CS 051G Homework Laboratory #2
Dirty Laundry

Objective: To gain experience using conditionals.

—

The Scenario. One thing many students have to figure out for the first time when they come to college is how to wash their clothes. Often by the time most students have figured it out, all their light-colored items are streaked with all sorts of interesting colors. In the hopes of helping next year's first-year students adjust more easily to college, we would like you to write a *laundry sorting simulator*.

The Approach. It is usually a good practice to develop programs incrementally. Start with a simplified version of the full problem. Plan, implement and test a program for this simplified version. Then, add more functionality until you have solved the full problem.

To encourage this approach, we have divided this lab into two parts. For the first, we will describe a laundry sorter with a very simple interface. You should plan, implement and test a program for this problem first. Then, in the second part of the assignment we will ask you to add additional functionality.

You are also encouraged to approach each of our two parts in this step-wise fashion. For example, in the first part you might begin by just writing the instructions to construct the necessary graphical objects without worrying about any of the mouse event handling. Once your program can draw the picture, you can move on to figure out how to handle events.

Getting Started. To start, drag the Laundry starter folder from the file server and then upload the resulting project into the Chrome browser.

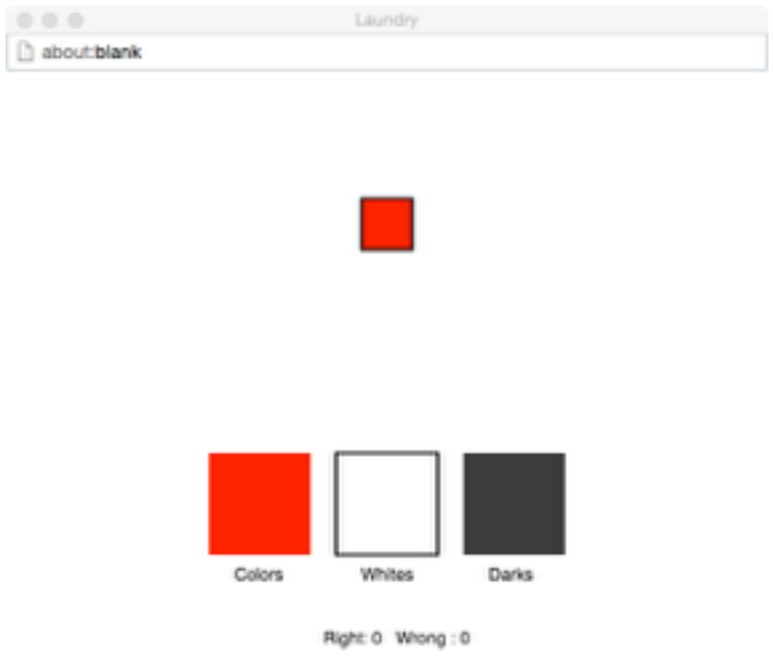
Part 1: The simulator should begin with three wash baskets on the screen (for our purposes they can just be rectangles or squares). One is labeled "whites", one "darks", and the last "colors". An image showing the kind of display we have in mind appears below. When the simulation begins, a color swatch will appear on the screen. The user ("laundry trainee") should then click in the corresponding basket. If the user is correct, the program should randomly select a new color for the next item and display it on the screen. For this part of the assignment you may just select among the three colors `white`, `red`, and `black` when creating items of clothing. If the user clicks on an incorrect basket, the original item remains in position for another try.

Design of Part 1. You will need to design an object that will `inherit graphicApplicationSize (...)` and that will display the wash baskets and the item to be sorted. Begin by laying out where all the items go on some graph paper. The picture should look more or less like the one below.

When the program begins, place all the wash baskets (with labels) on the screen. Then, add the item of clothing that is to be sorted. For simplicity you can make the first item always have color white (or you can be more ambitious and select the color randomly). The item should actually consist of two rectangles, a filled rectangle which is the appropriate color and a framed rectangle which is the same size, but lays on top of the filled rectangle to form a border (otherwise it will be awfully difficult to see a white item!)

Think Constants! When you lay out the wash baskets and item, use definitions to give readable names to all the relevant values. The width and heights of wash baskets and the item to be sorted, coordinates of the upper left corner of each of these, etc., are all good candidates for constants. Examples include:

```
def basketWidth: Number = 80
def laundryLocation: Point = 280 @ 100
```



Using names rather than numeric values in your program will make it easier to change things around and also make your program much, much easier to read (presuming you give them good names). Of course you will also need to use definitions to provide names for key items of your program like the three laundry baskets, the filled and framed rectangles for the laundry item, etc.

Identifying the Correct Basket (*Pay attention to the following as it will make your code much, much easier!*)

Once you have done the layout and figured out how to generate new items, all you have to do is to write the code for the method onMousePress. You will need to associate some information with an instance variable that will enable onMousePress to determine which is the correct basket. A particularly clever and useful way to do this is to use an instance variable – let's call it correctBasket – that will refer to one of the laundry baskets. In particular, you will make sure it always refers to the laundry basket that is the same color as the laundry item.

Whenever you generate a new laundry item, you will associate correctBasket with the rectangle/basket in which an item of its color should be placed. That way when the user presses the mouse on a basket, onMousePress can simply check (using the contains method) to see if the rectangle currently associated with the correctBasket contains the point where the mouse was pressed. Then, onMousePress will either select a new color for the item (if the user was correct) or wait until the user presses the mouse again (if incorrect).

A Warning! One odd feature of the simple interface that may bother you a bit is a result of the fact that the program selects laundry items randomly. Because the selection is truly random it sometimes picks the same color twice in a row. When this happens and you press the mouse on the correct basket for the first item you will get the feeling that the program ignored you. Even though it has actually displayed a new item, the new item looks just like the old one, so you may think nothing changed. Don't let this trick you into thinking that your version of the program (or ours) isn't working correctly. The more advanced interface in part 2 includes counters in the display that eliminate this problem.

Generating random numbers You will be using the grace library, random, for this project. To use it you must write something like:

```
import "random" as randomGenerator
```

right after the **dialect** statement. You can replace the identifier randomGenerator by any other legal Grace identifier that you like.

Now that you have access to the library, you will want to use the method integerIn (low) to (high) where low and high are the smallest and largest integers you want to be generated. For example, to generate a random number between 1 and 3 (inclusive), just write randomGenerator.integerIn (1) to (3) (assuming you used the name randomGenerator as the identifier to refer to the library).

If you wish, you can now use a conditional statement to associate each of the numbers from 1 to 3 with a different color. For example, if n is the random value generated, make the color of the item be white if n is 1, black if n is 2, and red if n is 3.

Design document If you go to the starter for this lab, you will find that it already contains a Laundry.grace file with comments and skeletons for all of the methods you will be implementing in this lab. To help guide you through the process, you will find // TODO DESIGN comments in all of the places where you have to figure out what variables to declare and what code you will write. (We've also annotated with // TODO DESIGN-2 the places where you might add design information for part 2 – not required, but recommended.)

The things you need to prepare before coming to the lab are described in the Lab Design Form at http://www.cs.pomona.edu/classes/cs051G/labs/laundry/Planning.docx, which is a Microsoft Word file.

1. down-load and read it.
2. work out the problems and fill in the required information.
3. print it out, and bring it to the lab with you.
4. the TA's will review it and give you feed-back.

The design is an important step in the coding process. You are sketching out the code to be written before you write it. This is like writing an outline before writing a paper. I promise you that thinking about the design before you start coding will make your life easier in the long run.

Try to convince yourself that your design is correct before you come to lab, as that will make your lab work much simpler. In particular, the trickiest part of the lab is using our hint above to make it easy to determine whether the user clicked in the correct basket, so look at this very carefully.

Writing and testing your Grace program Once you are in the lab and ready to code in Grace, copy Lab2-Laundry from the cs051G folder on your desktop to the CS51GWorkspace folder, also on your desktop. Open Chrome to www.cs.pomona.edu/~kim/minigrace and upload the file `laundry.grace` to get it showing in the edit window.

You can go ahead and click on "Run", but it will only bring up a blank window. Close it and go back to the edit window to work on your program. Be sure to write it in small pieces, and test it frequently by building it and running it. For example, you might write the code to draw the laundry item and then make sure it shows up where you want it. Do the same with the three laundry baskets and their labels, etc.

Once you have part 1 working correctly, make sure it is commented properly. In particular, the comments inside the method body should have been replaced by actual Grace code, but the comments before the method header should remain, and you should have appropriate comments for each definition and variable declaration.

In general, we don't put many – or any – comments inside the method body. Instead we place comments before the method declaration saying what – not how – the method does.

At this point, get one of your lab instructors or TAs to check it out to make sure it works properly and to give you feedback on your code. You may then proceed on to part 2. (You won't be turning in part 1, but you will need to use it as a starting point for part 2.)

Part 2: Once you get the basic version working, we would like you to jazz it up a bit. Here are the extensions we would like you to make:

1. Add labels (text items) at the bottom of the picture showing the number of correct and incorrect placements. This makes it clearer when the student succeeds in placing the item correctly. They should read something like "correct = nn", "incorrect = mm". The value in the first text item will be formed by using {...} to interpolate an instance variable which keeps track of the number of correct answers into the string being displayed. The other is similar.

2. Users should drag the items to the correct laundry basket rather than just clicking on the basket. Recall from the examples in class that you will need an instance variable to label the last mouse position before the drag so you can determine how far to drag the item. If the user presses the mouse button down outside the laundry item you should not drag the item, and it should not increase the correct or the incorrect counter. Also if a user only clicks in the correct basket (without dragging the laundry item), then that should NOT count as a correct answer.

3. Assign the item a randomly generated color by randomly choosing a color. You can do this either by using the method `color.randomColor` or randomly choose integers `redNum`, `greenNum`, and `blueNum` between 0 and 255 for each of the red, blue, and green components of the color. You can then create a color from those components by writing `color.r(redNum)g (greenNum)b (blueNum)`.

   Now define criteria for determining where each color should go. A simple criterion is based on the sum of the three color components. If the sum of the component numbers is less than 230, decide it is dark, if it is greater than 600, decide it is white. Otherwise it is colored. (Remember that there are parameterless methods `red`, `green`, and `blue` that return the number component of each of those colors. After you get the program working you might want to experiment with other criteria.

We will let you figure out most of the details of how to add the features for the more advanced versions. One piece of advice is that for the second enhancement, the code to determine whether the user got the laundry into the right basket will need to be in the `onMouseRelease` method, rather than the onMousePress method as it was in part 1. As a result, you might start work on this part by moving the code originally (i.e., for part 1) in onMousePress to onMouseRelease. Now the onMousePress method will merely need to record whether or not the mouse was pressed on the laundry and to record where the mouse was pressed. The purpose of the three methods that respond to mouse events are:

- onMousePress – for when the user first clicks on the item - though remember that they might miss.
- onMouseDrag – to do the actual dragging (assuming they clicked on it originally).
- onMouseRelease – if they were dragging it, check to see if they've dropped it in the right place when the mouse is released. Update the laundry item appropriately

While we do not require you to bring a design document for this part of the program, you will find it extremely helpful to make some notes and sketch this out in advance, realizing that your work in part 1 will help you better understand how to do part 2.

Grading Guidelines Your program will be graded using the point values in the table below. Please look these over carefully to make sure you don't lose any points (or conversely if you want to gain extra credit).

Table 1: Grading Guidelines

Value	Feature
	Design (1 pt total)
1/3 pt.	Variables and definitions
1/3 pt.	Pseudo-code in onMousePress
1/3 pt.	Test plan
	Style, Design, and Efficiency (9 pts total)
2 pts.	Descriptive comments (but not over-commenting)
2 pts.	Good identifier names
2 pts.	Good use of definitions
2 pts.	Appropriate formatting
1 pt.	Does not generate new objects unnecessarily
	Correctness (10 pts total)
2 pts.	Generates a new color for swatch only if previous one placed correctly
2 pts.	Swatch displayed in the correct initial position; returns to original location if incorrectly sorted
2 pts.	Updates number correct and incorrect properly
2 pts.	Drags swatch properly (lose 1 pt if can just press to increase score instead of dragging)
2 pts.	Appropriate behavior if user does unexpected things like starting to drag outside the swatch
	Extra credit (1 pt)
1 pt.	Does not update either # correct or # incorrect if user misses all baskets

Final remarks Be sure to do the basic version of the lab before attempting the more advanced features. Just work on adding one feature at a time, testing each thoroughly before working on the next.

Before turning it in, make sure you have tested it thoroughly. Try it by dropping the item in the correct basket as well as in the incorrect basket. Try pressing the mouse outside of the item ("missing" it) and then try dragging and releasing over a basket.

Turning in your program Your program is due at 11p.m. on Tuesday evening. Turn in your program as you did last week. Here are the instructions again:

- First, make sure you included your name and course number in a comment at the start of the program.
- Right click on "download". Select "Save link as". Navigate to the folder called "Lab2-Laundry" in your CS51GWorkspace folder. You should now rename that folder by clicking on it and pressing return, and typing in a more descriptive name in the same format as last time. I.e. "Lab2-BruceKim" – except replace "BruceKim" by your name! Note that dashes in names are OK, but don't include spaces or periods.
- Now open the "cs051G" folder icon on the desktop by double-clicking on it. Within the "cs051G" folder you should see a "dropbox" folder.
- Drag the folder you just created into the dropbox folder. When you do this, the computer may warn you that you will not be able to look at this folder. That is fine. Just click "OK".

If you should accidentally turn in a bad version of your program, you may drag another copy in as long as you change the name to be slightly different from the one you used before (e.g. Lab2-BruceKimRev1). The new name should make it clear which is the newer version.

Good luck and have fun!

Starter code This is a copy of the starter code for laundry. You can use this to help you write your design. You can write your design by hand or on the computer. However, you should be aware that copying and pasting from another electronic form (pdf, word, etc) to the Grace development environment invariably causes serious problems by inserting spurious characters that makes your Grace program choke.

I ask you NOT to start writing actual Grace code until the beginning of your lab. You will actually lose points in the grading of your design if you include Grace code instead of an English description of what is to be done. (See the basketball design example after this to see what we are looking for!)

```
// Include your name, lab section and a description of the program here

dialect "rtobjectdraw"

def laundryProgram: GraphicApplication = object {
    inherits graphicApplicationSize (600 @ 600)

    windowTitle := "Laundry"

    // Place all definitions and variable declarations here as well as initialization code.

    // Add comments saying what this method does
    method onMousePress(mousePoint: Point) -> Done {


    }

    // Add comments saying what this method does
    method onMouseRelease(mousePoint: Point) -> Done {


    }

    // Add comments saying what this method does
    method onMouseDrag(mousePoint: Point) -> Done {

    }

    // required to show window & start graphics system going
    startGraphics
}
```

Example Design for Basketball We wrote the pseudo-code in here as comments so we could print it out nicely. You don't need to do this if you don't want to. Note the comments for methods below. The lines before the method header will stay there after you finish the program as they describe *what* the method does. The comments after the method headers will be replaced by Grace code in the final program. They generally describe *how* the method works. Your final code should be readable enough that we can see how it works. However, the comments above the header about what it does is very helpful in trying to understand what each piece of your program is responsible for (i.e., it is higher-level and doesn't get bogged down in details).

```
// Basketball program (not an actual lab)
// Drag basketball into hoop
// Author: Kim Bruce, CS 51G, Lab section 47
//

dialect "rtobjectdraw"

def Basketball : GraphicApplication = object {
    inherit graphicApplicationSize (400 @ 400)

    // Location and dimensions of hoop
    def hoopWidth: Number = 100
    def hoopHeight: Number = 60

    // starting position of basketball and its diameter
    def startLocn : Point = 190 @ 300
    def ballSize : Number = 40

    // points scored
    var score : Number := 0

    // location mouse at most recently
    var lastMouse: Point

    // Text showing current score
    def display = textAt (150 @ 200) with "Your score is 0" on (canvas)
    // set fontsize to 16

    // basketball hoop is framed oval at (160,50) with size hoopWidth by hoopHeight
    // Style: Don't need to include details of location, etc. in both comments and def
    // statement, but do include it one place or the other
    def hoop = framedOvalAt (160 @ 50) size (hoopWidth @ hoopHeight) on (canvas)

    // the basketball is represented by an orange filled oval
    def ball : Graphic2D = filledOvalAt (startLocn) size (ballSize @ ballSize) on (canvas)
    // make the ball orange

    // Save where mouse is depressed
    method onMousePress (point: Point) -> Done {
        // remember where the mouse was pressed in lastMouse
    }

    // When mouse is dragged, move the ball with it
    method onMouseDrag (point: Point) -> Done {
        // if mouse was over ball before last drag then move the
        // ball by the distance between the old and new mouse
        // positions.
        // Remember the current position in lastMouse in preparation
        // for the next drag.
    }

    // increment the score and update the text if player scores,
    // i.e., if the mouse is over the hoop when it is released
    method onMouseRelease (point: Point) -> Done {
        // if the ball was being dragged and the hoop contains point
        // then update the score and the display showing the score.
        // Move the ball back to its starting position.
    }

    // required to pop up window and start graphics
    startGraphics
}
```