CS 141: Introduction to Computer Science
Winter Term, 2013

Lab 8

In this lab, we will practice arrays and loops while writing some useful classes.

Dates

Let's begin by demonstrating a use of arrays that can make it easier to write certain types of code. Open a new project in BlueJ and create a class `Date`. Give it private integer attributes `month`, `day`, and `year`, plus a constructor that takes these three numbers as arguments.

Next, we want a method `toString` that returns a string representation of the date. To make the display nicer, we want the month to appear as its abbreviation rather than just a number. Thus, we want today's date to appear as `Oct 25, 2012`. You could write the method to do this with a sequence of if statements, but it would be rather tedious to write. (Try it if you don't believe me.) Instead of the tedious code, type in the following alternative:

```
public String toString() {
  String[] monthNames = {"Jan", "Feb", "Mar", "Apr", "May", "June",
                         "July", "Aug", "Sept", "Oct", "Nov", "Dec"};
  return monthNames[month-1] + " " + day + ", " + year;
}
```

The method allocates an array in a way we have not seen before, giving the array contents as a list within braces. This simultaneously creates the array and initializes its contents. Once the array of month names has been created, converting the number representing a month into an abbreviation just requires that we look up the appropriate table entry. (Note that we have to subtract one from the month's numeric value since the first month is number 1, but the array is indexed from 0; another solution would be to put a dummy abbreviation in the array before January.)

Now do something similar to create a method `daysInMonth` that returns the number of days in the current month. Initially, ignore leap years so the numbers of days follows the rule "Thirty days hath September, April, June, and November" (The other days have 31, except for February, which has 28 without leap years.)

Once you have the basic version of `daysInMonth` working, let's modify it to take leap years into account. A common rule for determining if a year is a leap year is that leap years are multiples of four, but not 100 unless they are also multiples of 400. (So that 2012 and 2000 are leap years, but 2100 isn't.) In these years, February has 29 days instead of 28.

Once `daysInMonth` is working, use it to make the constructor validate its input. Specifically, the constructor should replace any year before 1 with 1 (we're not handling BC, at least for now). Then, it should replace any month outside the range 1–12 with 1. Finally, it should replace an invalid day (i.e. a day less than 1 or greater than the number of days in its month) with 1.

Next write a method `isBefore`, which takes another `Date` and returns whether the date of the object doing the comparison is before the date of its argument. (Remember that the method has access to its own attributes, but also the attributes of its argument. If the argument is called `other`, the method can access that object's attributes using the names `other.month`, `other.day`, and `other.year`.)

Now that our `Date` class has some basic functionality, let's use it to write the beginnings of a `Calendar` class, which will store a list of dates. (Actually, we're just going to manipulate the dates and not worry about attaching events to them, though you can do this if you have extra time...) Begin by creating a class `Calendar` whose attribute is an array of `Date` objects. The constructor should allocate a short (5–10 element) array for this reference and hand-create the `Date` for each cell.

Then write a method `print` that prints the dates stored in this array. Because you've written a `toString` method, it turns out that you can just pass the `Date` objects to `System.out.println`:

```
System.out.println(dates[0]);
```

where `dates` is the name of the array. Your `print` method should do something like this, but with the call to `System.out.println` in a loop so all the dates are printed.

Finally, we want to be able to reorder the dates into ascending order. This process is called *sorting*. You'll study various approaches to sorting in CS 142, but for now see if you can implement a simple algorithm: Begin by finding the index containing the earliest date. Then swap this date into position 0 using code like the following:

```
Date temp = dates[0];
dates[0] = dates[smallestIndex];
dates[smallestIndex] = temp;
```

(Print the dates after you've made this swap to make sure that you did it correctly.) Next find the index of the earliest remaining date (i.e. the earliest one in cells 1 to the end of the array) and swap it into cell 1. Repeat this for all the cells in order (using a loop).

If you have more time, see what other functionality you can add to the calendar. We mentioned handling BC dates earlier. Another obvious feature would be to store event names with the dates (likely by storing `Event` objects (or some similar name), each of which has a `Date`). You can also convert the array inside `Calendar` into an `ArrayList` so that more events could be added. Alternately, you could add a method to advance the date on a `Date` object so that the calendar could keep track of the current day and print appropriate reminders.