

# LiteBrite

**Design Questions:** 5:00pm September 26th

**Help Session:** 5:30pm September 26<sup>h</sup>, MacMillan 117

**Early Due Date:** 11:59pm September 29<sup>th</sup>

**Regular Due Date:** 11:59pm October 1<sup>st</sup>

**Late Due Date:** 10:00pm October 3<sup>rd</sup>

To run the demo, type in a shell:

```
cs015_runDemo LiteBrite
```

To begin coding, type in a shell:

```
cs015_install LiteBrite
```

## Silly Premise

It may seem unbelievable, and it probably should, but New York's power grid has gone out again! Elevators have stopped, air conditioning is a distant memory, and no computer can even boot – Linux, Windows, or even Macs! And they always work! The city engineers have tried their darndest, but nothing seems to be able to fix the problem. They have just one last thing they can try before protocol says to just issue an ultimatum declaring the year in New York to be 1850, and to leave it at that. What's this final trick they have up their sleeves, you ask? They're going to try redesigning the interface to control the power grid, and they need YOU to do it! They need something simple and straightforward to use... Something where you could just press a button and lights would appear... Something that harks back to our childhoods... And just for kicks, let's say it can light up the city with different colored lights, too. Now THAT would be cool (just don't tell the engineers.) What type of interface fulfills all these requirements you ask? LiteBrite, that's what!

You all remember LiteBrite as a kid: that game with the different colored pegs that was a lot of fun until your little sister ate all the pegs? Well, now you get to make one! Put an end to all the foolishness that New York's been having with its power. Give the engineers an interface they can understand. Let the citizens of New York have what they've always wanted – different colored street lights. Go forth and code!

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

Javadocs 1:3

## New Concepts Covered

- Parameters
- Accessor Methods ("get" methods)
- Return Types (using `return`)
- Local variables

- Handling user input
- Code reuse

## 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 palette with at least two color choices that are selected using “lite buttons”. The palette should have a current color specified by whichever “lite button” was clicked last. When a peg is added to the grid it should be the palette’s current color.

We are providing you with a partially written (“skeleton”) `Grid` and `Palette` class to model the grid and palette, as well as completely written `cs015.prj.LiteBrite.Lite`, `cs015.prj.LiteBrite.LiteButton`, and `cs015.prj.LiteBrite.LiteColor` support classes for the light pegs, the palette color buttons, and the actual colors, respectively. Their methods are described in the *Support Classes* section below.

Your job is to fill in the `Grid` and `Palette` class and use the support classes, along with the class(es) you create, to build your program.

## Helpful Hints

*Make sure you complete the LiteBrite design questions before you begin coding the assignment!*

You can see the demo of the program that the TAs have created by typing `cs015_runDemo LiteBrite` in a shell. This will help you get the feel of how your program should work.

Your first job is to decide what object(s) you are going to need in this program. Picking out the nouns from the program specification should help you get started. Also, be sure to look at the demo of the program and try to describe the objects that you see.

When designing the object(s) you want to create, 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 to put in your `App` class.

Next, define the purpose for the empty methods in the `Grid` and `Palette` class skeletons. Think about where in your code you will need to create instances of each of the object classes you have decided to use. Next, think about where in your code you will need to alter the properties of any of object instances you have created.

Finally, when you are confident about your design, log in, “cd” into the LiteBrite directory, and start writing your program. Begin by getting an empty frame to appear. Then 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.

Before you start programming, look over the slides from the first four lectures. Very special (not-so) secret tip: *review the **Parameters** lecture before beginning this assignment.* If you don't understand something that is covered in that lecture, see a TA during TA hours.

## Design Header

In the comment above your `App` class (called the header comment), write a brief description of your design. Note any differences or changes you made to the suggested design.

If there are any bugs that you know of in your program, write a note in the header comment explaining what they are and how you might fix them (if you have any ideas). This makes it easier for a TA to grade, and also shows the TA that you know what is wrong with your program. (Side note: not listing the bugs and hoping the TA won't notice them won't help your grade. The TA will think that you didn't test your program to see if it worked.)

## Handin Info

Remember to complete the Design Questions before starting the LiteBrite program. For instructions see the LiteBrite Design Questions handout.

You will need to "hand in" your program electronically. To hand in LiteBrite, type in a shell:

```
cs015_handin LiteBrite
```

This electronic handin should be completed by 11:59pm on Thursday, October 1<sup>st</sup>, or your program will be considered late. If you hand in your program by 11:59pm on Tuesday, September 29<sup>th</sup> it will be considered early, and you will receive extra credit on this assignment. The late due date for LiteBrite is 10:00pm Saturday, October 3<sup>rd</sup>. See the Standard Operating Procedures page on the website for details on the late policy and the early handin incentive.

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 TA room on the second floor of the CIT. If you see a TA anywhere besides the TA room, they are NOT on hours. Please be considerate of the TAs. They are students just like you and have their own work.

## Skeleton Classes

This is a listing of the classes that you need to fill in for this assignment.

---

**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. When you install the LiteBrite project, this class will already be in your LiteBrite directory, and you will only need to add code to it to run your program.

**Methods:**

App()

Constructs the application.

---

**Name:**

Grid

**Purpose:**

This class models a grid that can detect when a mouse has been clicked on top of it. It passes a `cs015.prj.LiteBrite.GridPosition` to the `insertLite` method.

**Methods:**

Grid(Palette p)

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

`void insertLite(cs015.prj.LiteBrite.GridPosition p)`

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

---

**Name:**

Palette

**Purpose:**

This class models a palette that can have `cs015.prj.LiteBrite.LiteButtons` added to it by instantiating them in the constructor. You can add as many or as few `LiteButtons` as you want!

**Methods:**

Palette()

Constructs an empty palette.

`void setColor(cs015.prj.LiteBrite.LiteColor c)`

This method is called automatically when a `LiteButton` is clicked. *You do not ever need to call* `setColor()`. However, if you want your `Palette` to respond to mouse clicks, you need to fill in this method.

---

## Support Classes

This is a listing of the predefined classes in package `cs015.prj.LiteBrite` that you will find helpful for this assignment.

---

**Name:**

`cs015.prj.LiteBrite.LiteButton`

**Purpose:**

This class models a color button on the `Palette` which controls the color of the light pegs.

**Methods:**

`cs015.prj.LiteBrite.LiteButton(Palette p, cs015.prj.LiteBrite.LiteColor myColor)`

Constructs a light button with a reference to your `Palette`.

---

**Name:**

`cs015.prj.LiteBrite.Lite`

**Purpose:**

This class models a light peg that will add itself to the `Grid` when constructed. Don't forget to set its color and position.

**Methods:**

`cs015.prj.LiteBrite.Lite()`

Constructs the light peg.

`void setPosition(cs015.prj.LiteBrite.GridPosition p)`

This method moves the `Lite` to the proper position on the `Grid`.

`void setColor(cs015.prj.LiteBrite.LiteColor newColor)`

This method changes the color of the light peg to the current color stored in `newColor`. You're correctly using the `Palette` when new `Lites` appear with the color of the last color clicked. Make sure to run the demo (by typing `cs015_runDemo LiteBrite` in a shell) to get an idea of what this means.

---

**Name:**

`cs015.prj.LiteBrite.LiteColor`

**Purpose:**

This class models the actual color you want to make your `Lite` pegs and `LiteButtons`. You know, like, red. Or green!

**Methods:**

`cs015.prj.LiteBrite.LiteColor(BLUE)`

Constructs a new color of the constant you send in. The following color constants are supported:

BLUE WHITE RED GREEN BLACK GRAY ORANGE YELLOW MAGENTA  
PINK CYAN