M6 (b) - Composition

Jin L.C. Guo

Image source: https://cdn.pixabay.com/photo/2017/11/05/21/21/container-2921882_960_720.jpg

Questions from previous lectures

- Static Keyword during import

  Imports static members from classes, allowing them to be used without class qualification. [REF]

- Can enum have non static and non final filed?

```
public enum Suit
{
   CLUBS, DIAMONDS, SPADES, HEARTS;
   int size = 0;

   void changeSize(int pSize) {
      size = pSize;
   }
   int getSize(){
      return size;
   }
}
```

```
Suit a = Suit.CLUBS;
Suit b = Suit.CLUBS;
System.out.println(a.getSize());  //0
a.changeSize(3);
System.out.println(b.getSize());  //3
```

You can, but need to have a good reason to use it.

Questions from previous lectures

- Is assertEquals comparing reference equality?

In `AssertUtils.class`, this function is called during assertEquals

```
static boolean objectsAreEqual(Object obj1, Object obj2) {
    if (obj1 == null) {
        return obj2 == null;
    } else {
        return obj1.equals(obj2);
    }
}
```

- Are we able to use reflection to modify final field?

If the underlying field is final, the method throws an `IllegalAccessException` unless `setAccessible(true)` has succeeded for this `Field` object and the field is non-static. [REF]

When you have doubt,

Use your debugger and reference the API specification.

Objective

- Design Principle:

Divide and Conquer

- Programming mechanism:

Aggregation and Delegation, Polymorphic Object Cloning

- Design Techniques:

Sequence Diagram

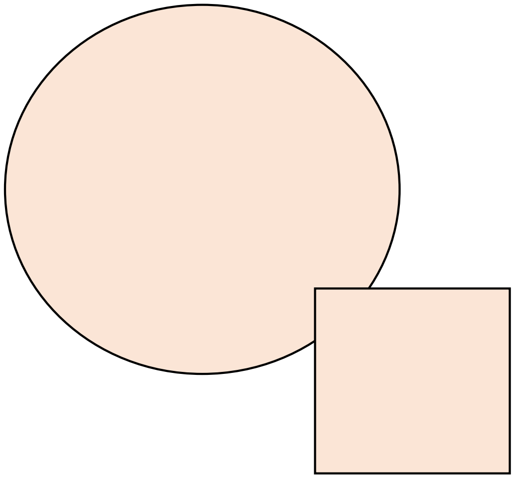
- Patterns and Anti-patterns:

Composite Pattern, Decorator Pattern, Prototype Pattern, God class 👎

2 reasons for composition

Composition Purpose 1

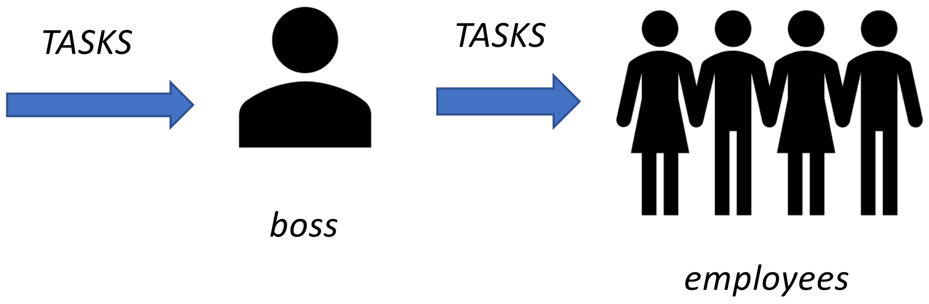
1 - Aggregation: Representation of collections



Composition Purpose 2

2

- Delegation: Redirect duties



TASKS

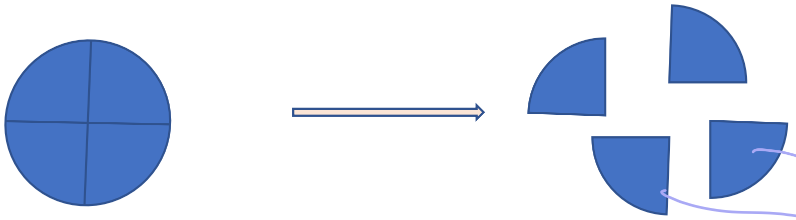
boss

TASKS

employees

Manage Complexity -- Divide and conquer

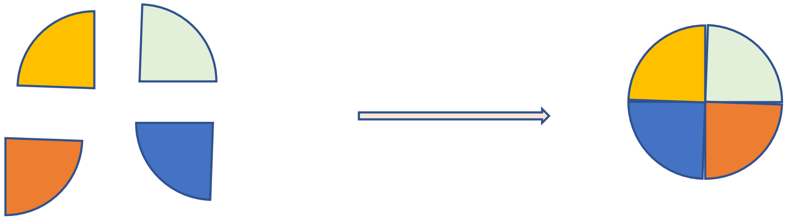
- Modularization
  - Decomposable



break it up

different functionalites

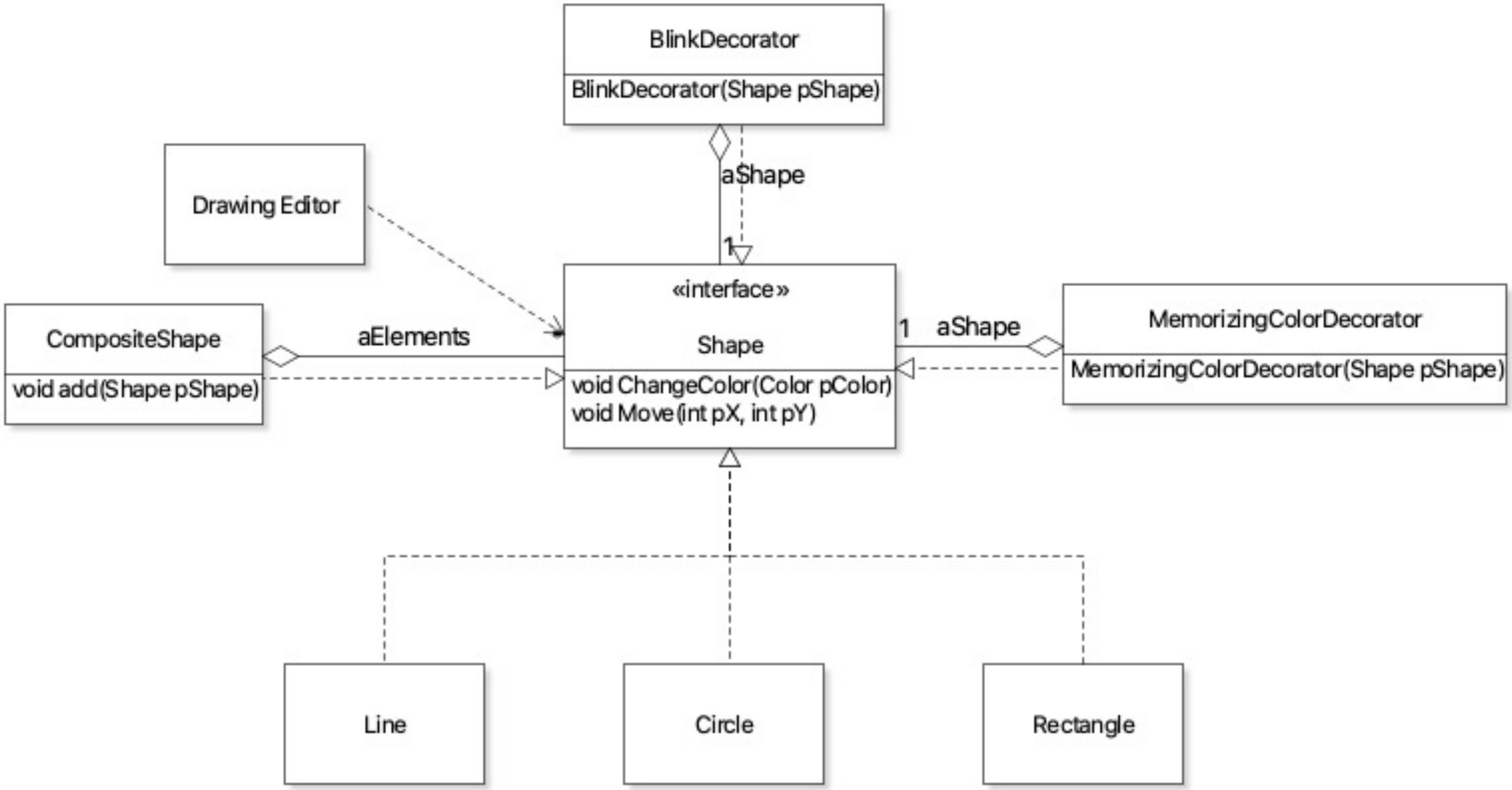
  - Composable



allows for definition of special behavior

mix and match functionalites
↳ decorators as seprate classes, treated equivalently

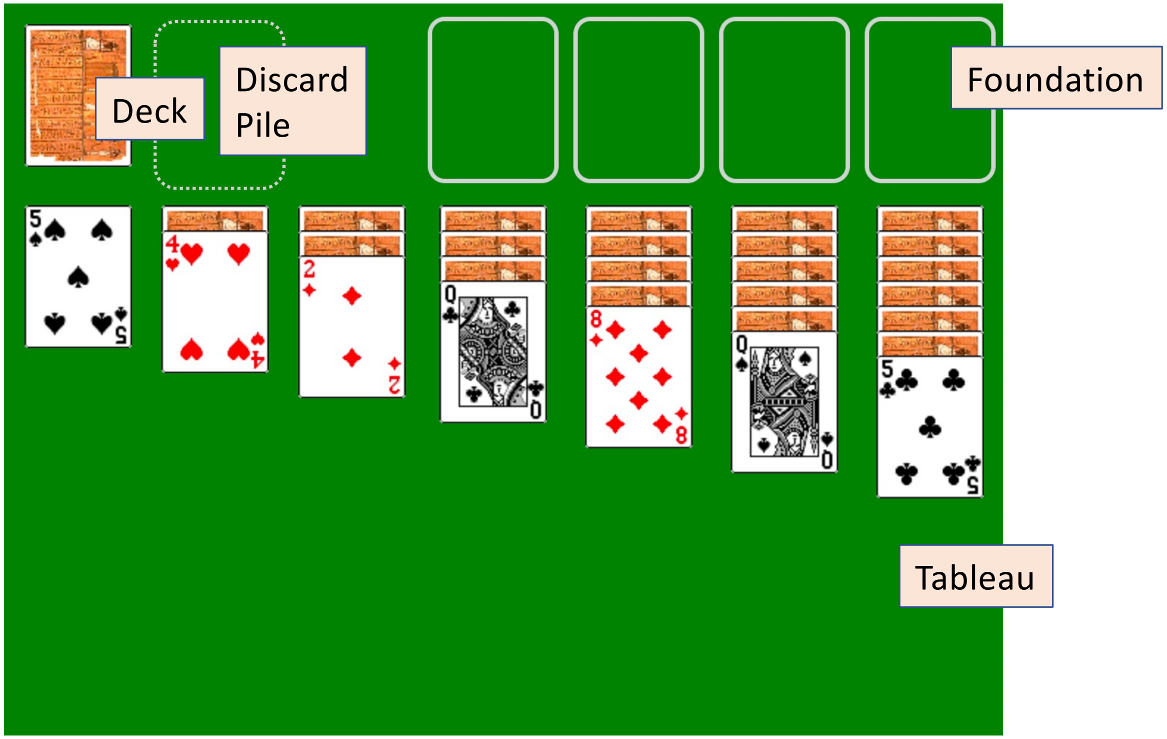
Example: the design of shapes:



Example: GameModel in Solitaire

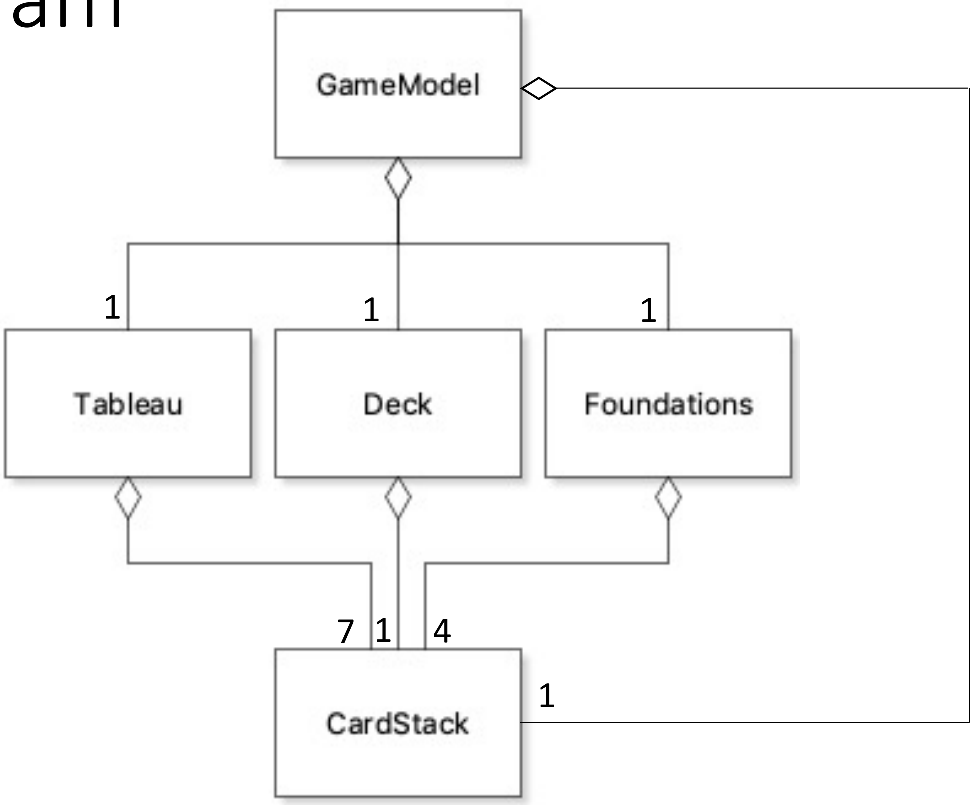
13 piles of cards?

God Class



The elements are the component, and also entities providing services.

Class Diagram



each class smaller with specific functionalities

Objective

- Design Principle:

Divide and Conquer

- Programming mechanism:

Aggregation and Delegation, Polymorphic Object Cloning

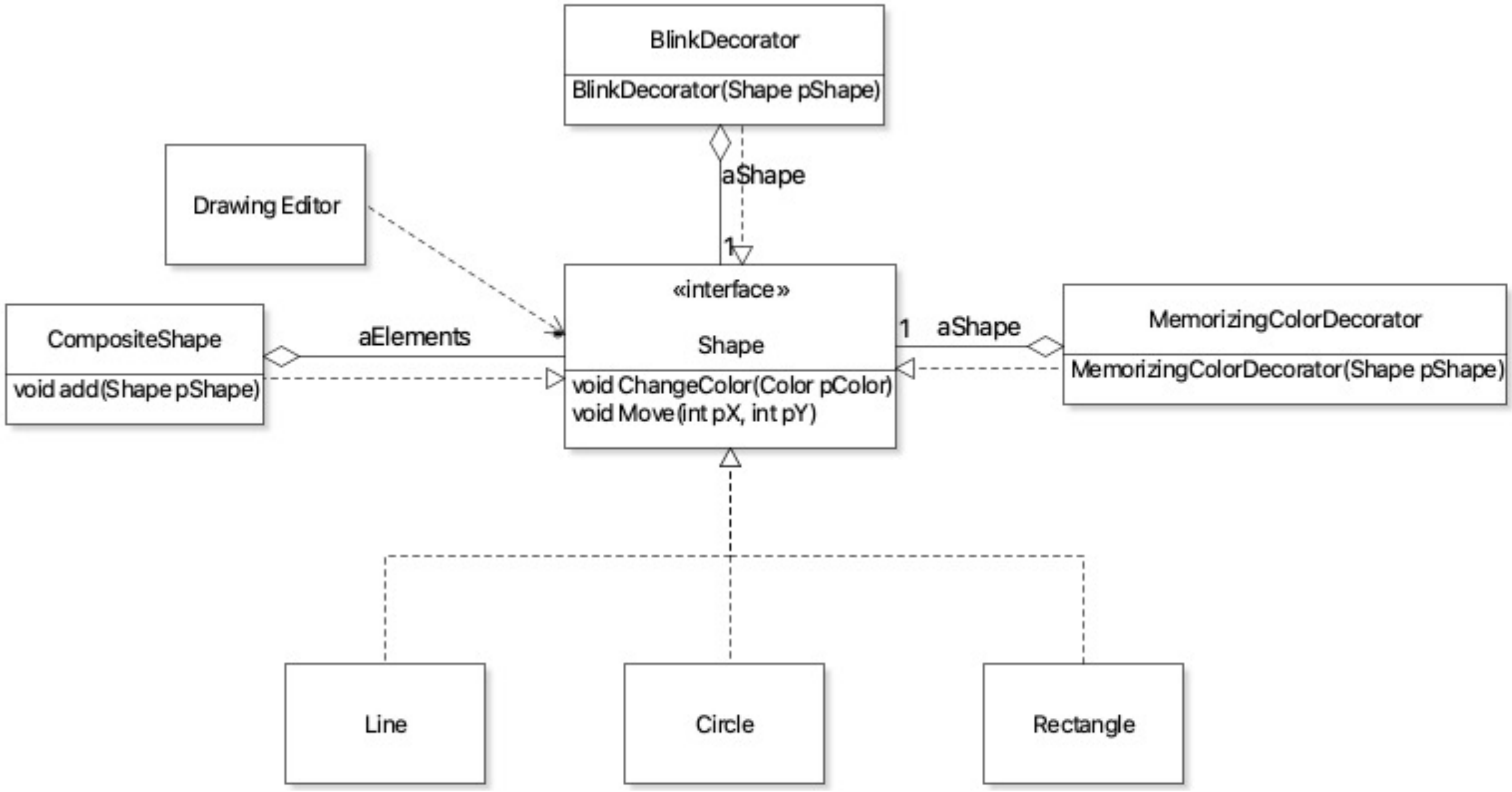
- Design Techniques:

Sequence Diagram

- Patterns and Anti-patterns:

Composite Pattern, Decorator Pattern, Prototype Pattern, God class 👎

So far, our design of shapes:



```java
/**
 * Aggregate a collection of shapes.
 * The client can get shapes and
 * add new shape on demand
 *
 */
public class ShapeManager
{
    private final List<Shape> aShapes = new ArrayList<>();

    public List<Shape> getShapes()
    {
        // return a copy of aShapes;
    }

    public void addShape(Shape pShape)
    {
        // add a copy of pShape;
    }
}
```

Activity1: How to design the function of making a copy of a Shape object?

Object Copying

- Copy Constructor

```
public  Line(Line pLine) {
    this.x_start = pLine.x_start;
    this.y_start = pLine.y_start;
    this.x_end = pLine.x_end;
    this.y_end = pLine.y_end;
}
```

- Static factory method

Line.newInstance(   )

```
public static Line newInstance(Line pLine)
{
    return new Line(pLine);
}
```

pretty good / safe options

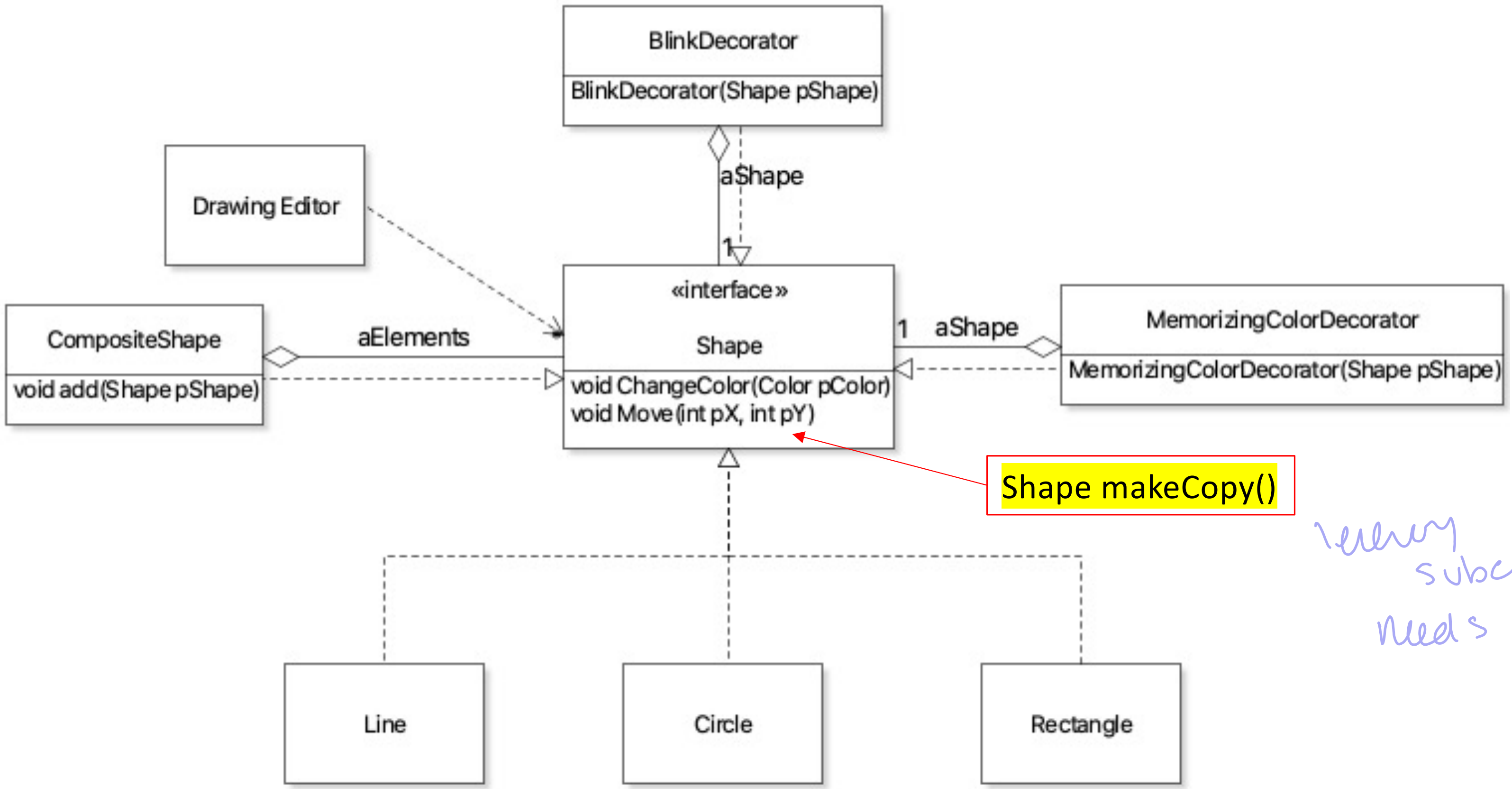
constraints :

```
public List<Shape> getShapess()
{
    // return a copy of aShapes;
    List<Shape> shapesCopy = new ArrayList<>();
    for(Shape sp:aShapes)
    {
        if (sp instanceof Line)
        {
            shapesCopy.add(new Line(sp));
        }
        else if (sp instanceof Circle)
        {
            shapesCopy.add(new Circle(sp));
        }
        else if (sp instanceof CompositeShape)
        {
            ……
        }
        ……
    }
    return shapesCopy;
}
```

How to achieve polymorphic object copying?

need to check
which shape
it is

switch = need flag



Shape makeCopy()

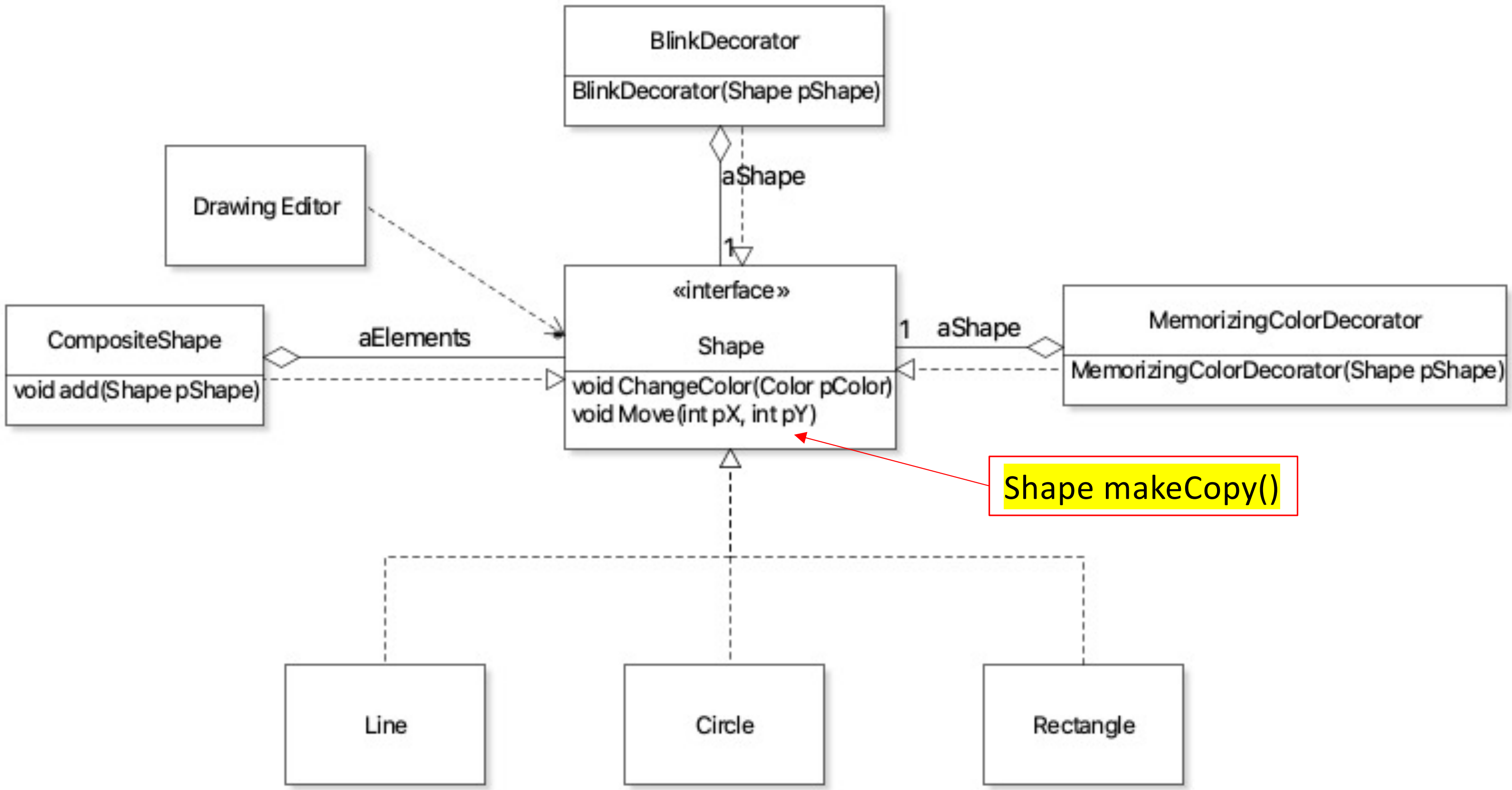
every
subclass
needs make copy

```
public List<Shape> getShapess()
{
    // return a copy of aShapes;
    List<Shape> shapesCopy = new ArrayList<>();
    for(Shape sp:aShapes)
    {
        if (sp instanceof Line)
        {
            shapesCopy.add(new Line(sp));
        }
        else if (sp instanceof Circle)
        {
            shapesCopy.add(new Circle(sp));
        }
        else if (sp instanceof CompositeShape)
        {
            ……
        }
        ……
    }
    return shapesCopy;
}
```

```
public List<Shape> getShapess()
{
    // return a copy of aShapes;
    List<Shape> shapesCopy = new ArrayList<>();
    for(Shape sp:aShapes)
    {



        shapesCopy.add(sp.makeCopy());



    }
    return shapesCopy;
}
```

```java
/**
 * Aggregate a collection of shapes.
 * The client can get shapes and
 * add new shape on demand
 *
 */
public class ShapeManager
{
    private final List<Shape> aShapes = new ArrayList<>();

    public List<Shape> getShapes()
    {
        // return a copy of aShapes;
    }

    public void addShape(Shape pShape)
    {
        aShapes.add(pShape.makeCopy());
    }
}
```

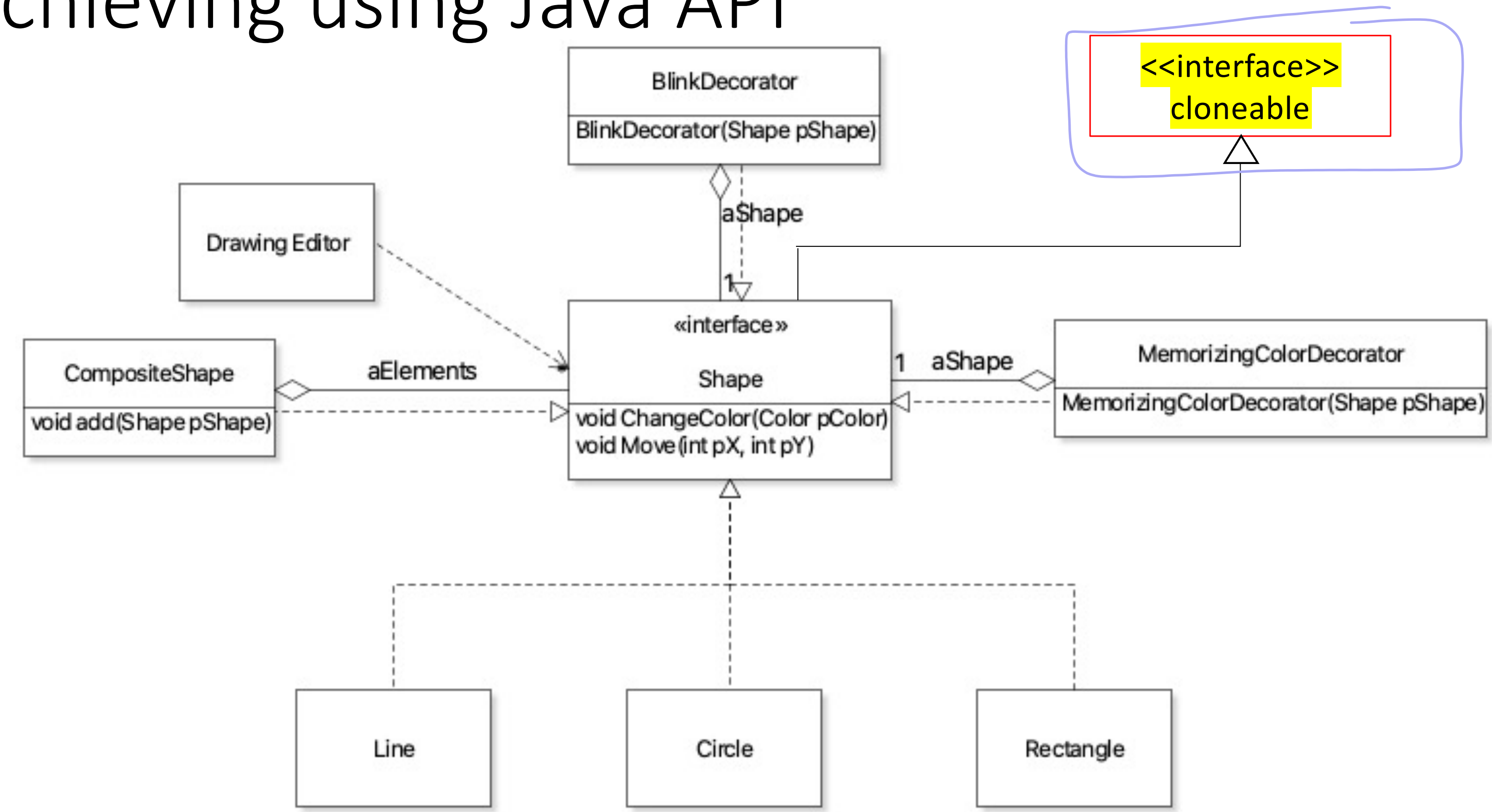
```java
public class Line implements Shape
{
    …


    @Override
    public Line makeCopy()
    {
        return new Line(this);
    }

}
```



Shape makeCopy()

Achieving using Java API

helps achieve polymorphic copy



Implements Cloneable

“Tough the specification doesn’t say it, in practice, a class implementing Cloneable is expected to provide a properly Functioning public clone Method.

In order to achieve this, the class and all of its super classes must obey a complex, unenforceable, thinly documented protocol. The resulting mechanism is fragile, dangerous, and extralinguistic: it creates object without calling a constructor.”

need to work around and follow instructions!

Effective Java by Joshua Bloch, 3rd ed., 2018.

Implements Cloneable

- java.lang.Cloneable

this interface does *not* contain the clone method.

implement this interface should override Object.clone with a public method.

A class implements the Cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class.

Invoking Object's clone method on an instance that does not implement the Cloneable interface results in the exception CloneNotSupportedException being thrown.

Override Object.clone()

```
protected Object clone()
          throws CloneNotSupportedException
```

Creates and returns a copy of this object.

x.clone() != x – two seperate objects

x.clone().getClass() == x.getClass()
↳ same runtime type

x.clone().equals(x) ↳ internally the same / logically equivalent

required in order to work

object should be obtained by calling super.clone - dont use constructors

the object returned by this method should be independent of this object

Shape extends Cloneable

```
public class CompositeShape implements Shape
{
    private List<Shape> aElements = new ArrayList<>();
    @Override
    public CompositeShape clone()
    {
        try
        {
            CompositeShape clone = (CompositeShape) super.clone();
            clone.aElements = new ArrayList< Shape>();
            for (Shape sp:aElements)
            {
                clone.aElements.add(sp.clone());
            }
            return clone;
        }
        catch (CloneNotSupportedException e)
        {
            assert false;
            return null;
        }
    }
}
```

return exact type so client doesn't need to downcast

Making a shallow copy

instead of calling constructor for a new shape

Objective

- Design Principle:

Divide and Conquer

- Programming mechanism:

Aggregation and Delegation, Polymorphic Object Cloning

- Design Techniques:

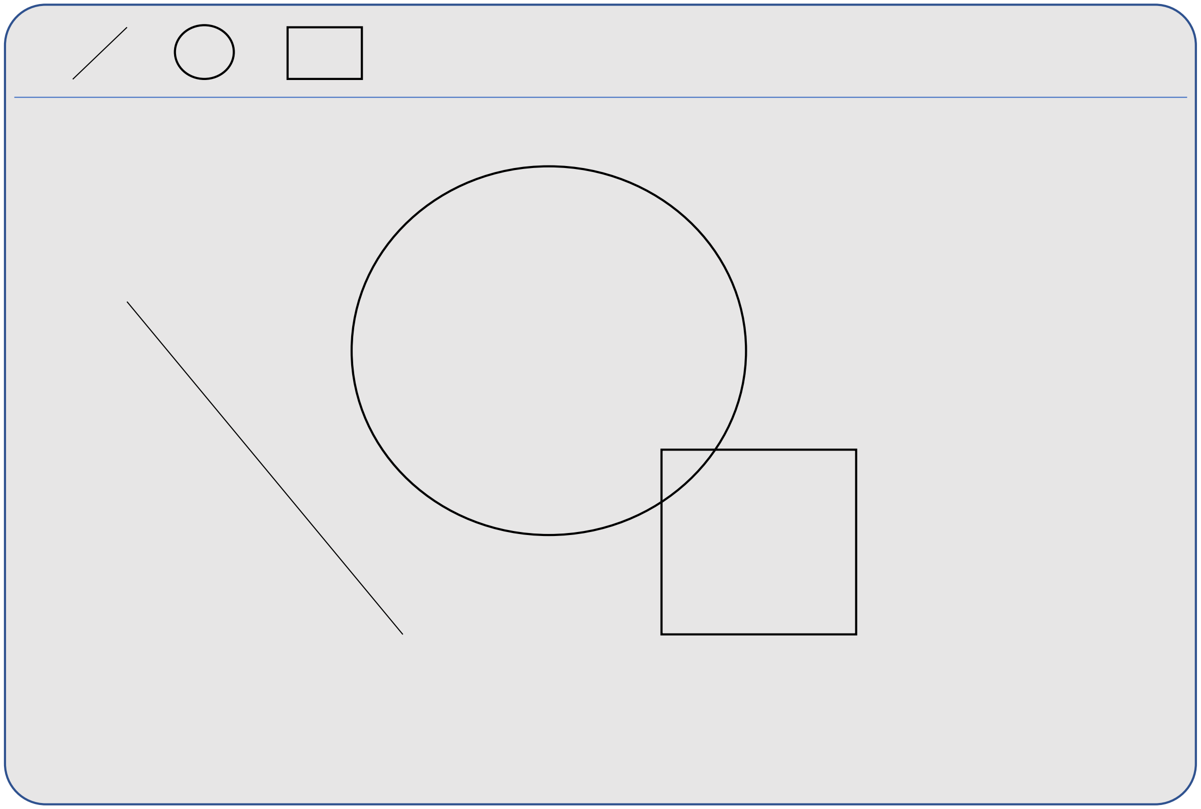
Sequence Diagram

- Patterns and Anti-patterns:

Composite Pattern, Decorator Pattern, Prototype Pattern, God class 👎

Design Problem

Allow the user to add shortcut to create predefined (any) shape, e.g., a red circle on top of a green rectangle what blinks twice.



```java
/**
 * Aggregate a collection of shapes.
 * The client can get shapes and
 * add new shape on demand
 *
 */
public class ShapeManager
{
    private final List<Shape> aShapes = new ArrayList<>();
                                         private Shape aPrototype;

    public List<Shape> getShapes()
    {
        // return a copy of aShapes;
    }

    public void addShape(Shape pShape)
    {
        // add a copy of pShape;
    }
}
```

```
/**
 * Aggregate a collection of shapes.
 * The client can get shapes and
 * add new shape on demand
 *
 */
public class ShapeManager
{
    private final List<Shape> aShapes = new ArrayList<>();
                                      private Shape aPrototype;

    public List<Shape> getShapes()
    {
        // return a copy of aShapes;
    }

    public void setProptypeShape(Shape pShape)
    {
        aPrototype = pShape.clone();
    }
}    public void addShape()
     {
         aShapes.add(aPrototype.clone());
     }
```

Prototype

- Intent
  - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
- Participants
  - Prototype

    declares an interface for cloning itself.
  - Product (Concrete Prototype)

    implements an operation for cloning itself.
  - Client

    creates a new object by asking a prototype to clone itself.

Activity 2: Consider what are the benefits and drawbacks of using Prototype Pattern?

Potential benefit:

- Concrete objects (e.g., objects of Line, Circle, Composite Shape, etc.) is going be hidden form the clients, so that it reduces the Classes the clients need to know about;
- You have the flexibility of adding or removing classes without affecting the client's code;
- The client can build complex object by updating fields of porotype.

Potential drawback:

- You need to override the clone method for all the subclasses of the porotype which might not be easy to achieve.