Summer Training 2009

Second Year

OOD: design and develop better

Lab4

**Goal: Apply Model-View architecture pattern to our GraFICS**

# Quick Intro to Design Patterns

1. What’s Design Patterns?!
2. Why Design Patterns?!
3. Categories of Design Patterns
   1. Architectural Patterns (Model-View-Controller **MVC**)
   2. Creational Patterns (Singleton, Factory, Abstract Factory)
   3. Behavioral Patterns (State, Command, Mediator)
   4. Structural Patterns (Composite)
4. Model/View Concept…Why?!
   1. Examples
      1. TV + Antenna
      2. PC + LCD + Data Show
      3. Excel data sheet + Chart
      4. Shape + GDI + OpenGL

# OOD Principles that are violated in Our Code

1. **Single Responsibility:**

Which is violated in the all shape classes since each class is currently responsible for two things: drawing and geometric of the shape

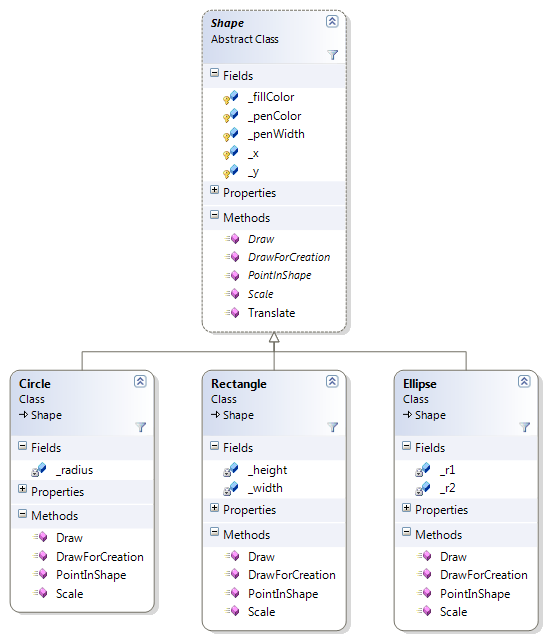
**Solution:** Model-View architecture *(this section)*

1. **Open for Extension, Closed for Modification:**

This is violated in the **mouseDown** event as we check the shapeToBeDrawn against all shapes types. So when a new shape is added to the project, we need to **MODIFY** the **mouseDown** event which violate this principle

*(Imagine that we add 100 shapes to the project… what will be your case!! )*

# Review class diagram until now!!

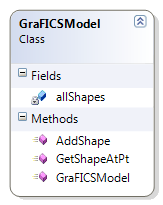


# Part1: Create a Model-View Architecture

## Hands-on: Switch current code to Model-View architecture

1. Creating the Model
2. Creating the View
3. Connecting the View with the Model

### FIRST: Creating the Model:



* 1. Create a new model class (**GraFICSModel**)
  2. Move the **allShapes** list from Form1 to **GraFICSModel**
  3. Add the following methods to **GraFICSModel**
     1. **public GraFICSModel():**

Default constructor… create a new instance from the list

* + 1. **void AddShape(Shape newShape):**

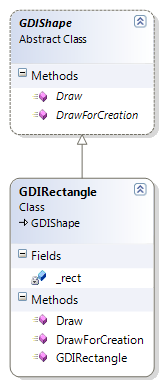
Add the given shape to the list

* + 1. **Shape GetShapeAtPt(int x, int y):**

Search the list and check if **PointInShape**… if so, then return the shape… else return null

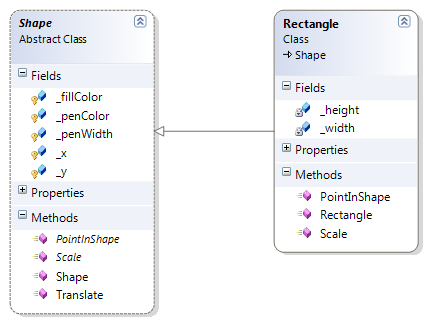
### SECOND: Creating the View:

Create the following class diagram: "\_**rect** is an object of **Rectangle** class"



1. Create a constructor that takes an object of **Rectangle** class and initialize **\_rect** by it
2. Move the **Draw** and **DrawForCreation** from **Rectangle** class to **GDIRectangle**

Now, the class diagram of the **ShapesLib** classes should look like the following:



Note that both **Draw** and **DrawForCreation** are removed from Shape class… this ensures the **Single Responsibility Principle**

### THIRD: Connecting the View (Form1) with the Model:

1. Add the following members to the view (**Form1**)
   1. Object from **GraFICSModel** and name it **Model**
   2. List of **GDIShape**'s to hold a **GDIShape** object for each **Shape** in the **Model**… name it **allGDIShapes**
   3. Object from **GDIShape**, name it **tmpGDIShape**, corresponding to **tmpShape** that is used for temporary drawing the shape during its creation
2. Apply the following modifications to **mouseDown**
   1. In SELECTION mode section, call the **GetShapeAtPt** method from the **Model** to get the selected shape (if any)
   2. In DRAWING mode section, create a new instance from **tmpGDIShape** and pass to it the created **tmpShape**
3. Apply the following modifications to **mouseUp**
   1. Add the **tmpShape** to the **Model**
   2. Add the **tmpGDIShape** to the **allGDIShapes** list
   3. Set the **tmpGDIShape** to null
4. Apply the following modifications to **Paint** 
   1. Loop on the **allGDIShapes** instead of old **allShapes** *(as* ***allShapes*** *now doesn't responsible for drawing)*
   2. Also, draw the tmpGDIShape instead of old **tmpShape** *(as* ***tmpShape*** *now doesn't responsible for drawing)*

# Part2: Add Polygon Shape and Draw it using Mouse Clicks in Model-View Architecture

## Hands-on OR Assignment according to section time:

### Starting from Section4Part2-Template Folder

Given the **Polygon** class that is inherited from **Shape** class and contains the implementation of all **abstract** methods, DO:

1. Add this shape to your shapes library
2. Create a GDIPolygon class for this polygon and place in it the suitable methods
3. Add a GUI button to the Form and handle the click event correctly
4. Handle the drawing of this shape using **mouseClick** to specify the points of the polygon (each **mouseClick** represents a new point in the polygon)

(Hint: use the **DrawForCreation** method to draw the polygon lines during the creation of it)

1. To finish the drawing, the user should **doubleClick** the mouse

**SHOW THAT:** After this, you will be able to select, translate and change the polygon color without writing any extra code.

# TODO Task for students: starting from Section4Part2-Template

Add to your project a free-Hand polygon shape which is same as the polygon but can contains a free-hand drawing **in addition** to the normal polygon lines, see the following figure for an example of such shape.

**To start the free-hand drawing**, the user should push the mouse button and drag it over the form… all points during the dragging should be added to the polygon… this will make the effect of free-hand drawing.

On the other hand, **to draw the normal polygon line**, the user should click the mouse at the points of the lines… only clicked points should be added to the polygon (same as normal polygon).