Project 3

MVC on the Clock

100 points

Assignment posted on April 05, 2018.

DUE: April 29, 2018, end of day

No late submission accepted; submit link canceled at stated time

Last day of project advising: 04/26

Learning Outcomes

- Recognize the need for arrays in programming tasks, and manipulate data in one and multi-dimensional arrays (b, c, i, j)
- Design and implement multi-class solutions to programming problems (b, c, i)
- Apply and utilize fundamental GUI components and operations (c, h, i)
- Utilize event driven programming and interfaces in advanced GUI applications (b, c, i)
- Utilize Graphics and Timer class in advanced GUI with animation (b, c, i)

Objectives

In completing this project you will experience

- The MVC architecture
- GUI with various components
- Listeners, event driven GUI elements
- Timer class
- Painting components
- Animation on GUI

Problem Statement

For this project you are to write a program such that it simulates a working clock allowing several control devices to change its state. A clock is an excellent real life object for which the MVC architecture easily applies, since the roles of Model, View and Controller are clearly distinguishable. The mechanical structure of the clock is represented by the model class named ClockWork. This class stores and processes the changing data for the moving clock without revealing the clock behavior on a display. The view class that represents the face of the clock, is expected to provide a rich graphical display of ClockWork activities. The control class has the tools the user can use to send input to ClockWork and by that to change the clock's state and behavior.

The face of the clock as shown in Figure 1 is the initial state of the clock, all three arms point to 12 hours. The arms of the clock move according to the following rules.

- The seconds arm (painted red) moves 1 tack forward every second
- The minute arm or long arm painted blue moves one tack forward after every full turn of the seconds arm
- The hour arm or short arm also painted blue moves one tack forward after every 12 tack steps of the minute arm

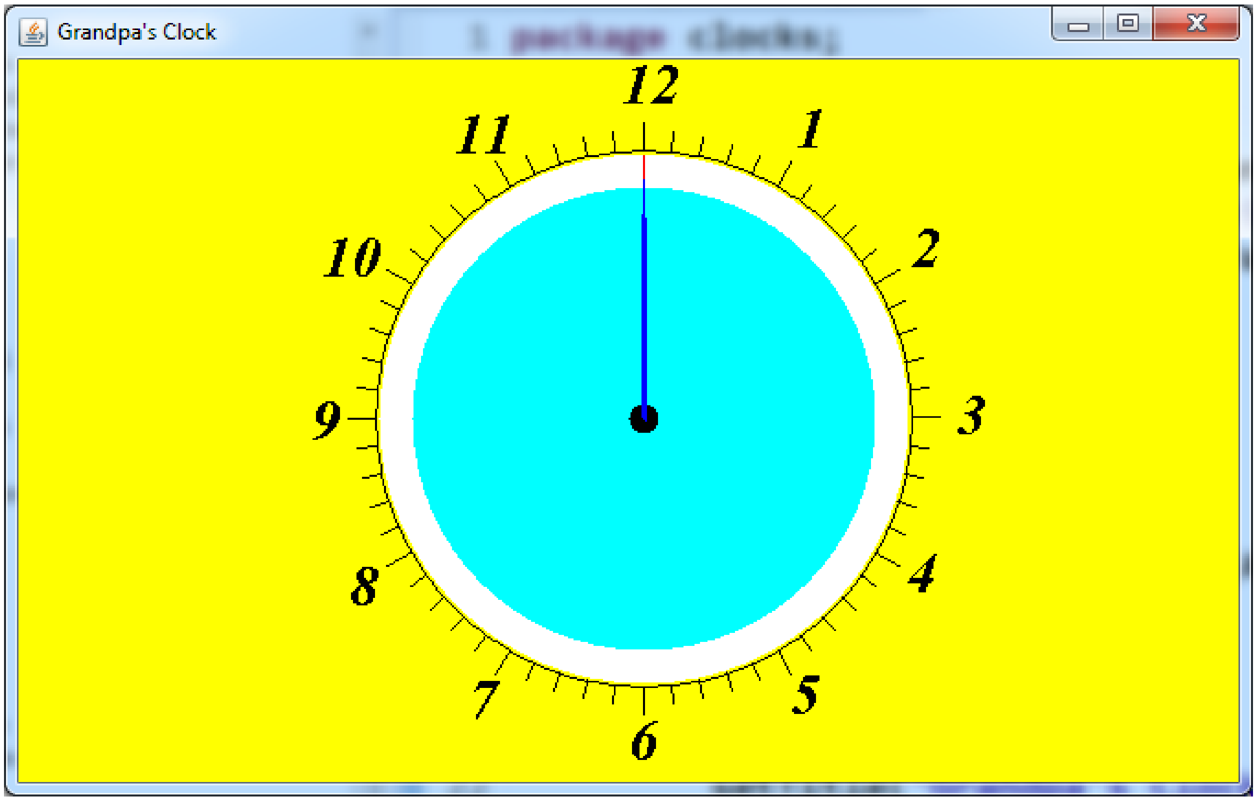


Figure 1

- After 2 hours 28 minutes and 36 seconds the state of the clock is shown on Figure 2.

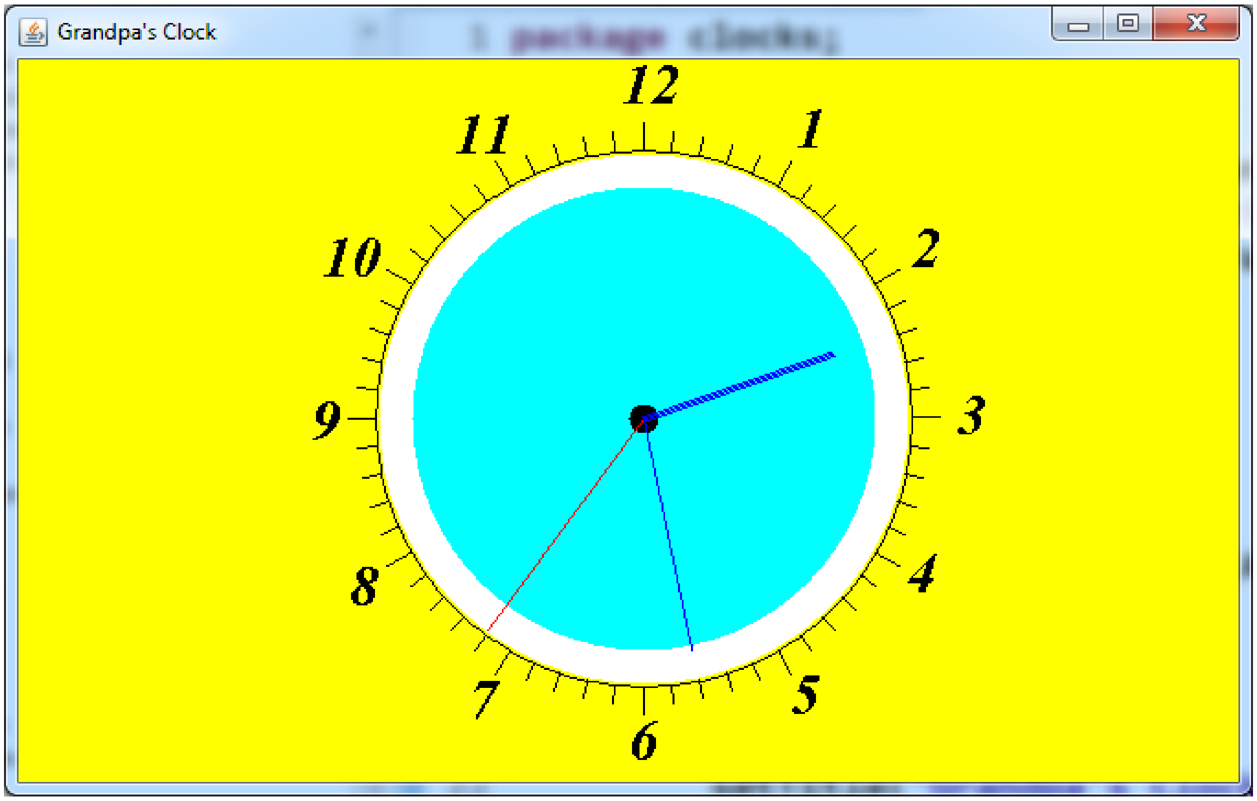


Figure 2

The controller class must provide the user the ability to

- start or re-start the clock
- stop the clock
- reset the clock
- set the clock to any selected time
- adjust the clock speed by selecting any ‘second’ unit between 0 and 1000 milliseconds (the latter is a real second).

The controller class must itself be GUI as shown on Figure 3

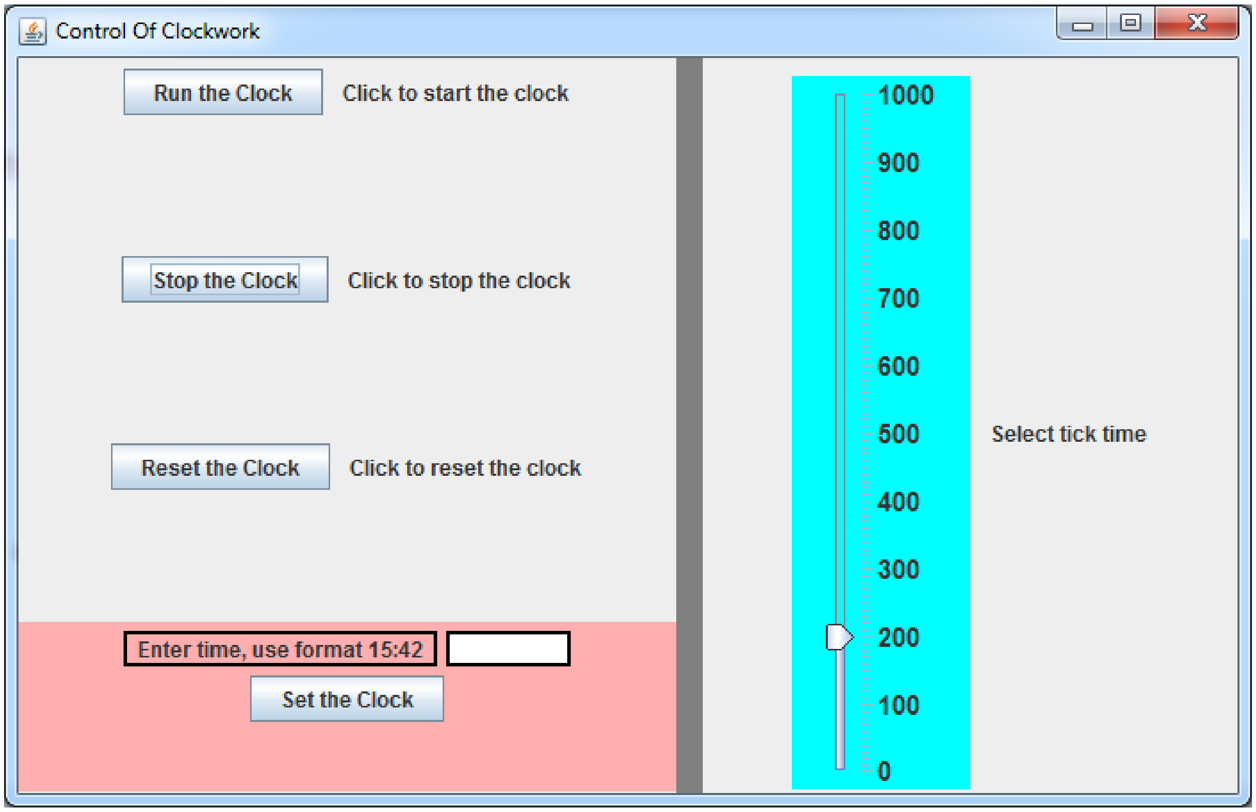


Figure 3

Analysis and requirements

Input and GUI functionalities.

- The structural input values necessary for building the GUIs are handled internally by the view class and the control class. A few of the structural parameters of the view must also be available for the model. These input are the choice of the programmer
- ‘On the fly input’ will be generated events fired by the active components of the control GUI.
- There have to be 60 tacks uniformly distributed around the perimeter of the clock face, the circular arcs between any two neighboring tacks represent one second time unit, the unit movement of the seconds arm.
- Every fifth tack is a bit longer than the rest, these longer tacks represent the hour positions.

- The coloring of the face is important, beyond its aesthetic value it helps the better perception of the clock movements.
- The longer tacks are numbered according to the hour values

The controller GUI is divided into two major regions. The components as shown on Figure 3 clearly identify themselves. According to the requirements described in the previous section,

- Button 'Run the Clock' starts the clock moving, and at the same time he button disables itself.
- A click of the buttons 'Stop the Clock' or 'Reset the Clock' makes the 'Run' button enabled again , a click of 'Set the Clock' does not.
- 'Stop the Clock' stops the clock at the current position.
- 'Reset the Clock' stops the clock and resets the arms to the initial position.
- On the bottom panel there is a bordered label, a bordered text field and the 'Set the Clock' button. The text field takes and input time value, and the button click sets the clock state to the input time. The input must be validated strictly to the required format. Invalid input is ignored and no change in the clock's state entails. In case of correct input the setting is executed; a running clock continues running from the newly set position, stationary clock remains stopped. The button click also vacates the text field whatever input if any was written there. The validation ensures that no input can crash the program and interrupt the clock activities.
- The only way for the windows to terminate the process is to apply the default close operation on either GUI.
- The east region is the host for the slider. Selecting a new tick time on the slider does not affect the running clock immediately. Immediate speed change can be fun to watch, but it would not be appropriate when the clock is used for measuring a time period. To have the tick time selection effective it must be followed by a 'Run…' click, which is only possible if a running clock is stopped first.

Output are the GUIs and the observable clock operations.

Design

The logic of the design is illustrated by the Class Chart at the end of the project description.

You will add four classes to the project

- **ClockWork,** non-GUI class handling all the application data needed for the working of the clock
- **ControlDevices**, a GUI class representing the control of the MVC design via active GUI components: buttons and slider; this class communicates to ClockWork only
- **FaceGUI,** a GUI class provides the visual representation (view) of the clock; uses data from ClockWork to update the GUI when notified by ClockWork
- **Test** contains the main method, instantiates the clock classes with the aggregation as shown on the class chart

1. **`ClockWork`** This class is the Model of the MVC paradigm.

`ClockWork`	
DATA FIELDS	
Clock parameters	

`-longArmIndex : int`
`-shortArmIndex : int`
`-secondsArmIndex : int`
	Store the current tack positions of the arms, a value from 0 up to 59

`-radius : int`
`-centerX : int`
`-centerY : int`
	Determined by the GUI size and location, values forwarded from the FaceGUI class when the frame is built.

`-x : int[]`
`-y : int[]`
`-xx : int[]`
`-yy : int[]`
	End-point coordinates of the tacks, each array is of length 60; arrays are instantiated at declaration

`-tickTime : int`	Stores the timer delay, the actual time period of one tick of the seconds arm; the delay is measured in milliseconds, a value between 0 and 1000
Functional & communication fields	
`-timer : Timer`	Controls the clock movement
`-listener : ActionListener`	Executes actionPerformed in the GUI
`-event: ActionEvent`	The parameter in actionPerformed
METHODS	
Constructor	
`+ClockWork()`	Instantiates event
Instance methods	
Setters and getters	getter(s) needed (in the View class) for the arm index fields and for the tack coordinates. If you create an array holding the arm index values and a 2-dimensional array holding the coordinate arrays, a single getter method would do in both cases; setter(s) needed for the circle data; again, a single

`+setTickTime(tick:int): void`	setter is recommended to set the values together
`+addActionListener(watch: ActionListener) : void`	setter for listener
`+loadTacks(): void`	Runs a **for** loop to populate the four end-point coordinate-arrays for the tacks; the points are on circles, Math.cos and Math.sin needed for the calculations. Try to figure out the formulas before you go to the Hint at the end of the description
`+makeTimer() : void`	Instantiates a new Timer object for timer; constructor parameters are tickTime for delay and a new TimerListener (an anonymous object from the inner class, see below); the method calls the start( ) method of the Timer class
`+stop() : void`	calls the stop( ) method of the Timer class
`+reset() : void`	Stops the timer; resets the arm index values to 0; calls connect( )
`-timeFormat(inp:String):boolean`	private helper method; validates the parameter inp if it is correct; returns true/false according to validation
`+setClock(timeToSet: String):void`	Calls timeFormat( ) to validate timeToSet; if not valid, the method returns; if valid, assigns shortArmIndex and longArmIndex according to the input and secondsArmIndex to 0; calls connect( )
`-connect( ): void`	Private helper; calls actionPerformed( ) with respect to listener as prefix and event in the parameter
Inner class	
`TimerListener`	Implements ActionListener
`METHOD`	

`+actionPerformed(event: ActionEvent): void`	updates secondArmIndex reduced modulo 60;
	if secondsArmIndex is 0, updates longArmIndex modulo 60;
	if longArmIndex is divisible by 12 updates shortArmIndex modulo 60;
	calls connect( )

2. `ControlDevices` This class is the Control of the MVC paradigm

`ControlDevices` **extends JFrame, implements ActionListener**	
DATA FIELDS	

`-work : ClockWork`
 	
`swing components as shown on Figure 3`
	To aggregate ClockWork to the class; initialized by the constructor
	
	JTextField, JLabel, four JButton, JSlider references; all can be instantiated at declaration

METHODS	
Constructor	
`+ControlDevices(cw: ClockWork)`	Initializes the data field **`work;`** calls **`makeSlider()`** and **`buildControlGUI()`**
Instance methods	
`-makeSlider( ): void`	Endows the slider as Figure 3 shows. Book and lecture material and your Lab 8 experience apply
`-buildControlGUI( ): void`	Creates the frame as Figure 3 show. Apply (as many) JPanels as appropriate (local variables). The GUI shown is built on the CENTER and EAST regions of the pane's BorderLayout, and the panel in the center applies a nested GridLayout ; your individual GUI design is welcome as long as the control provides all information and functionalities as required
`+stateChanged(event: ChangeEvent): void`	Defines the method from the ChangeListener interface; calls the setter method of the tickTime field of ClockWork passing the current value of the slider as parameter
`+actionPerformed(event: ActionEvent): void`	If the source is the --stop button, enables run and calls stop() of ClockWork --run button, disables run, calls makeTimer() of ClockWork --reset button, enables run and calls reset( ) of ClockWork --setClock button, calls setClock() of ClockWork passing the text from the text field as parameter and sets text field the empty string

3. `FaceGUI` This class is the View of the MVC paradigm.

`FaceGUI` extends JFrame implements ActionListener	
DATA FIELDS	
`-work : ClockWork`	To aggregate ClockWork to the class; initialized by the constructor
`-face : Face`	Inner class reference

`-radius : int`
`-centerX : int`
`-centerY : int`
	Circle data for the clock face; initialized when the GUI is built, used in Face as well as in ClockWork

METHODS	
Constructor	
`+FaceGUI(cw: ClockWork)`	Initializes the data field **`work;`** registers **`this`** as the ActionListener with work; calls buildClock()
Instance methods	
`-buildClock( ): void`	Makes the fundamental calls and parameter choices to build the GUI as in Figure 1. Initializes the circle data, such that the center is about the center of the frame; some vertical adjustment may be desirable; the radius must be less than the vertical half size to give room for the numbers The method calls the setter in ClockWork to pass the circle data to that class; calls loadTacks of ClockWork; instantiates the field **face**
`+actionPerformed(event: ActionEvent): void`	Interface method; face calls repaint( )
Inner class	
`Face`	extends JPanel Note: building the panel for the clock face requires substantial work, therefore it is justified to place the code of all the drawings in an inner class
`METHOD`	
`+paintComponent(g: Graphics)`	Superclass method The first statement must be the call of the same method with super prefix;

	The method --calls getter(s) to get arms index values and stores them in local variables; --calls getter(s) to get the tack coordinates and stores them in local arrays; --draws three circles and fills with white, cyan and black colors (that last fill overlaps, therefore the fills must be done in decreasing order of radii) --runs a for loop to draw the tacks all around the clock face --draws the numbers to clock; this is best done by choosing individually the parameters for the enclosing rectangles; set your favorite font before the drawing --draws the clock arms one by one; draw from the center to the other end such that the end is an appropriate adjustment of the tack coordinates

Implementation Hint

Here is a recommendable formula applicable in the for loop of the loadTacks method (ClockWork class, see the fields in the UML diagram):

```
x[k] = (int) (centerX + radius*Math.sin(2*k*Math.PI/60));

y[k] = (int) (centerY - radius*Math.cos(2*k*Math.PI/60));
```

Analogous assignments apply for the farther endpoints, but replace radius by radius + 10, and for every index divisible by 5 replace radius by radius +15 (the longer tacks)

Test class

In the main method

- Instantiate first a ClockWork object cw;
- Instantiate a ControlDevices object, pass cw as parameter in the constructor;
- Instantiate a FaceGUI object, pass cw as parameter in the constructor;

Evaluation

Documentation and Style (20 points).
Your program must conform with the Java Documentation and Style Requirements as set by the Computer Science Department. Emphasis will be placed on having the required banner and internal comments, indentation, and overall professional appearance. Comments must be clearly written with correct grammar and spelling.

Correctness (80 points).

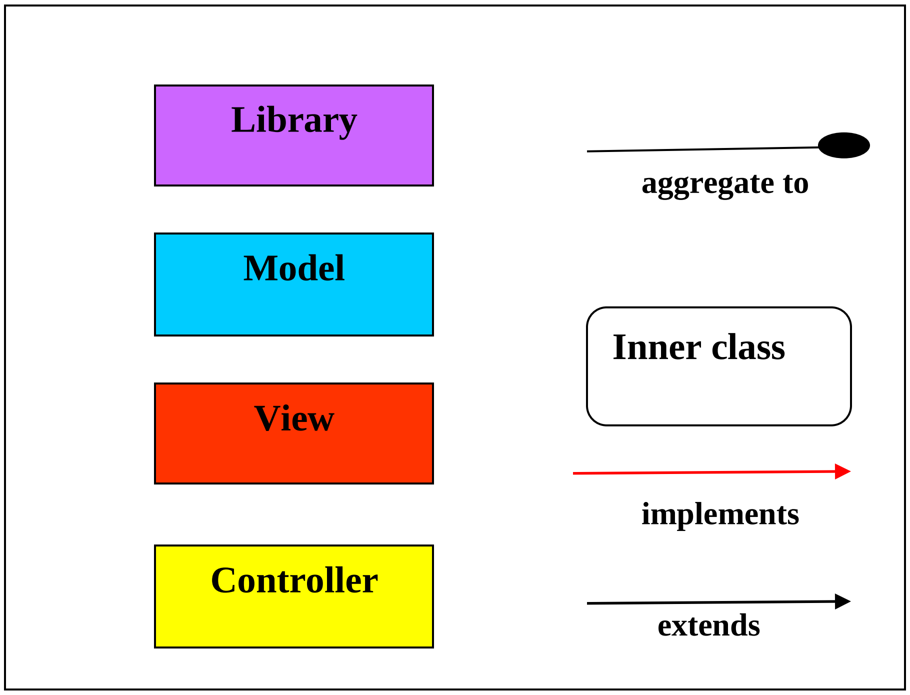
1.	ClockWork setup correct	05
2.	ClockWork methods correct	30
3.	ControlDevices built to specifications	10
4.	ControlDevices methods correct	10
5.	FaceGUI built to specifications	10
6.	Face inner class paintComponent( )	15

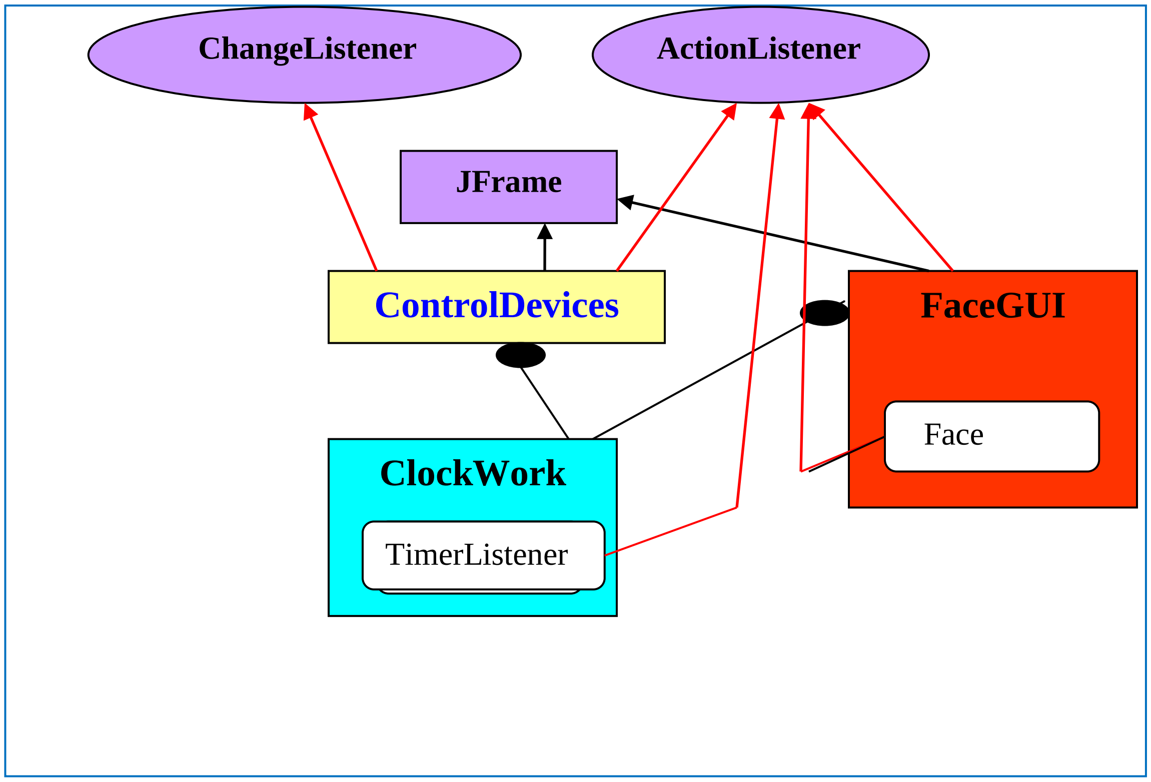
Total……………………………………….100 points

Submit: Upload the zipped project folder at the class site (not your lab site) on Blackboard

Class Chart

Color code:




ChangeListener
ActionListener
JFrame
ControlDevices
FaceGUI
Face
ClockWork
TimerListener