FA21 CSE 8B Homework 6: Power of Polymorphism

Due Date: Wednesday, November 10, 11:59 PM

Learning goals:

- Explore the power of polymorphism through:
  - The `extends` keyword (Inheritance)
  - Inheritance relationships with UML diagrams
  - Subclasses and superclasses

NOTE: This assignment should be completed INDIVIDUALLY. Pair programming is NOT allowed for this assignment.

IMPORTANT: You should NOT have to import any additional packages to complete this assignment. Any unnecessary imports may result in failure to compile on Gradescope.

THIS IS A LONG ASSIGNMENT WITH A LOT OF METHODS, SO PLEASE START EARLY!!!

Coding Style (10 points)

As always, we will be enforcing the CSE 8B Coding Style Guidelines. These guidelines can also be found on Canvas. Please ensure to have *COMPLETE* file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers. For reference, please see this mock PA with complete style. **These points are all-or-nothing, so please do NOT forget any headers or magic numbers/strings.**

Part 0: Getting started with the starter code (0 points)

1. Make sure there is no problem with your Java coding environment. If there are any problems, then review Assignment 1, or come to the office/lab hours before you start Assignment 6.
2. Download the starter code.
   a. If you are working on your local machine, then you can download the starter code from Piazza → Resources → Homework → `Assignment6.zip`. Download the starter code to a directory of your choice, then extract the `zip` file. The extracted folder should contain 8 Java files: `Assignment6.java`, `Cuboid.java`,

`IntegerList.java`, `List.java`, `MyObject.java`, `Shape.java`, `Sphere.java`, and `StringList.java`. Afterwards, open your terminal or command prompt, then navigate to the directory that contains those 8 Java files.

b. If you are working via UCSD Linux Cloud through your CSE 8B account, then use the commands below to copy the starter code to a new directory called `PA6`, to change your current directory to `PA6/starter`, and to print the files in the `starter` directory:

```
$ cp -r ~/../public/assignments/PA6 ~
$ cd ~/PA6/starter
$ ls
```

`ls` should print out 8 Java files: `Assignment6.java, Cuboid.java, IntegerList.java, List.java, MyObject.java, Shape.java, Sphere.java, and StringList.java`.

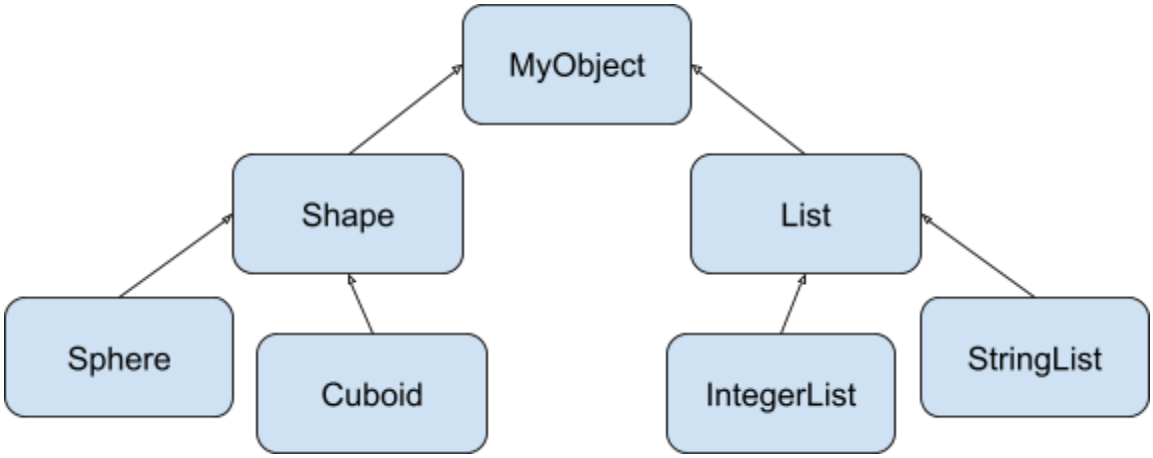
3. Try to compile and run the starter code, and you should expect the following output

```
bensonbudiman@Bensons-MacBook-Pro starter % javac *.java
bensonbudiman@Bensons-MacBook-Pro starter % java Assignment6
All unit tests passed.


index:
-1

mean:

sorted:
bensonbudiman@Bensons-MacBook-Pro starter %
```

You will have to implement a LOT of methods to finish this programming assignment.

Part 1: Overview

For this programming assignment, you will be implementing the classes for the UML diagram shown below:



Before you start programming, please take some time to review the starter code and to read the instructions below CAREFULLY. Some methods are already implemented for you, but you will need to supply those methods with a method header. **You should fully understand the purpose of each variable and the usage for each method before you implement anything.**

NOTE 1: you should NOT change any data field or method signature in the starter code. **As such, do NOT add any additional parameters to methods, and do NOT import any Java packages.** Feel free to add any helper methods if desired.

NOTE 2: You must implement and comment on everything with a `TODO`. **Do NOT forget to adhere to the CSE 8B style guidelines.**

NOTE 3: To be clear, the `List` class that you will be implementing is DIFFERENT from Java's built-in `ArrayList` class. **You will NOT (and should NOT) be using `ArrayList` in this programming assignment.**

Part 2: `MyObject.java`

First, you need to implement the object class called `MyObject`. This is a super class of all other classes in this assignment (as seen in the UML diagram). The `MyObject` class defines the default behavior of methods (e.g., the method `equals`, which is later overridden by subclasses) of all the subtype classes in this assignment.

The `MyObject` class has two fields: `String type` and `String highLevelType`. There are two `highLevelType` classes: `Shape` and `List`. Likewise, there are four `type` classes: `Sphere`, `Cuboid`, `IntegerList`, and `StringList`. As seen in the UML diagram above, a `Sphere` and a `Cuboid` "is-a" `Shape`, and an `IntegerList` and a `StringList` "is-a" `List`.

First, complete the constructor and getters as stated in the starter code. You may notice that there are a lot of getters, but the only getters that you will have to implement are:

1. `public String getType()`
2. `public String getHighLevelType()`

Do NOT forget about the constructor:

1. `public MyObject(String type, String highLevelType)`

Then, implement the following two methods:

1. `public boolean isComparableWith(MyObject anotherObj)`
   You should `return true` only when the current object (referring to `this` object - this entire writeup will use the same terminology for `this`) and the input `anotherObj` have the same `type` **OR** both of their `highLevelType` is `"Shape"`. Otherwise, you should `return false`.

2. `public boolean isAddibleWith(MyObject anotherObj)`
   You should `return true` only when the current object and the input `anotherObj` have the same `type` **AND** the `highLevelType` is `"List"`.

Part 2: `Shape.java` and `List.java`

`Shape` and `List` are two subclasses of `MyObject`. **Complete all remaining constructors and methods in those classes. The no-arg constructors are already provided to you, so you can use that as guidance for your other constructor. Remember, you must NOT change the existing signature or the fields.**

NOTE: Again, to be clear, the `List` class that you will be implementing is DIFFERENT than Java's built-in `List` class.

Shape

1. `public Shape(String shape, double volume)`
   This constructor has two arguments: `String shape` and `double volume`. This constructor should first set the `type` field with the `shape` input and set the `highLevelType` field with the string `"Shape"` **(HINT: use `super` to call the superclass constructor!)**. Next, if the input `volume` is strictly less than 0, then set the member variable `volume` to 0. Otherwise, assign the member `volume` as you would normally.

2. `public boolean equals(MyObject anotherObj)`
   This method overrides the `equals` method in `MyObject`. This method checks whether the current `Shape` object is considered equal to the input `anotherObj`. You should

`return true` only when they have the same `type` and same `volume`, and you should `return false` otherwise.

3. `public boolean isLargerThan(MyObject anotherObj)`
   This method overrides the `isLargerThan` method in `MyObject`. First, compare the `volume` of the current `Shape` and the input `anotherObj`. You should `return true` when two objects have the same `highLevelType` and the current `Shape`'s `volume` is strictly larger than that of the input `anotherObj`.

List

1. `public List(String listType, int length)`
   This constructor has two arguments: `String listType` and `int length`. This constructor should first set the `type` field with the `listType` input and set the `highLevelType` field with the string `"List"` **(HINT: use `super` to call the superclass constructor!)**. Next, if the input `length` is strictly less than 0, then set the member variable `length` to 0. Otherwise, assign the member `length` as you would normally.

2. `public boolean equals(MyObject anotherObj)`
   This method overrides the `equals` method in `MyObject`. This method checks whether the current `List` object is considered equal to the input `anotherObj`. You should `return true` only when they have the same `type` and same `length`, and you should `return false` otherwise.

3. `public boolean isLargerThan(MyObject anotherObj)`
   This method overrides the `isLargerThan` method in `MyObject`. First, compare the `length` of the current `List` and the input `anotherObj`. You should `return true` when two objects have the same `highLevelType` and the current `List`'s `length` is strictly larger than that of the input `anotherObj`.

Part 3: `Sphere.java` and `Cuboid.java`

`Sphere` and `Cuboid` are two subclasses of `Shape`. **Complete all remaining constructors and methods in these classes. You can also assume that all inputs for the constructors will be non-negative.**

Sphere

1. `public Sphere(double radius)`
   This constructor has one argument: `double radius`. This constructor should set the `type` field to the string `"Sphere"`, set the `volume` field based on the input **(NOTE: the**

volume of a sphere is $\frac{4}{3}\pi r^3$), and set the `radius` field to the input `radius`. **HINT: Setting the `type` and the `volume` will be done through calling `super`, and the setting of `highLevelType` should already have been implemented through the superclass `Shape`.**

NOTE: You might notice how there is no override method for `equals()` for the Sphere class. Because we only care about the `volume` of the Sphere when comparing Sphere objects (i.e., `radius` does not matter for comparison), we can just use the superclass Shape's `equals()`.

Cuboid

1. `public Cuboid(double length, double width, double height)`
   This constructor has three arguments: `double length`, `double width`, and `double height`. This constructor should set the `type` field to the string `"Cuboid"`, set the `volume` field based on the input **(NOTE: the volume of a cuboid is $l * w * h$)**, and set the remaining member variables `length`, `width`, and `height` respectively. **Follow the same hint as above!**

2. `public boolean equals(MyObject anotherObj)`
   Overrides the `equals` method in `Shape`. You should `return true` only when the superclass's `equals` method returns true on the input `anotherObj` and that `anotherObj` has the same `length`, same `width`, and the same `height` as the current object. Otherwise, you should `return false`.

Part 4: `IntegerList.java` and `StringList.java`

`IntegerList` and `StringList` are two subclasses of `List`. **Complete all remaining constructors and methods in these classes.**

IntegerList

1. `public IntegerList(int[] list)`
   This constructor has one argument: `int[] list`. This constructor should set the `type` field to the string `"Integer List"`, set the `length` field to the length of the input `list`, and set the `integers` field to input `list`. **HINT: Setting the `type` and the `length` will be done through calling `super`, and the setting of `highLevelType` should already have been implemented through the superclass `List`.**

2. `public boolean equals(MyObject anotherObj)`
   Overrides the `equals` method in `List`. You should `return false` if the input `anotherObj` does not have the same `type` as the current object. Next, two

`IntegerList` objects are considered to be equal when they have the same `length` and all the corresponding elements in `integers` have the same value.

3. `public int getIntegerAt(int idx)`
   Return the `int` value in `integers` at position `idx`. You can assume that `idx` will always be within the range of the `integers` (*i.e.* no need to handle out-of-bound errors).

4. `public boolean isLargerThan(MyObject anotherObj)`
   Overrides the `isLargerThan` method in `List`. The current `IntegerList` object is considered larger than the input `anotherObj` only when the sum of all elements in the current list `integers` is strictly larger than that of the input's sum of `integers`.
   Example: if `intList1 = new IntegerList(new int[]{-4, 6, 10})` and `intList2 = new IntegerList(new int[]{3, 4, 4, 6})`, then `intList1` would NOT be larger than `intList2`.

5. `public MyObject add(MyObject anotherObj)`
   Perform element-wise addition of the current `IntegerList` and the input `anotherObj`. In case two `IntegerList`s have different lengths, keep the exceeded part of the longer `IntegerList` unchanged. Then, `return` a newly created `IntegerList` that contains your result.
   Example: if `intList1 = new IntegerList(new int[]{-4, 6, 10})` and `intList2 = new IntegerList(new int[]{3, 4, 4, 6})`, then the result of `intList1.add(intList2)` would be `{-1, 10, 14, 6}`.

6. `public MyObject divideByInteger(int denominator)`
   If the `denominator` is less or equal to 0, then set the `denominator` to 1. Then, divide each element of the current `IntegerList` by the `denominator`. Finally, `return` a newly created `IntegerList` that contains your result.
   Example: if `intList1 = new IntegerList(new int[]{-4, 5, 13})`, then the result of `intList1.divideByInteger(2)` would be `{-2, 2, 6}`.

NOTE 1: Notice how `equals()`, `isLargerThan()`, and `toString()` are override methods, but `add()`, `divideByInteger()`, and `getIntegerAt()` are not. Why do you think that is the case?

NOTE 2: BOTH `add` and `divideByInteger` do NOT change the current `IntegerList` object. Both methods should `return` a newly created object. This same rule will apply to the same methods in the `StringList` class.

StringList

1. `public StringList(String[] list)`
   This constructor has one argument: `String[] list`. This constructor should set the `type` field to the string `"String List"`, set the `length` field to the length of the input `list`, and set the `strings` field to input `list`. **HINT: Setting the `type` and the `length` will be done through calling `super`, and the setting of `highLevelType` should already have been implemented through the superclass `List`.**

2. `public boolean equals(MyObject anotherObj)`
   Overrides the `equals` method in `List`. You should `return false` if the input `anotherObj` does not have the same `type` as the current object. Next, two `StringList` objects are considered to be equal when they have the same `length` and all the corresponding elements in `strings` have the same value.

3. `public String getStringAt(int idx)`
   Return the `String` value in `strings` at position `idx`. You can assume that `idx` will always be within the range of the `strings` (*i.e.* no need to handle out-of-bound errors).

4. `public boolean isLargerThan(MyObject anotherObj)`
   Overrides the `isLargerThan` method in `List`. The current `StringList` object is considered larger than the input `anotherObj` only when the sum of the lengths of all elements in the current list `strings` is strictly larger than that of the input's sum of lengths of `strings`.
   Example: if `strList1 = new StringList(new String[]{"iGQIL", "G92", "9"})` and `strList2 = new StringList(new String[]{"Animal", "Planet"})`, then `strList1` would NOT be larger than `strList2`.

5. `public MyObject add(MyObject anotherObj)`
   Perform element-wise concatenation of the current `StringList` and the input `anotherObj`. In case two `StringList`s have different lengths, keep the exceeded part of the longer `StringList` unchanged. Then, `return` a newly created `StringList` that contains your result. **NOTE: It does NOT matter which String you append/prepend as we will be testing based on the length of the Strings.**
   Example: if `strList1 = new StringList(new String[]{"iGQIL", "G92", "9"})` and `strList2 = new StringList(new String[]{"Animal", "Planet"})`, then the result of `strList1.add(strList2)` would be `{"iGQILAnimal", "G92Planet", "9"}`.

6. `public MyObject divideByInteger(int denominator)`
   If the `denominator` is less or equal to 0, then set the `denominator` to 1. Then, take the substring of each element of the current `StringList` where the `beginIndex` is 0

and the `endIdx` is the length of the element divided by the `denominator`. Finally, `return` a newly created `StringList` that contains your result.
Example: if `strList1 = new StringList(new String[]{"Animal", "Kingdom", "Theme", "Park"})`, then the result of `strList1.divideByInteger(2)` would be `{"Ani", "Kin", "Th", "Pa"}`.

Part 5: The 3 methods in `Assignment6.java`

Finally, within `Assignment6.java`, you will be using the power of polymorphism to sort arrays, to find the mean of an array, and to find an element in an array.

There are three methods that you need to implement in `Assignment6.java`. **All methods are required to be implemented in a polymorphic manner.** ***More specifically, do NOT separate objects with different subtypes by explicitly using a lot of if-else or switch statements. Furthermore, do NOT call the method `getType()` in these three methods.*** **You will lose ALL credit if you do not implement these methods in a polymorphic manner.**

1. `public static boolean sort(MyObject[] objList)`
   To sort an array of objects of different types, a comparison standard between two objects is required (i.e., for any two of the objects in the array, we need to be able to say "which object is larger?"). Fortunately, you should have already implemented a comparison standard, which is `isLargerThan()`. Apply the comparison standard (i.e., call method `isLargerThan()`) correctly based on the actual type of object. While you are sorting, if two objects are not comparable (**Hint:** use `isComparableWith()`), then print out the error message `NON_COMPARABLE_MSG` to the command line and `return false`.

   You can implement any sorting algorithm you want. If you are unfamiliar with sorting algorithms, then we recommend that you implement selection sort (see Liang, section 7.11, available in Piazza Resources), a simple, in-place comparison sorting algorithm. We will be explaining the selection sort in the discussion for this assignment.

   After calling this method, the input `objList` will be modified with all elements sorted from small to large based on the comparison standard of the object type. If the sorting is successful, then you should `return true` at the very end.

2. `public static MyObject mean(MyObject[] objList)`
   Sum up all objects in the input `objList` (**Hint:** use `add()`) and divide the result by the length of the `objList` to get the mean of the object of that type (**Hint:** use `divideByInteger()`). Similarly to `sort`, while you are summing objects, if two objects are not addible (**Hint:** use `isAddibleWith()`), then print out the error message `NON_ADDABLE_MSG` to the command line and `return null`.

3. `public static int findIndex(MyObject[] objList, MyObject target)`
   Find the index of the first object in `objList` that is considered equal (**Hint:** use `equals()`) to the input `target` object. If no match is found, then `return -1`.

After finishing these three methods, you can compile and run Assignment6. Your output should be the same as below:

```
PS C:\Users\Benson\Documents\GitHub\FA21_CSE8B_PA6\reference> javac *.java
PS C:\Users\Benson\Documents\GitHub\FA21_CSE8B_PA6\reference> java Assignment6
All unit tests passed.


index:
2

mean:
Integer List - Length: 4, Elements: [0, 4, 5, 3]

sorted:
Integer List - Length: 4, Elements: [1, 2, 3, 4]
Integer List - Length: 3, Elements: [-4, 6, 10]
Integer List - Length: 4, Elements: [3, 4, 4, 6]
```

Part 6: `unitTests()`

Keep in mind that the `main` method only provides one testing example. Therefore, you are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. **To get full credit, create a total of *at least 5* test cases (or tester methods) for the *three* methods in Assignment6.java (`sort`, `mean`, and `findIndex`). For example, you could create 2 tester methods for `sort`, 2 tester methods for `mean`, and 1 tester method for `findIndex`, or you could create 3 tester methods for `sort`, 1 tester method for `mean`, and 1 tester method for `findIndex`.** There are some comments within `unitTests()` suggesting what to test. We also suggest making some print messages in each of your test cases so that you will know which test case is failing. The `unitTests()` method should `return true` only when all the test cases are passed. Otherwise, you should `return false`.

NOTE: It is OK if your tests print to standard output. The Gradescope Autograder will not be reading anything from standard output.

Submission

VERY IMPORTANT: Please follow the instructions below carefully and make the exact submission format.

1. Go to Gradescope via Canvas and click on `PA6`.
2. Click the DRAG & DROP section and directly select the required files (**`Assignment6.java`**, **`Cuboid.java`**, **`IntegerList.java`**, **`List.java`**, **`MyObject.java`**, **`Shape.java`**, **`Sphere.java`**, **and** **`StringList.java`**). Drag & drop is fine. **Please make sure you don't submit a zip. Make sure the filenames are correct.**
3. **You can resubmit an unlimited number of times before the due date.** Your score will depend on your final submission, even if your former submissions have a higher score.
4. The autograder is for grading your uploaded files automatically. Make sure your code can compile on Gradescope. You will receive 0 on the Autograder section if your code does not compile correctly on Gradescope.

   NOTE: The Gradescope Autograder you see is a minimal autograder. For this particular assignment, the Gradescope Autograder will only show the compilation results and the results of a few testers. After the assignment deadline, a thorough Autograder will be used to determine the final grade of the assignment. **Thus, to ensure that you would receive full points from the thorough Autograder, it is your job to extensively test your code for correctness via `unitTests` and `main`.**