Learning Outcomes

Upon completing this assignment, students should be able to

- Create a *binary search tree* container class with *add*, *find*, and *toString*
- Store objects in a collection that behave *polymorphicly*.
- Use `git` to commit each "chunk" of working code. (Make proper use of `git`/version control.)

Introduction

Students will write a `OrderedShapes` program that reads a file full of `Shape` objects (class hierarchy below) and inserts them into a *binary search tree* according to their **area**. Then the user can enter commands to search for a shape with a given area, print out the whole collection in increasing order by area, print out the whole collection in decreasing order by area, or quit the program.

Note: Your use of `git` is part of the rubric for this program. You are expected to build your features one at a time, adding the new/changed *working* code to your `git` repository as you go along. There should be a fair number of commits in this repo, each with a **commit message** stating what was completed (this can go in the first line of the message; no need to write a novel).

Second note: the *right* way to do testing is to write tests for each feature as you go. This would be a good chance to practice this **but** you are not obligated to do this.

You can associate your local repo with your repo for turn-in on Gitea at any point. When you push the repo, the whole history of the project (all commits along with the date/time, commit message, and user who made the commit) is pushed. If you have problems pushing changed code, let Dr. Ladd know and he can help interpret `git` error messages.

Method

Getting Started

Start by creating a new directory for your solution to `F23-205-<ccid>-p005` (you **will** name the repo on Gitea with that format; call the directory whatever you want).

Initialize the directory with Gradle. Name the project `tree` and have the source package be `main`. Modify the created project so that the main class is called `main.TreeClientProgram` and so that `gradle run` passes the keyboard in to the running program.

Compile and run the hello world created by Gradle so you feel like your code works.

`menu.Menu`

Note: This is to use your simple, text-based `Menu` class. This is *not* to use JavaFX, Swing, or any other Java GUI. If you need assistance with `menu.Menu`, talk to the instructor.

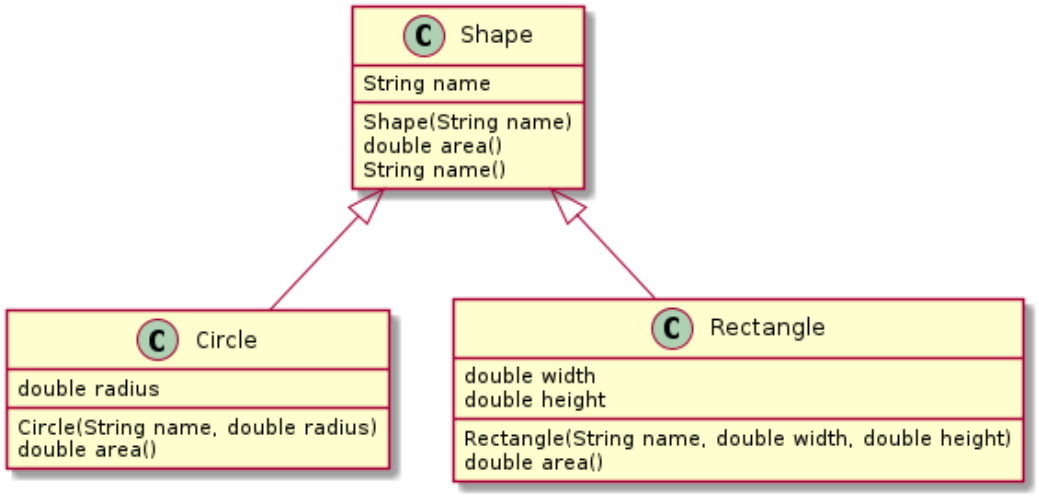
Add `menu.Menu` (from the first assignment in this class) to the project. There is only one menu for this program:

```
find - find a shape by area
dec - list all shapes in decreasing order by area
inc - list all shapes in increasing order by area
quit - quit the program
```

Add this menu to `main.TreeClientProgram` while modifying it to use our standard create-and-run structure. For now, the `run` method should loop, calling `match` on the menu until the user quits. There is nothing else to do right now.

Class Hierarchy

You will create a new package, `shapes` and put three classes in it. The classes are related as shown in the diagram below:



`Shape` The ultimate parent of the hierarchy. Keeps track of the *name* of the shape (an identifier) and has an `area` method that returns zero.
`Circle` Keeps a *radius* and overrides `area` to return the area of the circle.
`Rectangle` Keeps *width* and *height*, overrides `area` to return the area of the rectangle.

These are very short classes. They should have `toString` methods that returns the *type* of the shape, its *name*, and any specific information for the given type.

Input/Output

Modify your program to take a *shape file name* on the command-line. Your program must handle the **no-file-named-on-the-command-line** error before calling the `TreeClientProgram` constructor.

The constructor should take and save the data file name.

Modify `run` to load data from the file of shapes. Right now you do not have anywhere to store them. The load method opens the file and handles the **file-not-found** error.

The data file has the following format:

```
Shape
unspecified-shape-1
Circle
small-circle
1.5
Rectangle
square-B
12 12
Circle
big-circle
22.8
```

Notice that each shape begins with the name of the class on a line by itself. The next line is the name of the shape. A `Circle` then has a line with a radius on it and a `Rectangle` has a line with the width and height.

At this point, you can read the file and test your constructors. Next you will build a tree to hold the `Shapes` you read.

Binary Search Tree

Add a new package, `tree`. Put the class `BST` in the package. The class interface (the public functions it should have) is:

```
class BST {
  public BST();
  public void add(Shape newbie);
  public Shape find(double areaToSearchFor);
  public String shapesInDecreasingOrder();
  public String shapesInIncreasingOrder();
```

Feel free to have `private` helper methods.

Use the `toString` methods you put in the `Shape` classes to assemble the in-some-order strings.

Modify `TreeClientProgram.run` to put the shapes in a new tree and the menu handler to call the right methods on the tree.

Design Considerations

The tree must be kept ordered as stated above.

There are six (6) Java files across four (4) packages. No one of them is very long (and `menu.Menu` is already written). Work a little bit at a time, checking that it compiles and runs as you go.

Documentation

README

Must document how you tested. How do you know that it is right?

Must include instructions on how to **compile** and how to **run** the program as submitted.

Deliverables

Submission medium: `git` to Gitea at `cs-devel.potsdam.edu`.