W. P. CAREY
SCHOOL of BUSINESS
ARIZONA STATE UNIVERSITY

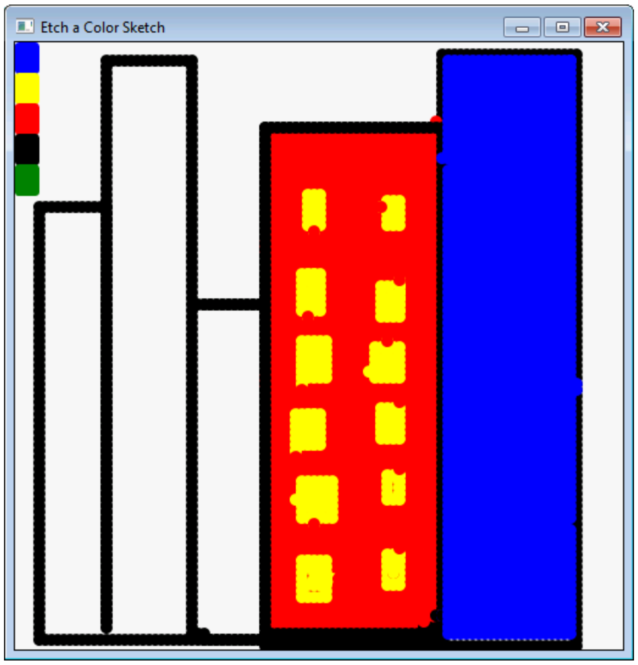
CIS 494 – Business Systems Development with Java – Spring 2015

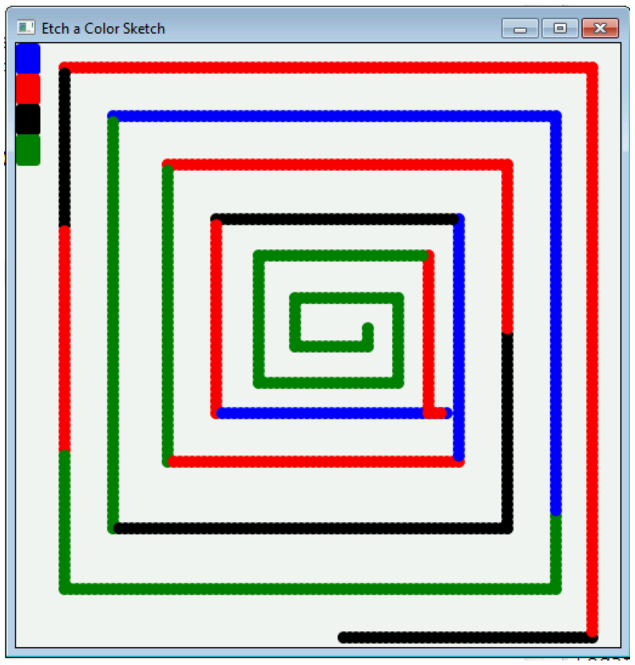
In-class Exercise 7: Key Events: Etch a Sketch

Due on Blackboard: Tonight, March 4 by 11:59 PM

Develop a JavaFX application that emulates the Etch-a-Sketch drawing toy by responding to arrow key events. Use the four arrow keys to move a dot in the specified direction.

Sample Output





Declare all GUI elements as ***instance variables***, i.e. within the class block, so the variables are in scope in all methods.

- Declare the following controls.

    - 5 Rectangles, one for each color, e.g. “blueBox”
    - 1 BorderPane, borderPane
    - 1 VBox, colorPane
    - 1 Pane, drawingPane
    - 1 Circle, dot (this is the dot that will “move” on the drawing pane).
    - 1 Scene, scene

- Declare instance variables:
    - xLocation, yLocation of the dot as doubles
    - xShift, yShift, which will determine how far the dot moves, as doubles
    - currentColor, a string to store the currently selected color.

Method: getPaintValue
Return type: Paint

This method should return a Paint object. From the Paint class, call the static method valueOf and supply with the current Color as an argument. It will return a Paint object with the specified color. Return that object. This is one line of code.

Method: initializeDrawingOptions
Return type: void

- Initialize xShift and yShift with a value from 0 to 10, which will determine how far the dot will move when the user clicks an arrow key.
- Initilize xLocation and yLocation with a value that will be the location where the first dot appears on the screen. Start with a random value of 250, 250 and configure this appropriate when your program starts working based on where you would like the first dot to appear.
- Set the currentColor to Black. Remember the data type of currentColor!
- Create a new instance of Circle. Pass its constructor your xLocation, yLocation, a radius in pixels (which will determine the size of the dot). Your last argument needs to be a Paint object. Call your getPaintValue method to get the Paint object. It should provide an object with the currently selected color. Assign the circle instance to your Circle variable, dot.
- Add the dot to your drawingPane.

Method: drawingPaneKeyPressedEvent
Parameters: e, of type KeyEvent
Return type: void

This method will process key events by looking at the parameter e. The parameter “e”, which is of type KeyEvent contains information regarding the KeyPressed Event, which happens when the user presses a key. The getCode() method within the e object will give you access to which key was pressed.

- Implement a switch statement that switches on the return value of getCode(). Have 4 case statements, one each for UP, DOWN, LEFT, RIGHT. These case values are constants, which are from the KeyEvent class. You only need to type UP, DOWN, etc. and nothing else.

    cases:
    UP: decrease yLocation by yShift
    DOWN: increase yLocation by yShift
    LEFT: decrease xLocation by xShift
    RIGHT: increase yLocation by yShift.
    default: nothing!

- After the switch, create a new Circle instance (in the same manner as you did earlier in the initialize method). Assign the Circle instance to dot.
- Add dot to your drawingPane.

Method: createScene
Return type: void

- Create a new BorderPane and assign it to borderPane.
    - Set a style for borderPane if you want!
- Create a new Pane and assign it to drawingPane.
- Set the event handler for drawingPane so that your method drawingPaneKeyPressedEvent is called *OnKeyPressed*. Remember to pass it “e”, so that that method can find out what key was pressed.
- Create a new VBox and assign it to colorPane
- For ***each*** of your Rectangles that represent a color
    - Create a new Rectangle and assign it to the appropriate Rectangle variable, like blueBox, etc.
    - Set the Height and Width to your preferred value like 20 pixels.
    - Set a Fill, to fill it with color. You can use the static constants from the Color class, e.g. Color.BLUE.

    - Set an event handler for the *MouseClicked* event. You only need two lines of code, which you could write inline by creating a new block of code using {} rather than creating dedicated event handler methods and calling them here. However, should you be more comfortable with writing dedicated event handler methods, feel free to do so.
        - Set the currentColor, e.g. “Blue” or whatever the appropriate color is for this Rectangle.
        - Set the focus to the drawingPane by calling its requestFocus() method.

Remember that the above needs to be done for each color rectangle.

- All add the color Rectangles to the colorPane. You can use the .addAll method to add them all in one go. The addAll method will accept a variable number of arguments.
- Put the colorPane on the left side of the borderPane
- Put the drawingPane in the center of the borderPane.
- Create a new Scene and use borderPane as its main node.

Enhancements

Can you implement an eraser, which erases the line using the same arrow key events?

Can you implement a technique that allows drawing to be turned off, so that the user can jump to another location without drawing?

<u>Submitting Files</u>

Submission should be made using a zip file that contains the entire Eclipse project folder. You will need to ***zip the entire project folder***. The folder will automatically contain the class source files as well as the compiled .class files.

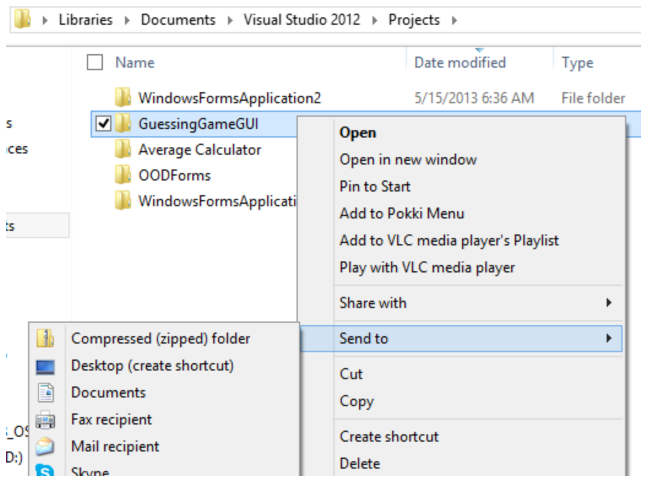
ZIP file should be named: **AX.zip or IC.zip**

where X is the in-class assignment number, e.g., A1.zip is the submission file for Assignment 1 and IC1.zip is the submission file for InClass1.

Note: *The ZIP filename is <u>independent</u> of the Project name. Do not name your project A1.*

How to Properly ZIP and Submit Your Project Files:

- Go to the folder within {Eclipse Workspace}\
- ZIP the entire top-level folder for your project by right-clicking your project folder and selecting Send to | Compressed (zipped) folder.
- Finally, submit the ZIP file using the submission link on Blackboard by the due-date and time listed on the assignment. Upload the ZIP file.



Using built-in windows zip tools: http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files

Verify your files BEFORE and AFTER submission:

- Check for actual class files being present in the folder before you zip it.
- ***Ensure that you are not zipping a short-cut to the folder.***
- After zipping, check file size. A file size under 4K likely does not contain all the files.
- Unzip, extract all files, and verify you see actual files, not a solitary short-cut.
- Uncompress your zip file before submitting and verify that files are present.
- Make sure you have ***submitted*** your file and not just saved a draft on Blackboard. ***A blue clock indicates a submission in progress, i.e. a draft, not a submission. The draft is accessible only to you. You will get a ZERO if you only ever save a draft on Blackboard and never submit your files.***

- Download your zip file after submitting, uncompress, and again verify that your files are present. Test your files in Visual Studio after uncompressing.

This takes an extra couple of minutes. Please do it if your grade is important to you. If you do this, you will not end up submitting a bad file. If you submit an empty file, or one containing only a shortcut, or a bad zip file, or a bad project file, you will receive a score of zero and your only recourse will be to do the makeup assignment at the end of the semester.