Design Patterns

Command

COMP3607
Object Oriented Programming II

Week 6

Outline

- Topics
  - Command Design Pattern

5

Command Design Pattern

The Command design pattern is an object behavioural pattern.

It allows the requester of a particular action to be decoupled from the object that performs the action.

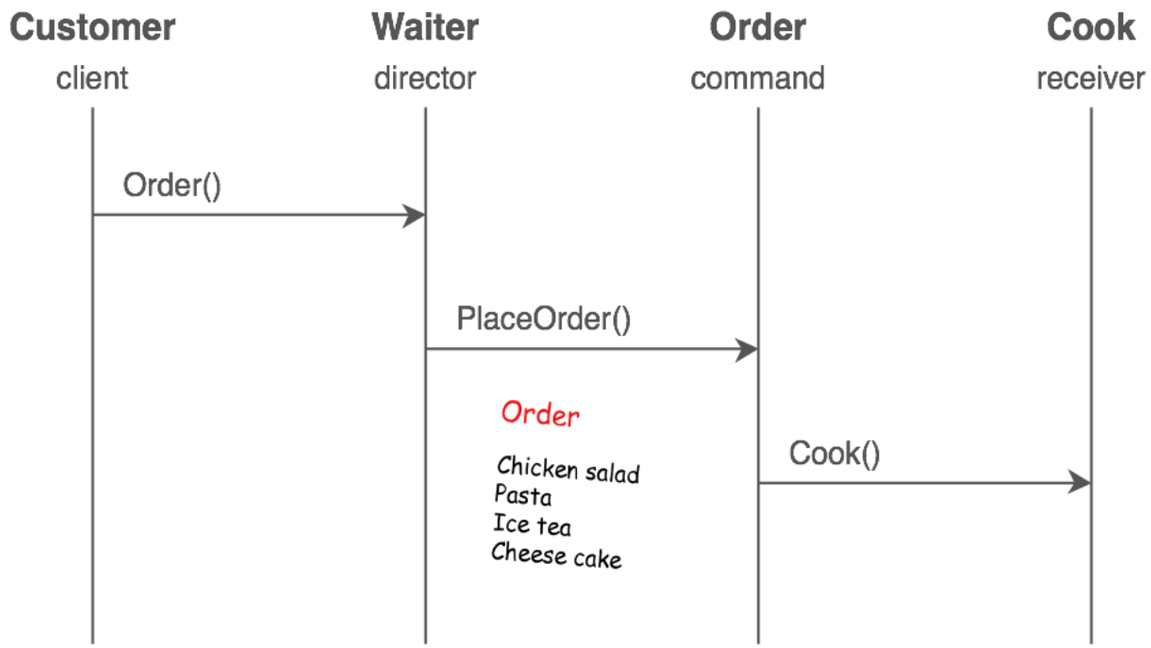
Example Scenario

One example of the command pattern being executed in the real world is the idea of a table order at a restaurant:

The waiter takes the order from the customer (command).

The order is then queued for the kitchen staff.

The waiter tells the chef that a new order has come in, and the chef has enough information to cook the meal.



Decoupling Producers from Consumers

The Command pattern declares an **interface** for all commands, providing a simple **execute( )** method which asks the **Receiver** of the command to carry out an operation.

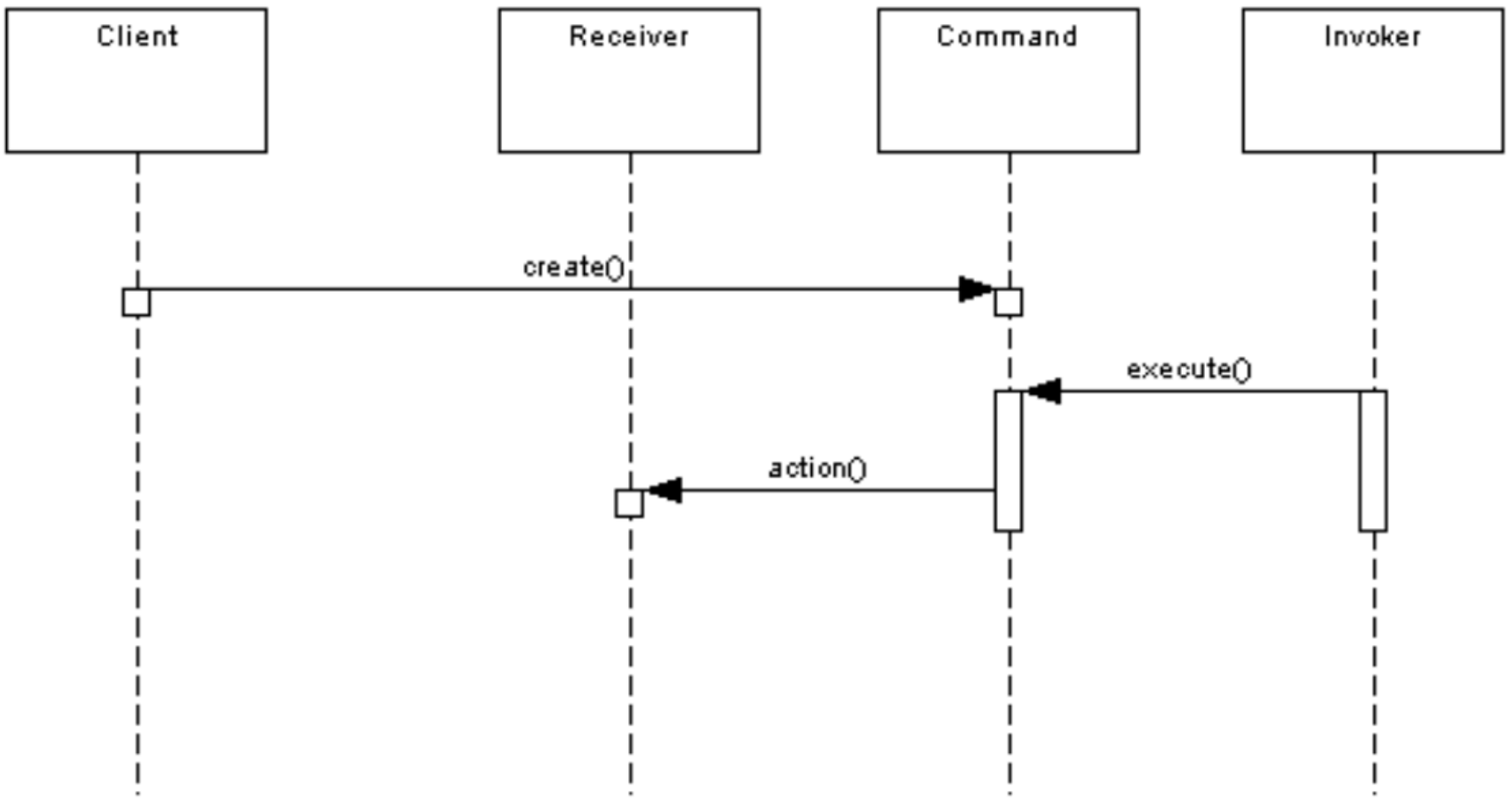
The **Receiver** has the knowledge of what to do to carry out the request.

The **Invoker** holds a command and can get the **Command** to execute a request by calling the execute method.

The **Client** creates **ConcreteCommands** and sets a **Receiver** for the command.

The **ConcreteCommand** defines a binding between the action and the receiver. When the **Invoker** calls execute the **ConcreteCommand** will run one or more actions on the **Receiver**.

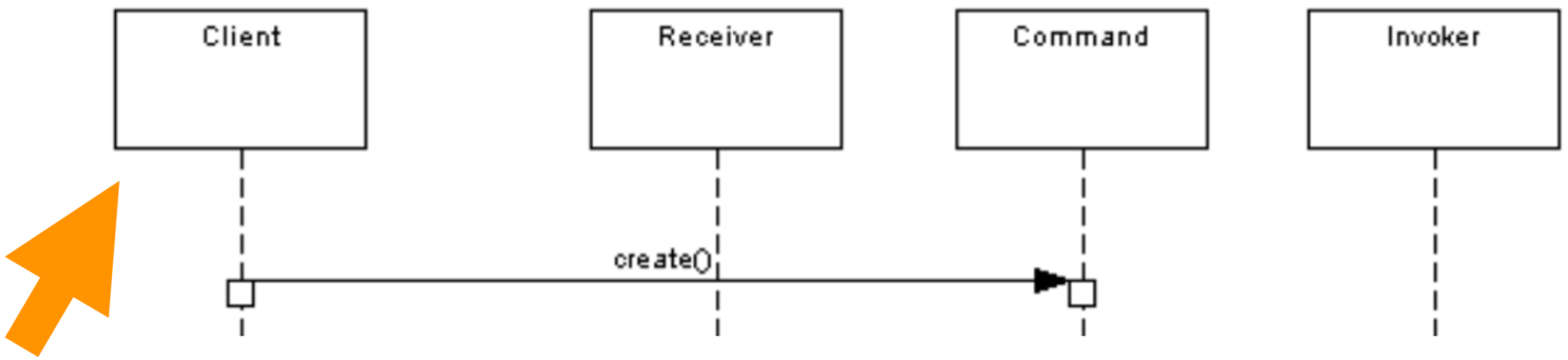
Sequence of Events



Sequence of Events

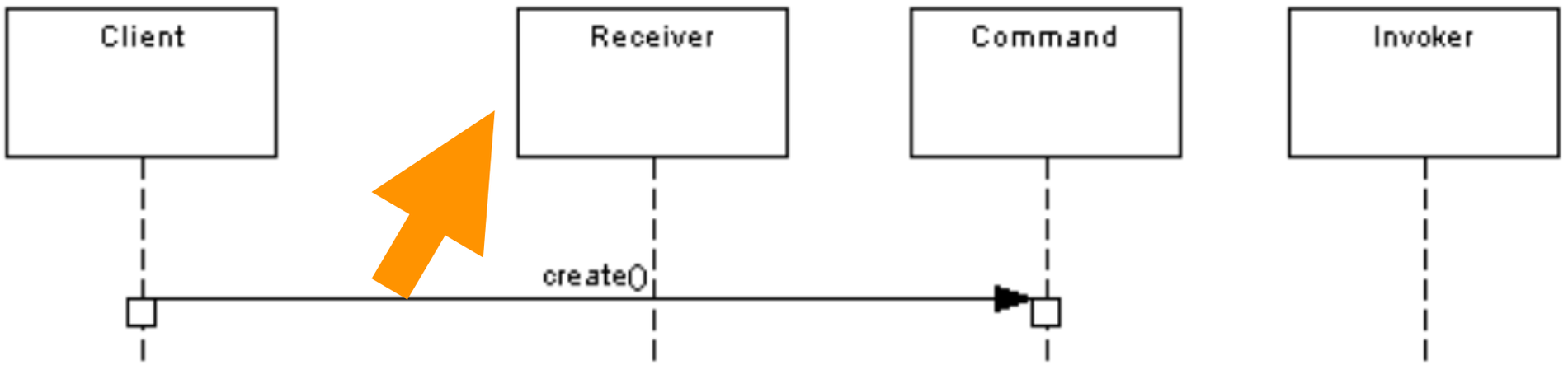
The Command pattern declares an **interface** for all commands, providing a simple **execute( )** method which asks the **Receiver** of the command to carry out an operation.

Sequence of Events



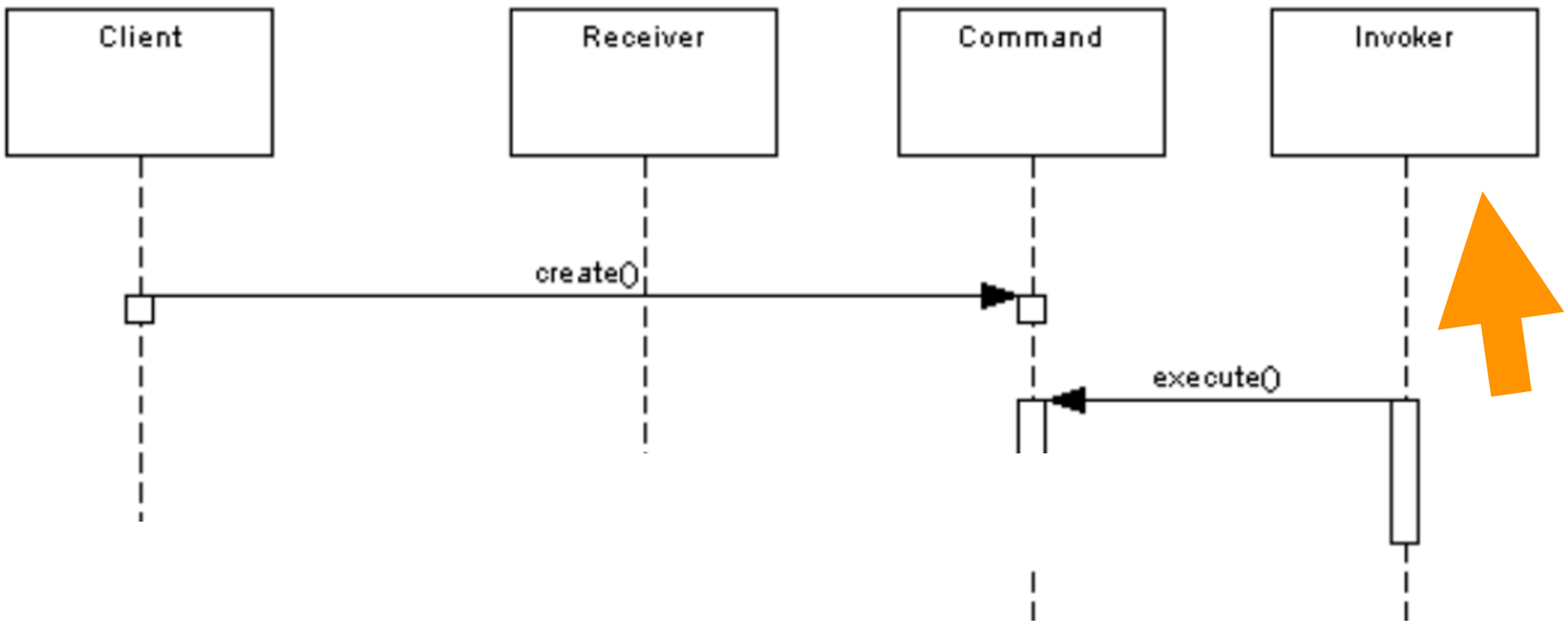
The **Client** creates a **ConcreteCommand** and sets a **Receiver** for the command

Sequence of Events



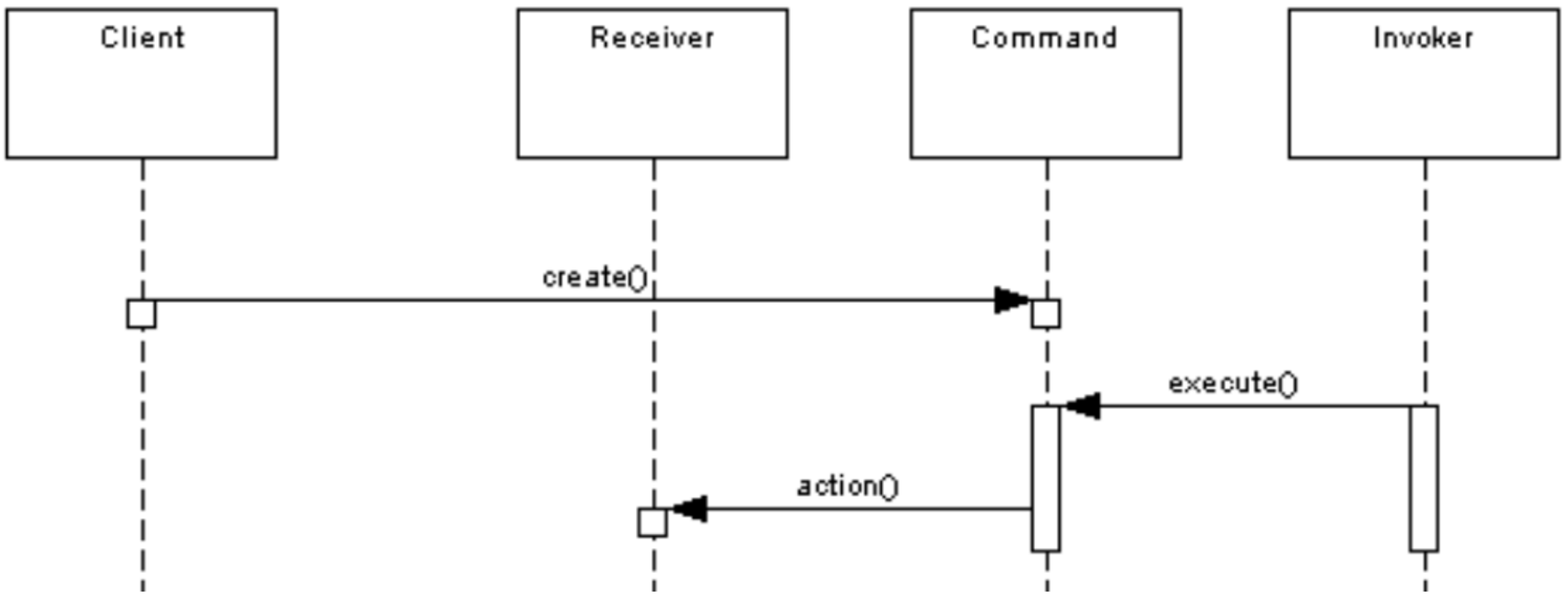
The **Receiver** has the knowledge of what to do to carry out the request. It doesn’t come into the picture just yet though.

Sequence of Events



The **Invoker** has a reference to **Command** and can get it to execute a request by calling the **execute( )** method.

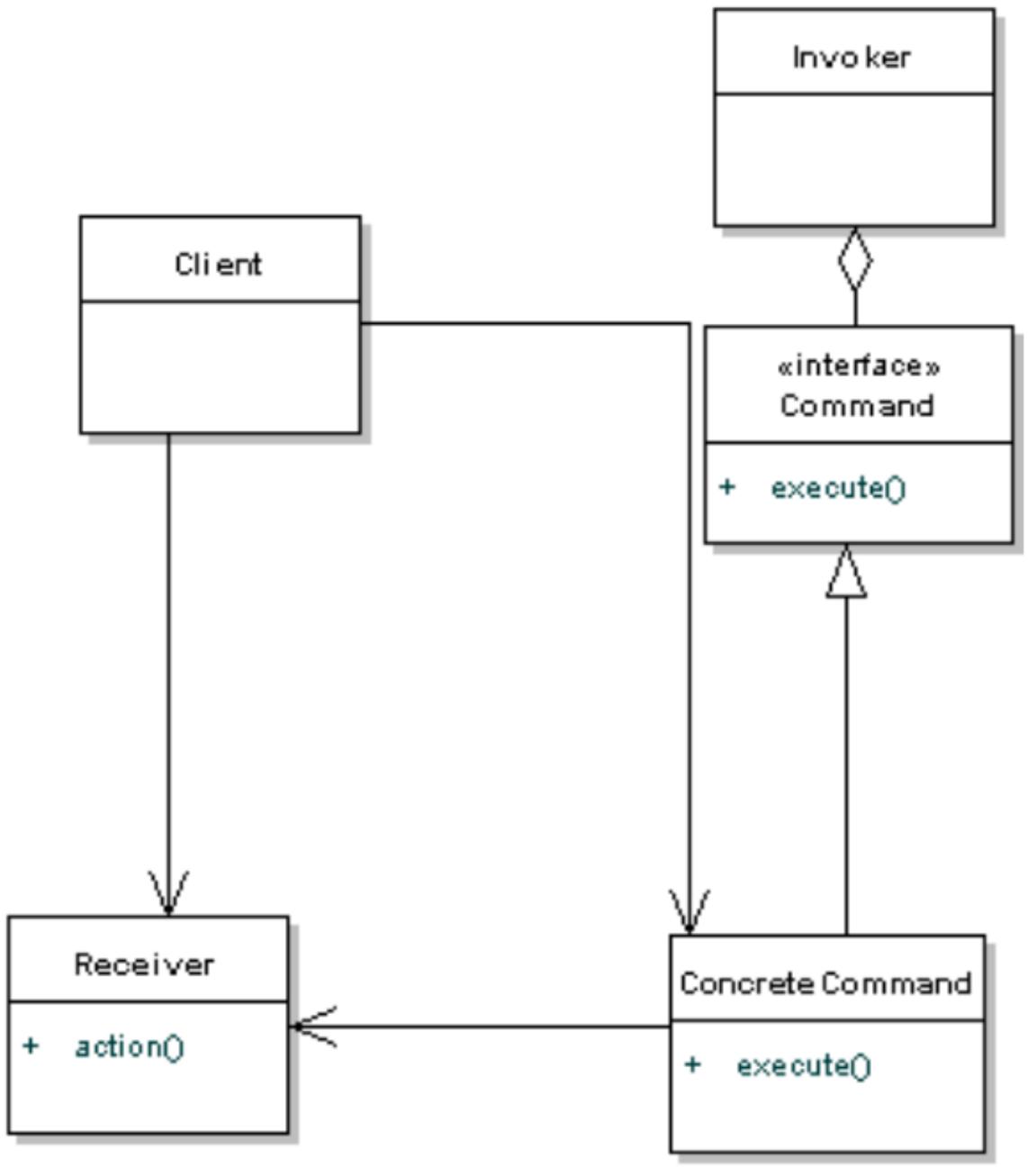
Sequence of Events



The **ConcreteCommand** defines a binding between the action and the **Receiver**.

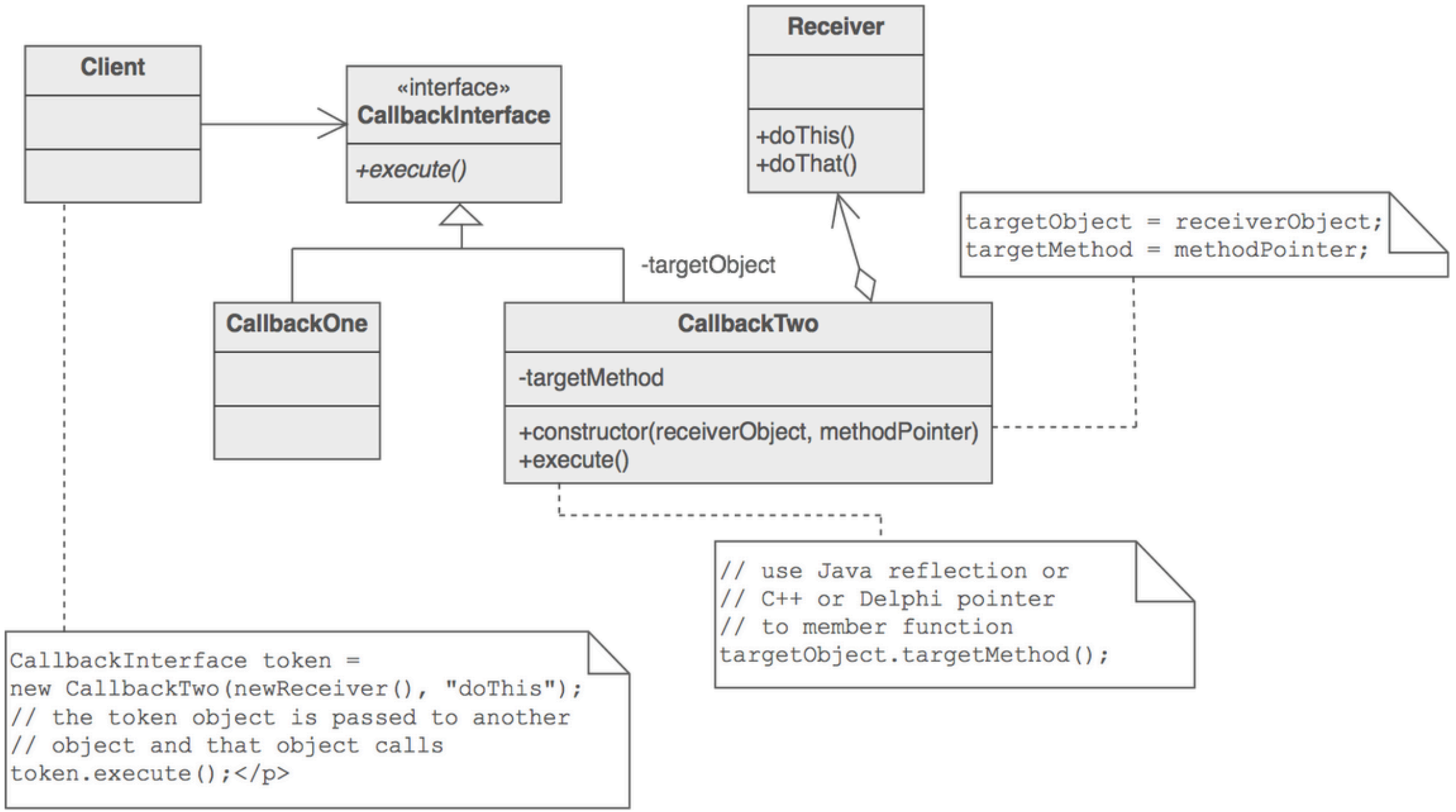
When the **Invoker** calls **execute( ),** the **ConcreteCommand** will run one or more actions on the **Receiver**.

Command (UML) Structure



https://dzone.com/articles/design-patterns-command

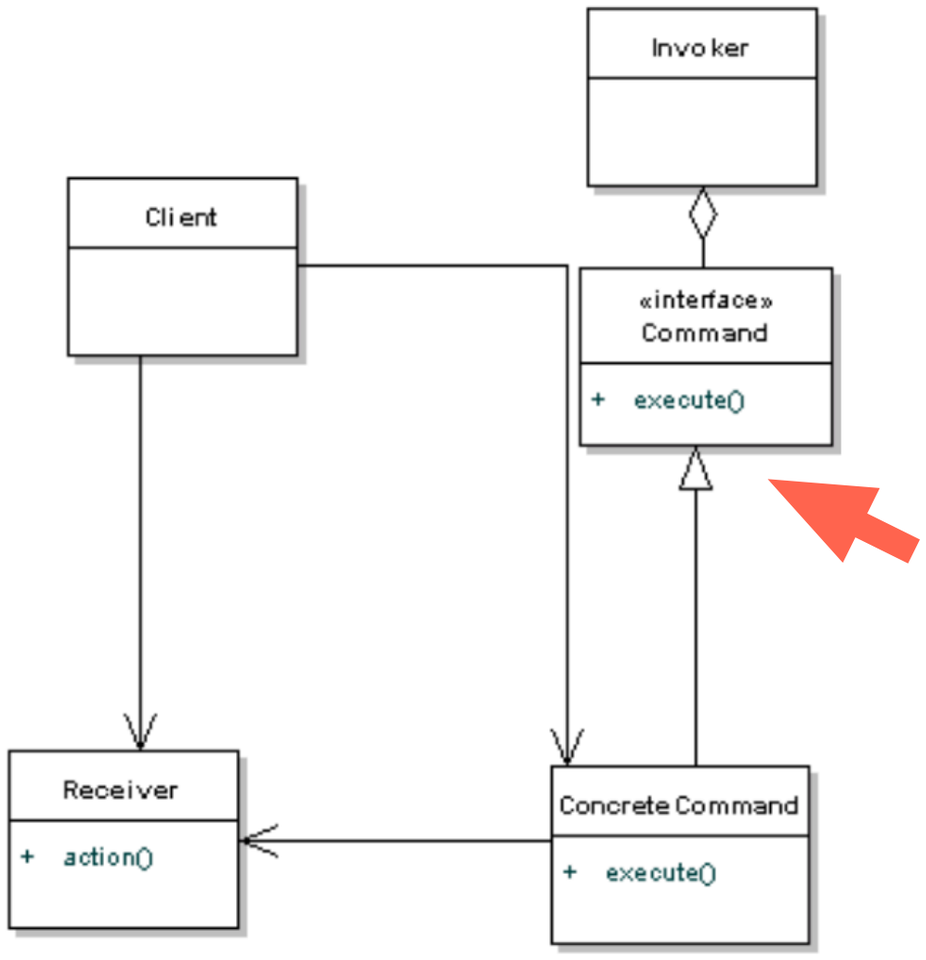
Command (UML) Structure



https://sourcemaking.com/design_patterns/command

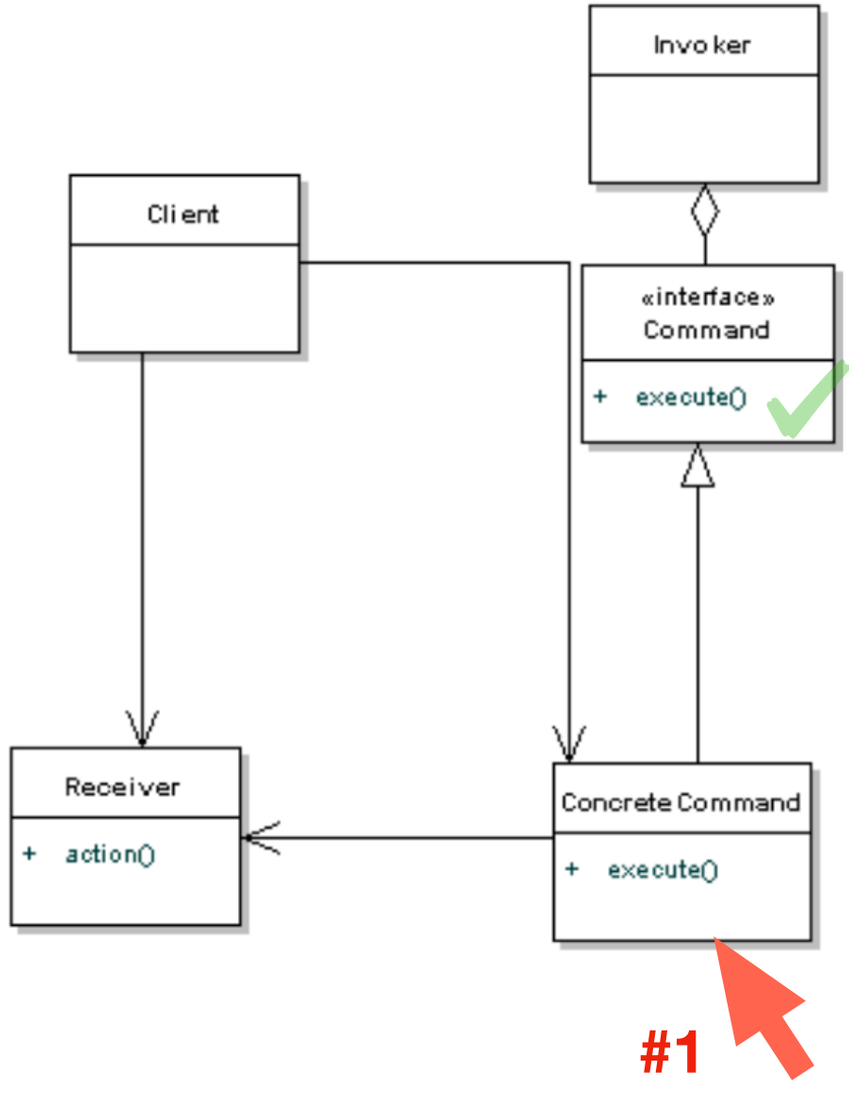
Code Example

```
1 //Command
2 public interface Command{
3   public void execute();
4 }
```



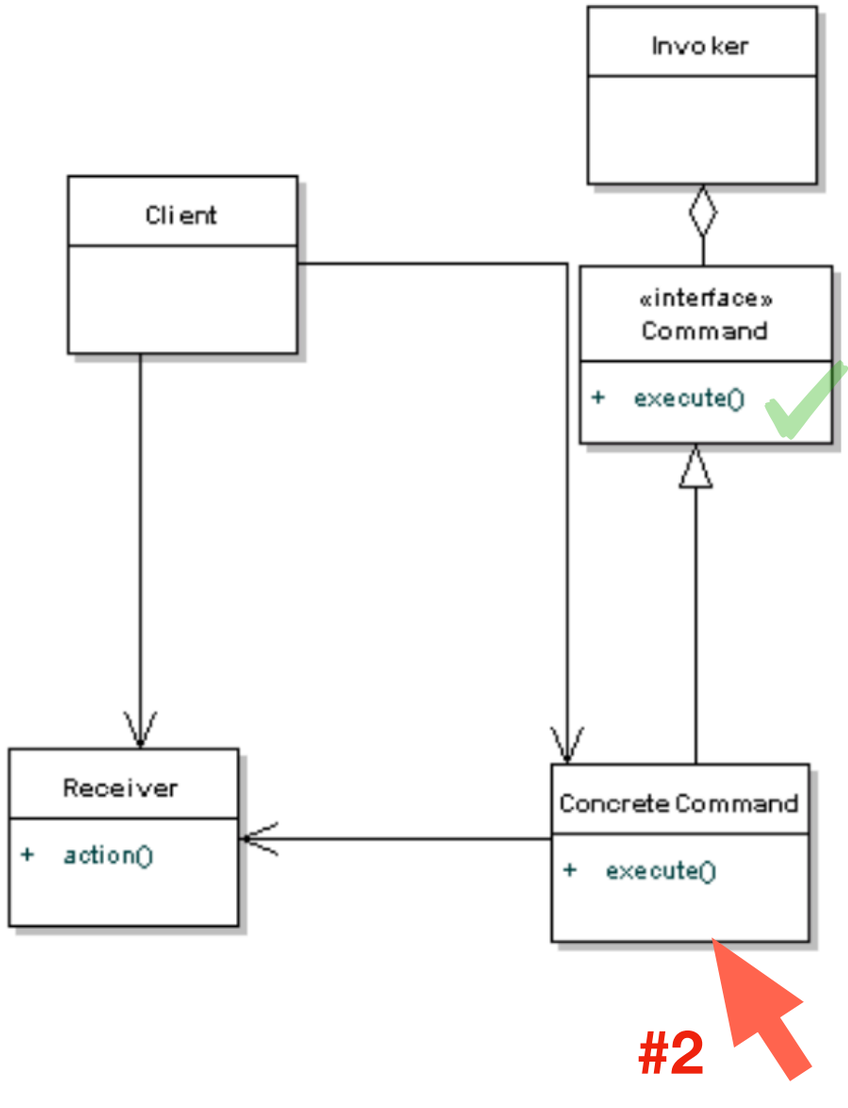
Code Example

```
1  //Concrete Command
2  public class LightOnCommand implements Command{
3    //reference to the light
4    Light light;
5    public LightOnCommand(Light light){
6      this.light = light;
7    }
8    public void execute(){
9      light.switchOn();
10   }
11 }
```



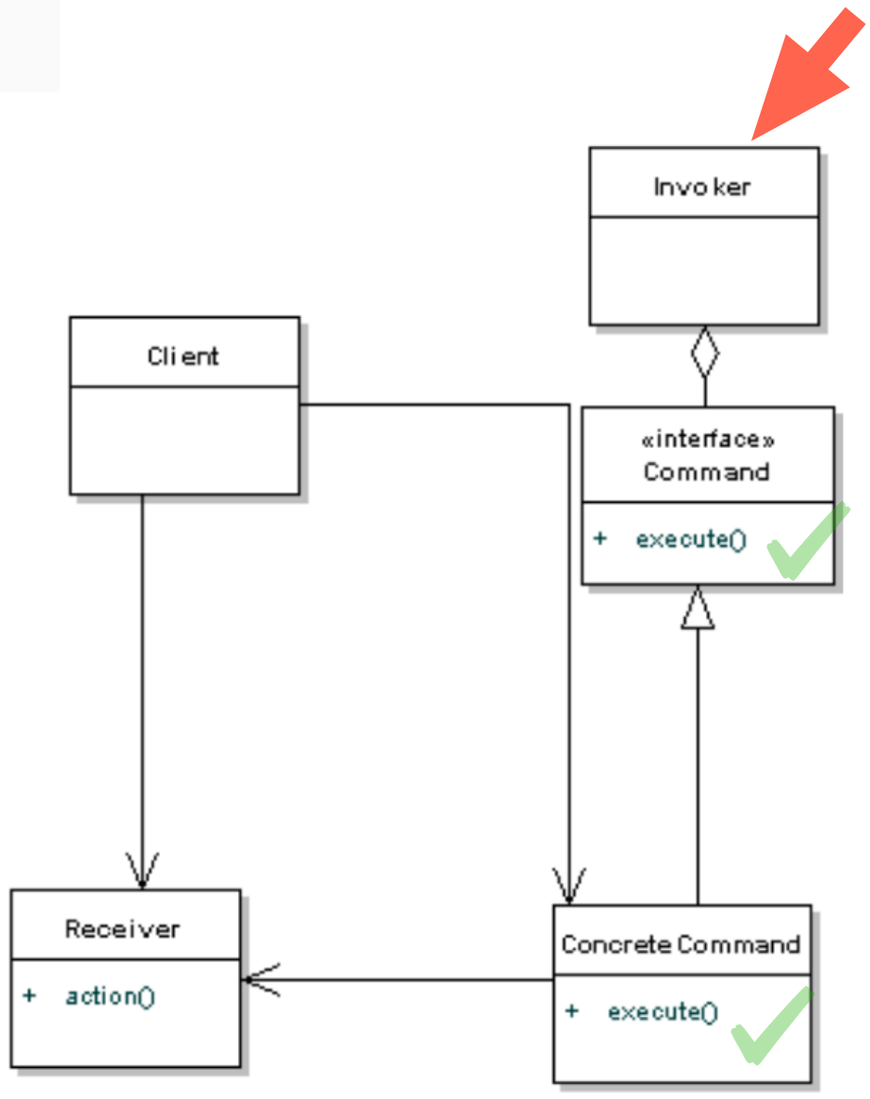
Code Example

```
1  //Concrete Command
2  public class LightOffCommand implements Command{
3    //reference to the light
4    Light light;
5    public LightOffCommand(Light light){
6      this.light = light;
7    }
8    public void execute(){
9      light.switchOff();
10   }
11 }
```



Code Example

```
1  //Invoker
2  public class RemoteControl{
3    private Command command;
4    public void setCommand(Command command){
5      this.command = command;
6    }
7    public void pressButton(){
8      command.execute();
9    }
10 }
```



Fill in the missing code fragments to complete the Command Design Pattern

Scenario:
A remote control can be used to turn on/off a light

```
//Command
public interface Command{
    
}
```

```
//Concrete Command
public class LightOnCommand {
  //reference to the light
  
  public LightOnCommand( ){
    
  }
  public void execute(){
    
  }
}
```

```
//Concrete Command
public class LightOffCommand {
  //reference to the light
  
  public LightOffCommand( ){
    
  }
  public void execute(){
    
  }
}
```

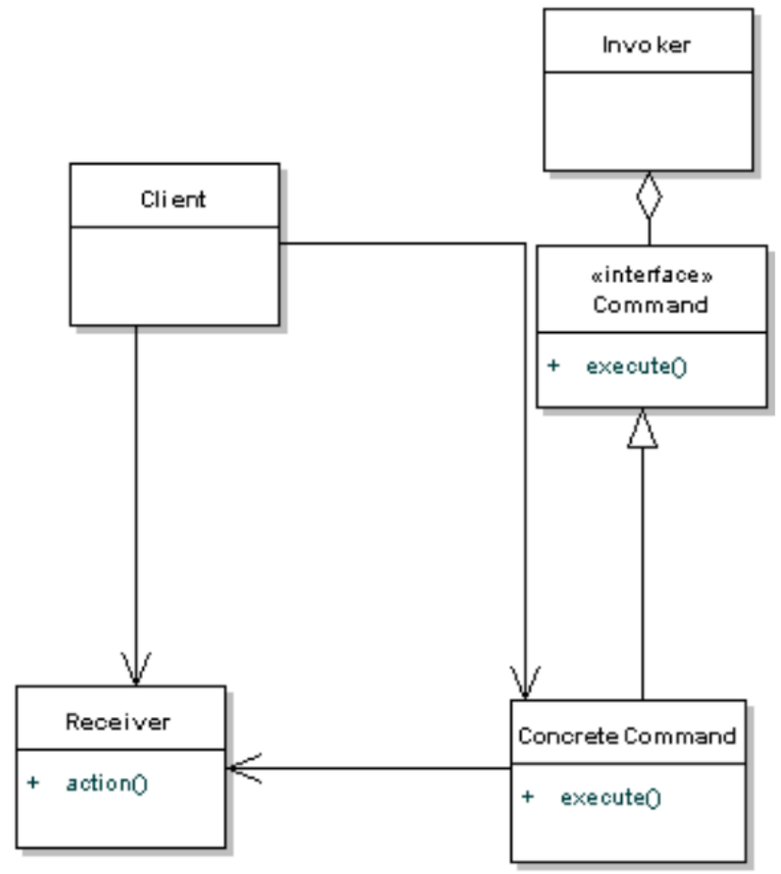
```
//Client
public class Client{
  public static void main(String[] args)    {
    RemoteControl control = ;
    Light light = ;
    Command lightsOn = ;
    Command lightsOff = ;

    //switch on
    control. ;
    control. ;

    //switch off
    control. ;
    control. ;
  }
}
```

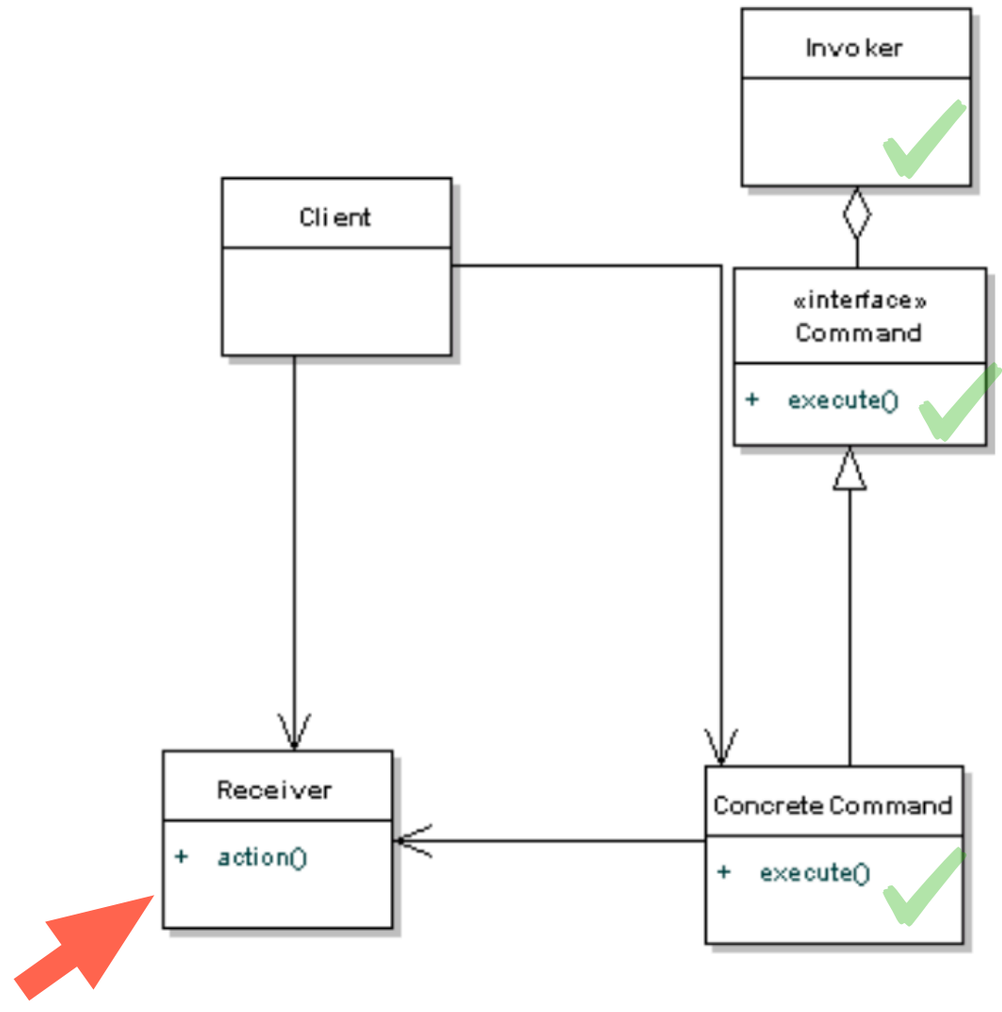
```
//Receiver
public class Light{
  
  public void switchOn(){
    
  }
  public void switchOff(){
    
  }
}
```

```
//Invoker
public class RemoteControl{
  ;
  public void setCommand( ){
    
  }
  public void pressButton(){
    
  }
}
```



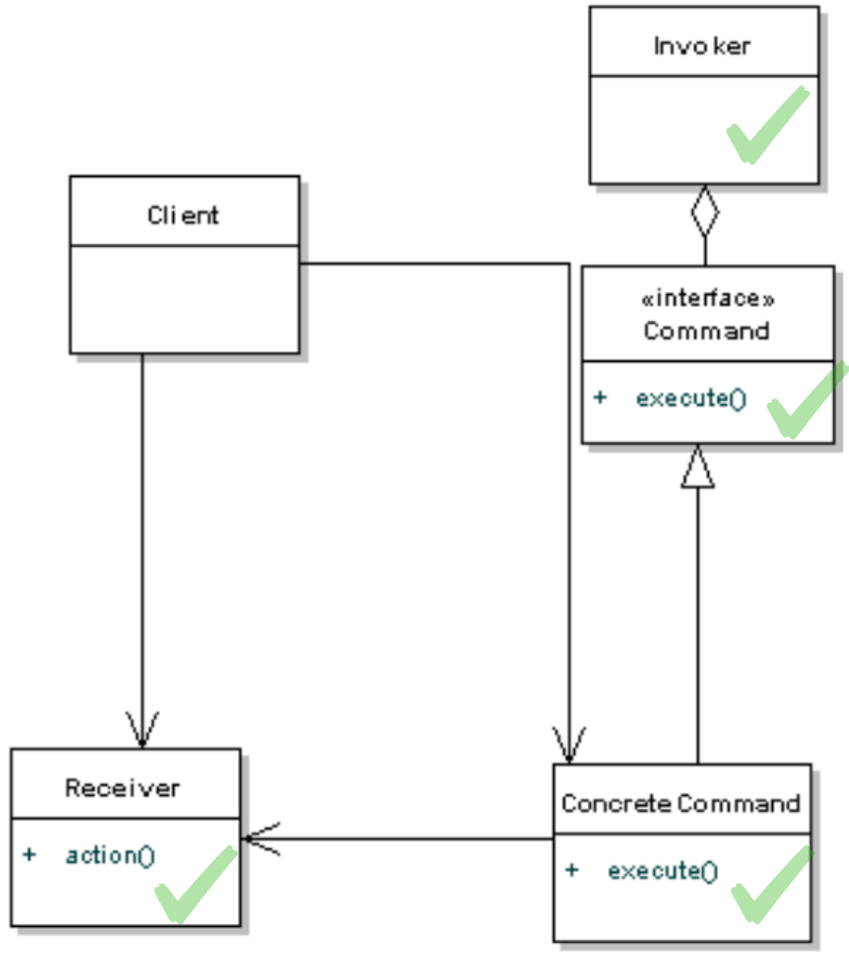
Code Example

```
1  //Receiver
2  public class Light{
3    private boolean on;
4    public void switchOn(){
5      on = true;
6    }
7    public void switchOff(){
8      on = false;
9    }
10 }
```



Code Example

```
1  //Client
2  public class Client{
3    public static void main(String[] args)    {
4      RemoteControl control = new RemoteControl();
5      Light light = new Light();
6      Command lightsOn = new LightsOnCommand(light);
7      Command lightsOff = new LightsOffCommand(light);
8
9      //switch on
10     control.setCommand(lightsOn);
11     control.pressButton();
12
13     //switch off
14     control.setCommand(lightsOff);
15     control.pressButton();
16   }
17 }
```



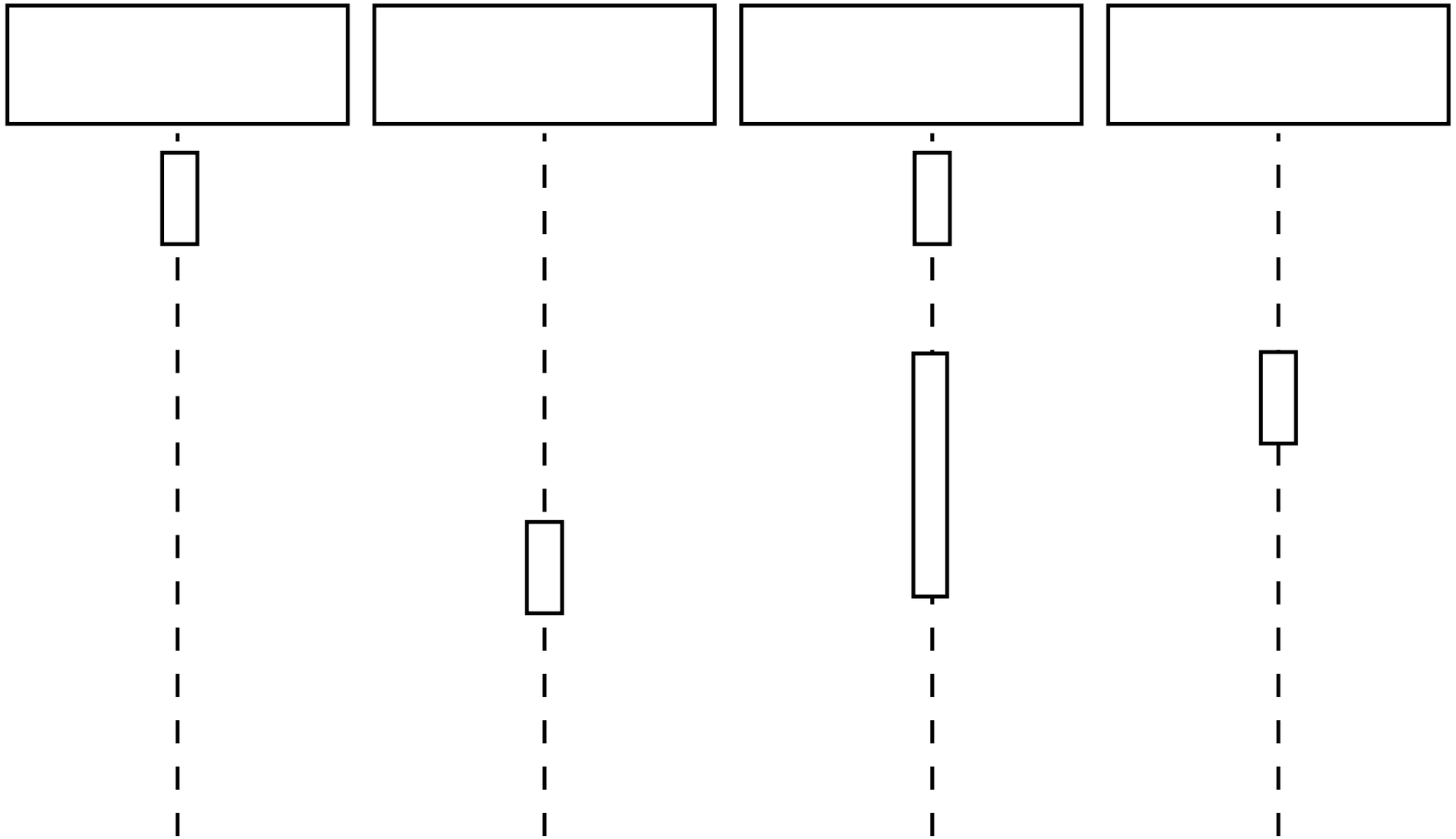
Command Pattern
-Steps-

1. Define a **Command** interface with a method signature like execute().
2. Create one or more derived classes that encapsulate some subset of the following: a "receiver" object, the method to invoke, the arguments to pass.
3. Instantiate a **Command** object for each deferred execution request.
4. Pass the Command object from the creator (aka Sender) to the **Invoker**.
5. The **Invoker** decides when to execute().

Exercise

10 mins

Complete the sequence diagram to illustrate the events that take place when a light is switched on for the code example given for the Command pattern.



Command Design Pattern
-Summary-

- Encapsulates a request as an object, thereby letting you parameterise clients with different requests, queue or log requests, and support undoable operations.
- Promotes "invocation of a method on an object" to full object status
- An object-oriented callback

Applicability:

The Command pattern is used when:

- A history of requests is needed
- You need callback functionality
- Requests need to be handled at variant times or in variant orders
- The invoker should be decoupled from the object handling the invocation.

Applicability
-Examples-

- The Command pattern is used often for multiple undo operations, where a stack of the recently executed commands are maintained. To implement the undo, all you need to do is get the last Command in the stack and execute it's undo( ) method.
- The Command pattern is useful for wizards, progress bars, GUI buttons and menu actions, and other transactional behaviour.

References

- Design Patterns: online reading resources and examples
  - Command:
    - https://www.oodesign.com/command-pattern.html
    - https://sourcemaking.com/design_patterns/command