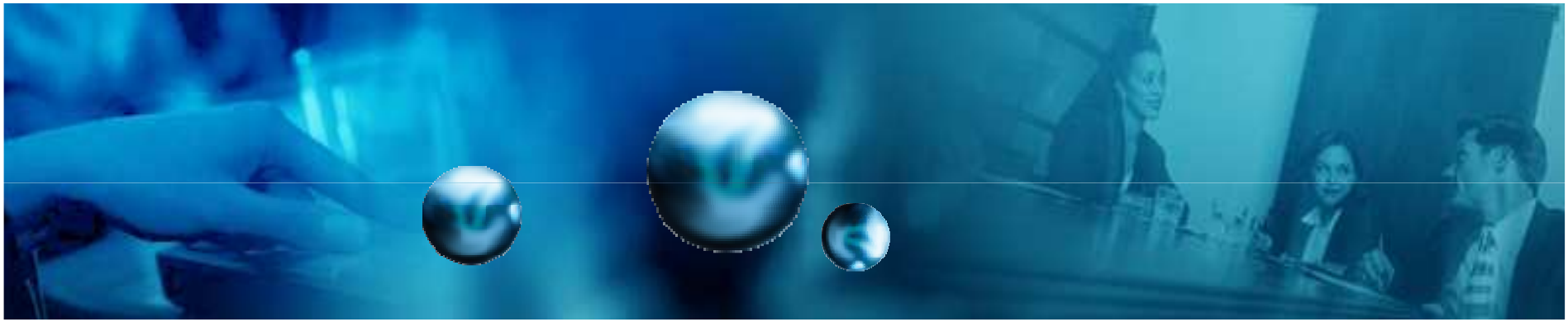
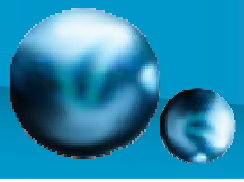
Command Pattern



XinniX Soft

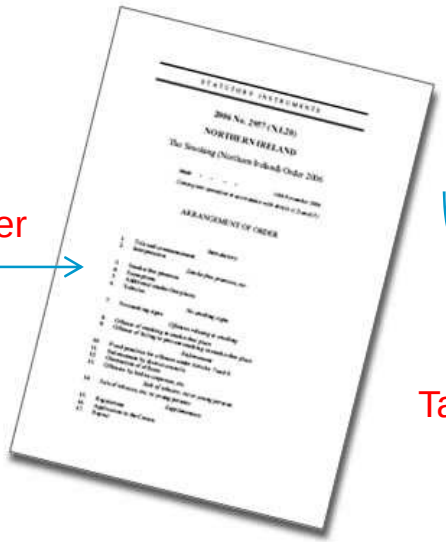


Introduction

- The Command pattern is a design pattern in which objects are used to represent actions. A command object encapsulates an action and its parameters.

Introduction

Create an Order

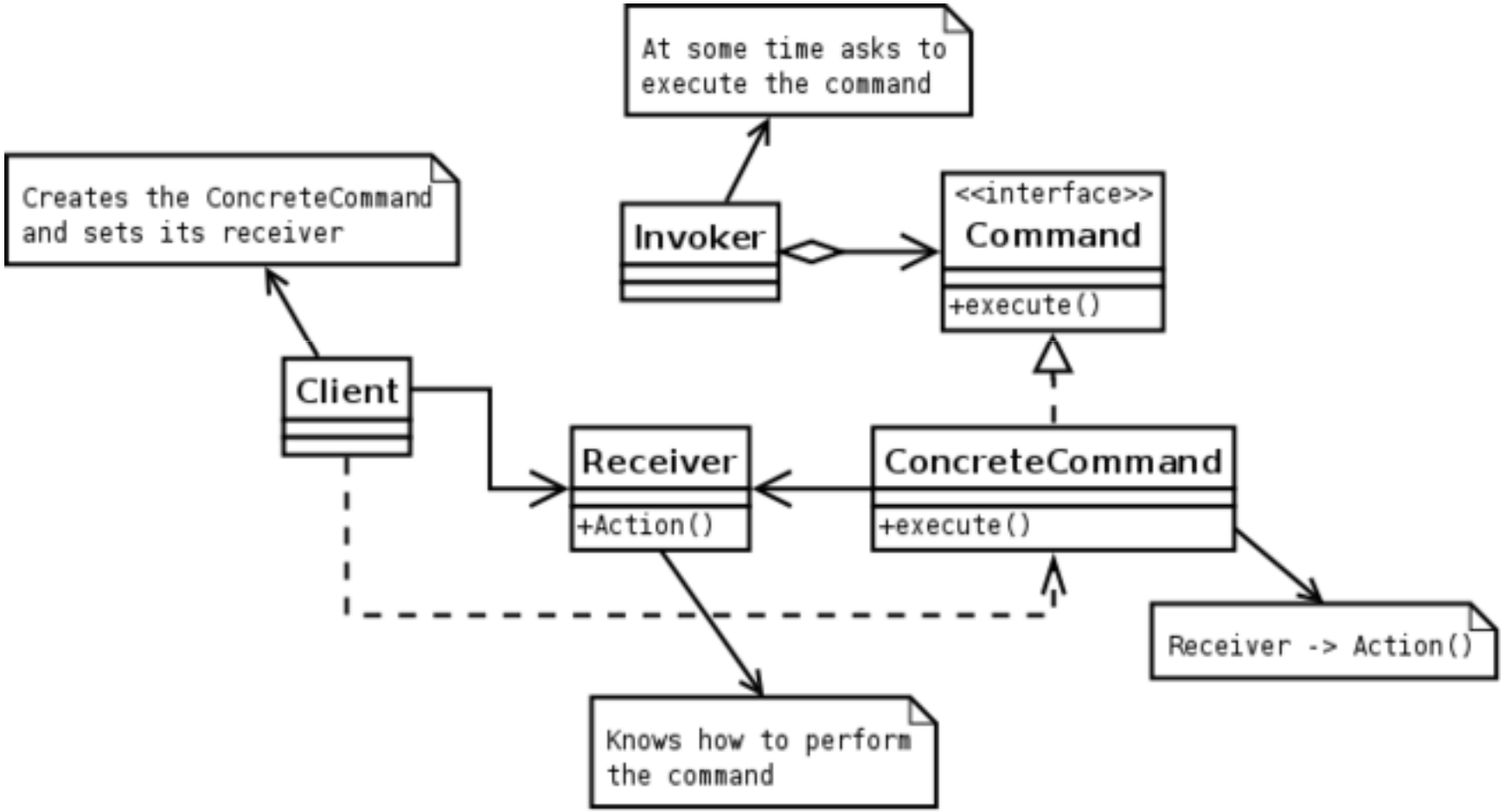


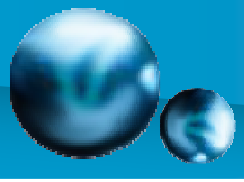
Take an Order

Order Ups

cook

Command Pattern UML

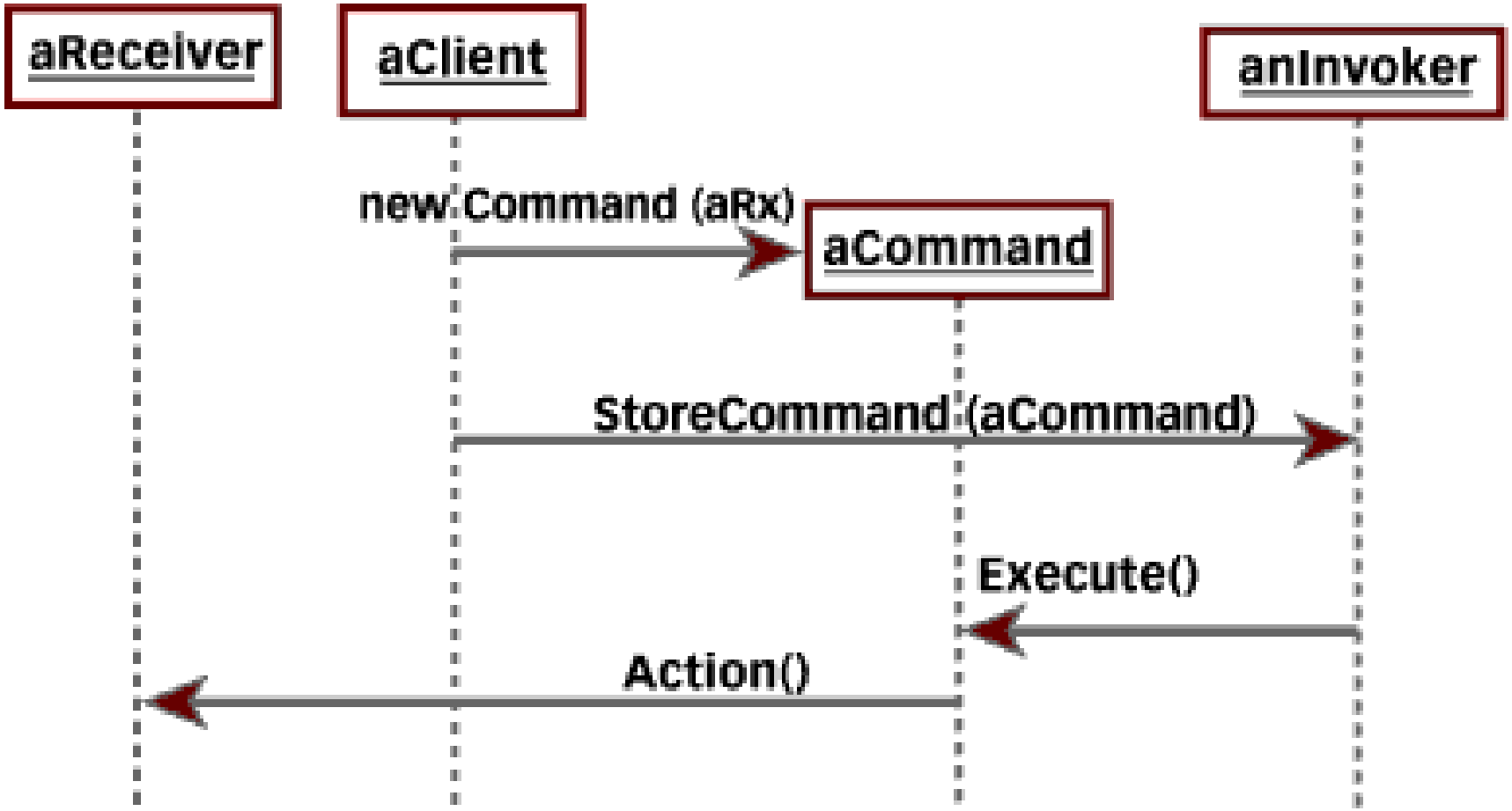




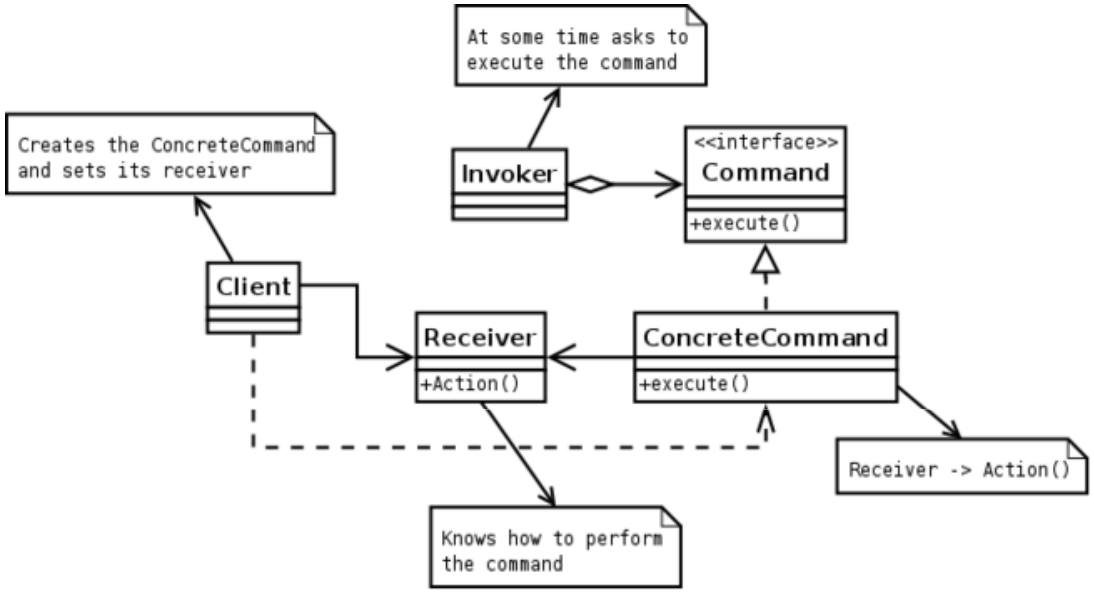
Participants

- Command
  - declares an interface for executing an operation.
- ConcreteCommand (PasteCommand, OpenCommand)
  - defines a binding between a Receiver object and an action.
  - implements Execute by invoking the corresponding operation(s) on Receiver.
- Client (Application)
  - creates a ConcreteCommand object and sets its receiver.
- Invoker
  - asks the command to carry out the request.
- Receiver
  - knows how to perform the operations associated with carrying out a request. Any class may serve as a Receiver.

sequence diagram



Implementation (Command)




```
public interface Command {
        public abstract void execute ( );
}
```

Implementation(invoke)

```
class RemoteSwitch {
        private Command onCommand;
        private Command offCommand;
public Switch( Command Up, Command Down) {
            UpCommand = Up;
            DownCommand = Down;
}

            public void   on( ) {
                        onCommand . execute ( ) ;
            }

            public  void    off( ) {
                        offCommand . execute ( );
            }
 }
```

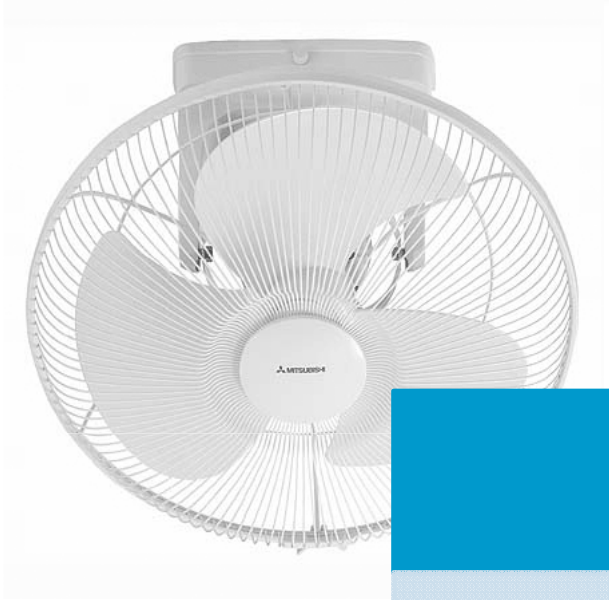
Implementation (Receiver - Fan)

```
class Fan {

public void startRotate() {
        System.out.println("Fan is rotating");
}

 public void stopRotate() {
        System.out.println("Fan is not
rotating"); }
}
```

Implementation (Receiver - Light)

```
class Light {

public void turnOn( ) {
        System.out.println("Light is on ");
}

public void turnOff( ) {
        System.out.println("Light is off"); }
}
```

Fan Vs Light



Fan
• startRotate() • stopRotate()

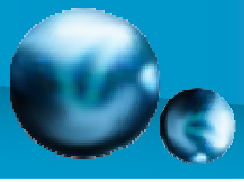
Light
• turnOn( ) • turnOff( )

Implementation(ConcreteCommand) [Light]

```
class LightOnCommand implements Command {
        private Light myLight;
        public LightOnCommand ( Light L) {
                myLight  =  L;
        }
        public void execute( ) {
                myLight . turnOn( );
        }
}
class LightOffCommand implements Command {
        private Light myLight;
        public LightOffCommand ( Light L) {
                myLight  =  L;
        }
        public void execute( ) {
                myLight . turnOff( );
        }
}
```

Implementation (ConcreteCommand) [Fan]

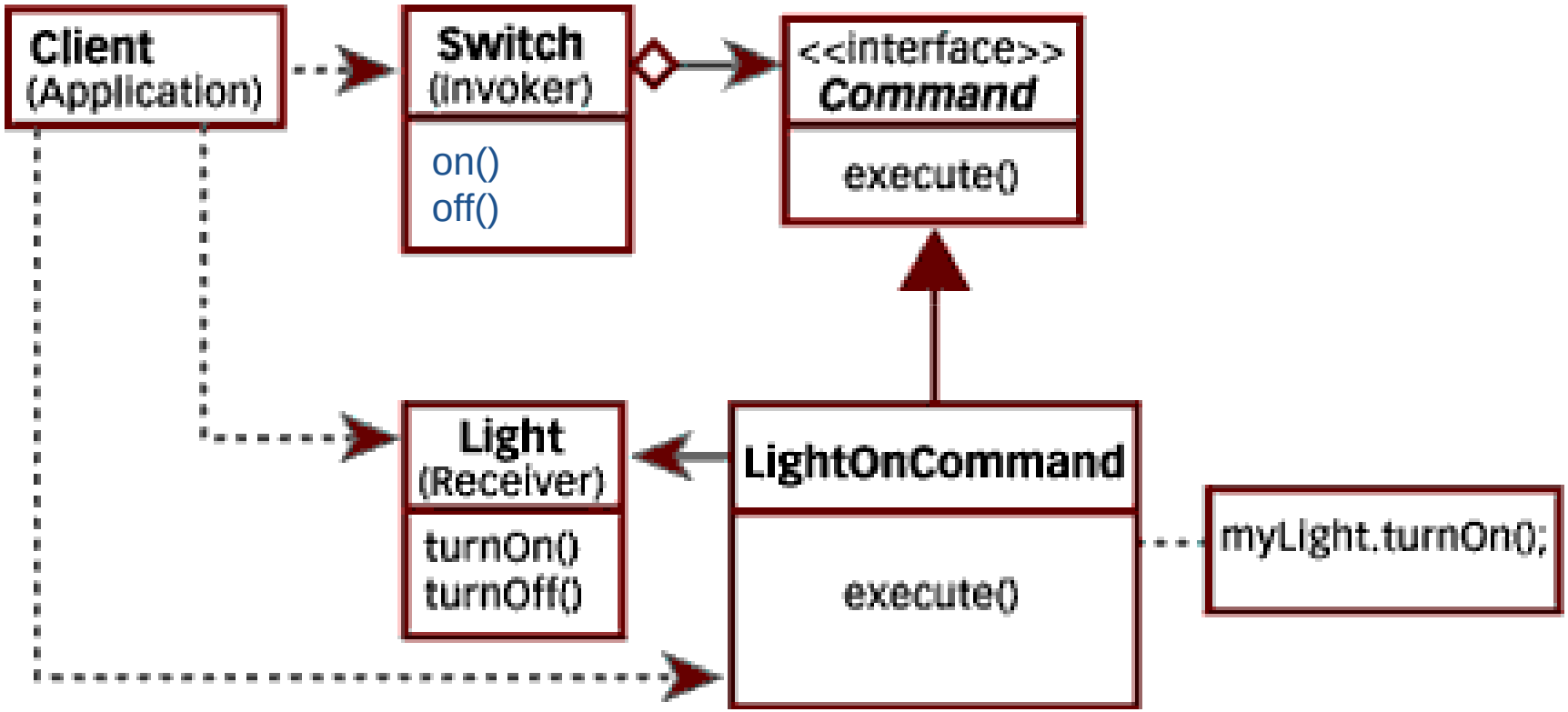
```
class FanOnCommand implements Command {
        private Fan myFan;
        public FanOnCommand ( Fan F) {
                myFan  =  F;
        }
        public void execute( ) {
                myFan . startRotate( );
        }
}
class FanOffCommand implements Command {
        private Fan myFan;
        public FanOffCommand ( Fan F) {
                myFan  =  F;
        }
        public void execute( ) {
                myFan . stopRotate( );
        }
}
```



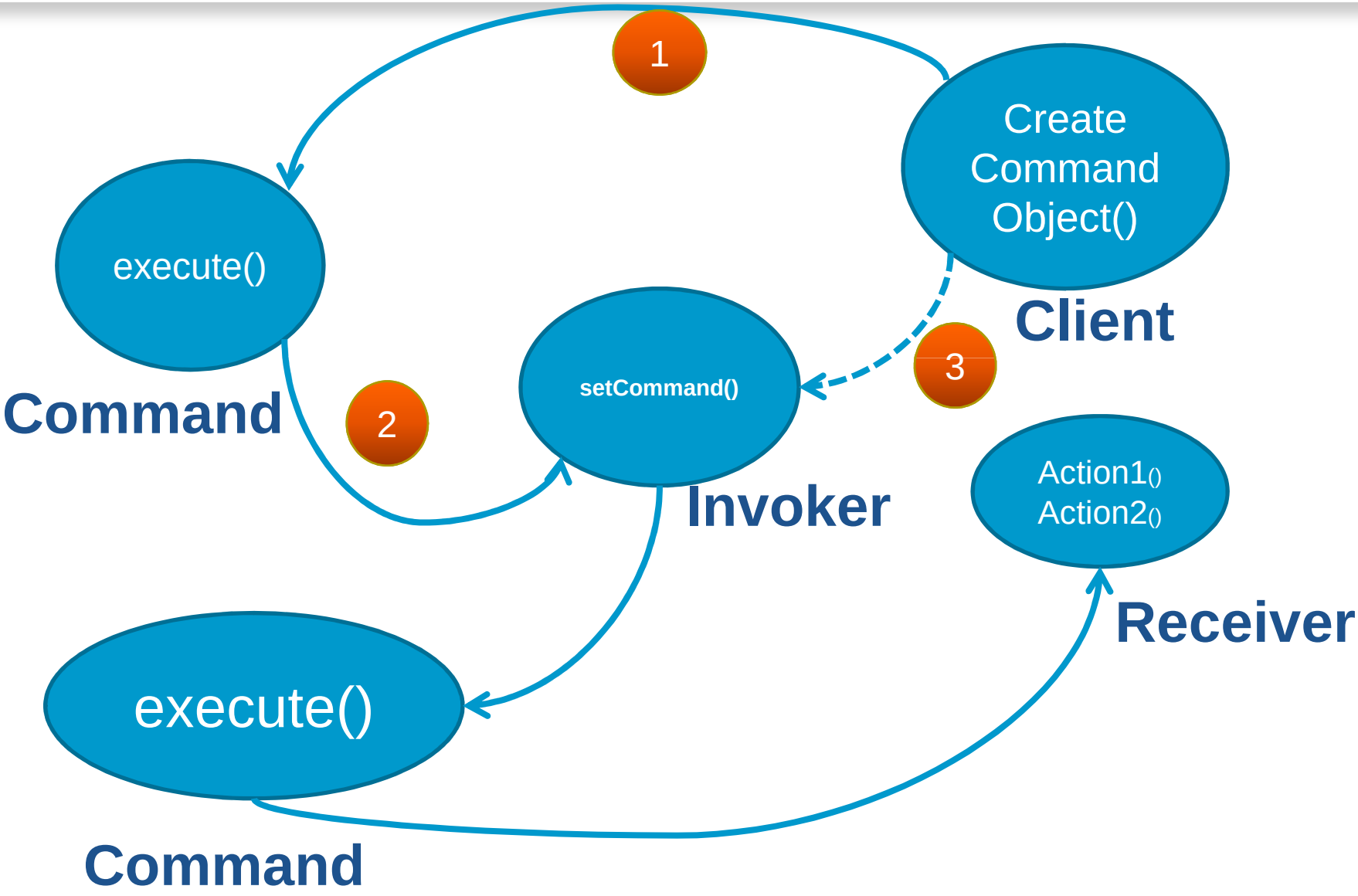
Implementation (client)

```
public class TestCommand {
public static void main(String[] args) {

Light light = new Light( );
LightOnCommand lightOn = new LightOnCommand(light);
LightOffCommand lightOff = new LightOffCommand(light);
RemoteSwitch remoteSwitch = new RemoteSwitch(lightOn , lightOff);
remoteSwitch.on();
remoteSwitch.off( );

Fan fan = new Fan( );
FanOnCommand fanOn = new FanOnCommand(fan);
FanOffCommand fanOff = new FanOffCommand(fan);
RemoteSwitch remoteSwitch2 = new RemoteSwitch(fanOn , fanOff);
remoteSwitch2.on();
remoteSwitch2.off( );
```

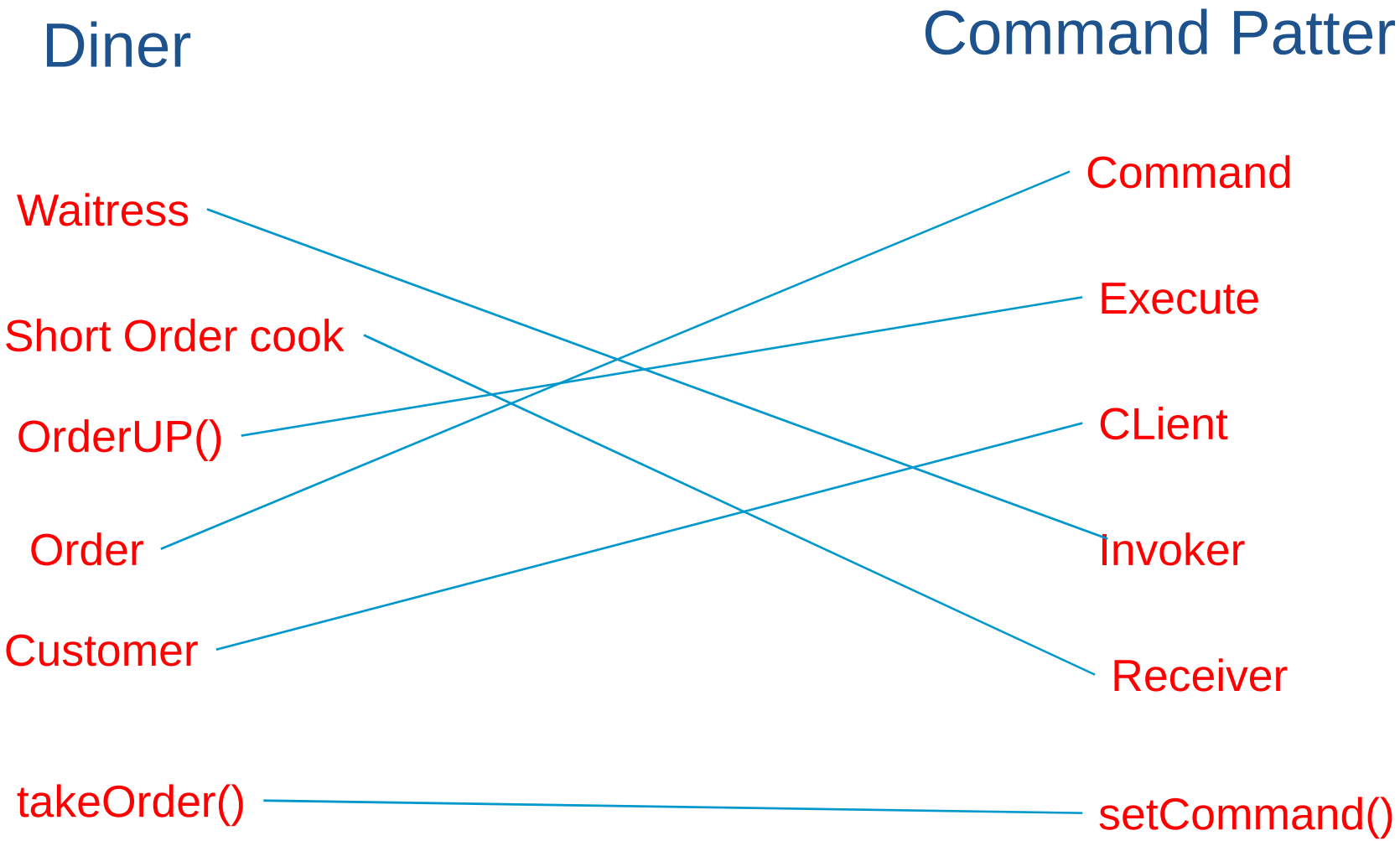
What is happen? UML Help you

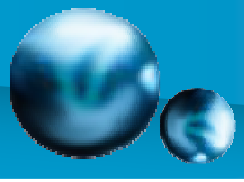


What ‘s happen? With state diagram



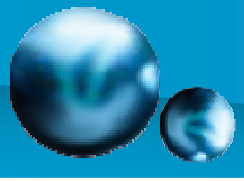
Questions?


Diner
Command Pattern
Waitress
Short Order cook
OrderUP()
Order
Customer
takeOrder()
Command
Execute
CLient
Invoker
Receiver
setCommand()



Conclusion

- The Command design pattern encapsulates the concept of the command into an object. The issuer holds a reference to the command object rather than to the recipient. The issuer sends the command to the command object by executing a specific method on it. The command object is then responsible for dispatching the command to a specific recipient to get the job done.



Reference

- Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides *Design Patterns Elements of Reusable Object-Oriented Software* Addison-Wesley, 1995 *Dr. Dobb's Journal,* January 1998"Java ReflectionNot just for tool developers," by Paul Tremblett http://www.ddj.com/articles/1998/9801/9801c/9801c.htm
- Sun's Reflection page http://java.sun.com/docs/books/tutorial/reflect/index