John Hibbert | CSC212 | Final Project Technical Doc | 12/3/2014

**public** **class** BoothOutletManager

**extends** Frame

**implements** MouseListener {

**private** **int** lastX, lastY;

Mouselistener checks these to see where the mouse is.

**private** Abutton blenderButton, microButton, hotDogButton, lavaLampButton, popcornButton, removeAmpsButton, removeIncomeButton;

The buttons needed to give the user access to the functionality. The first 5 add an appliance to the collection, while the last two remove them, one by removing the one with the highest amps, one by removing the one with the lowest income.

**private** DataOrganizer myDataOrganizer = **new** DataOrganizer();

The data organizer class is what manages the double-linked-list that forms the basis of the program.

**int** x;

**int** y;

These two variables are needed for drawing.

**int** applianceX

**int** applianceY

These two variables are passed to appliances in the data organizer in order to tell them where to draw

**final** **int** HOURSBOOTHWILLBEOPEN

A constant of how long the booth will be open. That is multiplied by the hourly profit to get the overall profit.

**final** **double** MAXIMUMAMPS

A constant of how many amps the circuit breaker can handle. If this is exceeded, the user cannot add more appliances until they have removed enough to get under this number,

**final** **double** BOOTHRENTALCOST

A constant that is subtracted from the overall profit from the event. This is the cost to rent the booth and have the privilege of selling wares.

**final** **int** ROUNDINGFACTOR

This is the decimal place to which the big decimals are rounded to. This is necessary to keep the numbers printing neat.

**boolean** overloaded = **false**;

This is a boolean that is flipped whenever the current amount of amps exceeds the MAXIMUMAMPS.

**public** BoothOutletManager() {

The constructor. In here, we do basic housekeeping and build the buttons.

**public** **void** paint(Graphics pane)

The paint method. Here we paint the buttons, the data organizer, the power strip itself and the output charts.

**public** **void** drawPowerStrip(Graphics pane)

This method draws the powerstrip visible in the program.

**public** **void** drawOutlet(Graphics pane, **int** x, **int** y)

This method draws the raised faces of the outlets on the power strip itself.

**public** **void** drawTotals(Graphics pane)

This method draws the chart that shows the costs, profits, and net.

**public** **void** drawAmps(Graphics pane)

This method draws a small label which tracks the number of amps as well as the maximum number of amps.

**public** **void** mouseClicked(MouseEvent event)

This runs the check method.

**public** **void** mousePressed(MouseEvent event)

This updates the mouse location and 'flips' the buttons.

**public** **void** mouseReleased(MouseEvent event)

This 'flips' the buttons again upon release.

**public** **void** mouseEntered(MouseEvent event) {}

**public** **void** mouseExited(MouseEvent event) {}

Neither of these two are used, but they must be implemented as the class implements mouselistener.

**private** **void** flipWhenInside()

This calls the flip method for the buttons if lastX and lastY are within the button. The appliance buttons should not flip when the circuit is overloaded.

**private** **void** checkAmpLevels()

This method compares the total amps to the maximum amps. If it is higher, it sets overloaded to true and changes the background color to darker to indicate that the lights have gone out.

**private** **void** check()

This method checks the location of the mouse compared to the buttons to see what it should do. Similar to the flipWhenInside, the appliance buttons should not be allowed to be used while the system is overloaded.

**public** **class** DataOrganizer {

Node head;

The head of the double-linked-list.

**public** DataOrganizer()

Just sets head to null. The labor is handled by the plugIn application.

**public** **void** plugIn(Appliance givenAppliance)

This is the add method. It creates a new node, adds the appliance to that node, then installs itself into the collection in the appropriate spot.

The appropriate spot is found as follows: First, check if the given appliance has a big plug. If it does, install it behind the first small plug, or at the end if there are no small plugs. If it does not have a big plug, install at the end no matter what.

**public** **void** updateLocations()

This method is there to update the location data of the appliances whenever we add or remove something. It iterates through the list and uses the appliance's setLocation method to give it a new x and y, then increments x.

**public** **void** unplug(Node givenNode)

This is the remove method. It removes a given node from the collection.

**public** Node findHighestAmps()

Returns the node containing the appliance with the highest value in its amps variable.

**public** Node findLowestIncome()

Returns the node containing the appliance with the lowest value in its hourlyIncome variable.

**public** **double** getTotalAmps()

Returns a double of the sum of the value of all the amps variables for the entire list.

**public** **double** getTotalRentalCost()

Returns a double of the sum of the value of all the rentalCost variables for the entire list.

**public** **double** getTotalHourlyIncome()

Returns a double of the sum of the value of all the hourlyIncome variables for the entire list.

**public** **void** paint(Graphics pane)

Paints the nodes in the collection.

**public** **abstract** **class** Appliance {

This class defines the functionality of the appliances, which must extend this class.

**protected** **double** amps;

The number of amps the appliance requires.

**protected** **double** rentalCost;

The cost to rent this appliance for the event.

**protected** **double** hourlyIncome;

The amount of money this appliance will generate in an hour.

**protected** **double** reductionFactor;

The factor by which the hourly income will be reduced for this appliance for each additional appliance of the same type is present.

**protected** **boolean** bigPlug;

This represents whether or not the appliance has a large plug that covers more than one outlet.

**protected** **int** x;

**protected** **int** y;

The X and Y locations of where the appliance should be drawn.

**public** **void** setLocation(**int** x, **int** y)

Sets the X and Y variable to the given values.

**public** **double** getAmps(){

**public** **double** getRentalCost()

**public** **double** getHourlyIncome()

Returns the value of the given variables.

**public** **boolean** hasBigPlug()

Returns the value of the bigPlug variable.

**public** **void** drawPlug(Graphics pane)

This instructs the paint method how to draw the plug. It checks if its bigPlug value is true or false to do so.

**public** **void** paint(Graphics pane){};

This method must be overridden by the child class, as each appliance will be drawn in a different way. Still, each appliance must be drawn, so adding it to this abstract class will mandate that.

**public** **class** Node {

The basic structure of the double-linked-list.

**private** Appliance thisAppliance;

The appliance that the node contains.

**private** Node next;

The next item in the list.

**private** Node prev;

The previous item in the list.

**public** Node()

The defualt constructor.

**public** Node(Appliance givenAppliance)

A constructor that sets the appliance to the given appliance.

**public** Node getNext()

**public** **boolean** hasNext()

**public** **void** setNext(Node givenNode)

**public** Node getPrev()

**public** **boolean** hasPrev()

**public** **void** setPrev(Node givenNode)

Setters, getters and has-checks for the prev and next variables.

**public** Appliance getItem()

**public** **void** setItem(Appliance givenAppliance)

A setter and a getter for the appliance variable. A hasItem shouldn't be necessary in this implementation.

**public** **void** paint (Graphics pane)

Calls the paint method of the appliance held in the node, if there is one.

**public** **class** Blender

**extends** Appliance

An example of a child of the appliance class. All 5 appliances will follow the same form.

**public** Blender()

The constructor. Set the value of the five important variables: amps, bigPlug, rentalCost, hourlyIncome, and reductionFactor.

**public** **void** paint (Graphics pane)

Draw a little picture of the given appliance, preferably with a width less than 50 pixels. In this method, drawPlug should also be called.

**public** **class** Abutton

**public** **class** UneFenetre **extends** WindowAdapter

I'm just wholesale copying these classes from the other assignments.