Design Patterns 2
Composite – Decorator - Visitor

Quan Thanh Tho

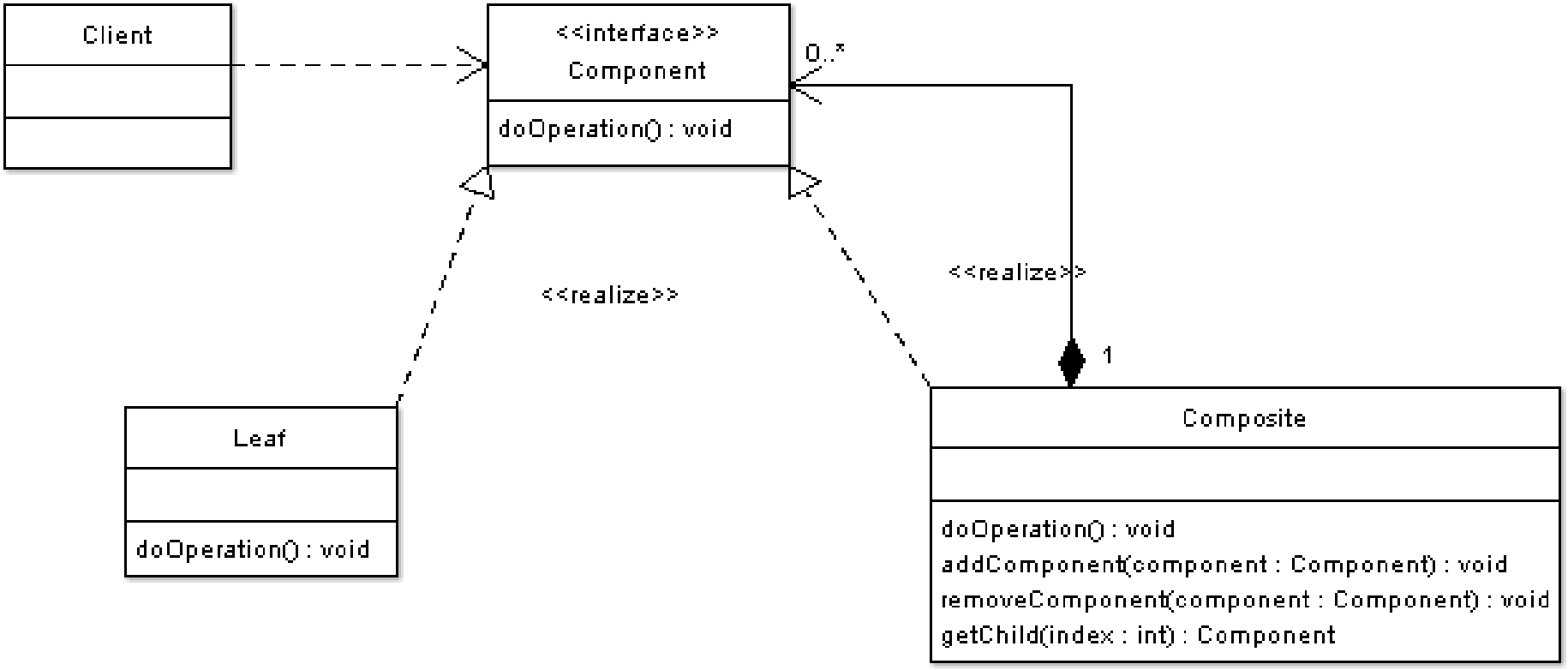
Agenda

- Composite
- Decorator
- Visitor

Composite

- Intent
  - The intent of this pattern is to compose objects into tree structures to represent part-whole hierarchies.
  - Composite lets clients treat individual objects and compositions of objects uniformly.

Class Diagram

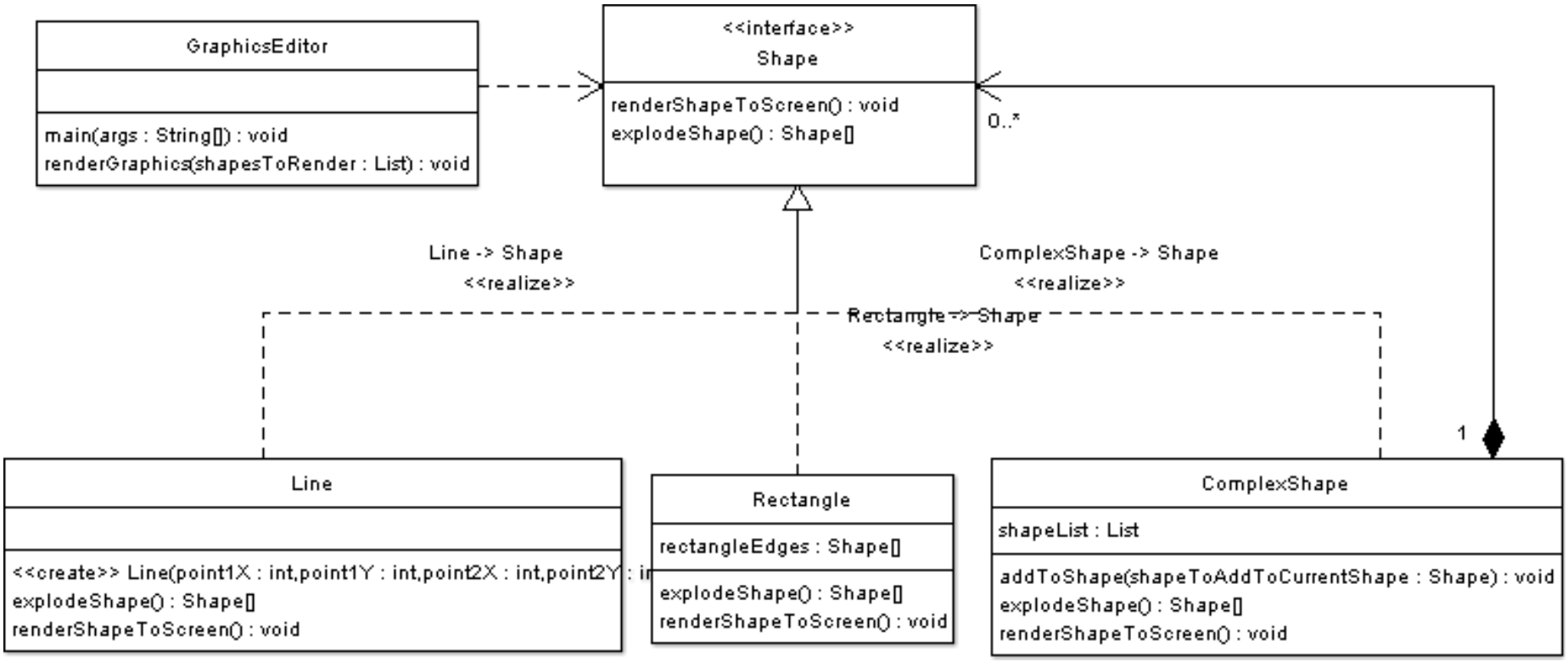


Participant Classes

- **Component** - Component is the abstraction for leafs and composites.
- **Leaf** - Leafs are objects that have no children..
- **Composite** - A Composite stores child components in addition to implementing methods defined by the component interface.
- **Client** - The client manipulates objects in the hierarchy using the component interface.

Example

- In graphics editors a shape can be basic or complex.
- A simple shape is a line, where a complex shape is a rectangle which is made of four line objects.
- Shapes have many operations in common such as rendering the shape to screen


GraphicsEditor
main(args : String[]) : void
renderGraphics(shapesToRender : List) : void
<<interface>>
Shape
renderShapeToScreen() : void
explodeShape() : Shape[]
0..*
1
Line -> Shape
<<realize>>
ComplexShape -> Shape
<<realize>>
Rectangle -> Shape
<<realize>>
Line
<<create>> Line(point1X : int,point1Y : int,point2X : int,point2Y : in
explodeShape() : Shape[]
renderShapeToScreen() : void
Rectangle
rectangleEdges : Shape[]
explodeShape() : Shape[]
renderShapeToScreen() : void
ComplexShape
shapeList : List
addToShape(shapeToAddToCurrentShape : Shape) : void
explodeShape() : Shape[]
renderShapeToScreen() : void

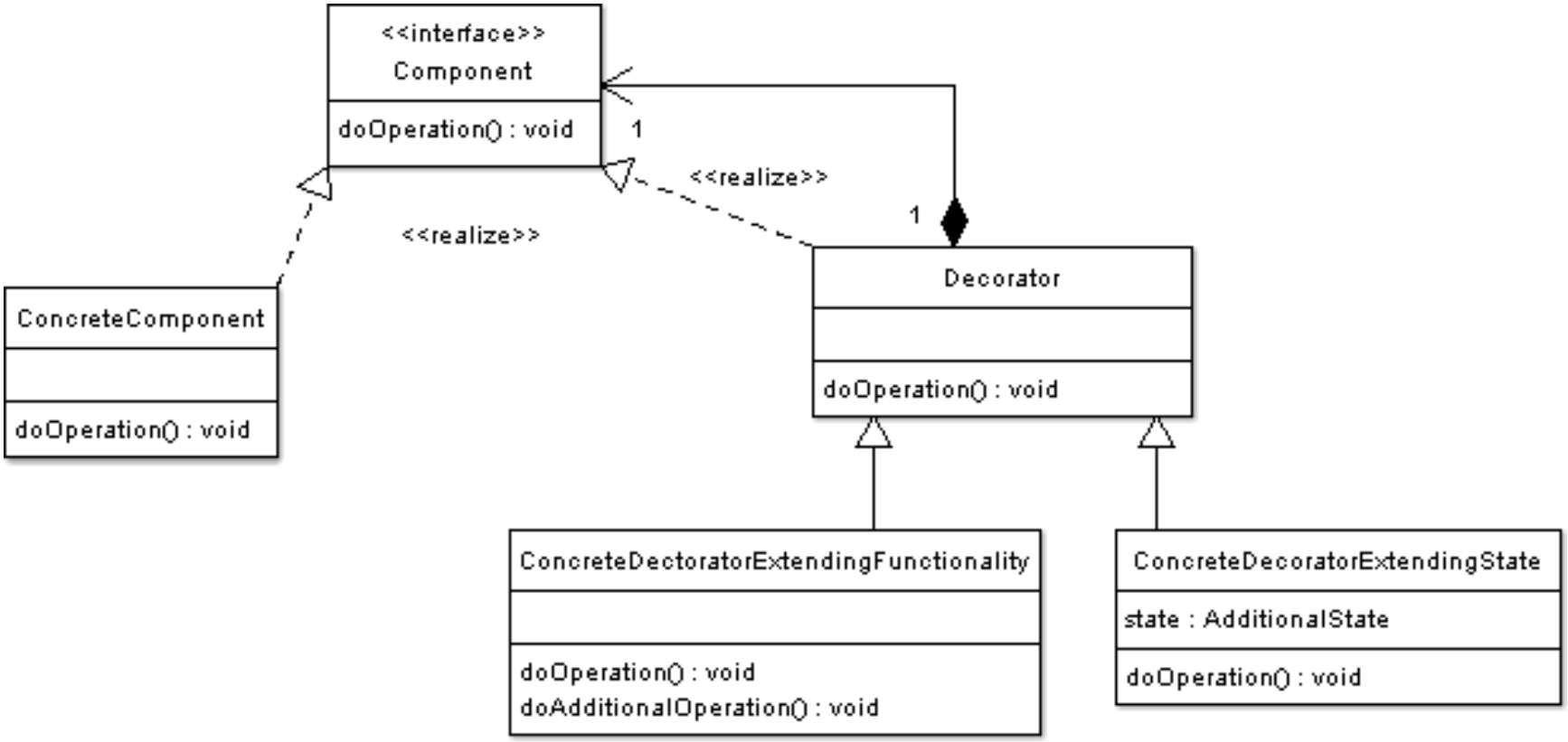
Code

- http://www.oodesign.com/composite-pattern-shapes-example-java-sourcecode.html

Decorator

- Intent
  - The intent of this pattern is to add additional responsibilities dynamically to an object.

Class Diagram

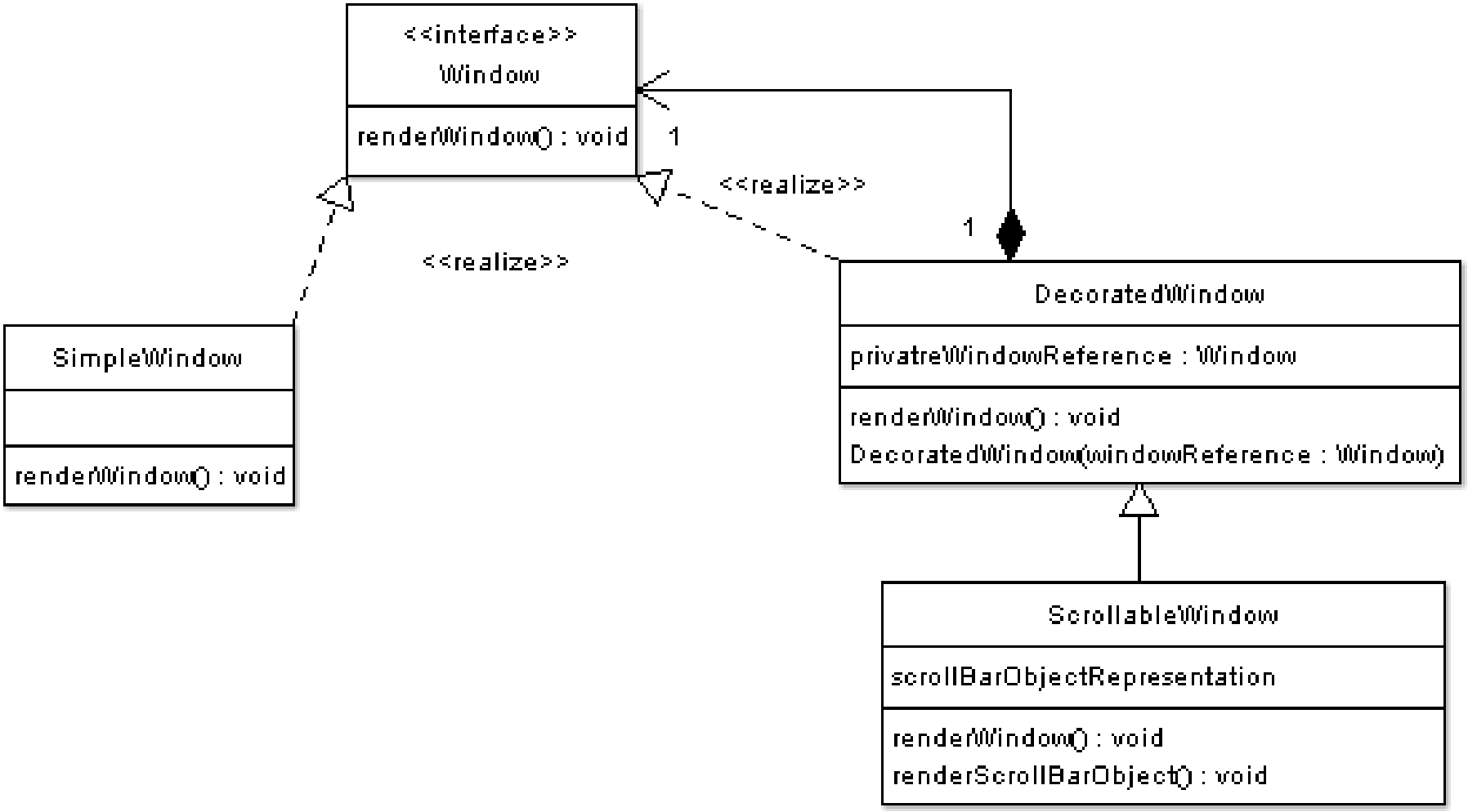


Participant Classes

- **Component** - Interface for objects that can have responsibilities added to them dynamically.
- **ConcreteComponent** - Defines an object to which additional responsibilities can be added.
- **Decorator** - Maintains a reference to a Component object and defines an interface that conforms to Component's interface.
- **Concrete Decorators** - Concrete Decorators extend the functionality of the component by adding state or adding behavior.

Example

- GUI Windows which can have scrollbar when needed

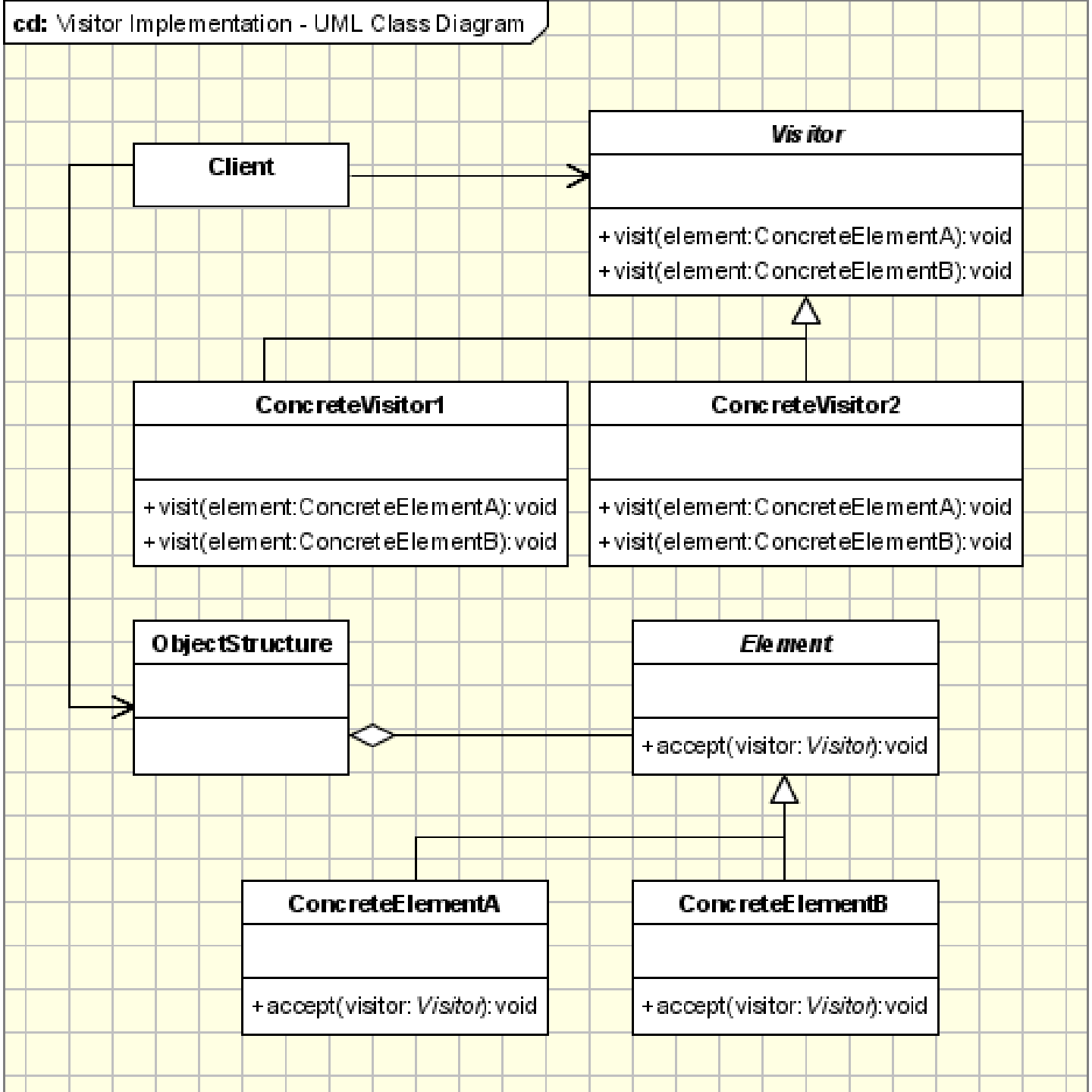

<<interface>>
Window
renderWindow() : void
1
<<realize>>
1
<<realize>>
SimpleWindow
renderWindow() : void
DecoratedWindow
privatreWindowReference : Window
renderWindow() : void
DecoratedWindow(windowReference : Window)
ScrollableWindow
scrollBarObjectRepresentation
renderWindow() : void
renderScrollBarObject() : void

Code

http://www.oodesign.com/decorator-pattern-gui-example-java-sourcecode.html

Visitor

- Intent
  - Represents an operation to be performed on the elements of an object structure.
  - Visitor lets you define a new operation without changing the classes of the elements on which it operates.

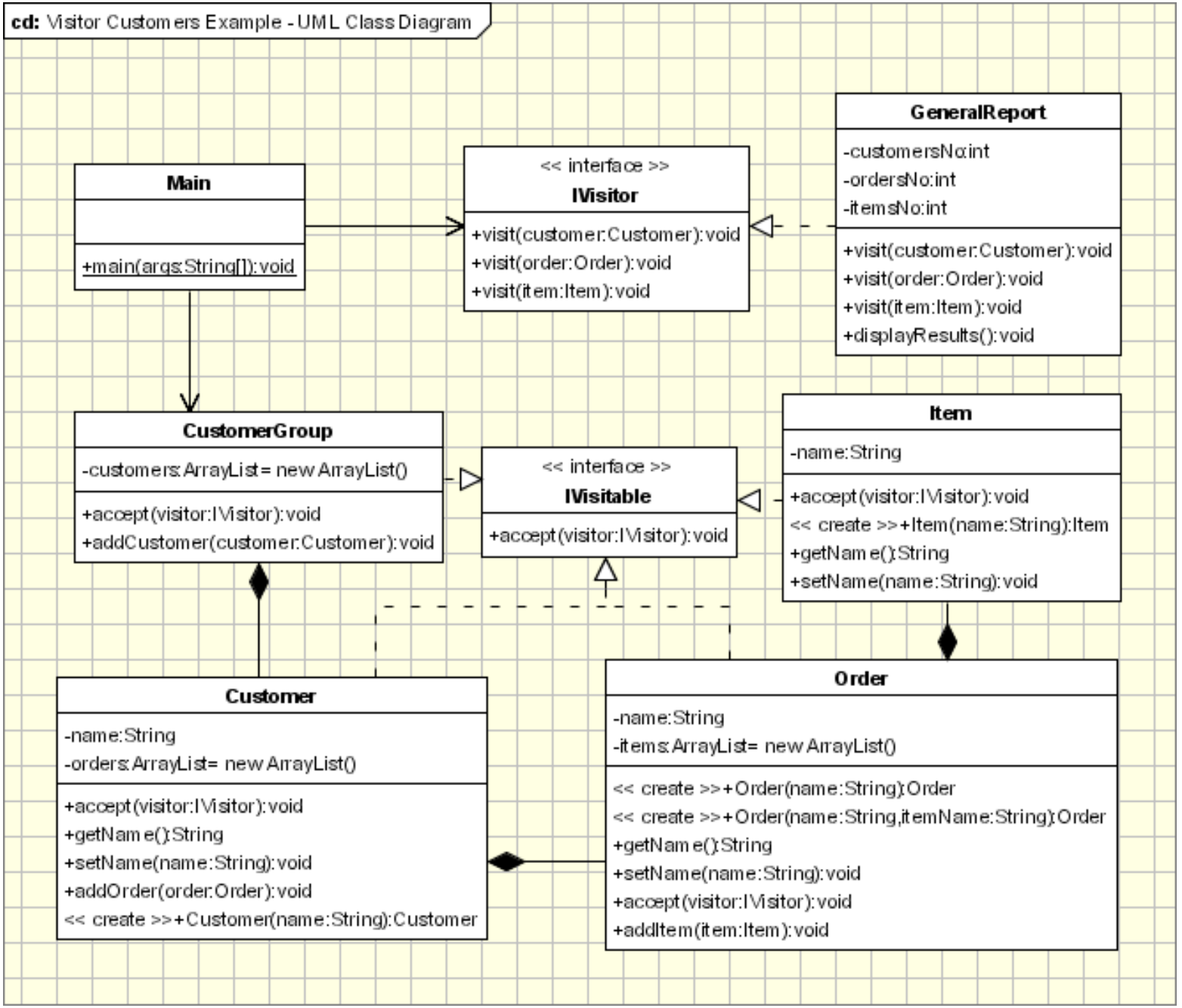

cd: Visitor Implementation - UML Class Diagram
Client
Visitor
+visit(element:ConcreteElementA):void
+visit(element:ConcreteElementB):void
ConcreteVisitor1
+visit(element:ConcreteElementA):void
+visit(element:ConcreteElementB):void
ConcreteVisitor2
+visit(element:ConcreteElementA):void
+visit(element:ConcreteElementB):void
ObjectStructure
Element
+accept(visitor:Visitor):void
ConcreteElementA
+accept(visitor:Visitor):void
ConcreteElementB
+accept(visitor:Visitor):void

Participant Classes

- **Visitor** - This is an interface or an abstract class used to declare the visit operations for all the types of visitable classes.
- **ConcreteVisitor** - For each type of visitor all the visit methods, declared in abstract visitor, must be implemented.
- **Visitable** - is an abstraction which declares the accept operation.
- C**oncreteVisitable** - Those classes implements the Visitable interface or class and defines the accept operation.
- **ObjectStructure** - This is a class containing all the objects that can be visited. I

Example

- We want to create a reporting module in our application to make statistics about a group of customers.
- The statistics should made very detailed so all the data related to the customer must be parsed.


cd: Visitor Customers Example - UML Class Diagram
Main
+main(args:String[]):void
<< interface >>
IVisitor
+visit(customer:Customer):void
+visit(order:Order):void
+visit(item:Item):void
GeneralReport
-customersNo:int
-ordersNo:int
-itemsNo:int
+visit(customer:Customer):void
+visit(order:Order):void
+visit(item:Item):void
+displayResults():void
CustomerGroup
-customers:ArrayList= new ArrayList()
+accept(visitor:IVisitor):void
+addCustomer(customer:Customer):void
<< interface >>
IVisitable
+accept(visitor:IVisitor):void
Item
-name:String
+accept(visitor:IVisitor):void
<< create >>+Item(name:String):Item
+getName():String
+setName(name:String):void
Customer
-name:String
-orders:ArrayList= new ArrayList()
+accept(visitor:IVisitor):void
+getName():String
+setName(name:String):void
+addOrder(order:Order):void
<< create >>+Customer(name:String):Customer
Order
-name:String
-items:ArrayList= new ArrayList()
<< create >>+Order(name:String):Order
<< create >>+Order(name:String,itemName:String):Order
+getName():String
+setName(name:String):void
+accept(visitor:IVisitor):void
+addItem(item:Item):void

Code

- http://www.oodesign.com/visitor-pattern-customers-report-java-sourcecode.html