COLBY COLLEGE
Department of Mathematics
and Computer Science

CS115 **Allen B. Downey**
Spring 1998

Assignment 6: Objects

Due: Wednesday 18 March

The purpose of this assignment is to use some of the predefined objects Java provides, like Strings and Rectangles.

Reading

Read Kamin et al., Sections 2.5.1 and 10.1 – 10.2.

Part One

Mess around with Strings and their methods

1. Create a new project named Name, and make it an application. Personally, I have stopped putting a π at the end of my project names, since it seems to cause more problems than it's worth. So my folder, project, and class are all named Name.

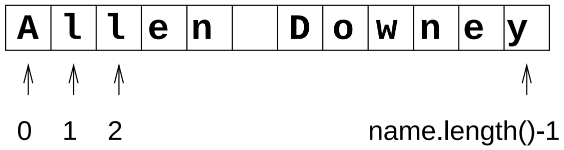
2. In `main`, create a variable named name with type String and initialize it with a String literal that contains your name. Notice that unlike most other Object types, you can initialize Strings with literals. You don't have to use the keyword `new` to create a new String.

3. Print the String.

   One of the methods you can invoke on Strings is `length()`. The way you invoke a method on an Object is similar to the way you extract a field from an object; you use dot notation. For example:

   ```
   int len = name.length();
   System.out.println (len);
   ```

   Notice the parentheses after `length`. They indicate that `length` is a method that takes no arguments, as opposed to a field named length. That probably (but not necessarily) indicates that `length` calculates the length of the String when you invoke it, as opposed to keeping the value in memory along with the String. All the methods you can invoke on Strings are described on page 326.



4. Type in and test the preceding code.

   Another method you can invoke on a String is `toUpperCase`. This method returns a new String that is the same as the old String except that all the lower case letters are converted to upper case. It *does not* modify the existing String. In fact, nothing you do can modify an existing String—they are immutable.

5. Invoke `toUpperCase` and `toLowerCase` on your String and print the results. You can print the results directly or store them in an intermediate variable.

Loopage!

Yet another method you can invoke on Strings is `charAt`. Unlike the other methods we have used, `charAt` takes an argument, an integer indicating which character you want. Something that indicates is called an index. In Java, as in many other languages, indices start are zero instead of one. See the figure for details.

WARNING: There is no character with index equal to the length of the String. The indices run from 0 to `name.length()-1`.

1. Write a for loop that goes through each letter of the String, selects each letter using `charAt` and prints each letter on a separate line.

2. Write a loop that prints the characters in the String backwards, all on the same line. Don't forget that you have to have a `println` somewhere after all the `print`s, or nothing will appear on the screen.

Send in the second String

1. Create another String with a creative name like `name2` and initialize it with another name, like Samuel Clemens.

2. Use the `compareTo` method to compare the two Strings. To compare two Strings, you invoke `compareTo` on one of the Strings and pass the *other* String as an argument. It's a little weird, but that's how it goes.

   There are two problems with this approach to comparing names. First, it gets fooled by upper and lower case letters. So e e cummings and my friend k claffy would not appear in the right place in the alphabet. Second, it compares the first names first and the last names last, and it should be doing the opposite. (Although in my high school year book I actually convinced the other editors that our pages should appear in alphabetical order by first name, just for the sake of being different and weird. I don't know how I pulled that off).

Fix the two problems

Let's write a method that gets things right.

1. Write a new method called `compareNames` that takes two Strings as arguments and that returns an integer, -1 if the first name precedes the other in the alphabet, 1 if it succeeds the other, and 0 if the names are the same.

2. One of the things you will have to do is separate the name into firstName and lastName. You may assume that the name will contain only two words, separated by exactly one space. You can find the position of the space using `indexOf`, and extract the first and last names using `substring`.

3. Once you have the two names, you will want to convert them to lower case (or upper case) and then compare them. If the last names are the same, you will have to use the first names to make the decision. Make sure you test this part of your program by giving it two names with the same last name (maybe yours and a sibling's).

NOTE REGARDING THE READING: In the book, they write a method called `compare` that takes *two Strings* as arguments. This is different from the built-in method called `compareTo` that gets invoked on one String, and receives the second String as an argument.

Neither way of writing a method is necessarily better than the other, although there are tradeoffs that we will get to later. But for now you should be aware of the difference, and know how to invoke either one.

Part Two

Using objects as parameters

1. Start with a clean copy of the Snowperson assignment. You can use my solutions if you want (they are on the General Server), or you can use your solution. If you use your solution, you might want to take a minute to review it, make sure you remember how it works, and clean it up a little.

2. Modify the Snowperson code so that every method that receives a bounding box as an argument gets a single Rectangle object instead of four integers. For example, the declaration of paintSnowperson would look like

   ```
   public void paintSnowperson (Graphics g, Rectangle bb)
   ```

3. Along with your modified code, turn in a paragraph that compares the two versions, discussing things like readability, debuggability, code length, encapsulation and abstraction.

   In the Java library, they had the option of using Rectangles to pass parameters to methods like drawRect and drawOval, but they chose to pass four integers instead. In fact, there are only a couple of methods in the whole Java library that use Rectangles at all. What do you think of this decision? Would you prefer a version of drawOval that takes a single Rectangle as an argument? What if you could have both?

What to turn in

1. Please print and prepare your code as usual, and include your short essay (maybe 200 words or fewer) on the back of one of the printouts or on a separate piece of paper. If your handwriting is not easily readable, feel free to type.