1. Modify the following code from factory pattern to Abstract Factory Pattern (only send Abstract Factory Pattern). All files & UML packaged in a zip folder.

Attachments: 04factoryPattern.pdf (Do not send this code-modify this code for abstract factory)

Your job is to refactor the code into an appropriate hierarchy. Utilize Dependency Inversion to find commonalities across the items and build an appropriate abstraction. Through your abstraction you should be able to ask any concrete item what its name (shape) is. You should NOT use a field for the area, provide the area as a behavior (why might this approach be important with regards to design?). Place all these items in a package named shape.

Once the above is done, build a simple factory (ShapeFactory) that has a createShape method (or methods if necessary), that takes necessary parameters, builds an appropriate shape(doesn't draw the shape, you just make a method called draw that returns), then returns it.

Finally, create and Submit an AbstractFactoryPattern that asks the factory for multiple shapes of each type.

Attachment Instructions for assignment: 04FactoryAbstractFactoryPatternStudentInstructions.pdf (Submit this code modified/refactored)

Output should look like this:

Inside Rectangle::draw() method.

Inside Square::draw() method.

Inside RoundedRectangle::draw() method.

Inside RoundedSquare::draw() method.

Turn in the following zip file labeled cscd212AbstractShapeFactory_firstname_lastname

NOTE: Nothing outside the package should be allowed to directly create a shape -- it must be done through the factory. Build your solution such that this is enforced.
all compiled source files
a capture of the output of your tester -- name that file ShapeTesterOutput -- if the capture is an image, make sure the format is .pdf, .png, or .jpg.
a UML diagram that represents all the items in the package