LiteBrite

Help Slides released: Friday, September 21
Early Handin: Tuesday, September 24 at 11:59pm
Regular Handin: Thursday, September 26 at 11:59pm
Late Handin: Saturday, September 28 at 10:00pm

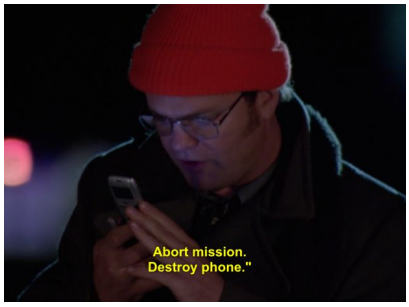
To run demo: `cs0150_runDemo LiteBrite`
To install: `cs0150_install LiteBrite`
To handin: `cs0150_handin LiteBrite`

Silly Premise

Oh brother. Due to recent suspicious activity on Schrute Farms Bed and Breakfast's computers, Dwight is now convinced that he has been hacked. Nowhere is safe. Using phones and computers is out of the question. Talking is obviously a recipe for disaster as microphones could be anywhere. Writing notes can only be done under a blanket so as to avoid the cameras, but no one is willing to go under Dwight's blanket... with him. Dwight realizes he has no need for communication anyways, but then he remembers he is a volunteer sheriff! As a volunteer sheriff he has a duty to serve the people. In this case, he must tell his only employee, his cousin Mose, that he should be on high alert. He has decided the only way to do this is to create a communication device that will be too bright to be picked up by the cameras. He calls it a LiteBrite, but soon realizes he doesn't have the coding skills he needs. Your job is to help Dwight create a LiteBrite to save the business (and Mose)!

Andy said, "Let there be LiteBrite," and there was LiteBrite.

JavaDocs 1:3



Collaboration Policy Reminder

From the collaboration policy:

Collaboration on project design is allowed on mini-assignments, excluding pseudocode. Otherwise, **no collaboration is allowed on project-specific details.** You may not discuss implementation or debugging of code for projects with anyone except the course staff.

Importantly, though, you may absolutely discuss general (i.e., not most assignment-specific concepts) CS15 concepts with anyone, including other current students. The following falls into this category:

- Non-coding questions that are explicitly asked on the mini-assignment
- Going over CS15 lecture slides, our (non-assignment) handouts, Javadocs, etc.
- Discussing object-oriented programming concepts, such as polymorphism
- General syntax questions. For example, “How do I declare an instance variable?”
- How to work remotely, and how to move and hand in files

Note that, when discussing topics that aren’t on the mini-assignment, any examples used must be from the lectures or your own creativity – you may not freely discuss how even broad design concepts, like containment, pertain to a specific assignment if they are not asked on the mini-assignment.

New Concepts Covered

- Parameters
- Containment and association
- Mutator Methods (“`set`” methods)
- Accessor Methods (“`get`” methods)
- Using `return`
- Handling user input (mouse clicks)
- Using built-in classes (javafx.scene.paint.Color)

Assignment Specifications

Create your own computerized LiteBrite. When the user clicks on the grid, colored pegs should be added at the proper location. There should be a color palette with at least two color choices that are selected using `ColorButton`s. The color palette should have a current color specified by whichever `ColorButton` was clicked last. When a lite peg is added to the grid, it should be the color palette’s current color.

Stencil Code vs. Support Code

We provide you with both stencil code and support code. “Skeleton” or “stencil” code refers to the classes that are the shell of the program. **You should never delete code that was given to you as stencil code, it is there to help you!** You will be adding in your own code to the stencil code we provide you with. On the other hand, support code is code that you will be able to use, but will not be able to see or add to. The Javadocs will describe what the support code does, and how to utilize it.

Stencil Classes

Below is a list of the stencil classes provided for this assignment. Your job is to fill in the rest of the provided `LiteBox` and `ColorPalette` classes, along with any class(es) you create, to build your program.

Name:

`App`

Purpose:

This class models an application. When you write your program you should fill in this class so that it contains your top-level object. A top-level object is a *singular* object that contains the rest of your program. *The App class should always **only** instantiate the top-level object, and should not instantiate any other objects.* When you install the LiteBrite project, the App class will already be in your LiteBrite directory, though you will need to edit it to run your program.

Methods:

`start(Stage stage)`

Starts the application. This is equivalent to a constructor for the App class, but is a special method used to start a graphical application. You should instantiate your top-level class here as if it were the App constructor, and *you do not need to use the Stage parameter.*

Name:

`LiteBox`

Purpose:

This class models a lite box that can detect when a mouse has been clicked on top of it. It passes a `cs015.prj.LiteBriteSupport.LitePosition` as a parameter to the `insertLitePeg` method.

Methods:

`LiteBox(ColorPalette palette)`

Constructs the grid (the lite box) with a reference to the instance of class `ColorPalette` indicated by the parameter `palette`. Note: Your top-level class should contain this and the `ColorPalette`, but it is not written for you; *you need to write this class yourself.*

`void insertLitePeg(cs015.prj.LiteBriteSupport.LitePosition position)`

This method is called automatically when the mouse is clicked inside the grid. *You do not ever need to call* `insertLitePeg()`. If you want your grid to respond to a mouse click, you need to fill in this method.

Name:

`ColorPalette`

Purpose:

This class models a color palette that can have `cs015.prj.LiteBriteSupport.ColorButton`s added to it by instantiating them in the constructor. You should add at least two, but no more than ten `ColorButton`s to the `ColorPalette`.

Methods:

`ColorPalette()`

Constructs an empty palette.

`void setColor(javafx.scene.paint.Color newColor)`

This method is called automatically when a `ColorButton` is clicked. *You never need to call* `ColorPalette`*'s* `setColor()` *method, just be sure to select a default color.*

However, to make your `ColorPalette` respond to mouse clicks, you need to fill in this method.

Support Code Classes

The javadocs have a list of the support code classes and what they do. We provide you with a description of their constructors and the methods you can call on them. See the Support Code handout online for more information about what support code is and how it interacts with your stencil code. To find out information about the support code we provide you, check out the CS15 Javadocs. In order to use support code, you must import the java package using the `import` keyword. This has already been done for you in the stencil code given, and as a result you can simply use `LitePosition` rather than typing out the full `cs015.prj.LiteBriteSupport.LitePosition` every time.

How to Get Started

Planning the Design

1. Your first job is to decide what object(s) you are going to need in this program.
    a. First look at the demo of the program and try to describe the objects that you see.

    b. Next, look over the list of predefined objects (see the *Support Classes* section), and decide how and where you want to use each one. You must also decide what the top-level object (which you will instantiate in your App class) might be.

2. Think about where in your code you will need to create instances of each of the object classes. Then, think about where in your code you will need to alter the properties of any of the object instances you have created.

Implementing the Design

1. Log in, run `cs0150_install LiteBrite`, `cd` into the LiteBrite directory, You can open all of the files in the LiteBrite directory by running `atom *` from your `~/course/cs015/LiteBrite` directory — the `*` means “all files” in the current directory.

2. When you run the program an empty frame should appear. We strongly suggest that after you have an empty frame, you work on getting the lite box and the color palette to show up and then doing the rest incrementally. Think about which classes use other

classes to operate. For example, a car uses an engine, so the engine must be created first. Use this to guide the order in which you fill in the classes.

3. You should continue to add small parts to your program, making sure that they work as you expect them to. This idea of writing your program incrementally will be very important as your programs get larger and harder to debug, so getting into a good habit now will save you a great deal of time in the future.

3. Java also has its own form of built-in support code called "packages." The package `javafx.scene.paint.Color` contains constants that represent color values (see a full list here) and more are available for your use (you'll see examples of these being used in the "Working With Objects" lecture): `BLACK`, `BLUE`, `RED`, `CHARTREUSE`, `PERU`, `MEDIUMAQUAMARINE`, `LEMONCHIFFON`, `BLANCHEDALMOND`, `OLIVEDRAB`, `PAPAYAWHIP`, `DODGERBLUE`. In order to use any of these colors you should call the package, just as we call the support code, by calling `javafx.scene.paint.Color.BLACK` or just `Color.BLACK`

4. Remember to refer to the CS15 Style Guide for tips on commenting, especially under the "Internal Documentation" section. Note that style will be factored into your grade!

5. Also remember to create a README (follow the guidelines in the *README* section)

Running your code

To run the program you must first `cd` (change directory) into `course/cs015/LiteBrite` directory, then compile as you normally would by running `javac *.java` in your shell. Run your program by typing `java LiteBrite.App`.

README

In CS15, you're required to hand in a README file (must be named README) that documents any notable design choices or known bugs in your program. Remember that clear, detailed, and concise READMEs make your TAs happier when it counts (right before grading your project).

You should also **include** the following information (in 1-2 sentences) in your README:

1. Which two classes are associated?
2. Name one getter method and one setter method used in your program.

You are expected to create your own README file. Please refer to the README guide for information on how to create a README, what information your README should contain, and how you should format it.

Handing in your code

In order to hand in your code, run the handin script at the top of this handout. The script will list all of the files you are about to hand in, `App.java`, `LiteBox.java`, `ColorPalette.java`, and any other classes you create and will prompt you to confirm. Once you've confirmed, you will receive an email stating that the handin was successful.

Note: The email is your receipt or proof that you've handed in the assignment successfully, so please do not delete it! *If you did not receive an email, we did not receive your handin*. You can run this script as many times as you would like; however, once you run the script, all past handins for this assignment are overridden. This means that if you run the script after the deadline, the project will be marked late, even if you handed in an earlier version on time. We will *not* accept emailed submissions or handins after the late deadline.

Remember that the TAs are here to help you with the assignment, the programming environment, or any concepts that you are not clear about. TA hours are posted on the website, and you can find them on hours in the Fishbowl (271) on the second floor of the CIT.

Minimum Functionality Requirements

MF Policy Summary: *In order to pass CS15, you will have to meet minimum functionality requirements for all projects. If you don't meet them the first time around, you may hand the project in again until you succeed, but you will keep your original grade. MF requirements are* ***not*** *the same as the requirements for full credit on the project. You should attempt the full requirements on every project to keep pace with the course material. An 'A' project would meet all of the requirements on the handout and have good design and code style.*

To meet MF for LiteBrite:

- The LiteBox and ColorPalette appear on screen.
- LitePegs appear in the proper location on mouse clicks.
- The color of each new LitePeg matches the color in the ColorPalette.