FA20 CSE 8B - Assignment 6

Introduction
In this assignment, you will explore the power of polymorphism to help implement complicated programs in a flexible and general way. You are also required to come up with your own tests to make sure your code has the correct behavior as described in this writeup. You should complete this assignment individually.

Part 1: Get started
1. Make sure there is no problem with your Java coding environment. If there is any, review assignment 1 or come to the office hour before you start this assignment.
2. Review everything on and before lecture 11
3. Download the starter code.
   If you work with your local machine, download it from Piazza -> Resources -> Homework -> Assignment6.java, BooleanVector.java, Cricle.java, DoubleVector.java, MyObject.java, Rectangle.java, Shape.java, and Vector.java.
   Start terminal/command line and navigate to the directory that contains the files listed above.
   If you decide to use Linux Cloud, use the following command to copy the starter code to a new directory called HW6:

   ```
   $ cd ~
   $ mkdir HW6
   $ cp -a ../public/assignments/PA6 ./HW6
   ```

   Note: you will not be able to compile the starter code because most of the methods are left unimplemented.

Implementation tip: you should not change any existing data fields or method signatures in the starter code. Some methods are already implemented to you, so you only need to fill in the methods with // TODO. You CANNOT add any additional fields, or import any Java build-in packages. You can only add helper methods in Assignment6.java file. As you are familiar with this course and our assignment format, we are trying to give you more succinct instructions.
Observe the starter code and read the instructions below to make sure you understand what each field means before you start to implement.

The inheritance relationship between classes for this assignment is shown in the UML below:

![UML diagram](image)

**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. MyObject class defines the default behavior of methods (e.g. the method equals, which is overridden later by subclasses) of all the subtype classes in this assignment. It has two fields: `String type` and `String highLevelType` that correspond to the classes shown in the table below. Complete the constructor and getters as we usually do. Then implement the following two methods:

- **public boolean isComparableWith(MyObject anotherObj):** Return true only when the current object (referring `this` object, the whole writeup will use the same terminology for this) and the input anotherObj have the same type **OR** both of their highLevelType are “Shape”.
- **public boolean isAddibleWith(MyObject anotherObj):** Return true only when the current object and the input anotherObj have the same type **AND** the highLevelType is “Vector”.

```java
// MyObject.java

public class MyObject {
    private String type;
    private String highLevelType;

    // Constructor and getters

    public boolean isComparableWith(MyObject anotherObj) {
        // Implementation
    }

    public boolean isAddibleWith(MyObject anotherObj) {
        // Implementation
    }
}
```
Part 3: Shape.java and Vector.java

Shape and Vector are two subclasses under MyObject. Complete all constructors and methods in those classes. The no-arg constructors are already provided to you. Remember, you cannot change the existing signature or the fields.

**Shape class** has one additional field: **double area**, which is the area of a shape.
- The constructor that takes two arguments should set the **type** field with the **shape** input and set the **highLevelType** with the string “Shape”. If the input area is less than 0, set it to 0.
- `public boolean equals(MyObject anotherObj)`: This method overrides the equal method in MyObject. It checks whether the current Shape object is considered equal to the input object. Return true only when they have the same **type** and same **area**.
- `public boolean isLargerThan(MyObject anotherObj)`: This method overrides the isLargerThan method in MyObject. Compare the areas of current Shape and the input object. Return true when two objects have the same **highLevelType** and the current Shape’s **area** is (strictly) larger than that of the input object.

**Vector class** has one additional field: **int length**, which is the length of the vector.
- The constructor that takes two arguments should set the **type** field with the **vectorType** input and set the **highLevelType** as string “Vector”. If the input length is less than 0, set it to 0.
- `public boolean equals(MyObject anotherObj)`: This method overrides the equal method in MyObject. Return true only when they have the same **type** and same **length**.
- `public boolean isLargerThan(MyObject anotherObj)`: This method overrides the isLargerThan method in MyObject. It compares the lengths of current Vector and input. Return true when two objects have the same **highLevelType** and the current Vector’s **length** is (strictly) larger than that of the input object.
Part 4: Circle.java and Rectangle.java

Circle and Rectangle are two subclasses under Shape. Complete all constructors and methods in these classes.

**Circle class** has one additional field: `double radius`, which is the radius of a circle.
- The constructor that takes one argument should set the `type` field to string “Circle” and `highLevelType` to string “Shape”. Then set the `radius` and `area` based on input. (You can assume all inputs are non-negative so you do not need to check for negative input)

**Rectangle class** has two additional fields: `double height` and `double width`.
- The constructor that takes two arguments should set the `type` field to string “Rectangle” and `highLevelType` to string “Shape”. Then set the `area` based on `height` and `width`. (You can assume all inputs are non-negative so you do not need to check for negative input)
- `public boolean equals(MyObject anotherObj)`: Overrides equal method in Shape. Return true only when the super class’s equal method returns true on the input object and it has the same length and same width as the current object.

Part 5: DoubleVector.java and BooleanVector.java

DoubleVector and BooleanVector are two subclasses under Vector. Complete all constructors and methods in these classes.

**DoubleVector class** has one additional field: `double[] doubleList`.
- The constructor that takes one argument should set the `type` field to string “Double Vector” and `highLevelType` to string “Vector”. Then set `doubleList` with the input list. (Recall Assignment 3)
- `public boolean equals(MyObject anotherObj)`: Overrides equal method in Vector. Return false if the input object does not have the same `type` as the current object. Then two DoubleVector objects are considered to be equal when they have the same length and all the corresponding elements in `doubleList` have the same value.
- `public double getDoubleAt(int idx)`: Overrides the default `getDoubleAt` method in `MyObject`. Return the double number in the `doubleList` at position `idx`. 
- **public boolean isLargerThan(MyObject anotherObj):** The current DoubleVector object is considered larger than the input DoubleVector object only when the sum of all elements in the list is larger than that of the input.

- **public MyObject add(MyObject anotherObj):** Perform element-wise addition of the current DoubleVector and the input. In case two DoubleVectors have different lengths, keep the exceeded part of the longer DoubleVector unchanged. Then return a newly created resulting DoubleVector.

  **Example:**
  If Vec1 is {1, 2, 3, 5, 8} and Vec2 is {1, 2, 3}, the returned vector of Vec1.add(Vec2) is {2, 4, 6, 5, 8}

- **public MyObject divideByInteger(int denominator):** If denominator is less or equal to 0, set it to 1. Then divide each element of the current DoubleVector by the denominator and return the newly created resulting DoubleVector.

  Note: add and divideByInteger do not change the current DoubleVector object. Both methods should return a newly created object. Same rule applies to methods in BooleanVector class.

  **BooleanVector class** has one additional field: boolean[] booleanList.

  - The constructor that takes one argument should set the type field to string “Boolean Vector” and highLevelType to string “Vector”. Then set booleanList with the input list.

  - **public boolean equals(MyObject anotherObj):** Return false if the input object does not have the same type as the current object. Then two BooleanVector objects are considered to be equal only when they have the same length and all the corresponding elements in booleanList have the same value.

  - **public double getBooleanAt(int idx):** Return the boolean value in booleanList at position idx.

  - **public boolean isLargerThan(MyObject anotherObj):** The current BooleanVector object is considered larger than the input object when the sum of all elements in the list is larger than that of the input. When calculating the sum, true is regarded as 1 and false is regarded as 0.

    **Example:** {true, true} > {false, true, false, false}

  - **public MyObject add(MyObject anotherObj):** Perform element-wise OR operation of the current BooleanVector and the input. In case two BooleanVector have different lengths, keep the exceeded part of the longer BooleanVector unchanged. Then return a
newly created resulting BooleanVector. OR operation means applying logic OR between two boolean variables.

**Example:**
If Vec1 is {true, false, false, true, false} and Vec2 is {false, false, true}, the returned vector of Vec1.add(Vec2) is {true, false, true, true, false}

- **public MyObject divideByInteger(int denominator):** If denominator is odd, regard it to be true and if it is even, regard it to be false. Then apply AND operation between each element of the current BooleanVector and the boolean interpretation of the denominator and return the newly created resulting BooleanVector.

  **Example:**
  If Vec1 is {true, false, false, true, false}, then the returned vector of Vec1.divideByInteger(3) is {true, false, false, true, false}; Vec1.divideByInteger(0) is {false, false, false, false, false}

**Part 6: Three methods in Assignment6.java**

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 explicitly by using a lot of if-else or switch statements. Do not call method getType() in these three methods. You will lose credit for doing that.

- **public static boolean sort(MyObject[] objList):** To sort an array of objects of different types, the 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 one is larger”). Apply the comparison standard (i.e. call method isLargerThan()) correctly based on the actual type of object. If two objects are not comparable (recall Part 2), print out the error message “Contains two or more objects that are not comparable” to the command line and return false. (You can use any sorting algorithm as you will, one simple way to sort will be explained in discussion this Friday). After calling this method, the input objList will be modified, with all elements sorted from small to large based on the comparison standard of object type. If the sorting is successful, return true.

- **public static MyObject mean(MyObject[] objList):** Sum up all objects in the input objList and divide the result by the length of the objList to get the mean object of that type. Similar to sort, If two objects are not addible (recall Part 2), print out the error
message "Contains two or more objects that are not addible" to the command line and return null.

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

After finishing all three methods, you can try to run Assignment6, and the output should be the same as follow:

```
$ java Assignment6
All unit tests passed.

index: 2

mean:
Double Vector - Length: 4, Elements: [0.0, 4.0, 5.666666666666667, 3.333333333333335]

sorted:
Double Vector - Length: 4, Elements: [1.0, 2.0, 3.0, 4.0]
Double Vector - Length: 3, Elements: [-4.0, 6.0, 10.0]
Double Vector - Length: 4, Elements: [3.0, 4.0, 4.0, 6.0]
```

**Part 7: Unit tests**

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 at least two test cases for each of the three methods in Assignment6.java. We 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 return false.

**Part 8: Coding style**

When coding in Java, there are several style rules that people usually follow to make the code clean and readable. In this course, you are asked to follow rules specified in link below:

[https://cseweb.ucsd.edu/classes/fa20/cse8B-a/styleguide.html](https://cseweb.ucsd.edu/classes/fa20/cse8B-a/styleguide.html)

Read the coding style guide carefully and refine your code for this and all future assignments.
Submission

Very important! Please follow the instructions below carefully and make the exact submission format. This is important since we will use scripts to grade so if you don't follow the same submission format you probably will receive a zero.

1. Go to Gradescope and click on PA6.
2. Click the DRAG & DROP section and directly select the seven required files (Assignment6.java, BooleanVector.java, Circle.java, DoubleVector.java, MyObject.java, Rectangle.java, Shape.java, and Vector.java). Drag & drop is fine. Please make sure you don't submit a zip. Just the file solely. Make sure the name of the file is correct.
3. You can resubmit unlimited 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 the use of the instructional team. You won't see the result of the autograder. As long as you uploaded your file you're good to go.