



M6 (b) - Composition

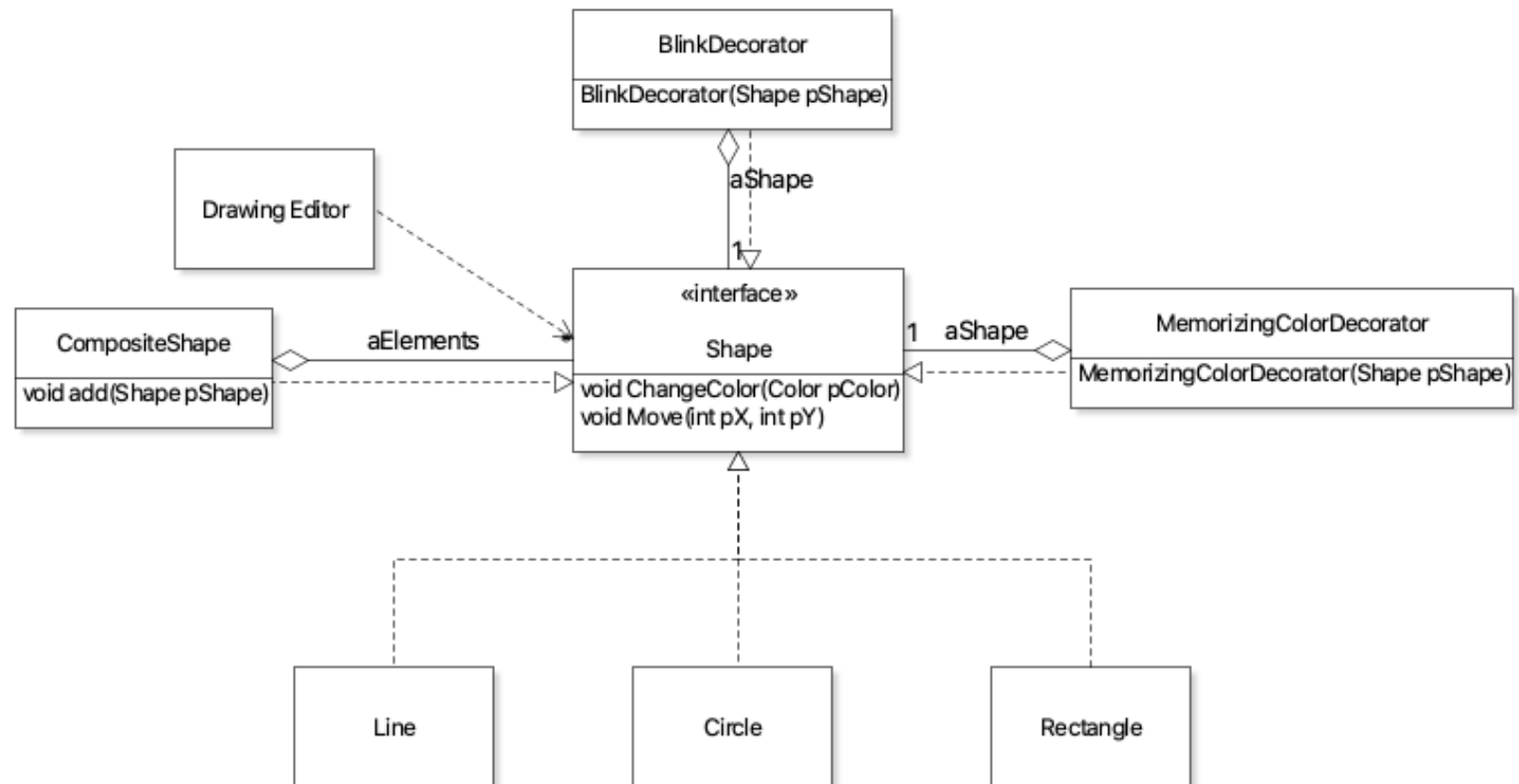
Jin L.C. Guo

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

Objective

- Polymorphic Object Cloning
- Prototype Pattern

So far, our design of shapes:



```
/**
 * 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;
    }
}
```

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

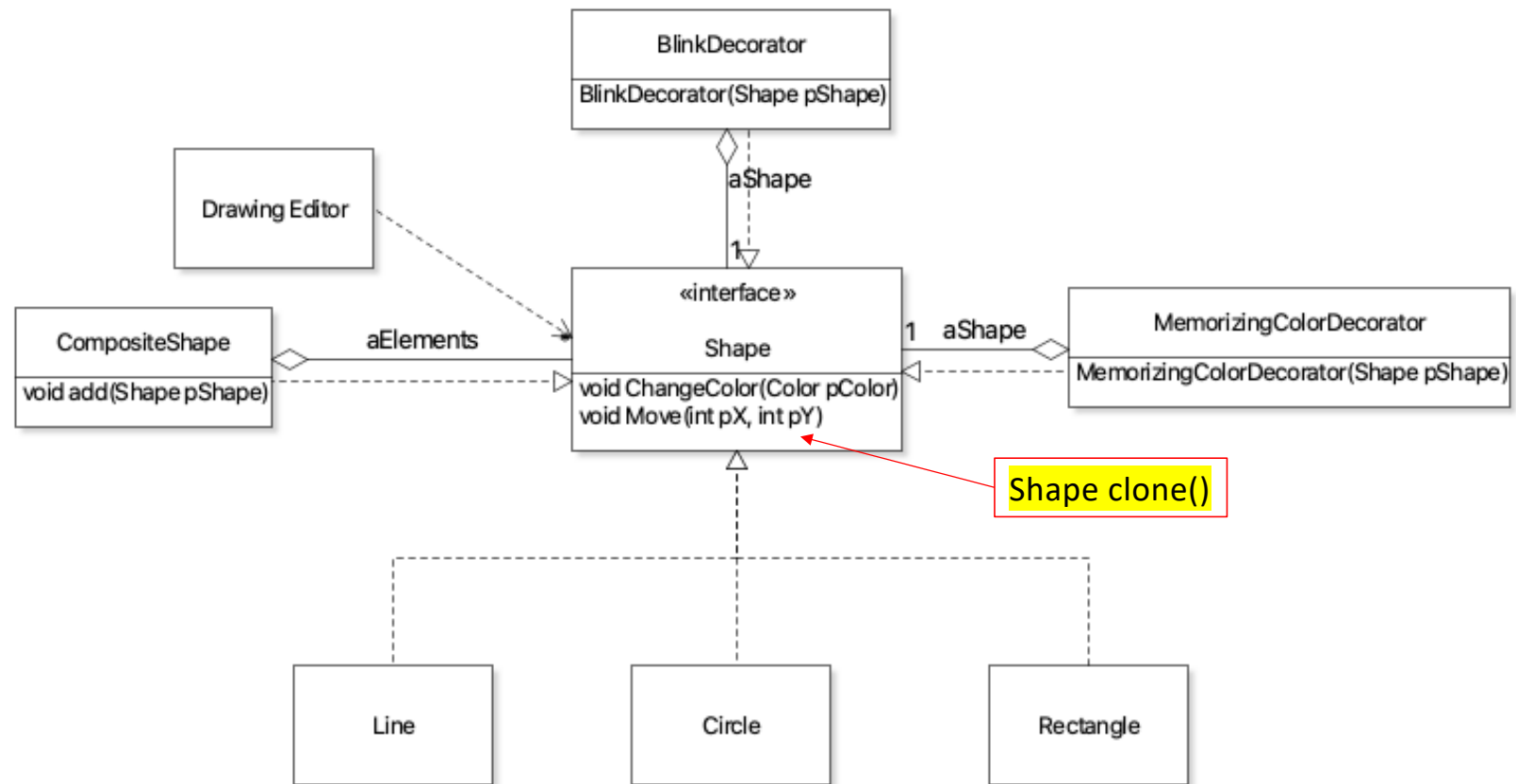
```
public static Line newInstance(Line pLine)  
{  
    return new Line(pLine);  
}
```

```

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?



```

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)  
    {
```

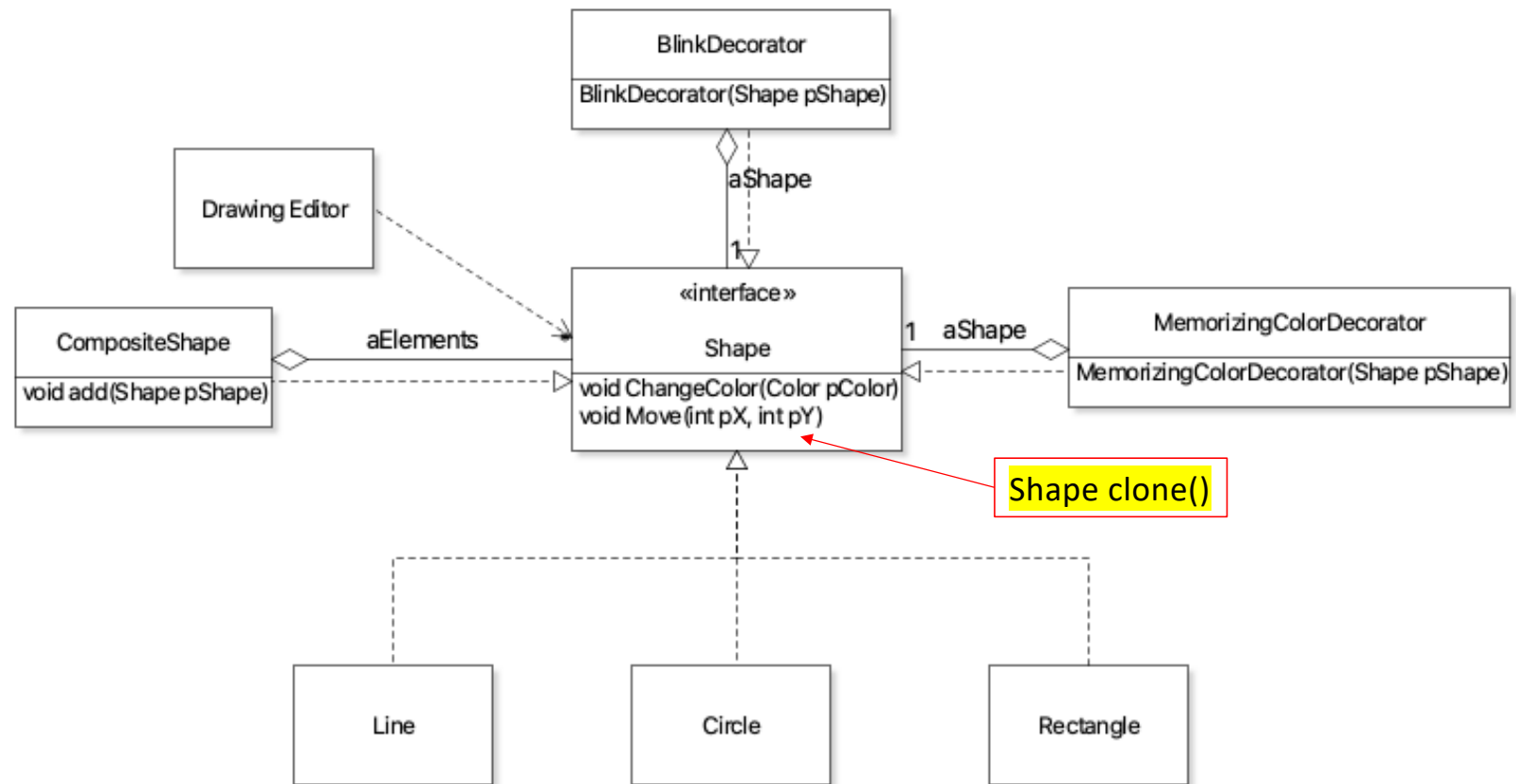
```
        shapesCopy.add(sp.clone());
```

```
    }  
    return shapesCopy;  
}
```

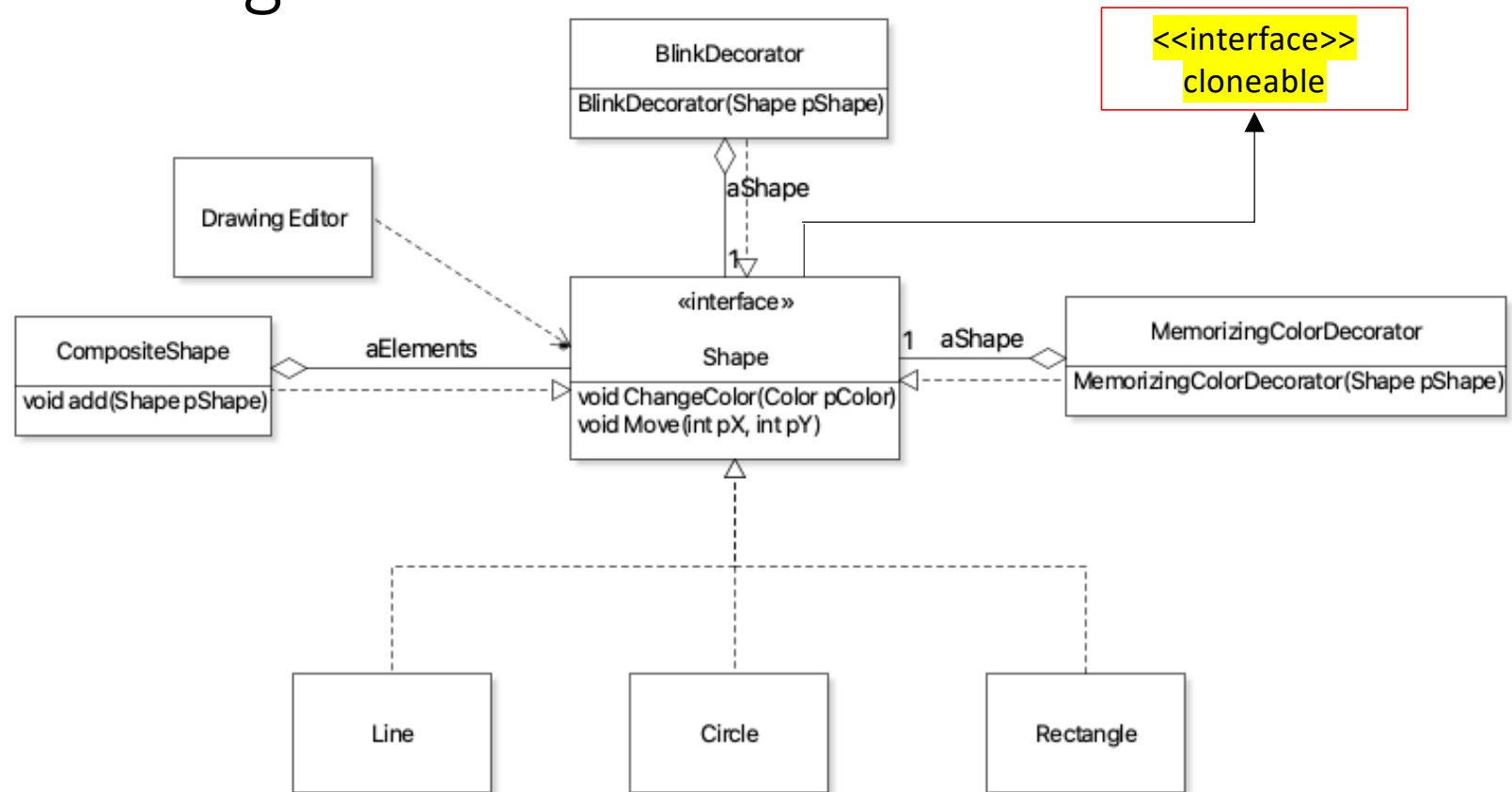
```
/**
 * Aggregate a collection of shapes.
 * The client can get shapes and
 * add new shape on demand
 *
 */
public class ShapeManager
{
    private final List<Shape> aShapes = new ArrayList<>();

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

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



Achieving this in Java



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`

`x.clone().getClass() == x.getClass()`

`x.clone().equals(x)`

object should be obtained by calling `super.clone`

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

```
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;
```

```
        }
```

```
    }
```

```
}
```

Making a shallow copy

```
/**
 * 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.*

What are the benefits and drawbacks of using Prototype Pattern?

- Please fill in the survey