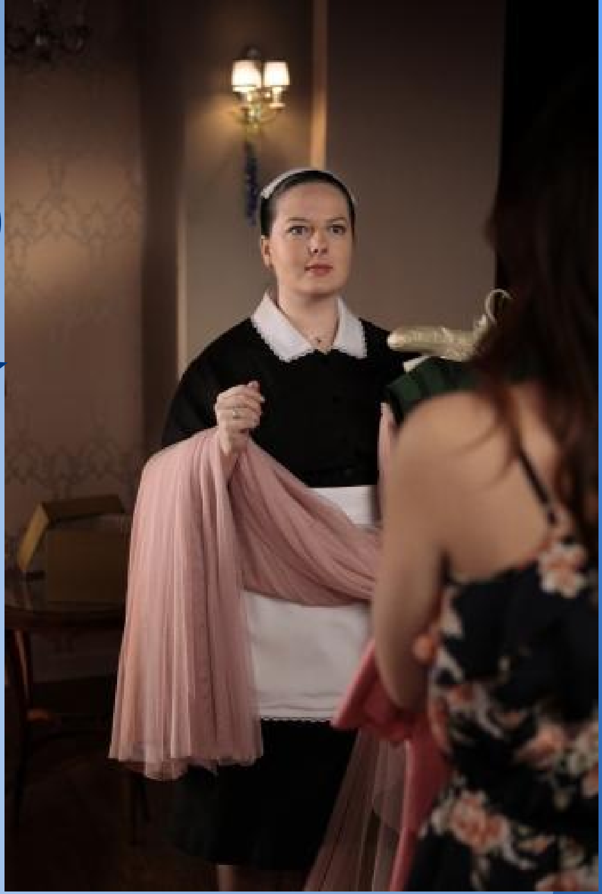
Servant Pattern

Mengyu Yin
Anisha Smith

Servant?

I want to have my dresses and coats washed clean.



I wanna fries, hamburger, and coke.

What is servant pattern?

A behavioral pattern used to offer some functionality to a group of classes without defining that functionality in each of them.

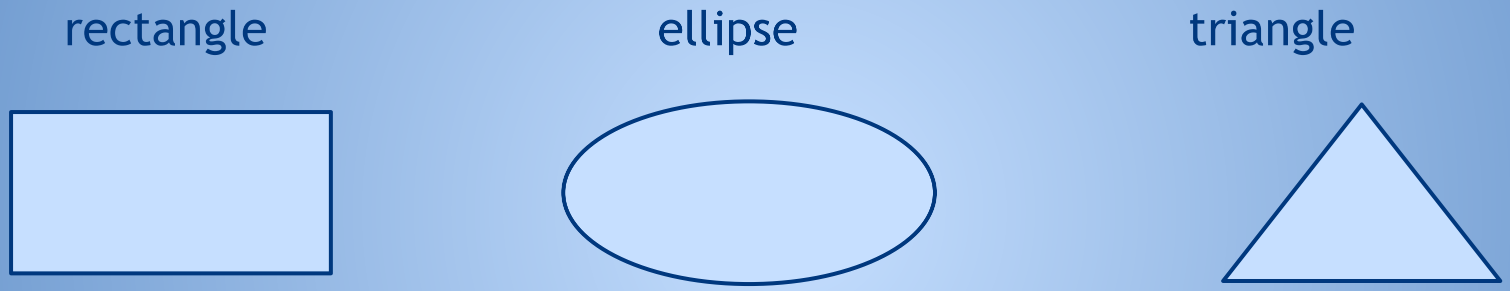
Another type of command pattern.

How is this applicable?

- Provides functionality without specificity
- Each class doesn't need its own definition of the behavior
- Objects are taken as parameters, method purely defines a behavior
- Anonymous

Example

Example classes representing geometric objects:



Using the servant design pattern, a method can be created that could do something to this series of distinct objects

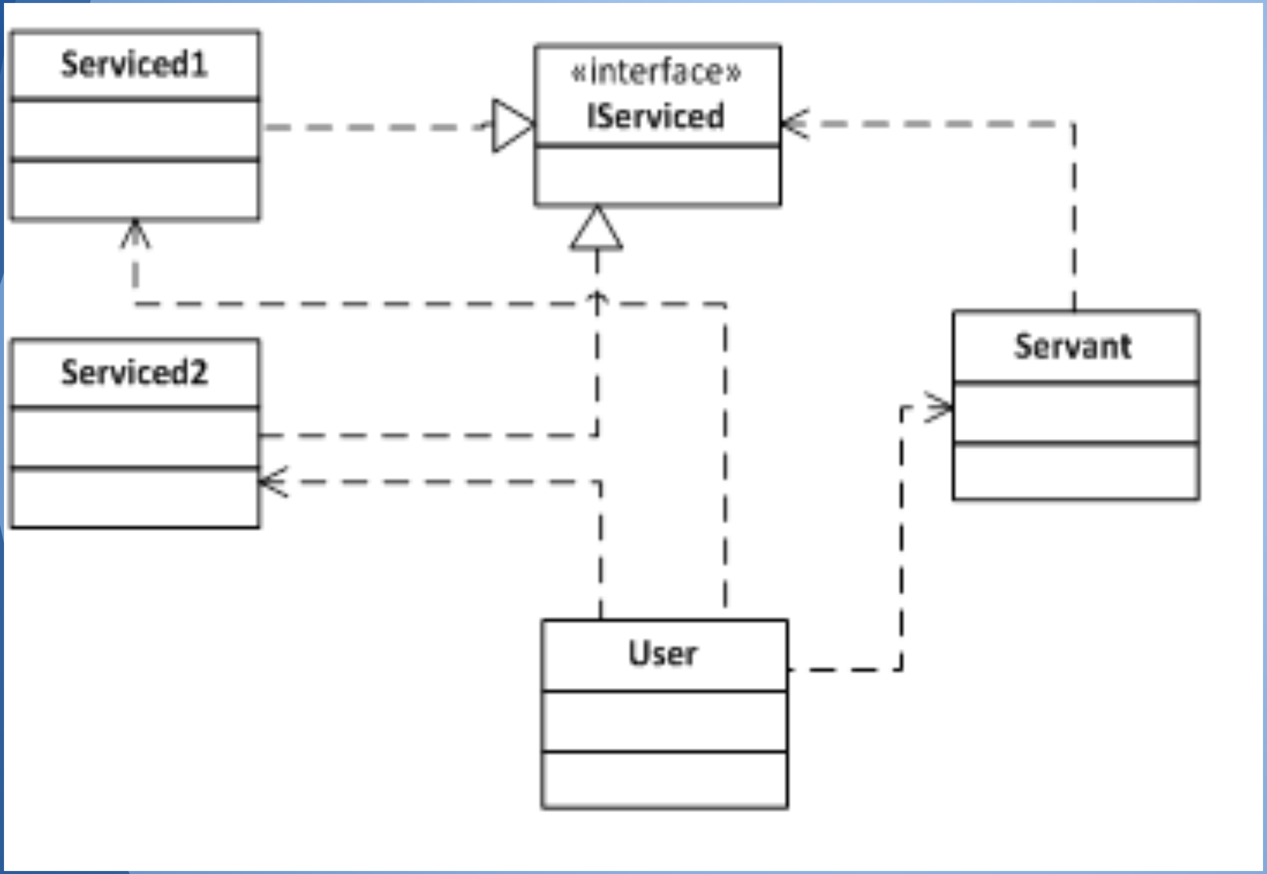
- Ex. a move function which would shift the objects in space

Code

```
/ Servant class, offering its functionality to classes implementing
// Movable Interface
public class MoveServant {
        // Method, which will move Movable implementing class to position where
        public void moveTo(Movable serviced, Position where) {
                // Do some other stuff to ensure it moves smoothly and nicely, this is
                // the place to offer the functionality
                serviced.setPosition(where);
        }

        // Method, which will move Movable implementing class by dx and dy
        public void moveBy(Movable serviced, int dx, int dy) {
                // this is the place to offer the functionality
                dx += serviced.getPosition().xPosition;
                dy += serviced.getPosition().yPosition;
                serviced.setPosition(new Position(dx, dy));
        }
}
```

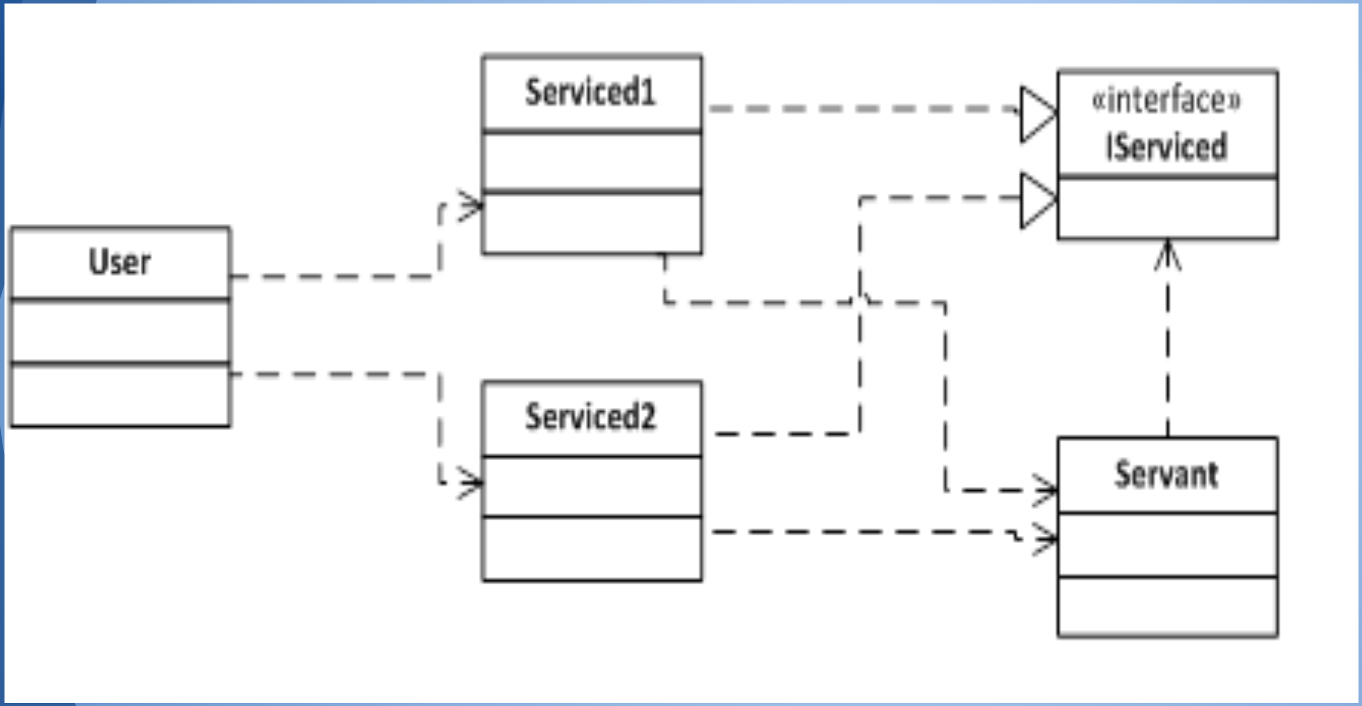
```
// Interface specifying what serviced classes needs to implement, to be serviced by servant.
public interface Movable {
        public void setPosition(Position p);
        public Position getPosition();
}
// One of geometric classes
public class Triangle implements Movable {
        // Position of the geometric object on some canvas
        private Position p;
        public void setPosition(Position p) {
                this.p = p;  }
        public Position getPosition() {
                return this.p; }}

public class Ellipse implements Movable {
        private Position p;
        public void setPosition(Position p) {
                this.p = p;  }
        public Position getPosition() {
                return this.p;  }}
```

Implementation I:



- User knows of servant method, calls it
- Serviced classes are unaware of servant
- Serviced classes interact with IServiced interface
- Serviced classes are then parameters passed to servant function

Implementation II:



- User is unaware of servant methods/classes
- User interacts with serviced classes
- Serviced classes use servant functions
- Servant functions provide service through IServiced interface

Advantages and Disadvantages

- specialized but still provides a common service

- allows for use over a variety of objects, no need to continually redefine for each

- generalized; some objects may be too specific

- requires that objects it acts on have common abilities

Sources

http://www.scribd.com/doc/49845211/Design-Patterns
http://shimonpeter.blogspot.com/2011/11/servant-design-pattern-in-java-example.htm
http://edu.pecinovsky.cz/papers/2006_ITiCSE_Design_Patterns_First.pdfl
http://en.wikipedia.org/wiki/Design_pattern_Servant