SYSC2004

Lecture

Canvas

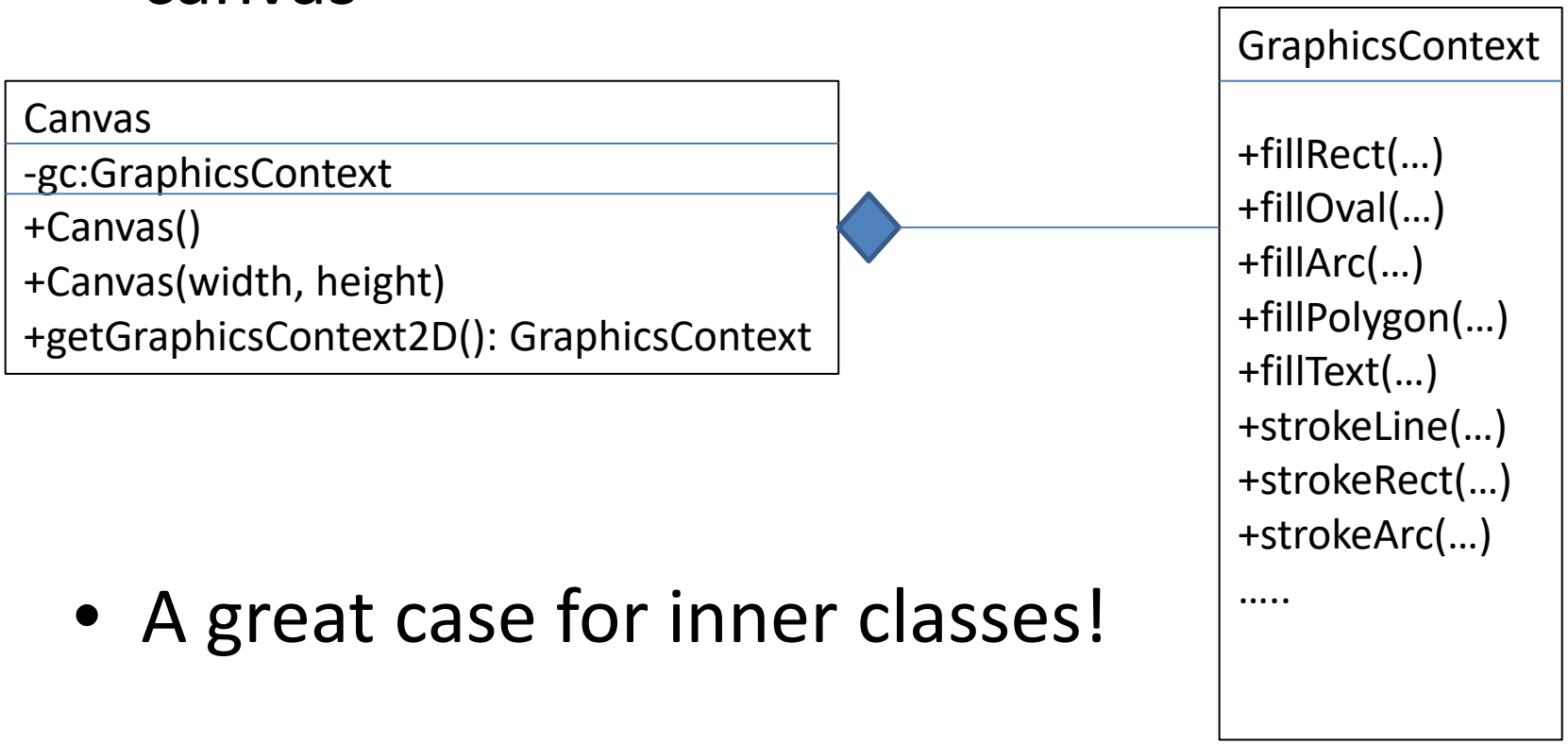
Observable Collections

Background - Canvas

- Canvas is different than most other JavaFX Control objects (which have pre-defined looks)
  - It is an empty space for custom drawing
  - Drawing of rectangles, circles, lines, arcs, points at specified (x,y) pixel locations
- Fun tutorial: https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm

Canvas – Programming Challenge

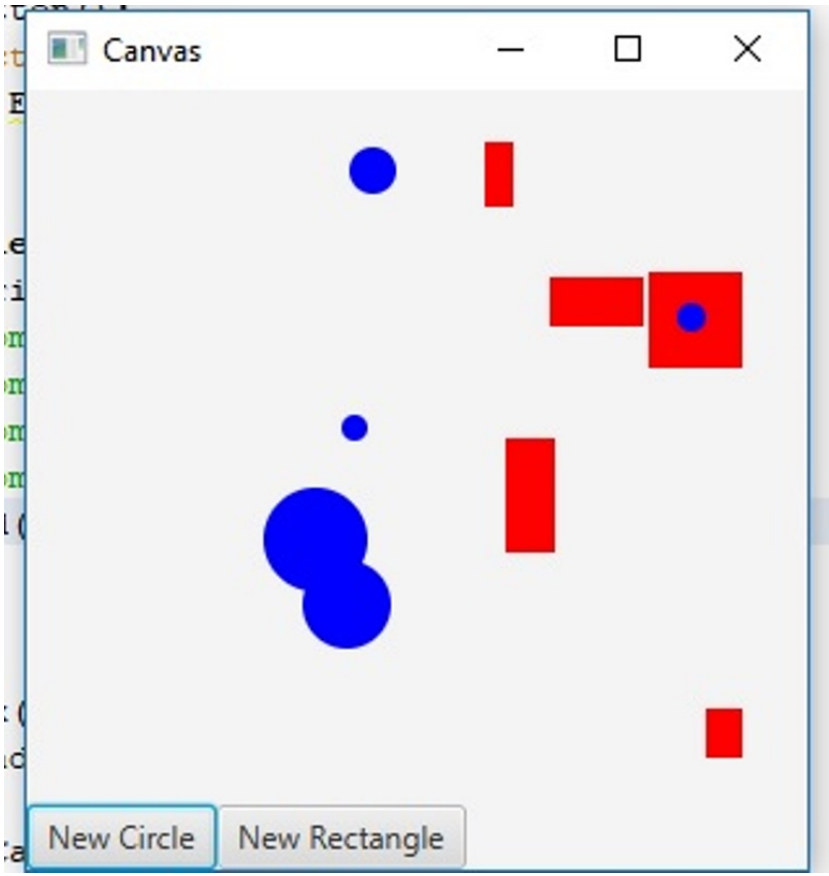
- You do not draw on the canvas; instead,
- You draw on the graphicsContext of the canvas



- A great case for inner classes!

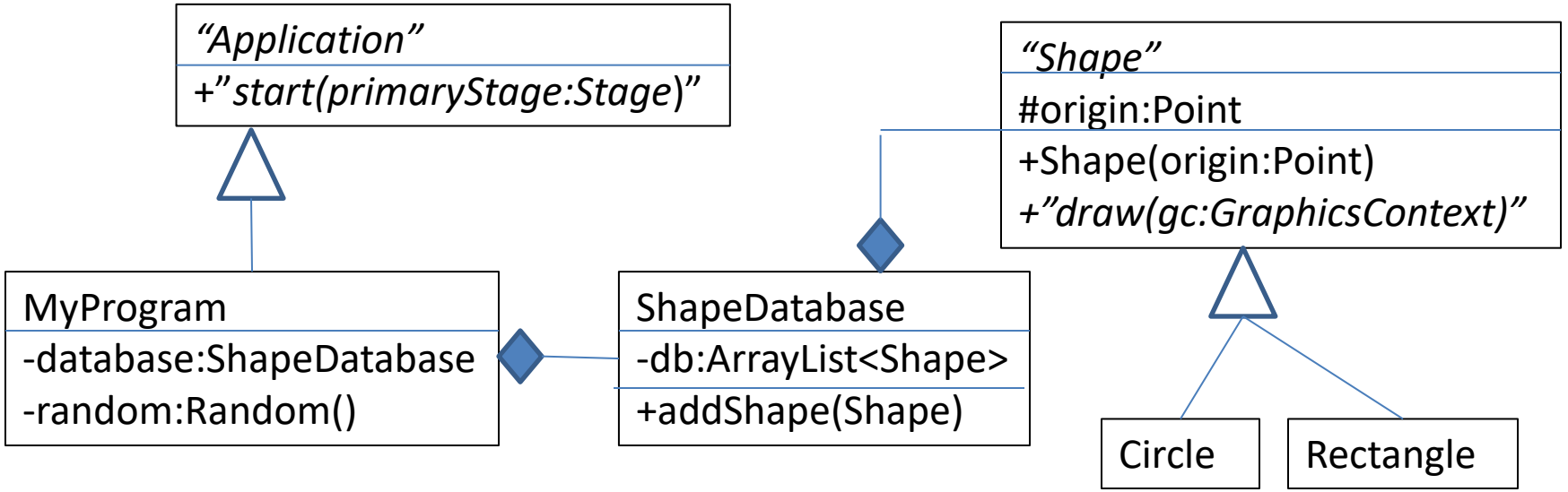
Sample Exam Question

- Write a JavaFX application that allows a user to add – at a click of a button - circles and rectangles of randomly generated sizes at randomly generated locations. Circles should be blue. Rectangles should be red.



Sample Exam Question

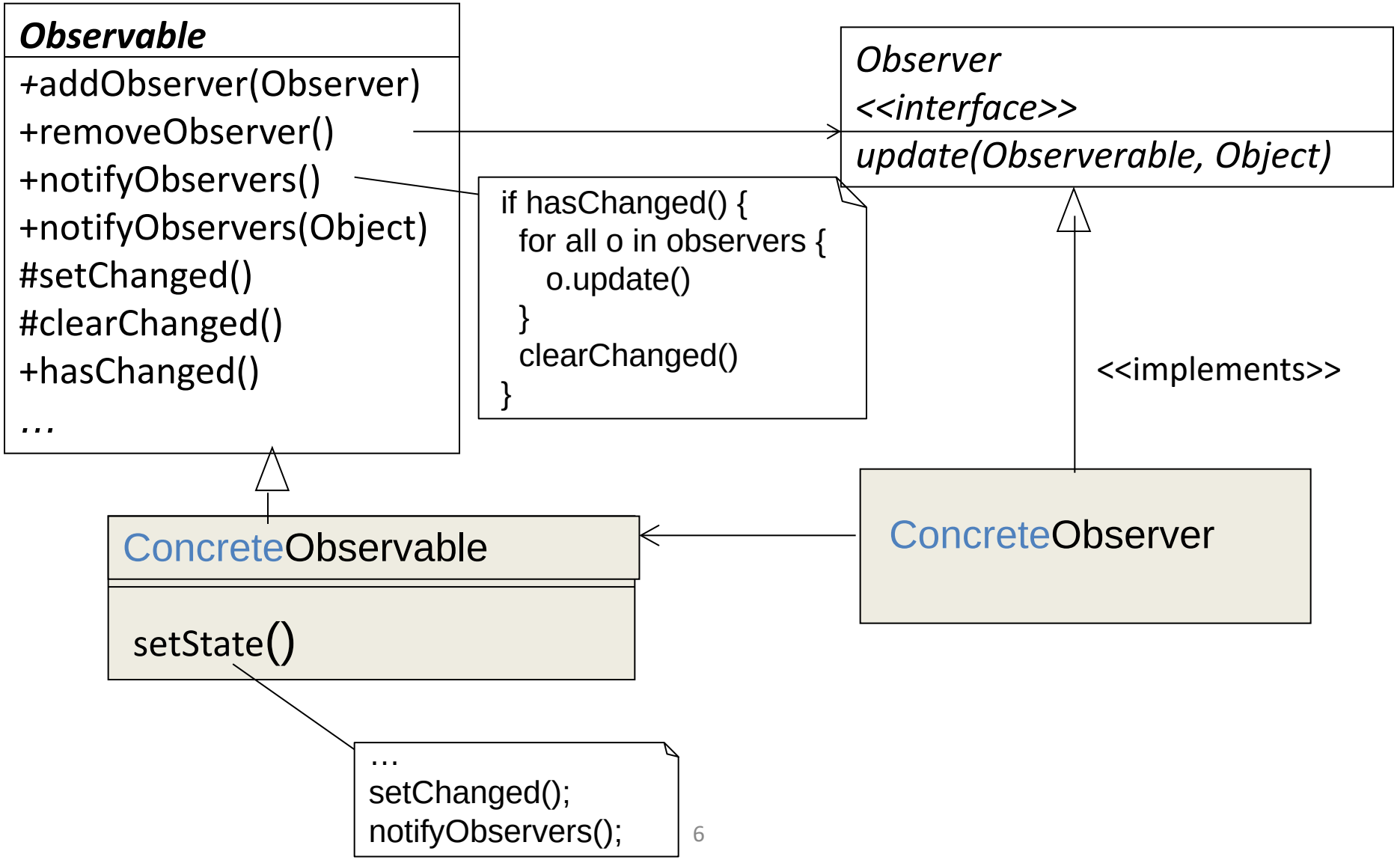
- Implement
  1. Use the given UML as a starting point.
  2. Make use of the Observer Pattern (Add to UML) : The Application must store each shape as it is created in a list. Adding a shape to the list triggers the drawing of that shape on the canvas
  3. Use Anonymous Inner Classes for Event Handling



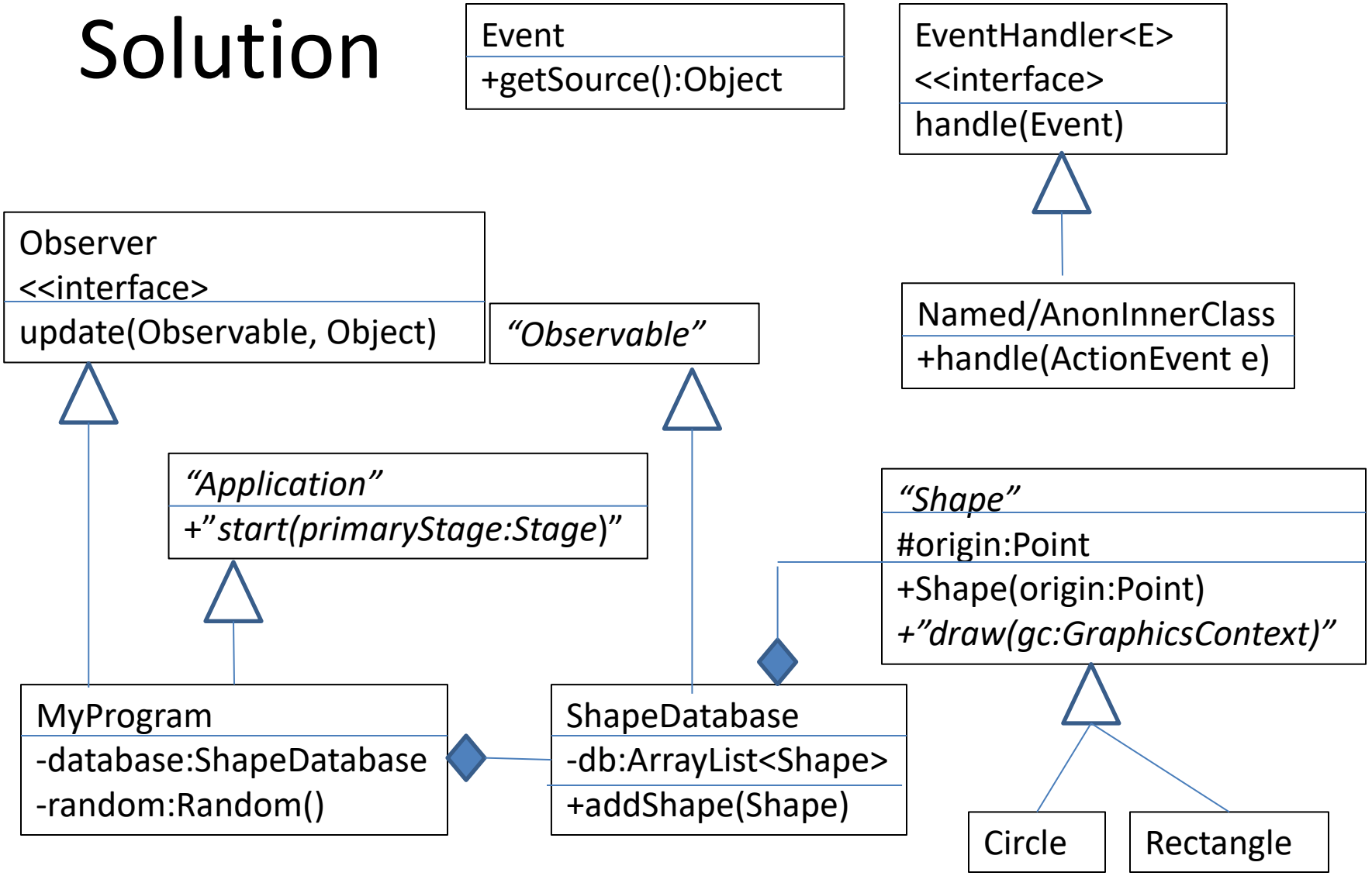
"" have been used to emphasize abstract portion. They are used for clarity and not part of actual UML

Anonymous Inner Classes will not be on the final exam.

Recall: Java Observer Pattern



Solution


Event
+getSource():Object
EventHandler<E>
<<interface>
handle(Event)
Observer
<<interface>
update(Observable, Object)
"Observable"
Named/AnonInnerClass
+handle(ActionEvent e)
"Application"
+"start(primaryStage:Stage)"
"Shape"
#origin:Point
+Shape(origin:Point)
+"draw(gc:GraphicsContext)"
MyProgram
-database:ShapeDatabase
-random:Random()
ShapeDatabase
-db:ArrayList<Shape>
+addShape(Shape)
Circle
Rectangle

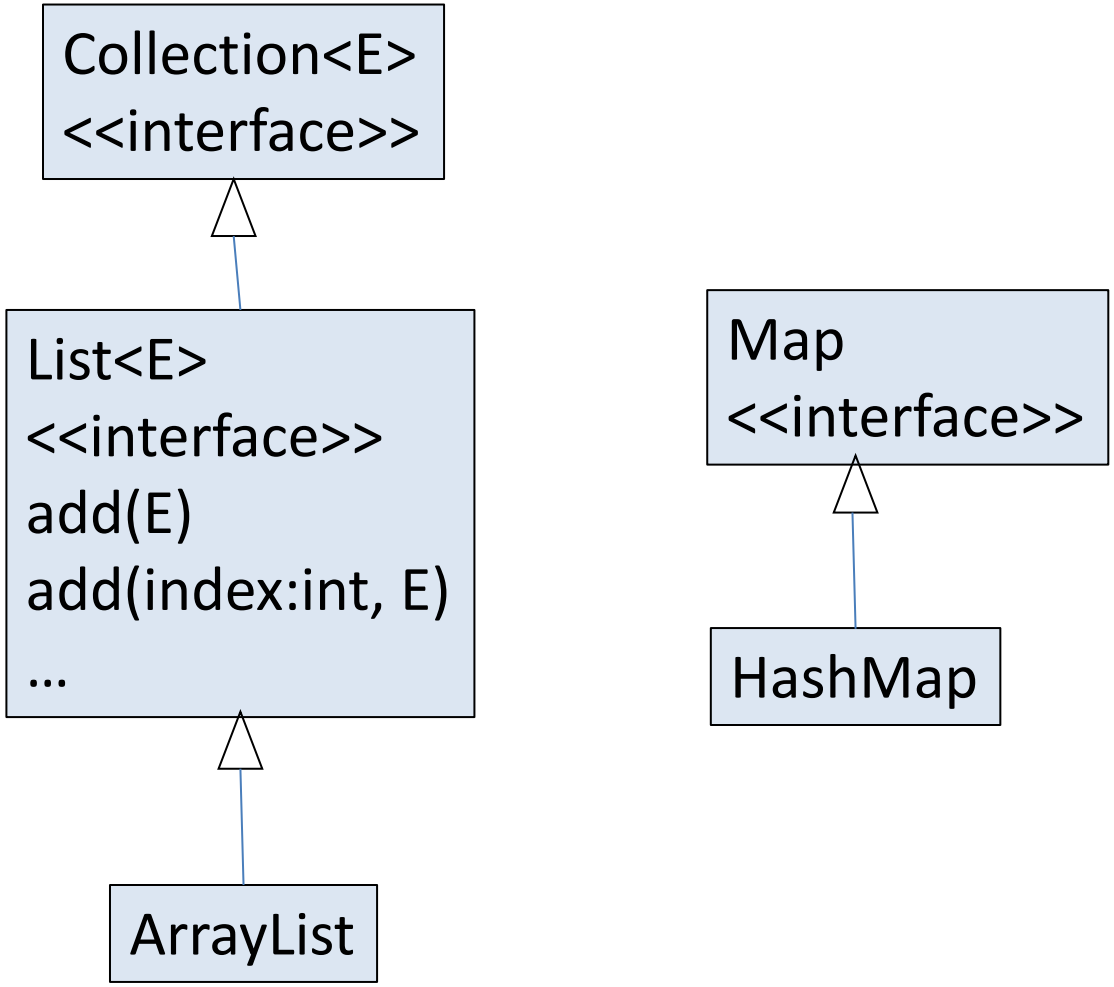
Pattern: Model View Controller (MVC)

- The exercise you've just done is organized according to the MVC pattern
  - Model – The business logic, the underlying data behind the GUI
  - View – The look (application and controls)
  - Controller – The feel (event handlers).
- Many, many versions of MVC but in common, and simply:
  - When a controller handles an event, it makes changes to the model
  - If needed, the model propagates its changes to the view
    - The controller does not directly update the view
    - The view observes the model

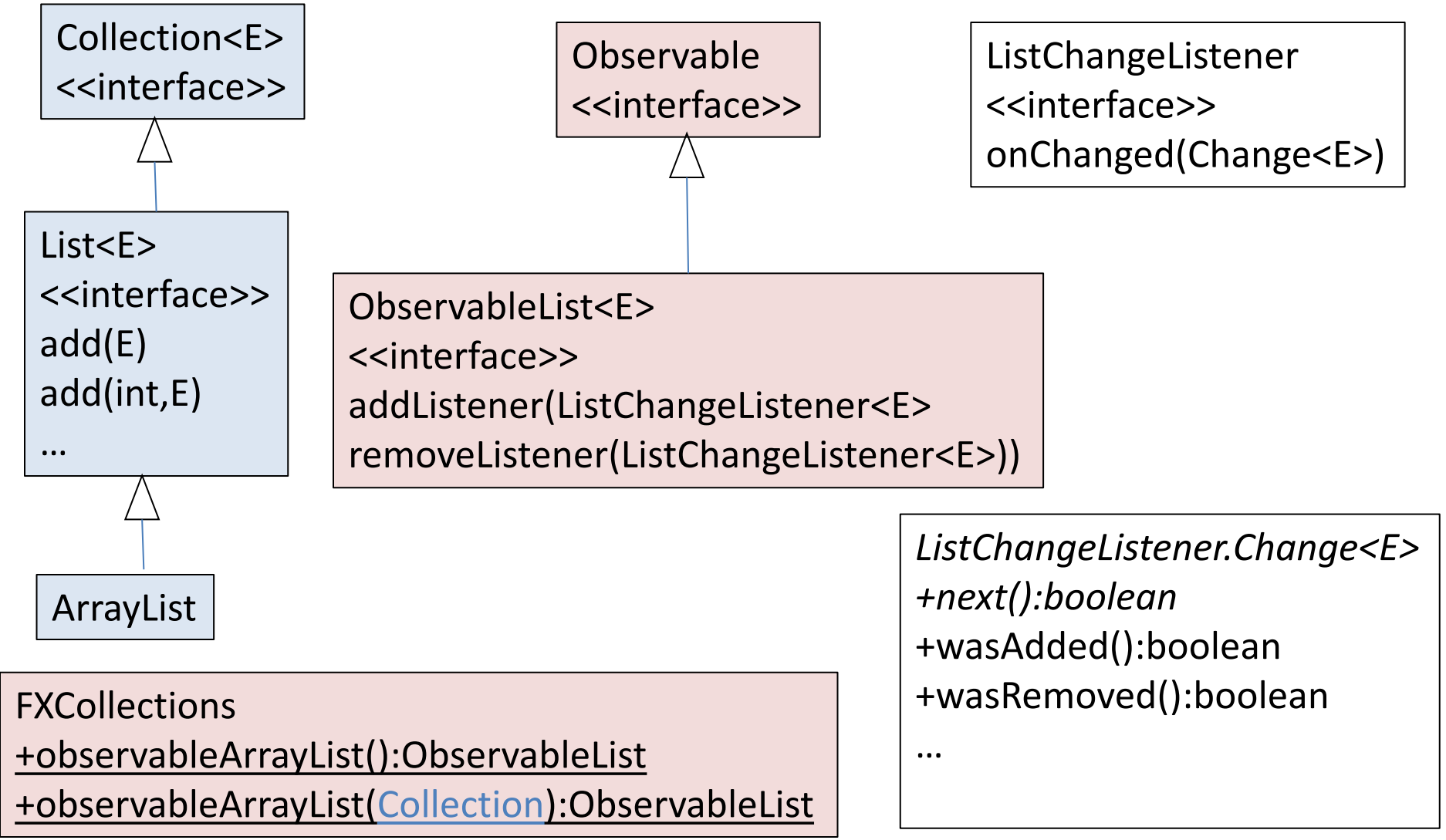
JavaFX's Observable Collections

- Useful for wrapping Collections into GUIs, with a MVC flavour
  - ObservableList
  - ObservableMap
- One-to-one mapping to basic Collection, but whenever a change is made to an ObservableCollection, registered observers are notified.

Recall: Collections



Observable FX Collections – Lists 1st



```
public class CollectionsDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
        // Wrap list with ObservableList.
        ObservableList<String> observableList =
                FXCollections.observableList(list);
        observableList.addListener(new ListChangeListener() {
                @Override public void onChanged
                (ListChangeListener.Change change) {
                        System.out.println("Detected a change! "); } });

        // Changes to the observableList WILL be reported.
        // This line will print out "Detected a change!"
        observableList.add("item one");

        // Changes to the underlying list will NOT be reported
        // Nothing will be printed as a result of the next line.
        list.add("item two");

        System.out.println("Size: "+observableList.size()); } }
```

https://docs.oracle.com/javafx/2/collections/jfxpub-collections.htm

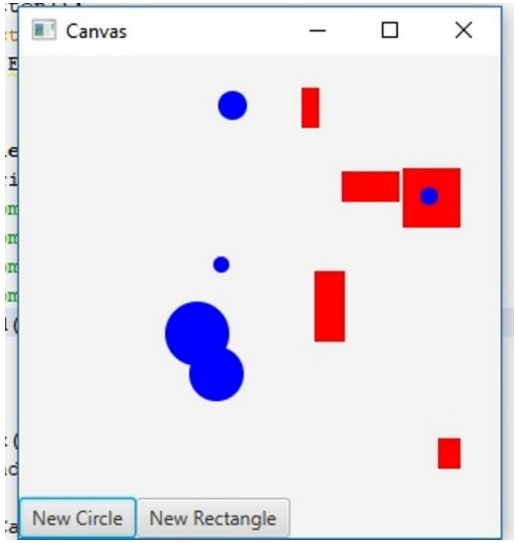
Same Example but meaningful Handler

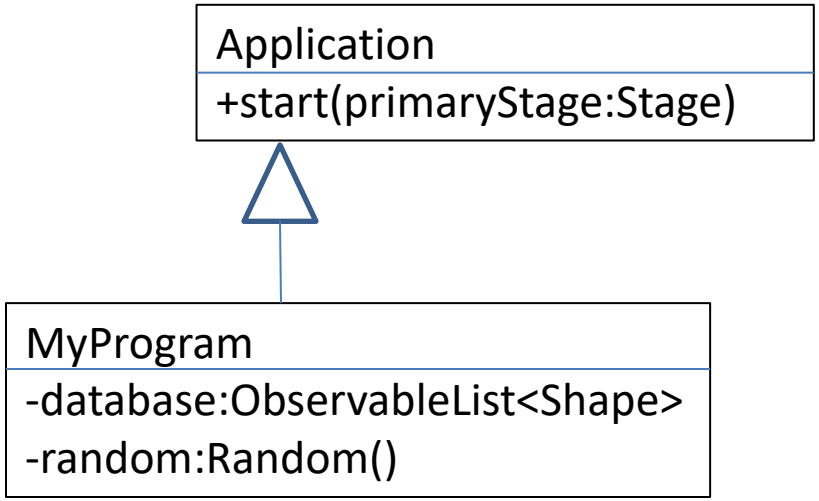
```
@Override
public void onChanged (ListChangeListener.Change c) {

    while (c.next()) { // Must be called before any other method
        if (c.wasAdded() ) {
                int index = c.getFrom();
                List<String> addedStrings =c.getAddedSubList();
                for (String s:addedStrings)System.out.println(s);
        } else if (change.wasRemoved() )  {
                int num = c.getRemovedSize();
                List<String> removedStrings = c.getRemoved();
        } ….
}
```

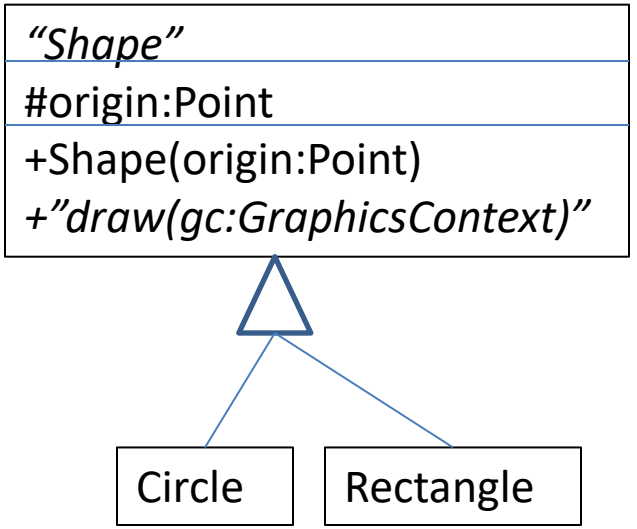
https://docs.oracle.com/javafx/2/collections/jfxpub-collections.htm

Second Sample Exam Question

- Repeat the previous exercise, but now using Observable Collections







Focus on the onChanged() method

```
@Override
   public void onChanged(Change change) {
      while (change.next()) { // Must be called before any other method
         if (change.wasAdded() ) {
                  int index = change.getFrom();
                  List<Shape> addedShapes =change.getAddedSubList();
                  for (Shape s : addedShapes) {
                        s.draw(gc);
           }
         }
      }
   }
```

Observable FX Collections – Maps 2nd