Whiteboard Design Document

adamyala — johno — ongck

Introduction

This sections introduces how different modules in our project interact. We model our project design to resemble a concurrent version of MS Paint. Our clients use our GUI to communicate with the Server. As the client draws on the **Canvas,** each **Stroke** is sent to our server through our protocol, and the model is updated in the server. This prompts the server to send an update to all users of that whiteboard, allowing users to work concurrently. The user can choose between all current whiteboards, or create a new whiteboard to work on. The user has the ability draw in different colors (selected in hex) and thicknesses, clear the entire whiteboard, and erase with different sized erasers.

Whiteboard Model

The server will hold a number of Whiteboards, each of which containing a **Drawing** (a **Sketch,** the top level of our ADT). Each whiteboard also stores its Board ID, Board Name, and a list of current subscribers. User's can subscribe to different Whiteboards or create a new one. User's may also change the name of a whiteboard. A Whiteboard has the following methods.

```
Whiteboard:
      // A length 9 unique String representing the board to
      // the server
      String getBoardID() :

      // Returns non-unique human-readable String
      //representing the board to users
      String getBoardName() :

      // Sets non-unique human-readable String
      //representing the board to users
      String setBoardName(String name) :

      // The drawing currently on the board.
      Drawing getDrawing() :

      // Returns the list of current users.
```

```
ArrayList<String> getSubscribers() :

// Adds the user to this board. This user will now
// receive a message whenever the board is updated.
void addSubscriber(String user) :

// Removes the user from this board.This user will no
// longer receive a message whenever the board is updated.
void removeSubscriber(String user) :
```

Abstract Data-type

ADT Definition

Drawing = Stroke or Custom or Sketch
Sketch = Sketch + Custom + Stroke

Our model uses an abstract data-type in order to represent all the possible contents in a whiteboard. We call the contents of a whiteboard a **Drawing**. We have created a **Drawing** interface to define some methods that bridge the gap between our ADT and our GUI. We then created 3 classes that implement **Drawing**--**Stroke**, **Custom**, and **Sketch**. These different hierarchies allow users to compose Drawings. **Custom** allows users to add photo's or other custom images; this class will be implemented if time allows. **Strokes** serve the purpose of real-time updates for freehand drawing and are another form of atomic **Drawing**. The user creates new **Strokes** locally and sends them to the server. The server then combines the whiteboard's **Sketch** with the received **Stroke** using **Sketch**'s **connect** method. The Server then responds with an updated sketch, that the GUI can display using **updateImage**. Erasing in our ADT is represented as drawing in the background color (white).

Interfaces

Drawing:

```
/**
*   Erases all components from the Drawing.
**/
void clear():

/**
*   Returns associated Java Graphics image associated with this
```

```
    *   Drawing. To display this drawing, the GUI must only draw
    *   this image.
    **/
    Image updateImage(Image background):
```

Classes

```
Stroke implements Drawing:
    // Represents an atomic drawing unit
    // The rep is --mutable--
    init:
        Start point
        End point
        Color
        Thickness

Custom implements Drawing:
    // Represents an atomic drawing unit
    // The rep is --mutable--
    init:
        Custom Image

Sketch implements Drawing:
    // Represents unconnected drawings
    // RI: Drawings only added to Sketch, never removed
    init:
        empty

    /** This adds a new drawing to the Sketch.
    *   <Sketch, Drawing> → <Sketch>
    **/
    void connect(Drawing drawing)
```

Domain Specific Language

We build a DSL called WhiteBoardLanguage to operate on our model.
Static Methods within our language:

- clear

- getImage

Protocol

Our protocol separates between messages from the client to the server and messages from the server to the client.

- CLIENT to SERVER
    - Create: "*newBoard \n \n*"
        - Client sends the message, server responds with a boardState message representing a new board.
    - Join: "*joinBoard [boardID] \n \n*"
        - Client sends the message, server adds client to the board's list of members and responds with a boardState message representing the board just joined.
    - Leave: "*leaveBoard [boardID] \n \n*"
        - Client sends the message, server removes client from the board's list of members and responds with confirmation text, "You left board [boardID]."
    - Update Drawing: "*updateBoardDrawing [boardID] [JSON of Drawing] \n \n*"
        - Client sends the message, server responds with a boardState message reflecting the client's (and anyone else's) changes.
    - Update Board name: "*updateBoardName [boardID] [String newName] \n \n*"
        - Client sends the message, server responds with a boardState message reflecting the client's (and anyone else's) changes.
- SERVER to CLIENT
    - Board state: "*[boardID] [JSON of server board] \n \n*"
        - Tells client about the state of a board using a JSON representing the Drawing currently on the board.
        - This is sent to all subscribed clients whenever a board is updated.
    - Boards list: *"BOARDS - [first board], [second board], [third board], …, \n \n"*
        - Sends the client a list of 9-digit integer IDs representing different boards and their non-unique names.

GUI

Three core elements:

- Drawing area
- Buttons used to alter the drawing

- interface to view, join, and leave whiteboards.

Our drawing area is a Canvas object which we display in the center of the screen. To its left go all of the buttons required for drawing. In a minimal implementation, these buttons are "Draw", "Erase", "Change Color", "Change Stroke", and "Clear". "Change Color", "Change Stroke" modify the brush the user is currently using. "Erase" Gives the user an eraser, and "Clear" erases the entire whiteboard.

Lastly, our whiteboard browser is represented as a dropdown list of names located above the drawing area. When a whiteboard is selected, the user automatically joins the selected whiteboard and it is opened in a tab inside the window.

Thread-Safety

We assure our project is thread-safe because both our GUI and Server operate in a single threaded environment. This means that there can be no unwanted interleavings between operations. By only mutating our model in the Server (a single threaded environment), we also assure that our model is threadsafe. The only space for concurrency is the interleaving of messages to and from the server, and the interleaving of user inputs to the GUI. However, all of these messages and inputs are processed in a single threaded fashion and ordering does not cause unwanted race conditions. As a result, our Project is threadsafe.

Testing Strategy

To ensure the correctness of our project, we will be testing our model, our protocol, and our GUI.

Model Testing:

We will doing automated J-unit testing. We will be testing all methods in the Drawing interface and and the additional public methods for each class in our ADT. We will be testing Stroke and Sketch.

Stroke:

- Constructor
    - color, thickness, startPoint, endPoint
- clear()
    - Check if this.color equals Color(0,0,0,0)

Sketch

- Constructor

    - Check if it creates an empty arraylist
- connect()
    - Add drawing to an empty sketch
    - Add drawing to a non-empty sketch
        - Check if the added drawing is equal to that in the sketch
- clear()
    - Check the color of all the drawings in the ArrayList to check if they are transparent

Protocol Testing:

We will be using automated J-unit testing to verify that our protocol is working properly. We will test the specifications of each method, while making especially rigorous tests for the client **updateBoard** calls and the server board message.

GUI Testing:

GUIs are most easily tested by hand, so our testing strategy focuses on testing that each feature works (methods defined in **Drawing** interface) and testing for possible concurrency conflicts.

Brush

- **Draw**
    - Test whether the correct curve appears on the screen as the mouse is dragged
- **Erase**
    - Test if anything can be drawn over with white
    - Test when weight is 0 (should throw exception)
    - Test when weight is a reasonable value (such as 10)
    - Test when weight is very large (such as 1000)
- **Weight**
    - Test when weight is 0 (should throw exception)
    - Test when weight is a reasonable value (such as 10)
    - Test when weight is very large (such as 1000)
- **Color**
    - Test various different colors
    - Should be able to paint over with another color

Global

- **Clear**
    - Test if board becomes blank