

CS 251L Homework 4

Object Orientation: Drawing Model Objects

Assigned: October 19, 2011

Due: October 26, 2011 @ 12:00pm (submitted before class starts)

Submission Method: Email solution to wokoun@unm.edu by due date.

Please identify your submission in the email subject header as being homework 4. You only need to attach the source files for your source files (not the compiled classes) to your email. You may attach multiple source files separately or archive them using a mainstream archiver (.zip, .sit, .tar.gz, etc.). If you are sending loose files, please create a proper email attachment rather than copy/pasting one after the other. If you send an unreadable file or archive, the instructor will generally notify you that you need to resubmit it, but the homework is not considered turned in until the instructor gets a readable file.

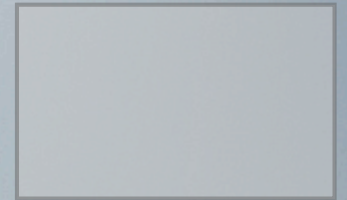
This is one of several assignments that will culminate in a simple drawing application. The application will have the ability to save and load drawings from a file and display them on the screen. In order to accomplish this, we will need to create objects that have the ability to save/load/paint themselves. The objects will have the data inside themselves to accomplish these tasks in response to messages to perform these actions.

This assignment is to create the data “model” for the drawing application, consisting of the nouns (objects), verbs (methods), and data required to create the application. The application will support four shape primitives (shown at right), which will each be a class. These four classes will have base classes, interfaces, and support classes to help them accomplish their jobs.

The assignment is to create the classes and interfaces as described on the next pages and then test them as described.



Line



Rectangle



Polygon



Ellipse

CS 251L Homework 4

For all classes:

- Create a constructor that takes no arguments. We'll need to use it later for I/O.
- Create a constructor that takes as arguments the various geometry data required to specify the position of the shape (do this step for concrete classes only).
- Create accessors and mutators (getters and setters) for each of the fields.
- Create a **toString()** method that outputs the names and values of all variables.
- Add class-level Javadoc comments to all classes and interfaces.

Paintable:

An interface that defines an object that can paint itself on the screen.

- Declare method: **paint()**

Storable:

An interface that defines an object that can save/load itself to/from a file.

- Declare methods: **save()** and **load()**

Shape:

An abstract class that will contain common shape properties and form a base for subclassing.

- Inherit interfaces: **Paintable** and **Storable**

Point:

A class that describes a point on a Cartesian (X-Y) plane. This class will be used to build up other shape types.

- Implements **Storable**
- Create fields: **x** and **y**, both of primitive type **double**

CS 251L Homework 4

Line:

A class that describes a line between two **Points**.

- Extends **Shape**
- Create fields: **p1** and **p2**, both of type **Point**
- Add empty methods for **paint()**, **save()**, and **load()**. These will be finished later.

Rectangle:

A class that describes a box anchored at some point and having a defined height and width.

- Extends **Shape**
- Create fields: **topLeft**, of type **Point**, and **width** and **height**, both of type **double**
- Add empty methods for **paint()**, **save()**, and **load()**.

Ellipse:

A class that describes an ellipse bounded by the specified rectangle coordinates. (This is how ellipses are specified on computers, rather than the way you've seen in math.)

- Extends **Shape**
- Create fields: **topLeft**, of type **Point**, and **width** and **height**, both of type **double**
- Add empty methods for **paint()**, **save()**, and **load()**.

Polygon:

A class that describes a polygon as a series of vertices.

- Extends **Shape**
- Create field: **vertices**, of type **Point[]**
- Add empty methods for **paint()**, **save()**, and **load()**.

Testing

In a **main()** method on **Shape**, create one instance of each of the four types of shapes (using geometry parameters of your choosing), then print each object with **System.out.println()** to see its fields.

Class Diagram

