Version Control System

~~Before starting the lab, go to GitHub by using your account and create your private repositories called CS102\_lab03\_a and CS102\_lab03\_b.~~ ~~Whenever you are done with one of the following parts of the lab you will need to add all of your changes, commit them, and push them to the corresponding~~**~~private~~**~~and remote repository.~~ ~~Make sure that your repositories are indeed private because if they are not then anyone on the web will be able to copy your homework, which will get you into trouble. For every part of the lab, you should have at least one commit that has a clear message of what was implemented in that commit and for which part.~~

(a) Shape

**(1)** ~~Design a class hierarchy to include classes Shape, Rectangle (with int sides, width, and height), Circle (with int radius ), Square (with int side), Triangle (with int sides called a, b, c). The Shape should be abstract with method double getArea(), getPerimeter()~~ ~~(Perimeter is a generic name, for circles, it's called the circumference), and Square should inherit getArea() from Rectangle. You can find formulas for the perimeters of some shapes~~[~~here~~](https://byjus.com/perimeter-formulas/)~~. You can find an area of a triangle by just knowing it's sides.~~

**(2)** Create another class, ShapeContainer, to hold a set of shapes. It should have methods void add( Shape s) and double getArea(), double getPerimeter() and String toString(). Write a ShapeTester class with a menu that allows the user to create an empty set of shapes (ShapeContainer), add as many shapes to it as they wish, compute & print out the total surface area of the entire set of shapes, and perimeter, and print out information about all of the shapes in the container by calling the toString() method for each shape. Experiment. Try to predict what would happen when you (i) comment out the getArea() method of the Circle class, and (ii) also make the Circle class abstract, before finally (iii) creating an instance of the (now abstract) Circle class to add to the shapes collection. Test your predictions.

**(3)** The customer is impressed with your work so far, and so asks you to extend the program. They want Shapes to be locatable (i.e. to have an x, y location, and getX(), getY(), and setLocation( x, y) methods). As a good designer you decide to first create a Locatable interface with these methods, then have the Shape class implement it. In this way all shapes automatically become locatable.

**(4)** Impressed, the customer wants even more! This time they ask for shapes to be Selectable, so you again start by creating a Java interface like for Locatable, having boolean getSelected() and setSelected( boolean) and Shape contains( int x, int y). Create a Selectable interface with these methods, then have the Shape class implement it. Change the toString() method of shape class so it shows whether the shape is selected or not. Add another option to your ShapeTester menu that allows the user to find the first Shape that contains a given x, y point and toggle its selected state. Provide another menu option that removes all selected shapes from the set of shapes. Good design practice suggests you should ask the ShapeContainer object to do the work of finding the first Shape containing the given point and of removing all selected shapes (rather than trying to do the work yourself in the ShapeTester class, which might/would require knowledge of the insides of the ShapeContainer class).