SLogo API

Turtle Movement and Image API

```
import javafx.geometry.Point2D;
import javafx.scene.Scene;

/**
 *
 *  This class updates the actual image placement in the
 *         Scene.
 */
public class ImageUpdater {

private Scene myScene; // Has access to a scene created by the main
//javafx class so that it can update locations

       /**
        *
        * @param newLocation
        *             the location to move the Turtle's image to.
        * @param turtleImage
        *             the actual image to move. Doesn't throw an   OutOfSceneException
because that should be handled by the TurtleHandler.
        */
       public void updateTurtleImage(Point2D newLocation, Image turtleImage);

       /**
        *
        * @param newColor
        *             The new color for the background of the scene.
        */
       public void setBackgroundColor(Color newColor);

       /**
        * Erases all lines and tells the TurtleHandler to reset the Turtle's
        * location back to home.
        */
       public void clearScreen();

       /**
        *
        * @param from
        *             the starting point of the line
        * @param to
        *             the ending point of the line
```

```
         * @param lineColor
         *            the color of the line to draw (whatever Turtle's pen color is)
         */
        public void drawLine(Point2D from, Point2D to, Color lineColor);

        /**
         *
         * @param shouldShow
         *            If true, show a reference grid, otherwise turn off the
         *            reference grid.
         */
        public void showReferenceGrid(boolean shouldShow);

}
```

```
import javafx.geometry.Point2D;

/**
 *
 *  This class is the actual visual representation of the
 *         Turtle.
 */
public class Turtle {
       private Point2D myPoint; // Has its x and y location
       private int myOrientation;
       private Image myImage;
       private Color myPenColor;
       private int myPenDown; // 1 if pen is down, 0 if pen is up
       private int myShowing; // 1 if the Turtle is showing, 0 if it's hidden

       /**
        *
        * @param orientation
        *             The amount to add to the current orientation/angle
        */
       public void updateOrientation(int orientation);

       /**
        *
        * @param newPoint
        *             The point to move to.
        */
       public void updateLocation(Point2D newPoint);

       /**
        *
        * @param newImageLocation
        *             The location of the image to be used as myImage
        * @throws ImageNotFoundException
        *              The location couldn't be found, so it doesn't update the
        *              image.
        */
       public void updateImage(String newImageLocation)
                     throws ImageNotFoundException;

       /**
        *
        * @param newColor
        *             the color to change the pen to.
        */
       public void changePenColor(Color newColor);

       /**
```

```
     *
     * @param isPenDown
     *            1 if the pen should be down, 0 if the pen should be up.
     */
    public void setPenPosition(int isPenDown);

    /**
     *
     * @param showing
     *            1 if the Turtle is showing, 0 if it's hidden.
     */
    public void setShowing(int showing);

    /**
     * @return Returns the Turtle's current point and orientation
     */
    @Override
    public String toString();

    /**
     *
     * @return myShowing, whether or not the Turtle is visible
     */
    public int getShowing();

    /**
     *
     * @return myPenColor
     */
    public Color getPenColor();
}
```

```java
import javafx.geometry.Point2D;

/**
 *
 *  This class handles the actual position of the Turtle, but
 *         not updating the image. It has an ImageUpdater to do this for it.
 */
public class TurtleHandler {
       private Turtle myTurtle = new Turtle();
       private int sceneXSize, sceneYSize; // Used for checking for
          //OutOfSceneException

       /**
        *
        * @param newLocation
        *            The location the Turtle is moving to. Don't update the
        *            location if it will be out of the scene. Tells the
        *            ImageUpdater to update the Turtle's image.
        */
       public void updateTurtleLocation(Point2D newLocation)
                     throws OutOfSceneException;

       /**
        *
        * @param isPenDown
        *            1 if the Turtle's pen is down, 0 if it's up.
        */
       public void setPenPosition(int isPenDown);

       /**
        * Sets the Turtle's location back to the home spot and then tells the
        * ImageUpdater to update the Turtle image back at that spot.
        */
       public void resetHome();

       /**
        *
        * @param newColor
        *            the color to change the pen to.
        */
       public void changePenColor(Color newColor);

       /**
        *
        * @param showing
        *            1 if the Turtle is showing, 0 if it's hidden.
        */
       public void setShowing(int showing);
	/**
	 *
	 * @param xSize
	 *            the sceneXSize
	 * @param ySize
	 *            the sceneYSize
	 */
	public void setSceneSizes(int xSize, int ySize);

}
```

Main API

```
public class Main extends Application {

        /**
        * Constants
        */
        public static final Dimension DEFAULT_SIZE = new Dimension(800, 600)
        private Scene myScene;


        /**
        * JavaFX thread starts here.
        * Creates the Stage and begins animation.
        */
        @Override
        public void start (Stage arg0) throws Exception;


        /**
        * The main animation loop. Updates one total frame.
        * @param root the root to have the updated display
        */
        public void advanceOneFrame (BorderPane root);


        /**
        * Adds the text field where the user can type in commands
        * @param textfield JavaFX TextField
        * @param root the root to have the added text field
        */
        public void addTextField (TextField textfield, BorderPane root);


        /**
        * Tells the parser to parse the userInput String (determined
        * by whatever was typed in the TextField)
        * @param userInput the user input
        * @return Returns True if XMLparser can parse the userInput String (which means
that the
        * userInput is valid). Otherwise, returns False.
        */
        public boolean sendUserInput(String userInput);


        /**
```

```
	* Displays a list of valid Commands (valid userInputs that the XMLparser could
parse)
	* @param userInput the user input
	*/


	/**
	public void showPreviousCommands(String userInput);


	* Returns a button that launches the HTML help page.
	* @param stage the primary stage
	* @param root the root to have the added button
	* @returns a button that launches the HTML help page.
	*/
	public launchHelpPage (Stage stage, BorderPane root);


	/**
	* Start of the program.
	* @param args
	*/
	public static void main (String[] args);

}
```

HelpPage API

```
import javafx.scene.Scene;

public class HelpPage extends Application {

        /**
        * Constants
        */
        public static final Dimension DEFAULT_SIZE = new Dimension(800, 600);


        /**
        * Creates the JavaFX Stage.
        */
        @Override
        public void start (Stage arg0) throws Exception;


        /**
        * Builds the scene for the help page.
        */
        public void buildScene();
}
```

Parser (Backend) API

```
public class Parser {
    public Parser () {
    }

    /**
     * Accepts a file for parsing
     *
     * @param file
     * file to parse
     */
    public void parseCommandFile(File file)  {
    }

    /**
     * Accepts a command for parsing.  Should utilize the command factory's addCommand
method
     *
     * @param command
     * command to parse
     * @return boolean
     * was parsing successful
     */
    public boolean parseCommand(String command) {
       return false;
    }

}
```

```java
public class CommandFactory {

    public CommandFactory () {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * adds a command to the queue for processing
     *
     * @param command
     * command to
     */
    public void addCommand(String command){

    }

    /**
     * cycles through the queue of command objects, and determines the order for
processing
     */
    public void processQueuedCommands(){

    }

}
```

```java
public abstract class Command {

    public Command () {
    }

    public void setDistance(int pixels){

    }

    public void setAngle(int degrees){

    }

    public void setPosition(Point position){

    }

    public abstract void executeCommand(){

    }
}
```

```java
public abstract class MathCommand {

    /**
     * accepts how many expressions are needed, depending on the subclass
     * ex. SUM would accept expr1 and expr2
     * LOG would accept expr1
     *
     * @param expressions
     * the expressions
     */
    public MathCommand (double[] expressions) {
    }

    /**
     * get the result of the math command
     * @return
     * the result
     */
    public abstract double getResult();

}
```