Presented by Group4

COMPOSITE PATTERN

What is Composite Pattern

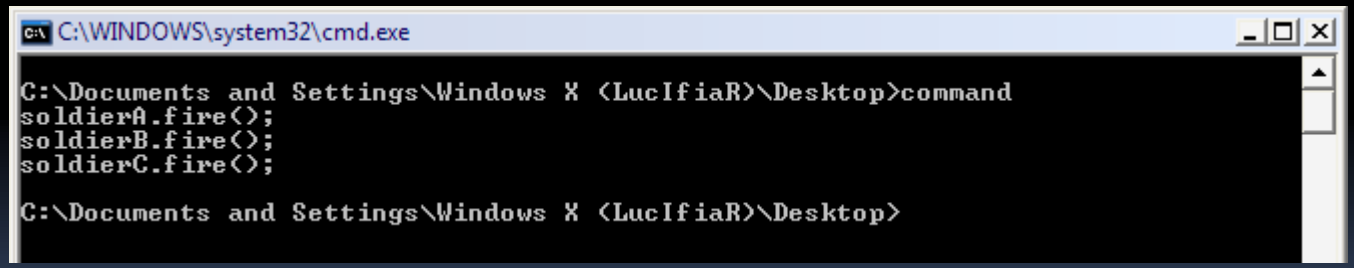
- Structural Pattern
- Allows a group of objects to be treated as single instance of object
- Intend to compose objects into tree structures to represent part-whole hierarchies

Motivation

- Complexity in tree-structured data
- Interface to treat complex objects uniformly
- Design to “has-a” relationship

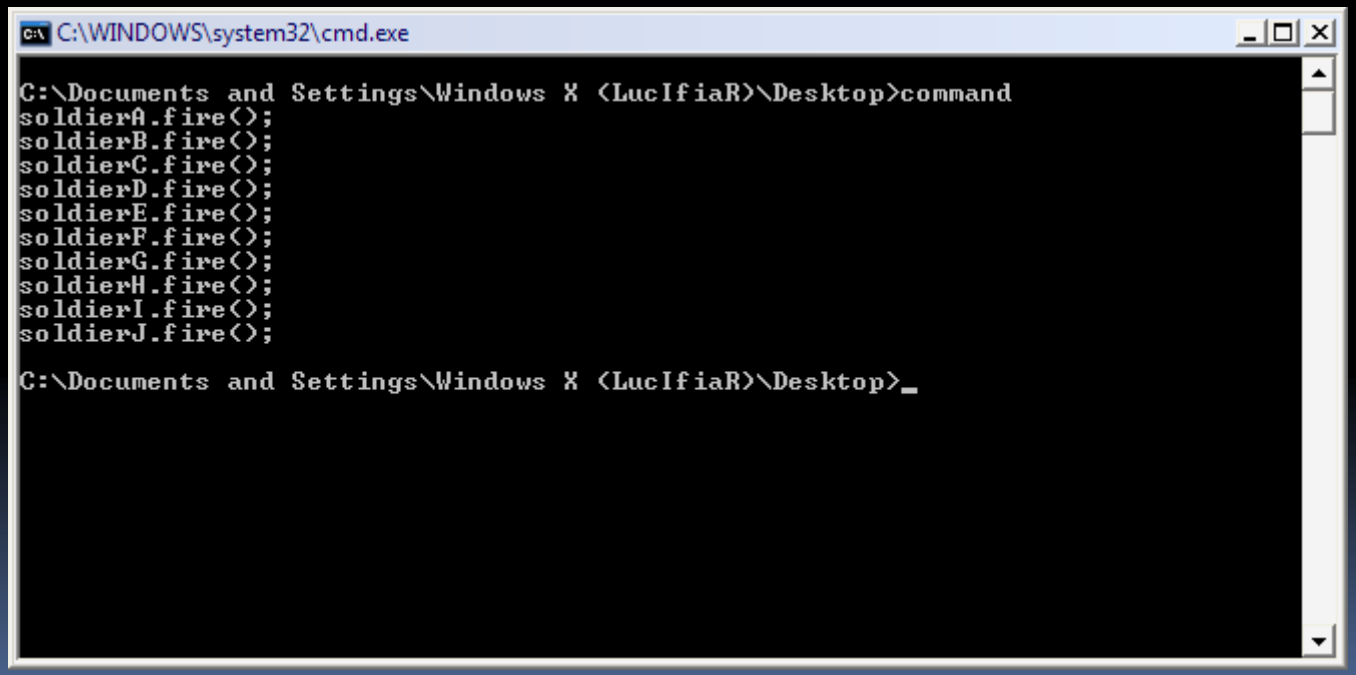
When to use

- Supposed you are a commander
- You're giving command "fire" to your soldiers
- If you has 3 soldiers, that's fine, right?



When to use (Contd.)

- What if you have 10 soldiers? Not going fine…

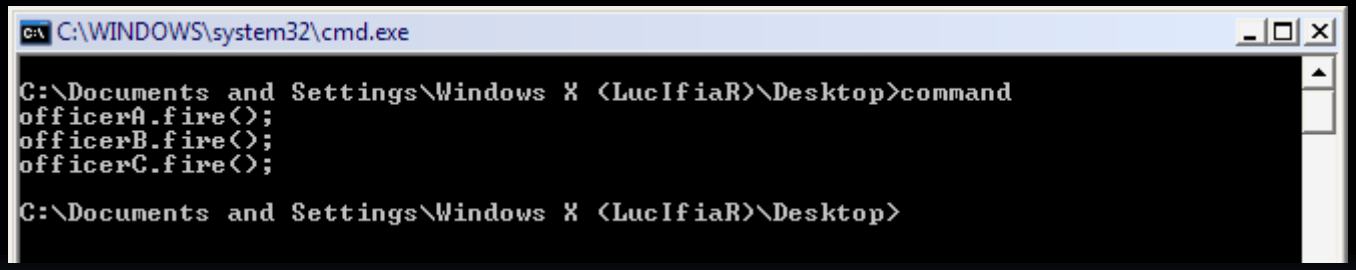


When to use (Contd.)

- What if you have 100 or 1,000 soldiers?
- I bet you'll be shot before giving your command to all of them…

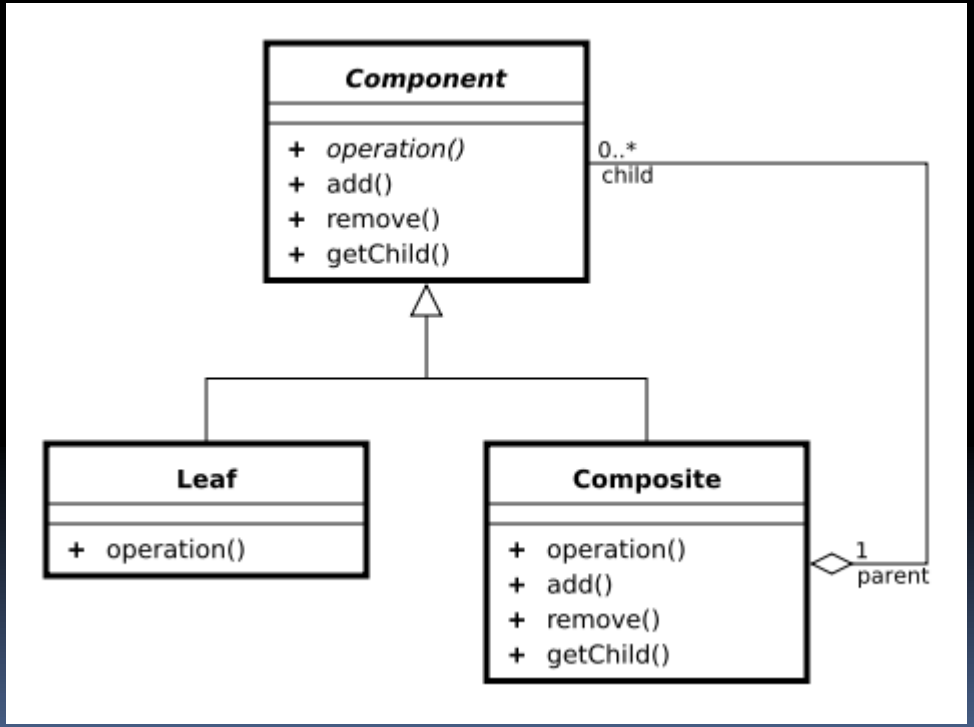
When to use (Contd.)

- How to make task simplier?
- Assign an officer to give command to their soldiers



- Now you have to command only 1-3 officers

Structure


Component
+ operation()
+ add()
+ remove()
+ getChild()
0..*
child
Leaf
+ operation()
Composite
+ operation()
+ add()
+ remove()
+ getChild()
1
parent

Structure (Contd.)

- **Component**
- is the abstraction for all components
- declares the interface for objects in the composition
- implements default behavior for the interface
- declares an interface for accessing and managing its child components
- (optional) defines an interface for accessing a components's parent in the recursive structure, and implements it if that's appropriate

Structure (Contd.)

- **Leaf**
- represents leaf objects in the composition
- implements all Component methods

Structure (Contd.)

- **Composite**
- represents a composite Component (component having children)
- implements methods to manipulate children
- implements all Component methods, generally by delegating them to its children

Examples

```
import java.util.ArrayList;

interface Commander {

   //Fire command
   public void fire();

}
```

Examples (Contd.)

```
class Officer implements Commander {

   //Collection of soldiers.
   private ArrayList<Commander> soldiers = new
     ArrayList<Commander>();

   //Group fire command
   public void fire() {
      for (Commander soldier : soldiers) {
         soldier.fire();
      }
   }
```

Examples (Contd.)

```
//Adds the soldier to officer's control.
   public void add(Commander soldier) {
      soldiers.add(soldier);
   }

   //Removes the soldier from officer's control.
   public void remove(Commander soldier) {
      soldiers.remove(soldier);
   }

}
```

Examples (Contd.)

```
class Soldier implements Commander {

   //Fire command.
   public void fire() {
      System.out.println("Fire!");
   }

}
```

Examples (Contd.)

```
public class Command {

    public static void main(String[] args) {
        //Initialize four soldiers
        Soldier soldier1 = new Soldier();
        Soldier soldier2 = new Soldier();
        Soldier soldier3 = new Soldier();
        Soldier soldier4 = new Soldier();
```

Examples (Contd.)

```
//Initialize three officers
    Officer officer = new Officer();
    Officer officer1 = new Officer();
    Officer officer2 = new Officer();

    //Composes the officers
    officer1.add(soldier1);
    officer1.add(soldier2);
    officer1.add(soldier3);
    officer2.add(soldier4);
```

Examples (Contd.)

```
officer.add(officer1);
      officer.add(officer2);

      //Prints the complete fire command (four
    times the string "Fire!").
      officer.fire();
   }
}
```

Questions?