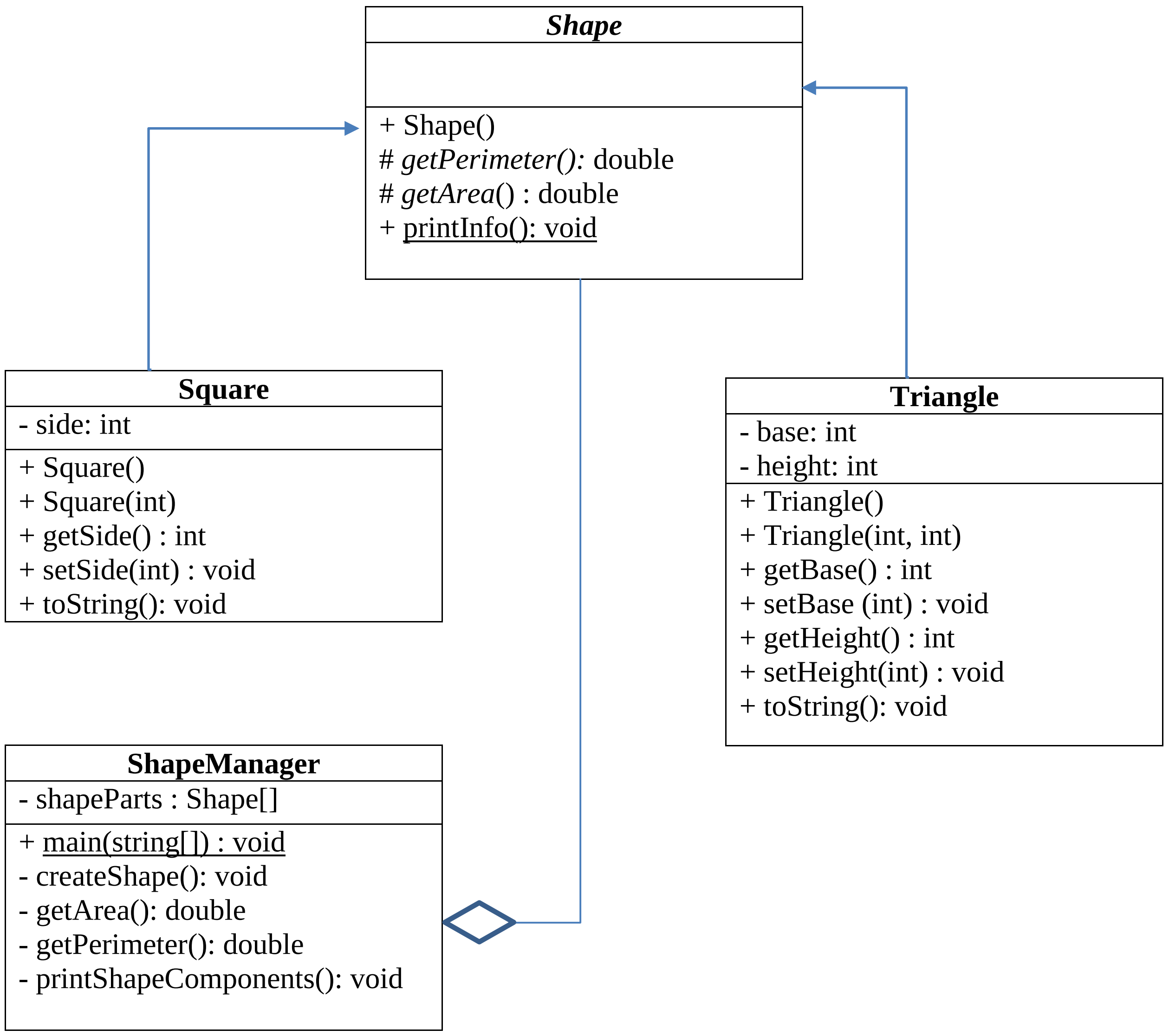
W. P. CAREY
SCHOOL *of* BUSINESS
ARIZONA STATE UNIVERSITY

CIS 494 – Business Systems Development with Java – Spring 2015

In-class Exercise 2: Object-Oriented Design

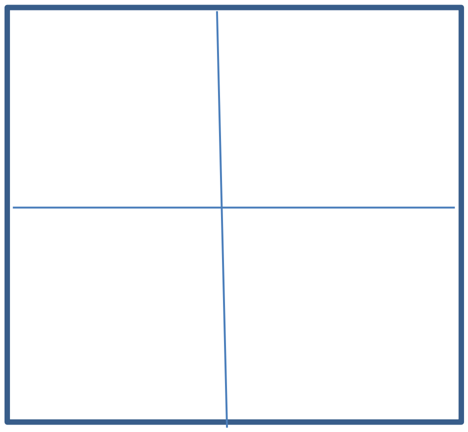
Due on Blackboard: Tonight, January 28 by 11:59 PM

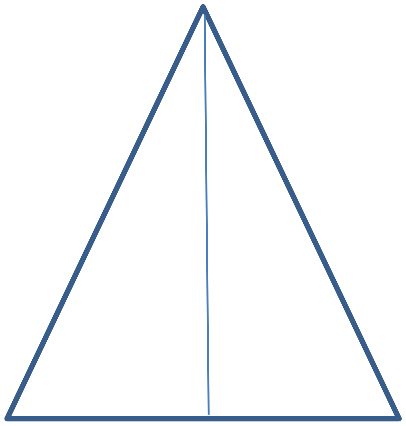
This purpose of this assignment is to give you practice in implementing abstract classes, concrete classes, implement inheritance, and use polymorphism.



In this project, there is a Shape abstract class, which defines properties of all Shapes. Square and Triangles are shapes, therefore they inherit from the Shape class.

The ShapeManager class maintains what are known as “Big Shapes.” A Big Square consists of smaller parts or components, namely 4 small instances of Squares. A Big Triangle similarly consists of 2 smaller Triangles.


Big Square


Big Triangle

The user of the program will specify if a Big Square or a Big Triangle should be created. Therefore, as the program is being developed, the developers do not know which Big Shape will need to be created at run-time. Accordingly, all the management of Shapes has to be done at a “higher” level of generic Shapes, i.e. the program will use an array of Shapes, not arrays of Squares or Triangles. The program will then utilize ***polymorphism*** to manage the actual shape objects, converting Squares/Triangles ***up and down the inheritance hierarchy*** into generic Shapes as needed.

Shape Class

Create an "abstract" class Shape with two abstract methods as specified in the UML diagram. Abstract methods CANNOT have a body. They should have no code blocks. There should be a semi-colon after the method declaration. Any class containing an abstract method must be prefixed with the keyword abstract.

Abstract classes cannot be instantiated and can only be used as a ***superclass*** (parent class). Other classes must inherit from the abstract class, implement the abstract methods, and then these ***subclasses*** (the child classes) can then be instantiated.

<u>printInfo Method</u>

- Put in a solitary println statement that outputs that this is a generic shape.

Square Class

Implement the Square class such that it inherits from the Shape class. You must use the "extends" keyword to implement inheritance.

- Since the Shape class is abstract, any class inheriting from it <u>**must**</u> implement any methods marked abstract in Shape. Accordingly, implement the getPerimeter() and getArea() methods.

  For each of the two methods, return the values as doubles. Make sure you use the same signature and return type as the methods in Shape. Use "@Override" to indicate that you are overriding the inherited methods.

- Try implementing and overriding a printInfo() method, which is static in the Shape class. *Note that the JVM should <u>not</u> let you override a static method. Comment out your method when you get an error.*

- Implement and override the toSring() method, which is non-static. The compiler should let you override this inherited non-static method. This method should return a string that says "Square with side x", where x is the value for its side.

Triangle Class

Implement the Triangle class such that it inherits from the Shape class. You must use the "extends" keyword to implement inheritance. Implement the instance variables, base and height, get/set methods, and constructors.

- Since the Shape class is abstract, any class inheriting from it **must** implement any methods marked abstract in Shape. Accordingly, implement the getPerimeter() and getArea() methods.

  For each of the two methods, return the values as doubles. Make sure you use the same signature and return type as the methods in Shape. Use "@Override" to indicate that you are overriding the inherited methods.

  Use the following formula for the perimeter of a triangle. Utilize the static .sqrt() method of the Math class to calculate the square root. Use the .pow() method of the Math class to calculate the exponents (give number and power as arguments).

$$Perimeter = base + height + sqrt\ (base^2 + height^2)$$

- Implement and override the toSring() method, which is non-static. The compiler should let you override this inherited non-static method. This method should return a string that says "Triangle with base x and height y", where x and y are the triangle's base and height.

Shape Manager Class

The ShapeManager class should contain your main method and is where the main program resides. ShapeManager has an "aggregation" relationship with Shape, illustrated with the empty diamond. This means that one ShapeManager will have many instances of Shape objects. Aggregation is a *part-whole* relationship in design terms. This entails that the whole (ShapeManger) is comprised of the parts (Shape) but that the whole *can* exist without the parts.

CreateShape Method

Description: This method should prompt the user and ask her if she wants a Big Square or Big Triangle. If the user wants a square, it should create a shape array of size 4, with each element being an instance of the Square class. If the user wants a triangle, it should create a shape array of size 2, with each element being an instance of the Triange class.

- Prompt the user and ask if the user wants a triangle or a square. The user is expected to enter an “S” or “T”.
- Create an instance of the Scanner class and use the nextLine() method to store the answer in a new variable.
- Use the .equalsIgnoreCase method of the String class to compare the user answer.
  If the user wants a Square
    - Ask the user for the length of the side
    - Read and store the side length. Remember to parse as an integer!
    - Create the shapeParts array as an array of **Shape** (not Square!), with **4** elements.
    - Loop through the entire shapeParts array.
        - Create instances of Squares using the Square constructor that accepts the side. Pass the length of the side as an argument. Assign the instance of the Square to the current element of the array (use index i). *This is polymorphism in action. We are assigning a Square to an array of Shapes. This is known as upcasting.*

  Else if the user wants a Triangle
    - Ask the user for the length of the base and height
    - Read and store the base and height. Remember to parse as an integer!
    - Create the shapeParts array as an array of **Shape** (not Triangle!), with **2** elements.
    - Loop through the entire shapeParts array
        - Create instances of Triangle using the **no-args** constructor, e.g. **Triangle().** Assign the instance of the Triangle to the current element of the array (use index i).
        - Within the loop, ***try*** to set the base of the current triangle (use shapeParts[i]), by calling the setBase() method.

          You should find that you are not able to do compile the program (if the IDE itself doesn’t generate the error while writing code). This is because the array element is a Shape and Shapes do not have a method called setBase(). So, we need to take the element of the array and store it in a variable of type Triangle. Since Triangles do have a method called setBase(), we will be able to access that method through a different variable.

(Continuing within the loop...)

  - ▪ Declare a variable tmpTriangle of type Triangle. Assign to this, the current element of the array (shapePart[i]). However, we have a type mismatch here. The right hand-side is of a different data type than the left hand-side. So, ***cast*** the array element on the right by as a Triangle by prefixing it with the data type within a set of parantheses, e.g. (Triangle)

    This is polymorphism in action too. We have now take a Shape and ***downcast*** *it into a Triangle.*
  - ▪ Use the setBase() and setHeight() methods of tmpTriangle and pass them the base and height values accordingly.

(Continuing CreateShape Method here…)

- Call printShapeCompoenents
- Print the total area of the shape by calling the getArea() method of the ShapeManager class. This method returns a double. You can pass the return value as an argument straight to the printf() method. Remember to use formatting instructions to format the double. *You MUST at minimum provide it with the formatting instruction that the argument is a float, or the program will crash while running.*
- Print the total perimeter of the shape in the same manner as you just printed the area above.
- Remember to call close() for your Scanner object!

<u>getPerimeter Method</u>

Description: This method should return the total perimeter of all the parts in the array.

- Loop through the entire shapeParts array. Call the getPerimeter() method of all the parts in the array and add them up. Make sure you declare a variable to store the total.
- Return the total

Note: This is not geometrically accurate as it double counts any sides that are common to different parts..

getArea Method

Description: This method should return the total area of all the parts in the array.

- Loop through the entire shapeParts array. Call the getArea() method of all the parts in the array and add them up. Make sure you declare a variable to store the total.
- Return the total

printShapeComponents Method

Description: This method should print to the console a list of all the components/parts of the Big Shape.

All classes have an in-built method called "toString()" that returns a string representation of that class. Use the toString() method to output a list of all the parts.

- Loop through the entire shapeParts array
  - Call the toString() method of each element of the array and print that string to the Console.

Main Method

- Call createShape()!

Learning Points for Abstract Methods

- At what places do you need to use the `abstract` keyword?
- Try deleting the `abstract` keyword from the class declaration and compiling/running the project? Does it let you? If not, why not?
- Note the structure of the abstract methods. Is there a body? Are there traditional method blocks?
- Try adding a method block to an abstract method. Does it let you?

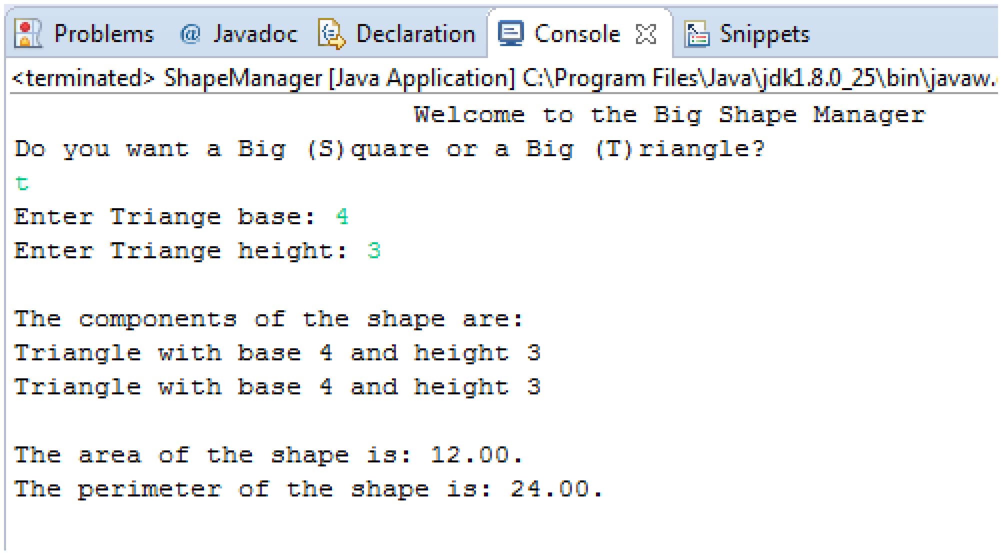
Learning Points for Inheritance

- Change the accessibility level of the abstract methods of Shape to private. What happens?
- Can you access private methods of the superclass from the subclasses?
    a. Are you able to access private methods of the superclass by using super.methodName()?
- You are able to override non-static methods but not static methods!

Learning Points for Polymorphism

- You were able to assign a Triangle/Square to a variable of type Shape (array of Shapes). Try doing it the other way around. Can you assign a Shape to a variable of type Triangle/Square?
- While you can store a variable of type Triangle (subclass) in Shape (superclass), you will find that you will be able to access methods from Triangle (subclass) only through a variable of that subclass. So, you should always downcast a superclass variable into a subclass variable when you want to treat it like a subclass variable.

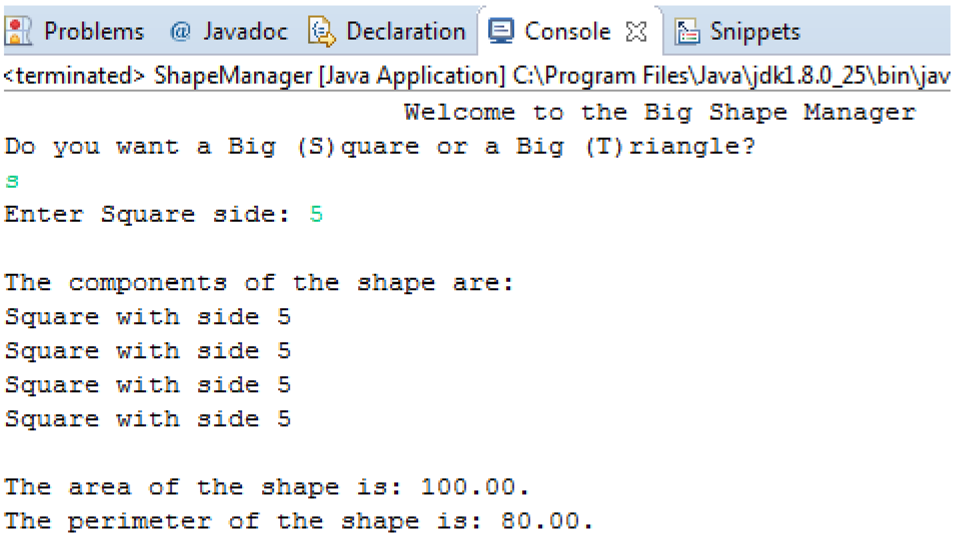
Sample Output



```
<terminated> ShapeManager [Java Application] C:\Program Files\Java\jdk1.8.0_25\bin\javaw.
                          Welcome to the Big Shape Manager
Do you want a Big (S)quare or a Big (T)riangle?
t
Enter Triange base: 4
Enter Triange height: 3

The components of the shape are:
Triangle with base 4 and height 3
Triangle with base 4 and height 3

The area of the shape is: 12.00.
The perimeter of the shape is: 24.00.
```



```
<terminated> ShapeManager [Java Application] C:\Program Files\Java\jdk1.8.0_25\bin\jav
                          Welcome to the Big Shape Manager
Do you want a Big (S)quare or a Big (T)riangle?
s
Enter Square side: 5

The components of the shape are:
Square with side 5
Square with side 5
Square with side 5
Square with side 5

The area of the shape is: 100.00.
The perimeter of the shape is: 80.00.
```

<u>Submitting Files</u>

Submission should be made using a zip file that contains the entire Eclipse project folder. You will need to ***zip the entire project folder***. The folder will automatically contain the class source files as well as the compiled .class files.

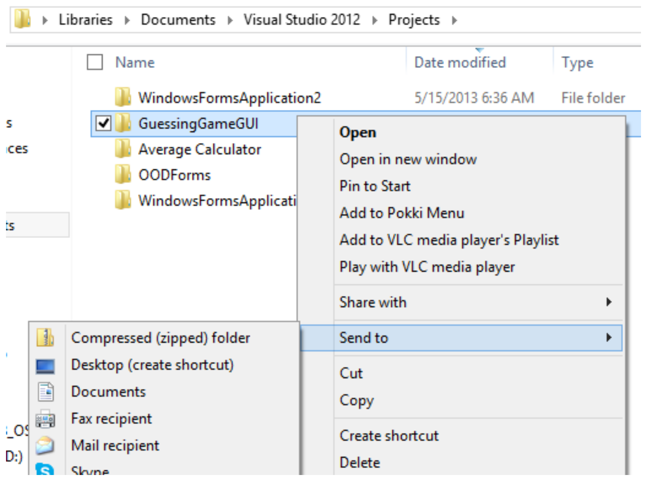
ZIP file should be named: **AX.zip or IC.zip**

where X is the in-class assignment number, e.g., A1.zip is the submission file for Assignment 1 and IC1.zip is the submission file for InClass1.

Note: *The ZIP filename is <u>independent</u> of the Project name. Do not name your project A1.*

How to Properly ZIP and Submit Your Project Files:

- Go to the folder within {Eclipse Workspace}\
- ZIP the entire top-level folder for your project by right-clicking your project folder and selecting Send to | Compressed (zipped) folder.
- Finally, submit the ZIP file using the submission link on Blackboard by the due-date and time listed on the assignment. Upload the ZIP file.



Using built-in windows zip tools: http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files

Verify your files BEFORE and AFTER submission:

- Check for actual class files being present in the folder before you zip it.
- ***Ensure that you are not zipping a short-cut to the folder.***
- After zipping, check file size. A file size under 4K likely does not contain all the files.
- Unzip, extract all files, and verify you see actual files, not a solitary short-cut.
- Uncompress your zip file before submitting and verify that files are present.
- Make sure you have ***submitted*** your file and not just saved a draft on Blackboard. ***A blue clock indicates a submission in progress, i.e. a draft, not a submission. The draft is accessible only to you. You will get a ZERO if you only ever save a draft on Blackboard and never submit your files.***

- Download your zip file after submitting, uncompress, and again verify that your files are present. Test your files in Visual Studio after uncompressing.

This takes an extra couple of minutes. Please do it if your grade is important to you. If you do this, you will not end up submitting a bad file. If you submit an empty file, or one containing only a shortcut, or a bad zip file, or a bad project file, you will receive a score of zero and your only recourse will be to do the makeup assignment at the end of the semester.