## Lab 4

### Preliminaries

The name of your Eclipse project should be abc123-lab4, where you replace abc123 with your abc123 id. The name of the file that contains your main method should be Lab4.java. You will also several Java files for implementing different kinds of Length objects: Length.java, Meter.java, Inch.java, Foot.java and Yard.java; these Java files will be placed into a Java package named "length". To submit the project, export the project and upload the zip file to Blackboard.

The primary goal of this assignment is to become more familiar with inheritance in Java. The design could probably be improved in a number of ways, but in its current form, this assignment will make you consider how methods and variables are inherited, in particular, how methods inherited from the superclass work with methods in the subclass.

### Task

The task is to read a file data.txt, which consists of one length on each line. Each length should be stored in an appropriate Length object. All the Length objects should be stored in an ArrayList<Length>. Each Length object should be printed out. The minimum and maximum length should be printed out. Then, the sum of the Lengths should be printed out in different ways as shown below.

Suppose data.txt consists of the following lengths.

1 meter

1 inch

1 foot

1 yard

401.336 meters

15839 inches

1319 feet

439 yards

Each length will be on one line, a number followed by the unit. Note that a unit might be spelled two different ways.

Each length should be stored in an object of the appropriate type: a Meter object, an Inch object, a Foot object or a Yard object, where Meter, Inch, Foot and Yard are subclasses of Length. All of these objects should be stored in an ArrayList<Length> unless you want to manage a dynamically-sized array yourself. You should not depend on having exactly eight lines in the input.

This data should result in the following output:

class length.Meter: 1.0 meter

class length.Inch: 1.0 inch

class length.Foot: 1.0 foot

class length.Yard: 1.0 yard

class length.Meter: 401.336 meters

class length.Inch: 15839.0 inches

class length.Foot: 1319.0 feet

class length.Yard: 439.0 yards

Minimum is class length.Inch: 1.0 inch

Maximum is class length.Inch: 15839.0 inches

Sum of Lengths Adding from First to Last

class length.Meter: 1609.344 meters

class length.Inch: 63360.0 inches

class length.Foot: 5280.0 feet

class length.Yard: 1760.0 yards

Sum of Lengths Adding from Last to First

class length.Meter: 1609.3439999999998 meters

class length.Inch: 63360.0 inches

class length.Foot: 5279.999999999999 feet

class length.Yard: 1760.0 yards

Each line printing a Length object includes the class name obtained by calling the getClass method; "length.Meter" indicates that the Meter class is in the length package. The remainder of the line prints the value and the unit of the Length object.

### Project Structure

The main method will be in Lab4.java in the src directory. The other .java files should be placed in a package named "length". In Eclipse, you can create a new package named length by (File -> New -> Package), then when you create a new Java class (other than Lab4.java), you should fill in the name of the package as "length" (without the double quotes). This will put the .java file in the src/length directory and include a "package length;" declaration at the top of the file.

The length package will contain an abstract class named [Length](http://cs.utsa.edu/~cs3443/laboratories/Length.java). You should also have classes named Meter, Inch, Foot and Yard, which will all be subclasses of Length (extends Length).

### Length.java

A nearly complete [Length.java](http://cs.utsa.edu/~cs3443/laboratories/Length.java) has been provided for you. The only coding you need to do in Length.java is the compareTo method. In Eclipse, after you have created a package named length, then download Length.java into the src/length directory, and refresh the project (right click on the project -> Refresh).

Length.java is an abstract class, which means some methods are left to be implemented by subclasses: add, getUnit, and toMeters. The add method should add another Length to the object being called (e.g., adding any kind of Length object to a Foot object). The getUnit method should return the String for describing the unit (e.g. "foot" or "feet" for Foot objects). The toMeters method should return the number of meters as a double (e.g., converting feet to meters for Foot objects).

Length.java does implement some common features of Length objects. It declares a private instance variable named length. This instance variable is very private; even the code of subclasses cannot access this instance variable directly. Code in the subclasses need to use the getLength and setLength methods implemented in Length.java to access and modify this instance variable.

The initial Length.java includes a compareTo method that fulfills the "implements Comparable<Length>" declaration in the class header. However, the compareTo method needs different code. This method should return a negative integer if this (the object being called) is less than other, zero if this is equal to other, or a positive integer if this is greater than other. Directly comparing the value of the length instance variables will not work as you could be comparing inches to meters. You need to use the toMeters method to compare the two Lengths correctly.

The toString method returns a String including the class name, the length, and the unit name, just like most of the lines in the example output.

### Subclasses of Length

Each subclass of Length should be in the length package, should have a constructor and should implement the three methods add, getUnit, and toMeters. The constructor should have a double parameter named length. The code should consist of one line super(length); to invoke the constructor in Length.java.

The getUnit method should return the singular name of the unit if the length is 1.0, else it should return the plural name.

The toMeters method converts the length of this object to meters. Use the international standard conversions:

* 1 inch = 0.0254 meters
* 1 foot = 0.3048 meters
* 1 yard = 0.9144 meters

Using any other numbers in your conversions is wrong, and you won't get a good reference from your instructor if you apply to NASA. [One of their missions failed because of incorrect conversions.] You should use constants to store these numbers, e.g., the Inch class should have a constant:

/\*\*

\* 1 inch = 0.0254 meters

\*/

public static final double METERS\_PER\_INCH = 0.0254;

The add method needs to be able to add any kind of length (other) to the object being called (this). You need to call other with toMeters, convert meters to the unit of this object, and add the result to this's length (using the getLength and setLength methods).

### Summing the Lengths

To sum the Lengths, you should create Meter, Inch, Foot and Yard objects with a value of 0, and then add all the Lengths to each of these objects. Your main method will compute these sums in two ways: once summing from the first Length to the last Length and the other summing from the last Length back to the first Length. Because of roundoff, you do not get exactly the same results.

### Reading in the Data

Assume that the name of the file is data.txt. In Eclipse, the file needs to be in the top directory of the project. Reading all the lengths in the file should be accomplished by a sequence of code like:

Scanner in = null;

try {

in = new Scanner(new File("data.txt"));

} catch (FileNotFoundException exception) {

throw new RuntimeException("failed to open data.txt");

}

// need more code for other parts of this assignment

while (in.hasNextDouble()) {

double value = in.nextDouble();

String unit = in.next();

Length length = null;

// code here to use the value of unit to create the

// right type of Length object and store it in length.

System.out.println(length);

// need more code for other parts of this assignment

}

// need more code for other parts of this assignment

### Comments

Create javadoc comments for your Java classes and your methods, constructors, and comments. This is not as much makework as it sounds as methods in the subclasses of Length will inherit the javadoc from Length.java.

### UML Class Diagram

Create a UML class diagram similar to that shown in the [Chapter 10 notes](http://cs.utsa.edu/~cs3443/ch10.html) for this lab. This class diagram should include the Lab4 class, the Length class, and all the Length subclasses. Each class should list all the instance variables, constants and methods in that class. Note that the abstract methods in the Length class should be italicized.

### Rubric

* An incorrect submission will possibly get zero points. A project that does not compile will receive at most 50 points total.
* (80 pts.) The output is correct (no extraneous output), mostly produced by Strings coming from Length's toString method.
* (10 pts.) There are javadoc comments: at least one for each class and comments for each method, constructor and constant. An exception is the add, getUnit and toMeters methods in subclasses of Length; these will inherit javadoc comments from Length.java.
* (10 pts.) A UML class diagram is included that has all the classes (including the Lab4 class) and all the constructors, methods, variables and constants of each class.