

Fitting it Together

Steven R. Bagley

Course Overview

- OO-Basics
- OO Design (breaking the problem down)
- Design Patterns (reusable bit-solutions)
- OO Mechanisms (Under the Hood)

Sketching Application

- Follow the whole process from the design document to the implementation
- Identify objects
- Refine the design — might mean more objects!
- See where Design Patterns can be used

SKETCHING APPLICATION

This sketching application will allow the user to create a Picture by combining a series of Shapes to form the image. The choice of shape can be selected from a menu. Shapes can either be Circles, Rectangles, Triangles or Polygons.

Shapes are constructed by the user using the mouse to draw out the shape, clicking to define each parameter in the shape (e.g. a Rectangle would be defined by clicking once to set one corner point, and then clicking again to set the opposite corner-point, a circle would be defined in terms of its centre and radius). Polygons are ended, by double-clicking to define the last point.

Once defined, a Shape's shape is fixed however its position can be changed by dragging it around the screen. The colour of a Shape can be changed at any point from a menu of eight colours (Red, Orange, Yellow, Green, Blue, Indigo, Violet and Black).

Find Candidate Objects

- `Picture` — Stores the whole picture
- `Shape` — Pictures are made of these
 - `Rectangle`
 - `Circle`
 - `Polygon`
 - `Triangle`

Find Candidate Objects

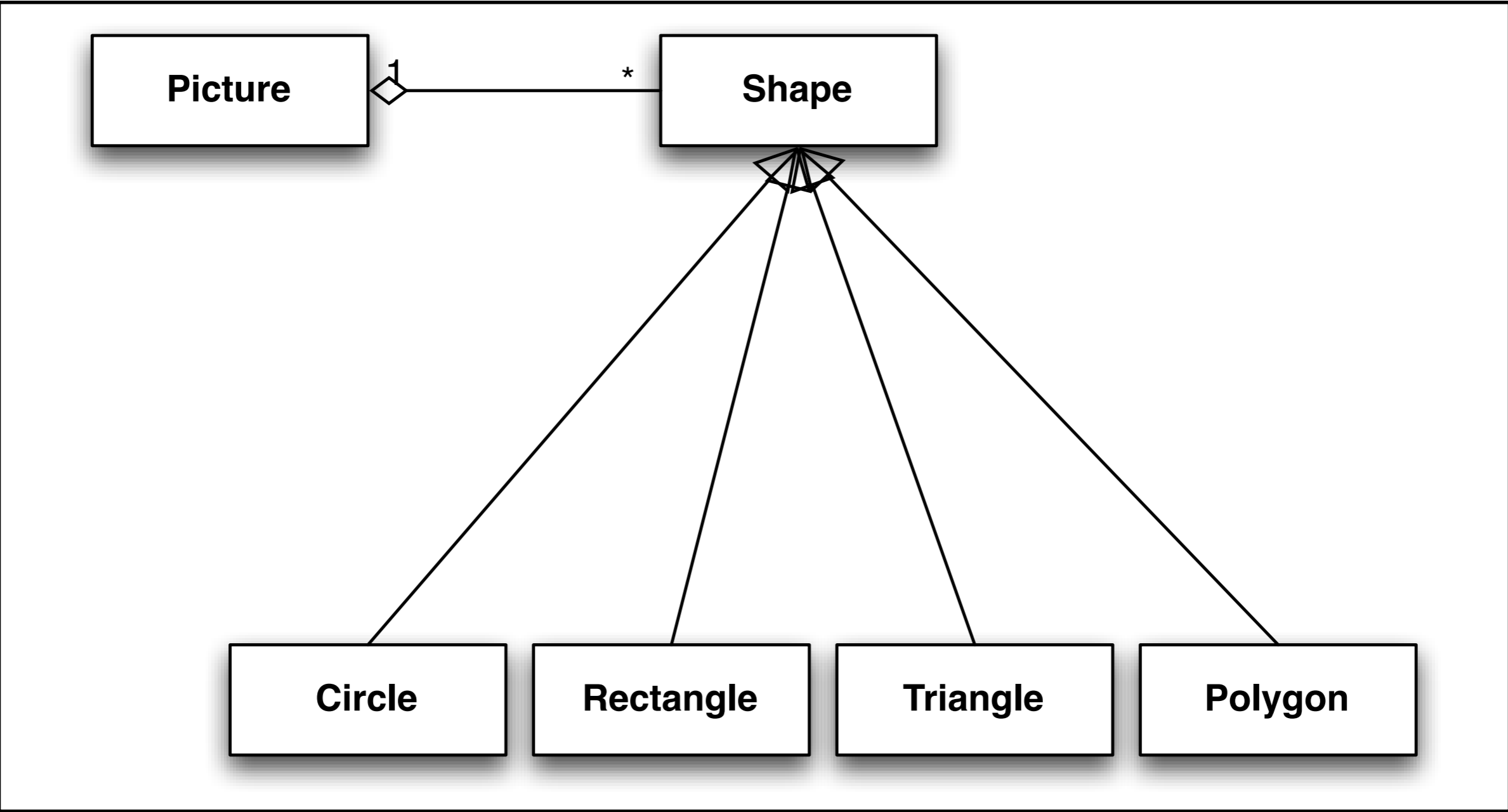
- `Picture` — Stores the whole picture
- `Shape` ← Pictures are made of these
- `Rectangle`
- `Circle`
- `Polygon`
- `Triangle`



Inheritance?

Inheritance

- `Shape` is obviously a base-class
- `Polygon` and `Circle` derive from `Shape`
- What about `Rectangle` and `Triangle`?
- Are they `Polygons` or `Shapes`?



Spotting Methods

- Add Shape to `Picture`
- Remove Shape?
 - How do we specify `Shape` to remove?
 - Pointer to object represents identity
- `Shapes` have colour; Accessor and Mutators
- `Shapes` can be moved

Picture

Operations

AddShape

RemoveShape

Collaborators

Shape

Shape

Operations

SetColour

GetColour

Move

Collaborators

Drawing Shapes

- Do we need a `Draw()` method?
- Call `Draw` on a `Picture`, which calls `Draw` on each `Shape` to draw the image on screen
- Ties implementation to a specific drawing paradigm
- Model-View-Controller

Model-View Controller

- The **Model** represents the data but has no notion of how to display it
- The **View** knows how to display things but has no knowledge of the data
- The **Controller** links the two together, directing the **View** to use the data from the **Model**

MVC and SketchApp

- We'll use MVC, but won't consider how we write the **View**
- Our **Model** is the `Picture` class
- Don't need `Draw` method, but do need accessors for the `Shapes`
- `Shapes` need accessors too...
- `Shape Accessors` will be shape specific

Picture

Operations

AddShape

RemoveShape

GetNumShapes

GetShapeAt

Collaborators

Shape

Circle

Operations

Centre

Radius

Collaborators

Polygon

Operations

GetNumPoints

GetPointN

Collaborators

Rectangle

Operations

BottomLeftPosition

Width

Height

Collaborators

Construction

- Next question though, is how do we create these classes
- Reading the description one thing immediately jumps off the page
- Construction is done in stages
- This implies using the **Builder** pattern

Why Builder?

- Object created over a series of events
- Object not valid until all events completed
- Builder pattern lets us hide object away until construction is complete
- Let's consider a concrete example

Rectangle Building

- User clicks once — starts drawing a rectangle
- User moves mouse until rectangle is of correct size
- User clicks again setting rectangle size
- Implies `RectangleBuilder` will need a method to respond to mouse clicks

RectangleBuilder

Operations

OnMouseClicked

Collaborators

How it works...

- `RectangleBuilder` will implement a *State Machine*
- Mouse clicks will move between states
- When we reach the finish state, we can get the object out of the Builder

Starting State

- Mouse Click in the start state:
 - Deletes any pre-existing object
 - Defines the starting position
 - Moves into State 2

State Two

- Mouse click in State Two
 - Sets the end position
 - Calculates width/height (based on stored start position)
 - Creates `Rectangle` Object
 - Moves back to Start State

Methods and Directors

- Need a method to get the created object
- What will be the director?
- Easy – the **Controller** (another object!)
- Controller receives mouse events from the **View**
- Pass these onto the builder

Other Shapes

- Other shapes can be built in the same way
- `PolygonBuilder` needs to be able respond to double clicks as well
- How does the controller change builders?
- Strategy Pattern?
- Abstract Builder class

AbstractBuilder

Operations

OnMouseClicked

OnMouseDoubleClick

GetShape

Collaborators

Shape

SketchController

Operations

GetPicture

SetBuilder

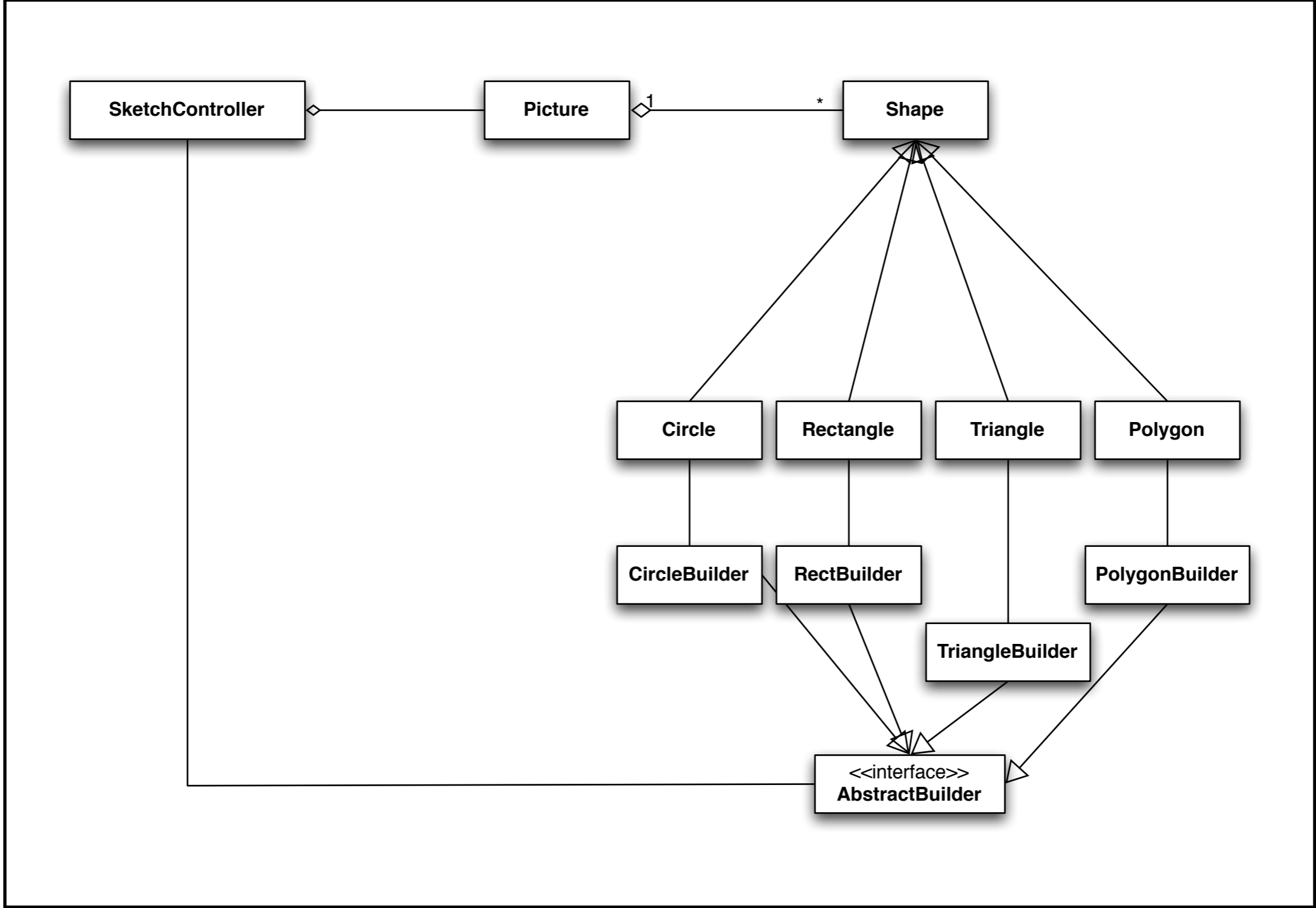
OnMouseClicked

OnMouseDoubleClick

Collaborators

Picture

AbstractBuilder



Dragging

- Last part of the description, we are going to consider is dragging
- Need to know whether the mouse click is over a Shape
- If it isn't, we start building a new Shape
- If it is, then we want to drag
- More methods needed...

Finding Shapes

- Mouse click will give us an x-y co-ordinate
- Need to find `shape` that is under those co-ordinates
- Method on `Picture`, `GetShapeUnderPoint`
- How does it test shapes?
- Shape needs an `IsPointInside` method

Picture

Operations

AddShape

RemoveShape

GetNumShapes

GetShapeAt

GetShapeUnderPoint

Collaborators

Shape

Shape

Operations

SetColour

GetColour

Move

IsPointInside

Collaborators

Implementation

- Don't have to wait till design is finished to start implementing
- Could start implementing by only considering rectangles, and get that working
- Then extend that code to support the other shapes

Conclusion

- Break down the description, this will lead to some objects and methods
- As you consider how these are implemented, you'll create more objects and methods
- Might also remove some — could have considered to do without `Rectangle`, and `Triangle`