to interact with the IDE.

In terms of the inputs of the program, there are various resource files needed to support the functionality of the IDE. For the front-end, text files are needed to fill in the texts on the buttons and other interactive components on the screen; for the back-end, a syntax properties file and a library of language translations are needed for the parser.

Exceptions are handled within the main sections of our project. For the front-end, exceptions will be thrown for input errors such as invalid file path or invalid color. The back-end will throw exceptions if the input command line is invalid or the chosen language is not supported by the library.

User Interface
==========
<img src="https://raw.githubusercontent.com/duke-compsci308-spring2015/slogo_team02/master/DESIGN/UIDesign.JPG?token=AIax0PQpP1XHfz0LUGG4-Dj2VWBeelV6ks5U6lWNwA%3D%3D">
The User interface is represented by the figure above. The user interface has several components which are described below

* Turtle Playground

This is where the turtle acts upon the code given. This is represented by the aliceBlue colored location in the figure above.

* Previously Used Commands

The scroll pane on the right side of the scene represents a list of previously used commands and also suggested commands. This allows the user to click on it and reload the previously used commands into the codeField.

* Terminal

The black window on the lower right side represents the output window. This window displays the results from the user's print statement or displays error messages which were caught in the model.

* Button Pane
    * Change Language allows the user to switch between the avaliable language to code in. At final product, this will be represented by a scroll down instead of a button.
    * Background allows the user to customise the background based on an image. When clicked, it will open a file chooser for the user to pick their color.
    * Image allows the user to change either the color or the image of the turtle. When clicked it will open a popup box asking for the new color or to open a file chooser to select the new shape.
    * Help allows the user to open up a HTML webpage where there will be a [help guide](http://www.cs.duke.edu/courses/compsci308/spring15/assign/03_slogo/commands.php) for the user to see before being able to code in the program.
    * Enter allows the user to submit their code. When enter is clicked, the code inside the codeBox is sent to the controller as a string to be parsed in the model.

* CodeBox

This is represented by the white text area in the lower left of the scene. This is where the user can input their code.

Design Details
==========

The explanation of each of the Model API is listed below.
#####Model External API

public void changeLanguage(String language)
This method takes in a String that represents the language that the commands and user interface will change to. The precondition is that the language passed in must be supported by the program library.

public void processCommand(String code)
This method takes in a string that represents a complete SLogo program. It sends the string to the parser and executes the program. This method catches a ParseException; if an exception occurs, the model stops executing the program and prints an error message to the user terminal.

#####Model Internal API

public void parse(String userInstructions, Program curProgram)
This method takes in a string and a program, and sends both to the Parser instance variable of the class, which parses commands from the string passed in and adds them to the given program. This method throws a ParseException, which typically occurs when the program entered by the user has a syntax or logic error and thus cannot be executed.

public void execute(Program toExecute)
This method takes in a Program object and calls execute on that program.

ViewAbstract is an abstract class which has the public API methods. This is so if the user wants to create their view in Swing, they would be able to access this abstract class to implement the methods in a difference GUI language other than JavaFX. ViewFX is the concrete class that has the implementations of ViewAbstract for JavaFX.

The explanation of each of the View API is listed below

#####View External API

View class' external APIs are the public methods called by the Model and call the internal APIs where detailed functionalities are defined. The methods and their descriptions are as follows:

public void drawTurtle(Point2D new, String ID)
Moves the location of the given turtle from the original location (obtained from the ViewTurtle instance with corresponding ID) and calls lineDrawer() to draw the line (color of which is obtained from the ViewTurtle instance) between the original point and the new point.

public void moveTurtle(Point2D new, String ID)
Moves the location of the given turtle to the given point without drawing a line. Checks from the corresponding ViewTurtle instance whether the pen is up or down.

public void clearScreen(TextBox text)

Removes all the turtles by calling removeViewTurtle() for every ViewTurtle in the ArrayList and clears the terminal by calling clearTerminal(TextBox text), and initializes the panel by setting a ViewTurtle located at (0,0).

public void rotate(double Angle, String ID)
Rotates the specified ViewTurtle by the given amount of angle (Obtains the original angle from the ViewTurtle instance)

public void printToConsole(String message)
Calls printConsole(String text) by passing the 'message' to print on the console screen.

public void addTurtle(Point2D point, String ID)
Calls initializeViewTurtle() by passing the parameters and creates a ViewTurtle object at the specified 2D point with given ID and stores it in the ArrayList<ViewTurtle>.

#####View Internal API
The internal view API contains all the public methods used by the external View API that is not displayed to the user while using it. The methods and their description are as follows:-

public Line lineDraw(Point2D new, Point2D old, String ID, Color lineColor)
This method takes in the old point 2D and the new Point2D and generates a line between them with the color that is specified in the lineColor parameter. This method is called in the external API method known is drawTurtle. If there is a situation where there are multiple turtles, the ID tag can be used to distinguish which turtle is belongs to.

public void clearTerminal(TextBox text)
This method takes in the TextBox which represents the console in the user interface. It clears it by setting all the text in the textbox to a blank string. This method is called in the clearScreen() method in the External API class.

public void removeViewTurtle(ViewTurtle turtle)
This method is called inside the clearScreen() method. It removes the ViewTurtle which stores the shape so it is not visible within the game window anymore

public void initializeViewTurtle(Point2D point, String ID)
This method is called inside addNewTurtle(). It creates the turtle shape and adds it into the game window. It creates the shape at points (x,y) in the grid and also sets the center of the shape to (0,0). This is because when a square is initialized in java, the center is the top left and not the actual center of the square.

public void sendCommand(String text)

This method takes the string inputted in the text window (This is where the user codes his program) and sends it to the controller to be handled by the model side for parsing.

public void printConsole(String text)
This method takes the string text and fills it up in the console to show display it to the user. This can also display error and other erroneous messages.

API Example Code

When the method fd 50 is typed, and the run button is hit, the following commands are executed:-

* executeCommand is called in my Controller to send the String to the parser in model
* Model.
* move() which is a public View API is called by the model. It then moves the turtle by 50px and creates a 50px line

Design Considerations

For the view, there was a consideration on whether to do an abstract for view. This was because it would be redundant in the implementation of it in JavaFX. However, the goal is to make this project very modular and able to adapt to multiple languages hence an abstract view class would allow easy implementation of the methods in other Java GUI languages such as Swing.

Aside from that, there was a design consideration on whether to pass the turtle object between model and view. This makes it easier to code but then it allows access to parameters where if the user changes it, it can cause unwanted effects to the user. In the end, we decided to have a turtle in the model and a turtle in the view where they both handle different parameters. The turtle in the model holds general parameters while the ViewTurtle holds JavaFX specific parameters such as Color and Shape. This makes it much more modular and restrict the user from accessing certain parameters.

We also had a design consideration over how we should implement model view controller. One of the consideration which we ended up not following is having controller initialize both view and the model with the following program loop occur controller->model->controller->view->controller. This made more sense initially as model didn't need an instance of view and view didn't need an instance of model as they both pass through their data through the controller. However in the end, we set up a model->view->controller design explained in the design overview.

Team Responsibilities
================

During our first team meeting, we decided to set up two different groups to work on the project. There is the model group (backend) and view group (front end). The frontend and the backend group would work seperately and convene together twice a week to merge the code together.

* View
    * Jangsoon - Turtle Playground and View Turtle
    * Sivaneshwaran - Setting up the GridPane, Buttons, ObservableListeners

* Model
    * Sierra & Yancheng - Pair Program to create the Model