RECITATION 5

COMPOSITE DATA TYPES: ARRAYS AND CLASSES

For many interesting programs, we don’t just want to use variables for a single data type, like int or boolean. We want to deal with lots of variables all at once. This week, we’ll talk about two different ways of bundling your data up together in programs. Programmer-defined data types generally make the programmer responsible for reserving the memory for each object, using the **new** command.

ARRAYS

You’ve seen one array type already: remember that Strings are one-dimensional arrays of characters. Java represents the String “Frodo Baggins!” like this.

F	r	o	d	o		B	a	g	g	i	n	S	!

You can access bits of the string by giving their position in the array. In computer science, you generally start counting array positions at 0. Suppose we set

```
String name = “Frodo Baggins!”;
```

in the Java code. Then name[0] = ‘F’, the first character in the array. Another nice thing you can do is slice out substrings; for example, name.substring(0, 5) is “Frodo”. If you’re reading carefully, you should be thinking that something is wrong here, because name[5] is really the position of the space; but you always specify the slice from start to end + 1, so the substring doesn’t actually wind up having the space. The most common mistakes with arrays involve trying to get an element before the start or after the end of the array, like

```
char error1 = name[-1];
```

or

```
char error2 = name[20];
```

Both of these assignments attempt to look outside the bounds of the array for data, which is generally a bad idea, since we have no idea what’s there. These mistakes will usually generate run-time errors.

You can make arrays of anything, including integers, characters, doubles, and other arrays. Here’s an array declaration that works to draw Polygons in a Java applet; the first point is at x[0], y[0], the second at x[1], y[1], etc. Java’s drawPolygon methods sort out the coordinates for point 1 (x_coords[0], y_coords[0]), through however many points you have asked for.

```
/* define an array of integers called x_coords, and set it up with 4 numbers;
        to close the polygon, you need to include the first point at the end */
int [] x_coords = {0, 50, 50, 0, 0};
int [] y_coords = {0, 0, 50, 50, 0};
page.drawPolygon(xcrds, ycrds, 5);
```

You access individual members of the array by using their subscripts (positions) in the array. The first position is 0, so y_coords[2] would be 50 in the example above.

To do string matching in Java, you pretty much have to use arrays:

```
/* define a function named Match that returns an integer giving the position
        of the first match for char c in String str */
int Match(String str, char c)
{
        /* length of the string we’re searching; we need to be sure that
                we don’t fall off the end */
        int length;

        /* counter for our position in the array */
        int j = 0;

        /* get the length of the String str */
        length = length(str);

        /* check every position in the string, from start to end, until
                there’s a match */
        while (j < length)
        {
                /* compare the letter at position j in the String to the
                        char c we’re looking for
                if (str[j] == c)

                        /* if there’s a match, we can quit and send back
                                our current position */
                        return j;

                /* increment the position counter by 1;
                        this only happens if we haven’t found a match yet */
                j++;
        }

        /* if we have gotten to this point, the character we’re looking
                for isn’t there, so we return a weird value */
        return –1;
}
```

Remember that the most insidious problem with arrays is wandering off their ends by mistake. In the above code, j must be greater than –1 and less than the length of the string. (Satisfy yourself that this really checks every letter in the string without walking off the end.) If the value we return is –1, then str[-1] is not defined! (Neither is str[length].) So if we call this routine from our main program, we'd say:

```
int position;
String name = “Socrates”;
char [] letters = {‘x’, ‘r’, ‘q’, ‘n’, ‘y’, ‘c’, ‘s’};

int letterpos = 0;

/* loop over the letters in the letters array */
while (letterpos < 7)
{
        /* search for current letter in the String name */
        position = Match(name, letters[letterpos]);

        /* if we found a valid match, then we can look it up in the String array*/
        if (position > -1)
                System.out.println(name[position] + “ found at “ + position);

        /* but if the position is –1, then the character doesn’t match */
        else
                System.out.println(“No match found for “ + ch1 + “ in “ + name);

        /* increment the position counter to check for the next letter */
        letterpos++;
}
```

Exercise: what if you wanted to search an array of strings for the characters in letters?

II. CLASSES

Often, arrays are a bit limited. Every array has to contain the same type of data. It's generally nice for the programmer to be able to define certain particular data types. In Java, you do this by defining classes. For example, you can define a new class for an x, y point by gluing two integers together. This code describes another kind of composite type, defined by the program.

```
public class Point
{
        int x;
        int y;
}
```

We call the variables inside the class definition (x and y, here) the class variables, because they must be accessed through the class.

What might you want to do with Points? You might want to move them, or rotate them, or draw lines between them. The methods that operate on Points also get defined as part of the Point class. This idea of grouping data of a particular type with the methods allowed to work on the data is a key feature of object-oriented programming, because it tends to make code more modular and more portable.

For starters, we'd like to initialize Points. We can do this by writing a constructor for them. The constructor is a special method in your class. It has no return type (basically, it's sending back a new instance of class Point). It must have the exact same name as the class. In our example, it takes 2 parameters, an integer for the x position and another integer for the y position. We'll also include a routine to move points around by a certain x and y distance.

```
public class Point
{
        int x;
        int y;
        Point(int init_x, int init_y)
        {
                x = init_x;
                y = init_y;
        }
        void MovePoint(int dx, int dy)
        {
                x += dx;
                y += dy;
        }
}
```

Now, we can include this class in a project, initialize 2 new points in our main function, and move them by saying:

```
        public static void main (String [] args)
        {
                Point p1 = new Point(1,2);
                Point p2 = new Point(5,9);
                p1.MovePoint(5,3);    //now p1 is at (6,5)
                p2.MovePoint(-3,1);   //now p2 is at (2,10)
        }
```

If you look at this code carefully, you can see that p1 and p2 are just the names of two different Point-type variables. The class Point defines what a Point is and does. p1 and p2 are objects (particular examples) of type Point. The class is the blueprint for a type; it defines what kinds of variables the type holds and what methods can work on these variables. The object is a particular instance of that type, with actual values assigned to its variables.

You've already used classes and objects in the applet homework. Your drawing space is defined using an object called page, of class Graphics. This means that when you use the drawing methods in the Graphics class, you had to call them via the page object, as below.

```
import java.applet.*;
import java.awt.*;
public class Class1 extends Applet
{
        public void paint(Graphics page)
        {
                setBackground(Color.white);

                /* all of these methods are defined in class Graphics and are called
                        via the page object */
                page.drawOval(60,70,100,50);
                page.setColor(Color.blue);
                page.fillOval(60,70,10,50);
                page.setColor(Color.red);
                page.drawRect(0,0,50,50);
                page.drawString("Happy Birthday!");
        }
}
```

Most interesting programming problems can be broken down into logical chunks of data that fall nicely into a class structure. The program then consists of managing the behavior of these objects. For example, if you were writing a game to play checkers, you might start by defining one class for a checker (to keep track of color, x position, and y position), and another for the 8x8 game board, maybe as a 2D array with each cell holding a checker, or nothing. Then you could write methods in the game board class to check the legality of each move, make players take turns, and decide when the game is over.

Next week, we'll cover classes and methods in more detail, and provide you with a programming example to help you appreciate objects at work. The next programming assignment will involve writing 2 or 3 classes, along with the regular main program, to solve some interesting problem, most likely recursively.