[**Department of Computer Science**](http://www.cs.umd.edu/)

[**CMSC131**](http://www.cs.umd.edu/class/fall2017/cmsc131) Fall 2017

**Project:** Marquee

**Due Date:** Tuesday, Nov 28, 8:00 pm

## Overview

For this project you will implement the data (or model) part of the **MVC** design pattern for a program that implements a marquee. A **marquee** is a display unit as seen in old theaters and sports arenas, where a message is formed from a series of blinking lights. By varying the arrangement of lights over time, the marquee can produce the effect that the text is being scrolled from right to left across the window. Your task in this project will be to implement this type of marquee. More details about the program are provided in the **Specifications** section. You may want to take a look at the [**Sample Run**](#gjdgxs) section before you read the description.

## Objectives

To practice two-dimensional arrays.

## Grading

* (40%) Public Tests
* (50%) Release Tests
* (10%) Style

## Code Distribution

The project's code distribution is available by checking out the project named **Marquee**. The code distribution provides you with the following:

* A class named **Marquee** - This class represents the driver that allow us to run the Marquee. You should not modify this class.
* A class named **PublicTest** - This class represents public tests for the project.
* A package named **cmscMarqueeLib**  - This package provides support classes for the project. You should not modify or add anything to this package.

## Specifications

You will write a program that lets a user display a message in a marquee. The marquee display consists of a 2-dimensional array of cells where each cell can assume the colors red or white (see the [**Sample Run**](#gjdgxs) section). The program will prompt the user for a message to display and will process that message so it can be moved across the display from right to left. For this project we are providing the view and controller part of the **MVC** architecture, and you must implement the model part. In other words, we will do the displaying, but you need to implement what is to be displayed at each step of the animation.

### Animations

In order to understand what you must implement, let's first overview how we can create animations. An animation consists of a sequence of still images, which are rapidly updated, and so creates the illusion of motion. The redrawing process is triggered by a timer event, which fires regularly at some small time interval (e.g., every 10th of a second).

### How Does the Marquee Program Works

For this project we have implemented two classes named **MarqueeDisplay.java** and **MarqueeController.java**. These classes implement the view and controller part of the **MVC** architecture respectively. The MarqueeDisplay class has a method named **displayMessage()**, which takes a two-dimensional array of cells and displays the array on the marquee display. The **MarqueeController** class implements a timer that calls the **displayMessage()** method of the **MarqueeDisplay** class, with a particular two-dimensional array of cells, at a particular rate.

Where does the array that is being displayed comes from? This is the result of calling a method named **step()**, which is defined in the class **MarqueeDataManager**. The MarqueeController class calls step() to obtain the next two-dimensional array to display. Each call produces a new 2-dimensional array, which has been shifted slightly relative to the prior one; this creates the illusion of motion. Your task for this project is to implement the step() method. You do not need to modify the MarqueeDisplay or the MarqueeController class. If you define step() correctly, the support classes will take care of the rest.

### Marquee Library

The **cmscMarqueeLib** has the support classes you need for this homework. The classes you will find in this package are:

* **Cell**: This class implements a cell (the equivalent of a pixel) of the marquee display. It is an object that stores the color of the cell. There are two colors, BACKGROUND\_COLOR and FOREGROUND\_COLOR (defined below). The Cell constructor is given the color of the cell as its only argument. For example, the call **Cell(FOREGROUND\_COLOR)** is used to in creating a new foreground cell. Note that Cell is an immutable class, which means that it is safe to copy references to them.
* **ConfigValues**: There are constant parameters that define things like the height and width of the characters and the marquee; this class defines these constants. You should use these constants by name (rather than their integer values) since we may change them. The constants you will need in your program are:  
    
    
  + **EMPTY\_COLUMNS\_BETWEEN\_CHARS**: Consecutive characters in the Marquee are separated by a small number of columns. (See the [**Sample Run**](#gjdgxs).) This integer constant gives the number of such columns between each pair of characters.
  + **CHARACTER\_HEIGHT**: The height of a character in cells.
  + **CHARACTER\_WIDTH**: The width of a character in cells.
  + **MARQUEE\_HEIGHT**: The height of the marquee in cells. For this project you may assume that the marquee height equals the character height.
  + **MARQUEE\_WIDTH**: The width of the marquee in cells.
  + **FOREGROUND\_COLOR**: The color of a cell used to represent a character.
  + **BACKGROUND\_COLOR**: The color of a cell used to represent the background of the display.

The constants **FOREGROUND\_COLOR** and **BACKGROUND\_COLOR** are of type **Color**. This class is defined in the java.awt package of the Java class library.

* **DataManager**: This is an interface that your **MarqueeDataManager** class must implement. The interface defines only one method named step() which has the following prototype:  
   **public Cell[][] step();**
* **MarqueeDisplay**: This class implements the marquee display.
* **MarqueeController**: This class implements the controller of your system.
* **Alphabet**: As you can imagine, everyone can define different 2-dimensional Cell object arrangements in order to generate the English alphabet. To avoid this situation, we have defined the Alphabet class. This class defines a method called **toIntArray()** which has the following prototype:  
   **public static int[][] toIntArray(char inputChar);**  
   The method returns a 2-dimensional integer array that corresponds to the specified character. For example calling the method with letter 'E' will return the following array:  
   1111111  
   1000000  
   1000000  
   1000000  
   1111000  
   1000000  
   1000000  
   1000000  
   1111111  
   As you can see those array entries with 1 represent cells of the two-dimensional array that must have a foreground (red) color and those with 0 a background (white) color. The method **toIntArray()** only returns arrays for the English alphabet ('a'-'z' and 'A'-'Z') and the space character (' '). Digits or other symbols cannot be transformed into arrays. Although you can provide lowercase letters, those are transformed into uppercase letters. Notice that in order to display a character you must transform the array of integers into an array of Cell objects.

### Files

* **Marquee.java**: This class has a main method which starts the marquee program. The main method creates a MarqueeDisplay object, a MarqueeDataManager object and a MarqueeController object using the previous two objects. It then proceeds to start the animation by calling the controller start method. You must not modify the contents of this method.
* **TestsSupport.java**: Provides tests support for your program. Do not modify.

### Your Assignment

Your assignment is to implement the **MarqueeDataManager** class. (Everything else is provided.) The class implements the DataManager interface therefore it must define the step() method. The step() method will return the next 2-dimensional array of Cell objects to display in the animation process. The MarqueeDataManager class must define a constructor with the following prototype:

public MarqueeDataManager(String message, int animationPattern);

The message parameter represents the message you want to display. The animationPattern is an integer that represents which animation pattern you are interested. We use 0 (zero) as the animation pattern for this project. When this class has an honors section (e.g., CMSC131H) students in that section will use this value to distiguish between the animation pattern for the project and animation patterns for the honor part. Your constructor should set up the state of the MarqueeDataManager object so that every time the step() method is called, a 2-dimensional array of Cells representing the next stage in your animation is returned.

### MarqueeDataManager step() Method Restrictions

The following restrictions must be observed as you implement the step() method:

* The first call to the step() method must return a 2-dimensional array that represents an "empty" display, that is, only containing pixels with the background color.
* Each step() method call moves the message to the left along the marquee by exactly one column. For example, the first call results in an entirely empty marquee and the second call will display the leftmost column of the first letter.
* The message wraps around the display in the following manner:
  + (Step n) The rightmost column of the last letter appears in the leftmost column of the marquee,
  + (Step n+1) The marquee is empty,
  + (Step n+2) The leftmost column of the first letter appears in the righmost column of the marquee.
* Each character displayed is separated from the preceeding character by a gap consisting of some number of empty columns. The constant **EMPTY\_COLUMNS\_BETWEEN\_CHARS** indicates this number of columns. (Note: If you have m letters then there will be m-1 gaps between these letters to be filled.)
* The size of the array returned by step() must have a same dimensions size of the marquee, namely, **MARQUEE\_HEIGHT** by **MARQUEE\_WIDTH**.

### Hints on Implementing the Marquee

You can follow any approach (algorithm) you understand is best in implementing the MarqueeDataManager class. The approach that we used in our implementation is based on a "sliding window" principle. Here is a general outline of the approach, and a figure is provided below to illustrate it.

* **Setting up the padded array:**
  + Convert each character of the message to display into a 2-dimensional array of Cell objects by calling the Alphabet.toIntArray() method and then translating the array of 0-1 values into a 2-dimensional array of Cell references.
  + For each individual character, concatenate the 2-dimensional arrays of Cell objects representing the message (being sure to add the empty columns between each character.) We will refer to this 2-dimensional array as the **message array**.
  + Pad both sides of the message array with a block of "empty" columns of cells that is exactly the same size as the marquee. We will refer to this as the **padded array**.
* **The sliding window and step():**  
    
   Once you have created the padded array, each call to step() returns a 2-dimensional subarray of this array called the "**window**". Each window is exactly the same in size as the marquee. The window starts at the left end of the padded array and each call to step() causes it to slide one column to the right. Copy the contents to a 2-dimensional array, which is then returned as the result of step().  
    
    
    
  

### Sample Run

Here is an example that show a couple of snapshots of the marquee as it displays the message "FEAR THE TURTLE". By the way, in order to stop the marquee just close the display window.











You can also find a video for the application at [Video](http://docs.google.com/demoMarquee.swf). Make sure the browser window is large enough, otherwise you will not see the animation.

## Requirements

* Follow the style guide provided at [Style Guide](http://www.cs.umd.edu/class/fall2015/cmsc131/content/resources/StyleGuidelines.html).
* You should not modify the project specifications. For example, do not add extra behavior or output.

## Additional Project Information

Additional project information can be found at [MarqueeProjectNotes.pdf](http://docs.google.com/MarqueeProjectNotes.pdf).

## Submission

Submit your project from Eclipse (within Java perspective) by right-clicking the project folder and selecting "submit" . After you have submitted your project, you should visit the submit server. There you can obtain feedback about how well your project is performing. The number of times you can run our tests on your project (before the due date) is limited. The earlier you begin working on the project, the more opportunities you will have to see how your project performs on our tests before the due date!

## Academic Integrity

Please make sure you read the academic integrity section of the syllabus so you understand what is permissible in our programming projects. We want to remind you that we check your project against other students' projects and any case of academic dishonesty will be referred to the [Office of Student Conduct](http://www.jpo.umd.edu/).