Lab 1

In this lab, you’ll practice writing methods of a Java class by implementing a version of our first data structure, the Bag.

Implementing an Array-based Bag

Log into LabApps and select the Eclipse application. After you’ve entered your password and given permission to run the application, it will ask for a location for your workspace; accept the default.

Once Eclipse opens, select the “Workbench” option, which appears as a curved arrow near the top right-hand corner of the window. This will bring up a view similar to what you saw me use in class.

In order to start programming, you need to first create a project; in the File menu, select New and then Java Project. Name the project OurBag. Then create a Java file; in the File menu, select New and then Class. Fill in the name OurBag, make sure that you’re not creating the stub for public static void main (uncheck that box), and hit Finish.

Now let’s begin implementing the class. The “big picture” idea is that objects of this class will store a collection of strings (i.e. objects of type String), just like a bag stores a collection of groceries (or whatever) in the physical world. The class will use an array for the underlying storage. Begin by adding attributes (variables) to the OurBag class. It should have an array of strings called vals and an integer num. The latter will store the number of values in the bag, which is different than the length of the vals array; in general, the array will have empty cells for values that haven’t been added yet.

Your first method should be a constructor that takes no arguments. (Recall that a constructor is a method without a return type and whose name is the same as the name of its class. It is called whenever an object of that class is created.) The constructor should set num to 0 (since the bag is empty) and create an array for vals; begin by creating one of length 3.

Next write a method size that returns the number of values in the bag (i.e. the num attribute).

Next write a method add, which should take a string and add it to the array. It should store the string in the first empty array cell (use num to determine which one this is), and increment num (since the bag now has one more value). For now, assume that the array has an empty cell.

In order to test that your code works, you’ll need to create a class with the main method to test it. Use the New menu to create a new class called Main similarly to how we created OurBag, but this time select the checkbox to create the stub for public static void main. Once the class is created, write code in its main method to create an instance of OurBag (use new like we did in class yesterday). Then print this object’s size, add a value or two, and print its size again. Run your program to check that add and size seem to work. (To run the program, hit the round green button containing a white triangle in the menu bar.)

Next write a method contains that takes a String and returns a boolean indicating whether that string is in the bag. Use the equals method of String to see if two strings are equal (e.g. s1.equals(s2) will return whether s1 and s2 are equal). We’ll talk about why you don’t want to use == later in the class.

Once contains is written, modify main to test that contains returns false before a value is added to the bag and true afterwards.

Next, it’s time to revisit add. Our first implementation makes the unfortunate assumption that the array always has room for another string. Since we’d like to allow our bag to grow bigger than 3 elements, we need to remove this assumption. To do this, add should check if the array is already full before putting a new value into it. (To see if it’s full, compare num to vals.length.) If the array is full, create an array that is twice the current length and copy the values from the old array into the new one. Then assign this new array to vals; this copies the reference to the new array into the attribute, effectively replacing the old array.
with the new one. Check that this method works by changing main to insert more strings.

After all that, you’ve written a basic bag implementation. If you have additional time, try to implement the methods below (or other useful bag methods). As you implement each method, be sure to modify main to test it.

- A method `clear` that takes no arguments and returns nothing, but empties the bag. (Hint: you can do this without touching the `vals` attribute.)

- A method `remove` that takes a `String` and removes that value if it appears in the bag. If the bag contains more than one copy of the `String`, only one of them should be removed.

- A method `addAll` that takes another `OurBag` object and adds all members of the argument bag.