CS 335 — Software Development

The Interpreter and Visitor Patterns

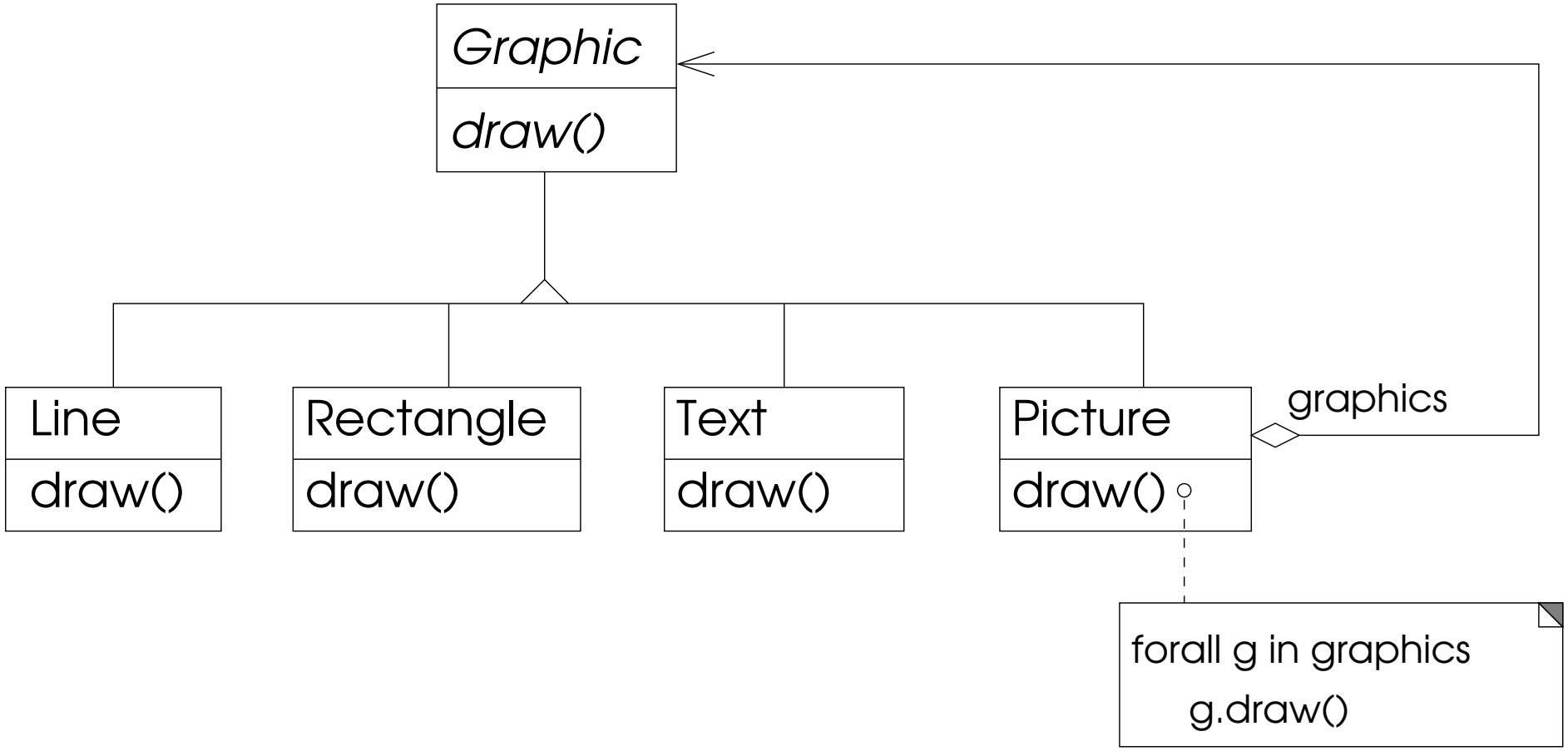
Oct 8 and 10, 2007

Interpreter

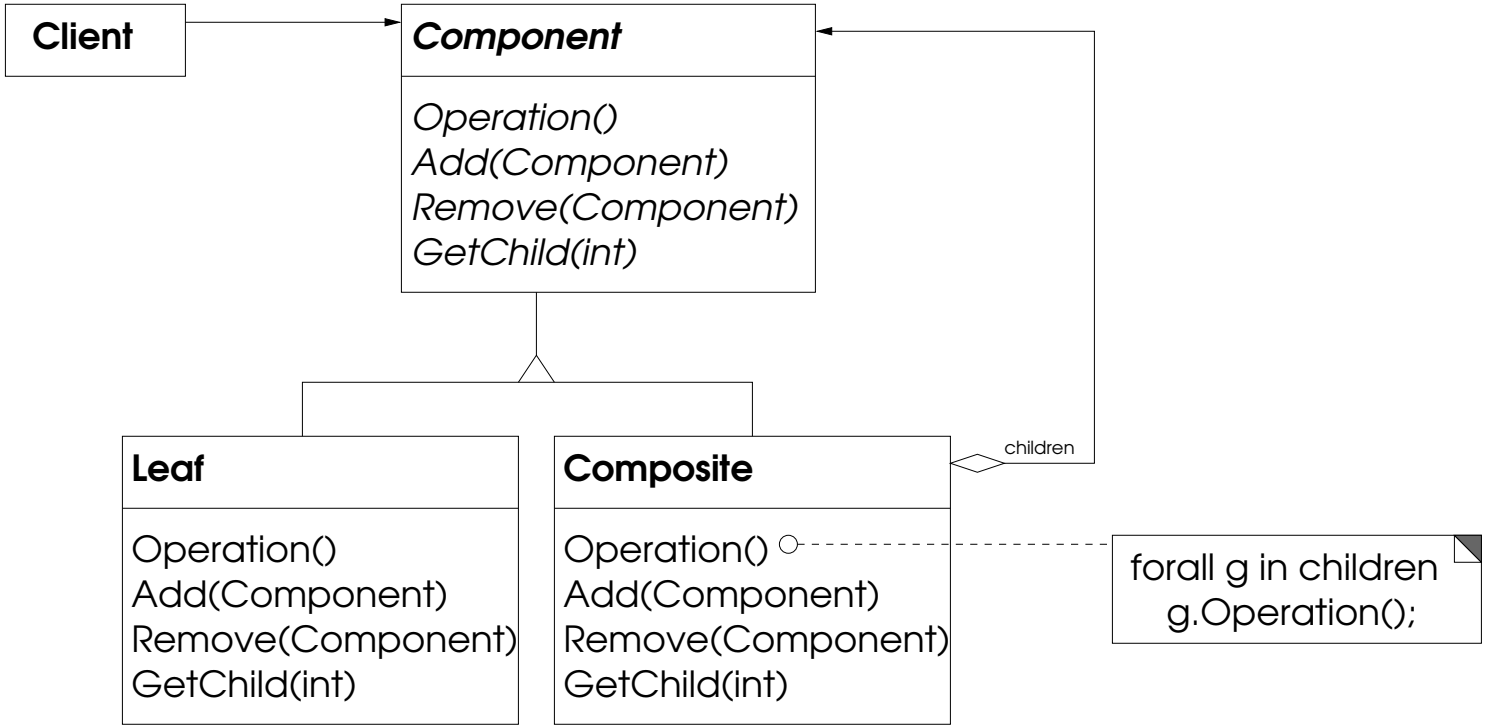
Given an language, define a representation for its garmmar along with an interpreter that uses the representation to interpret sentences in the language.

Gamma et al, *Design Patterns*, pg 243.

Composite


Graphic
draw()
Line
draw()
Rectangle
draw()
Text
draw()
Picture
draw()
graphics
forall g in graphics
g.draw()

Composite

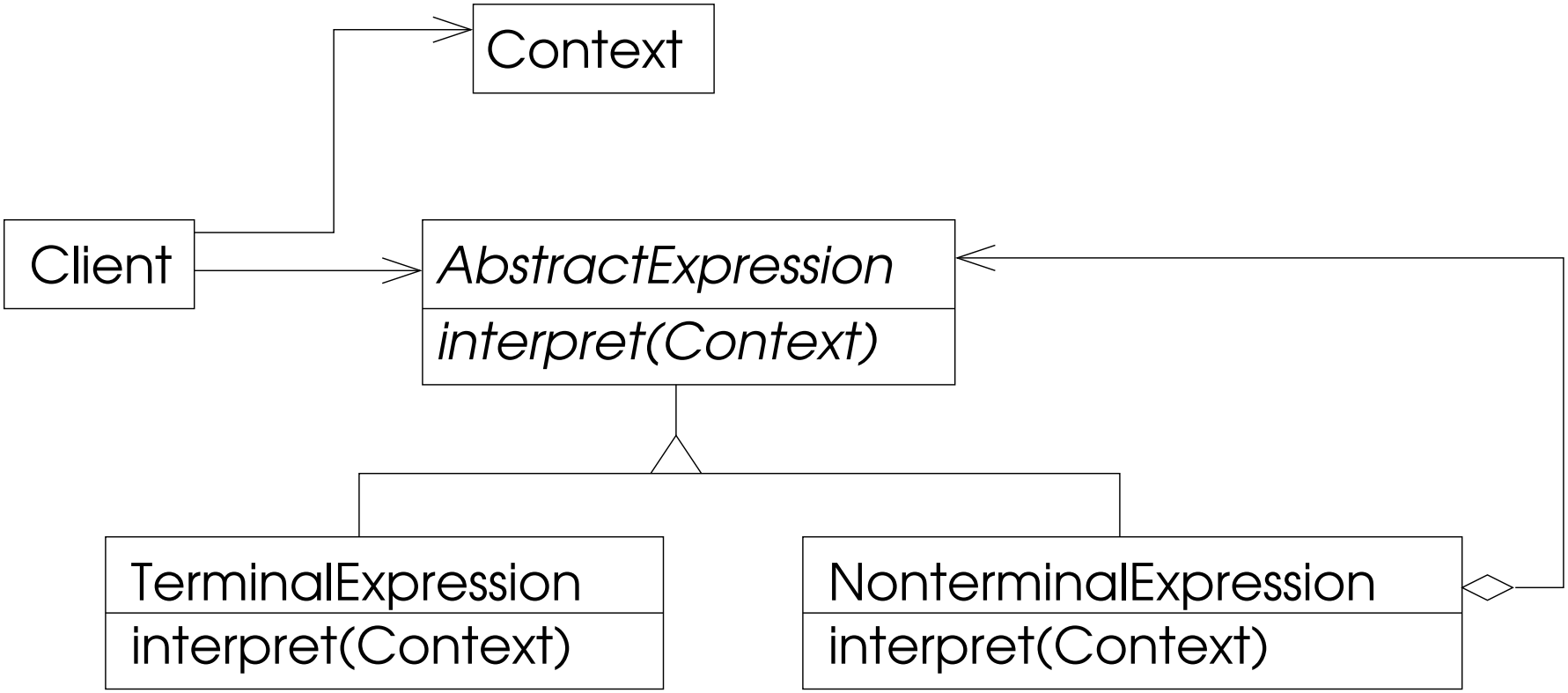

Client
Component
Operation()
Add(Component)
Remove(Component)
GetChild(int)
Leaf
Operation()
Add(Component)
Remove(Component)
GetChild(int)
Composite
Operation()
Add(Component)
Remove(Component)
GetChild(int)
children
forall g in children
g.Operation();

Composite

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Gamma et al, *Design Patterns*, pg 163.

Interpreter



Functional vs OO

```
interface Animal {
    String happyNoise();
    String excitedNoise();
}

class Dog implements Animal {
    String happyNoise() { return "pant pant"; }
    String excitedNoise() { return "bark"; }
}

class Cat implements Animal {
    String happyNoise() { return "purrrrr"; }
    String excitedNoise() { return "meow"; }
}
```

Functional vs OO

```
class Chicken implements Animal {
    String happyNoise() { return "cluck cluck"; }
    String excitedNoise() { return "cockadoodledoo"; }
}
```

Functional vs OO

```
interface Animal {
    String happyNoise();
    String excitedNoise();
    String angryNoise();
}

class Dog implements Animal {
    String happyNoise() { return "pant pant"; }
    String excitedNoise() { return "bark"; }
    String angryNoise() { return "grrrrr"; }
}

class Cat implements Animal {
    String happyNoise() { return "purrrrr"; }
    String excitedNoise() { return "meow"; }
    String angryNoise() { return "hissss"; }
}

class Chicken implements Animal {
    String happyNoise() { return "cluck cluck"; }
    String excitedNoise() { return "cockadoodledoo"; }
    String angryNoise() { return "squaaaack"; }
}
```

Functional vs OO

```
datatype Animal = Dog | Cat ;

fun happyNoise(Dog) = "pant pant"
  | happyNoise(Cat) = "purrrr"

fun excitedNoise(Dog) = "bark"
  | excitedNoise(Cat) = "meow"
```

Functional vs OO

```
fun angryNoise(Dog) = "grrrrr"
  | angryNoise(Cat) = "hisssss"
```

Functional vs OO

```
datatype Animal = Dog | Cat | Chicken;

fun happyNoise(Dog) = "pant pant"
    | happyNoise(Cat) = "purrrr"
    | happyNoise(Chicken) = "cluck cluck";

fun excitedNoise(Dog) = "bark"
    | excitedNoise(Cat) = "meow"
    | excitedNoise(Chicken) = "cockadoodledoo";

fun angryNoise(Dog) = "grrrrr"
    | angryNoise(Cat) = "hisssss"
    | angryNoise(Chicken) = "squaaaack";
```

Functional vs OO

	Dog	Cat	Chicken
happyNoise	pant pant	purrrr	cluck cluck
excitedNoise	bark	meow	cockadoodledoo
angryNoise	grrrrr	hisssss	squaaaaack

Visitor

Represent 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.

Gamma et al, *Design Patterns*, pg 331.