IBSL Computer Science 1-2

Problem Set #12: Barcodes!

CONGRATULATIONS!!!

YOU HAVE COME A LONG WAY IN PROGRAMMING. Believe it or not, you now have the ability to create just about any program you desire. Those of you that have learned to look through the ORACLE method pages have really packed away a lot of info. Those of you that have simply gone on line and "grabbed code" have sort of cheated yourself but I still think you've learned a fair amount. This assignment represents a culmination. It has some very specific requirements and those of you that know how to use methods will be able to complete this easily. Those who are still writing monolithic code will have a tough time.

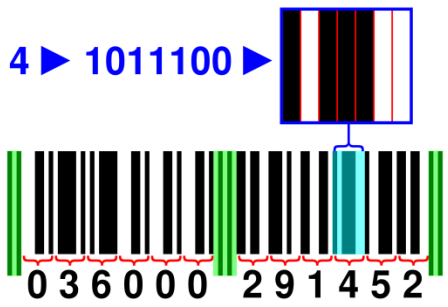
In short, you are going to write this program and then I will change the requirements **after** you are done. If you have compartmentalized the code, the modifications will take less than a period. If you haven't, well …..

Why? Well, this is real life! That's what happens in the industry all the time. You make something for the customer and then find out she wanted steak and not chicken. … again, that's life.

First off you are going to do a quick little activity on Barcodes, before you delve into this homework assignment…actually do it, if you skip it you will ask me silly questions like "how do barcodes work" and I will just tell you to actually read the assignment! So take a look at this:

Make your own Barcodes!

We see barcodes almost everywhere we look these days. Their purpose is simple, to provide an easy way to scan in a string of numbers, without having to bother to type them in. There are many varieties of barcodes (it is really just a general term to describe a black and white pattern that represents data), but the most common one that we see is the Universal Product Code (UPC)…this is the code that is on almost everything we buy in the super market. In this worksheet we'll use the codes for the 12 digit UPC, commonly called the UPC-A.

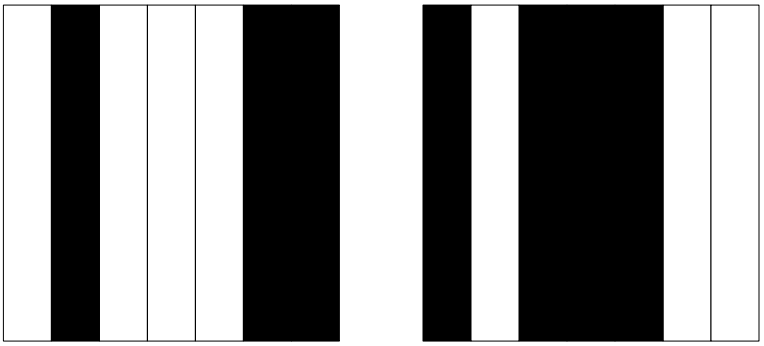


Each number is actually represented by a grouping of 7 bars. They touch, so 3 bars together looks like one fat bar. The shaded area on the ends and middle are called "guard bars," they help the optical scanner detect the actual numbers. There are two sets of codes: one for the left hand numbers (the first 6 digits) and one for the right hand numbers (the last 6 digits). This is done to make sure there a clear distinction between the

company and product identification codes. The table below indicates how the patterns are drawn (0 means leave the bar white, 1 means color the bar black)

Left Hand Codes		Right Hand Codes	
0	0001101	0	1110010
1	0011001	1	1100110
2	0010011	2	1101100
3	0111101	3	1000010
4	0100011	4	1011100
5	0110001	5	1001110
6	0101111	6	1010000
7	0111011	7	1000100
8	0110111	8	1001000
9	0001011	9	1110100

Example: Writing “4” using the left and right hand codes



Use the table above to fill out a barcode for the number: 056321 079805.

Every barcode comprises 12 digits. There is a mathematical **rule** that determines whether the barcode is valid. (NOTE: The same is true for credit cards and ISBN numbers etc. They are used to make sure the number has been entered correctly and helps reduce errors in transmission). The rule for UPC-A barcodes is:

Assume your number comprises 11 digits. The 12^{th} digit will be chosen to make the following expression valid. We will call the left most digit x_1 and the right most digit x_{12}. Now, choose the right most digit so that:

$$(3x_1 + x_2 + 3x_3 + x_4 + 3x_5 + x_6 + 3x_7 + x_8 + 3x_9 + 3x_{10} + 3x_{11} + \mathbf{x_{12}}) \% 10 == 0$$

In other words, the manufacturer chooses the first left-most 11 digits, then the 12^{th} digit (the rightmost) is chosen to make the expression above equal to 0 when divided by 10. Note that we are multiplying every other digit by 3 and summing. Why is this done? It is done because whenever anything is transmitted electronically there is a chance of error due to static in the line. This is a fundamental problem and is covered in chaos and fractal theory but is inherent in data transmission (a fascinating subject!). So, the right-most digit is called a checksum. Try this out by looking at some UPC labels and seeing if this is true. Again, credit cards and ISBN numbers do the same thing with different variations on the multipliers or the number to "mod" by.

Your assignment:

Write a program that presents to the user a window with a pull-down menu. This menu must have the following options:

Enter an 11-digit UPC code

Enter full UPC code

Change color

Save UPC image

Enter a *** code**

Explanations for each:

Enter an 11-digit UPC code

The user enters an 11 digit upc code into a text field. You take that 11 digit code and by using the expression above for a valid 12 digit code, figure out the missing 12th digit and replace the text-field contents with a valid upc code.

Enter full UPC code

Take in a 12-digit upc code. Check for validity. If it is invalid, use an **alert** window to report that it is not a valid code. If it is valid, pop-up a VBox or an HBox window with the UPC barcode. Note in the above example the guard bars are slightly longer than the regular number bars. Yours should be also.

Change color

If the user wants to change colors, you should provide them with the option of changing the black pattern to one of the following colors: Green, Blue, Red. If this happens, you will redraw the image so that it displays color.

Save UPC image

If chosen, you should bring up a textbox that allows the user to give a filename. There is a way to convert a graphic in Java to a .jpg or .pdf. Figure it out. Save the image to filename specified.

Enter a *** code**

This remains to be figured out. I'm going to give you a different kind of code. If everything you have done is modular with methods etc. you should be able to quickly modify your code. I may give you ISBN or some credit card or some such with a set of rules. It will be an alternating pattern using the same number representation (for 0's through 9's) specified above.

Once your program is up, the cmd window (console) should have zero activity … no error messages or anything.

Design decisions: You may either use a whole bunch of rectangle objects (in the Java sense) and just lay them out next to each other, or you may use a "canvas" and literally draw the rectangles and fill them with black (or red, or green etc.). Either way works. When I did it I chose the second option.

Remember that all barcode digits are a pattern of 7 bars (white or black). I would suggest doing this in two parts: First take a minute to find pattern/patterns in how the bars are generated (hint: Left vs. Right is a good place to start)…the more general your description of the bars the less you'll have to hardcode. Second, make sure you build a function (method) that lets you

specify where the bars are drawn. Third, if you haven't taken a look at the switch() statement yet…now's the time!

A couple of tips: remember that the first 6 digits will use the left hand code and the second 6 will use the right hand code and use your number splitter function from a while back to help you out. This task has a lot of moving parts…a strong plan/pseudocode will help you immensely!

I will give you one very big hint. Set up the basic pattern for a given number in an array. Every digit is an alternating pattern of white (0) and black (1). Just to be a very nice person, here was my fundamental definition of the digits.

```
int [][]leftcodes = {
        {0,0,0,1,1,0,1},   // 0
        {0,0,1,1,0,0,1},   // 1
        {0,0,1,0,0,1,1},   // 2
        {0,1,1,1,1,0,1},   // 3
        {0,1,0,0,0,1,1},   // 4
        {0,1,1,0,0,0,1},   // 5
        {0,1,0,1,1,1,1},   // 6
        {0,1,1,1,0,1,1},   // 7
        {0,1,1,0,1,1,1},   // 8
        {0,0,0,1,0,1,1}    // 9
 };

//Here are the arrays for the end and midbars to the pattern … I mean the guard bars here.
 int [] endbar = {1,0,1};
 int [] midbar = {0,1,0,1,0};
```