pd8 $hw53_{16}$ 2019-03-18 ray. lee.

0. A list of elements of a given type can be represented using an array of that type. For example, in $hw50_{16}$, we created three separate arrays for three respective types (int, double, and String). Each particularly typed array stored values (or objects for the String array) as elements. A fourth array, of type int, stored the type of an element at a particular index, using integers 0, 1, and 2 as its elements (0 = int, 1 = double, 2= String).

1. A list of elements of disparate types can be implemented without polymorphism two ways, at least. Such programs share a program structure like

```
public class List_inArraySlots {

private int[] typeOfElement;

private int[] intElements;

private double[] doubleElements;

private String[] stringElements;

private int filledElements;

private static final int INITIAL_CAPACITY;
```

2. **Stumbling block**

   This structure constitutes a maintenance nightmare: Every time the add method is invoked, four elements are passed as arguments. One of the arguments indicates the type of the element to be stored at that particular index, and another one of the arguments indicate the value (or reference to a String object) to be stored as the element. Two of the four arguments are completely ignored, which is very unproductive because it wastes both the time of a programmer to consider those cases and the memory of the computer. In addition, every time a new type is introduced, a new array of that type must be created in order for elements of that type to be appended to the array.

3. **Workaround**

   Polymorphism in Java allows implementing a list of elements of disparate types by creating an Object array that stores wrapper classes, subclasses of the Object class, as elements. The object's attribute stores a value that can be differently typed than the other objects' attribute's value.

4. Polymorphism in Java supports applying a method to every element when all of their types have an implementation of that method.

   For example, + can be used as addition between ints and concatenation between strings. If used on a string and an int and if the int comes after the string, the string and int can be concatenated to form a single string. The implementation can be defined in the class that represents the type, as in String, Double, Integer, etc. Alternatively, the implementation can be inherited, the way class String inherits method toString() from class Object. These examples show that it is immaterial *how* the method becomes part of the class.