Project 4: Message Display Terminal

1 Objective

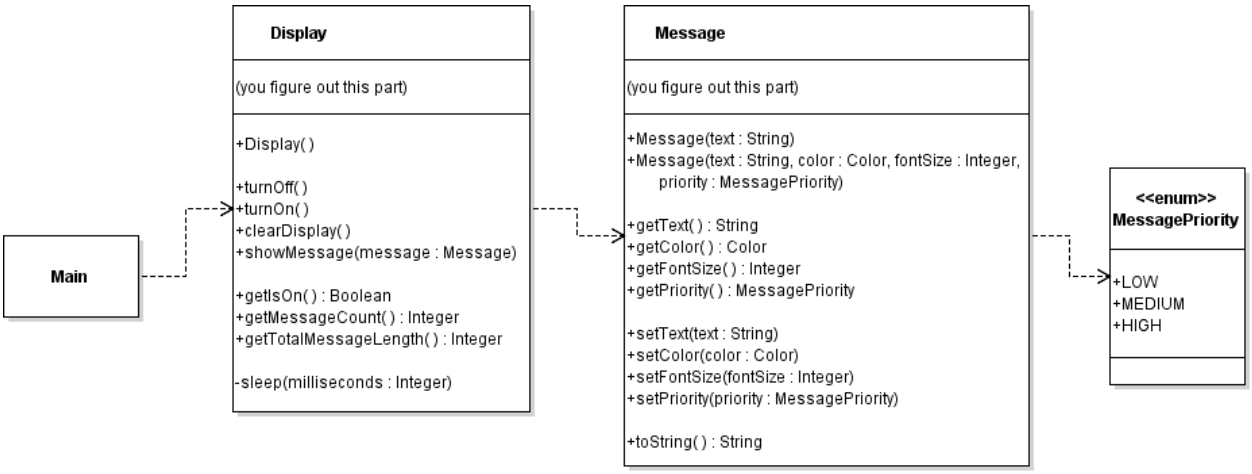
This project is the first time you will create your own objects, along with methods, private data, etc. This is not a complete working system with user interaction; it is just a chance for you to create classes, then instantiate and test them[1]. You will create two classes, one of which uses the other ("has-a" relationship).

2 About Message Display Terminal

The display shows messages that are sent to it. It uses message content to determine the text, size, color, and length of time before the next message is displayed.

3 Classes You will Create

Here is the UML Class Diagram for the objects you are to create, and how they relate to each other. Please pay attention to notation (including parameter and return types). There is a lot of information here; please understand it all (and ask questions if you are unclear) before starting to code!



[1] You are writing Supplier code here, and not Client code. You will create a simple Main as a demonstration, but that is not considered a client deliverable and thus will not be graded.

4 More Details

4.1 Message

Messages consist of:

1. the text of the message
2. the color the text should appear
3. the font size that should be used for the text
4. the message priority (High, Medium, or Low)

The defaults for a Message should be (1) black text, (2) font size of 12, (3) low priority.

MessagePriority should be an enumerated type. These are discussed in slides, videos and in an appendix at the end of your textbook.

Message's toString function should return the message text.

4.2 Display

When started, the Display should show a picture of a computer monitor[2], with its "screen" portion set to a dark color (e.g., dark gray). When turned on, the display's screen portion should be a light color (e.g., white). When client code requests that the display show a message:

- If the display is **on** it should display that message at the next available position on the screen, on a new line, with the requested color and font size. Low-priority messages should remain on the screen for .25 seconds before the next message is displayed; medium, .5 seconds; high, 1 second. These messages affect message count and total length. If the text gets too low on the screen, call clearDisplay which, when the display is on, should draw a light-colored rectangle over the "screen" portion of the display. You may then continue to display text on the screen.
- If the display is **off**, no messages should be shown, counted, etc. Call clearDisplay which, when the display is off, will draw a dark-colored rectangle covering the "screen" portion of the display.

Display's showMessage method maintains the appropriate internal counts and totals. Note that we are not storing all messages as a group; we must handle necessary accounting at message display time.

4.3 Main

Main will demonstrate that the Display and Message classes are working harmoniously. It should instantiate a Display, then instantiate messages and ask the Display to show them.

5 Constraints and Assumptions

- Create *no* static methods.
- Mark *each* instance variable and each method as either public or private (and use no other modifier); follow the UML where it gives guidance and make smart decisions where it does not.
- Use *exactly* the method names shown, or instructor test code may fail (if so, you will lose points). Look carefully at the parameter data types and the return data types; they give you significant clues.

[2] You should find a picture on the internet (e.g., a JPEG file). Find content that is licensed under Creative Commons, so as not to misuse intellectual property.

- Throw an IllegalArgumentException if any of these preconditions are violated:
  - The "usual suspects" are misused (null or empty strings, null object, or enumerated type references).

6 Code Implementation

Follow our Course Style Guide found in Canvas.

7 Testing

Write unit tests for the Message class (only). Test constructors, all methods, and preconditions. And do not forget test code is *still code* and can have bugs, so be suspicious if everything passes miraculously the first time, i.e., test that you can induce failures and get the appropriate error messages shown.

8 Extra Credit

Without extra work, long messages will simply run off the end of the "screen" portion of the display, onto the monitor edges, and off into the ether afterwards; that is not ideal. For extra credit, fix that; back up, word by word, until the message fits[3]. Remaining text should move to the next line (again, subject to fit and perhaps overflowing yet again). To assist with that, create a private method called displayLine that takes parameters including the line to display, the font to use, the color to use, and the number of milliseconds to pause after displaying the text. It should return a string containing any overflow text, so that showMessage can iterate until the whole message has been displayed. None of the extra credit work should affect the Display's statistics regarding message lengths.

You will need to research the Graphics method getFontMetrics; this method will retrieve a FontMetrics object that will be especially useful in determining the width of actual text.

You do not need to test the extra credit work; while interesting and necessary if this were production code, that is more work than I am requesting for this project.

9 Submitting Your Work

Use BlueJ to create a .jar file. Be sure and specify "include source." Submit the .jar file.

10 Hints

- Do not duplicate code; avoid this wherever possible. As an example, do not forget it is legal (and proper) for one constructor to call another constructor; this is good practice and helps avoid duplication and the creation of extra code paths to test.
- Everything you write in this project is Supplier code. You should not ask questions of the user nor print anything out for them to see. Provide a simple Main class (with a runnable main method) to show a quick demo; printing in this code is fine (and expected).

[3] You are not responsible for handling single-word strings that will not fit on a line; if they spill over, that is okay.

11 Grading Matrix

Achievement	Max
Message class	
Constructors	5%
Set methods	5%
Get methods	5%
Minimized code duplication	5%
Preconditions	5%
Display class	
Constructor	5%
Set/get methods	5%
Message display	10%
Message timing	5%
Messages move down display	10%
Display clears when full	10%
Actions when on/off	10%
General coding	5%
Testing of Message class	5%
Style and Documentation	10%
Extra credit	5%
Total	**105%**