

# Module 19: "Command"



**TEKNOLOGISK**  
**INSTITUT**

# Agenda

- ▶ Introductory Example: Toggling Lights
- ▶ Challenges
- ▶ Implementing the Command Pattern
- ▶ Pattern: Command
- ▶ Overview of Command Pattern
- ▶ .NET Framework Example: WPF Commands
- ▶ Command Extensions and Variations

# Introductory Example: Moving Parts

```
public class Light
{
    private bool _on;

    public Light() => _on = false;

    public void Toggle()
        => _on = !_on;
}
```

```
Light light = new Light();

string input = Console.ReadLine();
if( input.ToLower() == "toggle" )
{
    light.Toggle();
}
```

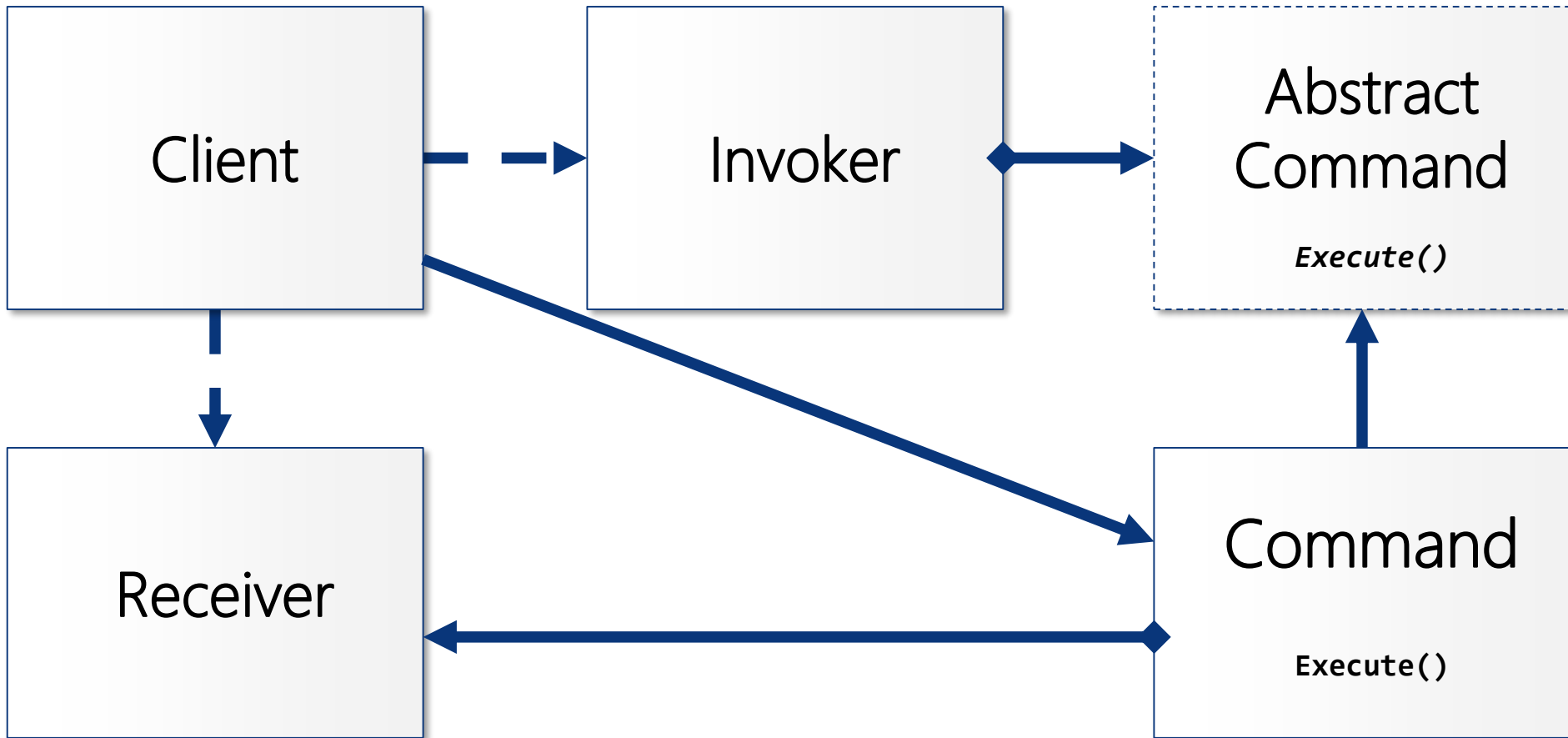
# Challenges

- ▶ How do we decouple the client from the actual light?
- ▶ What if we want to toggle more lights?
- ▶ What if we want to control other kinds of lights?
- ▶ What if we want to support timed controls?

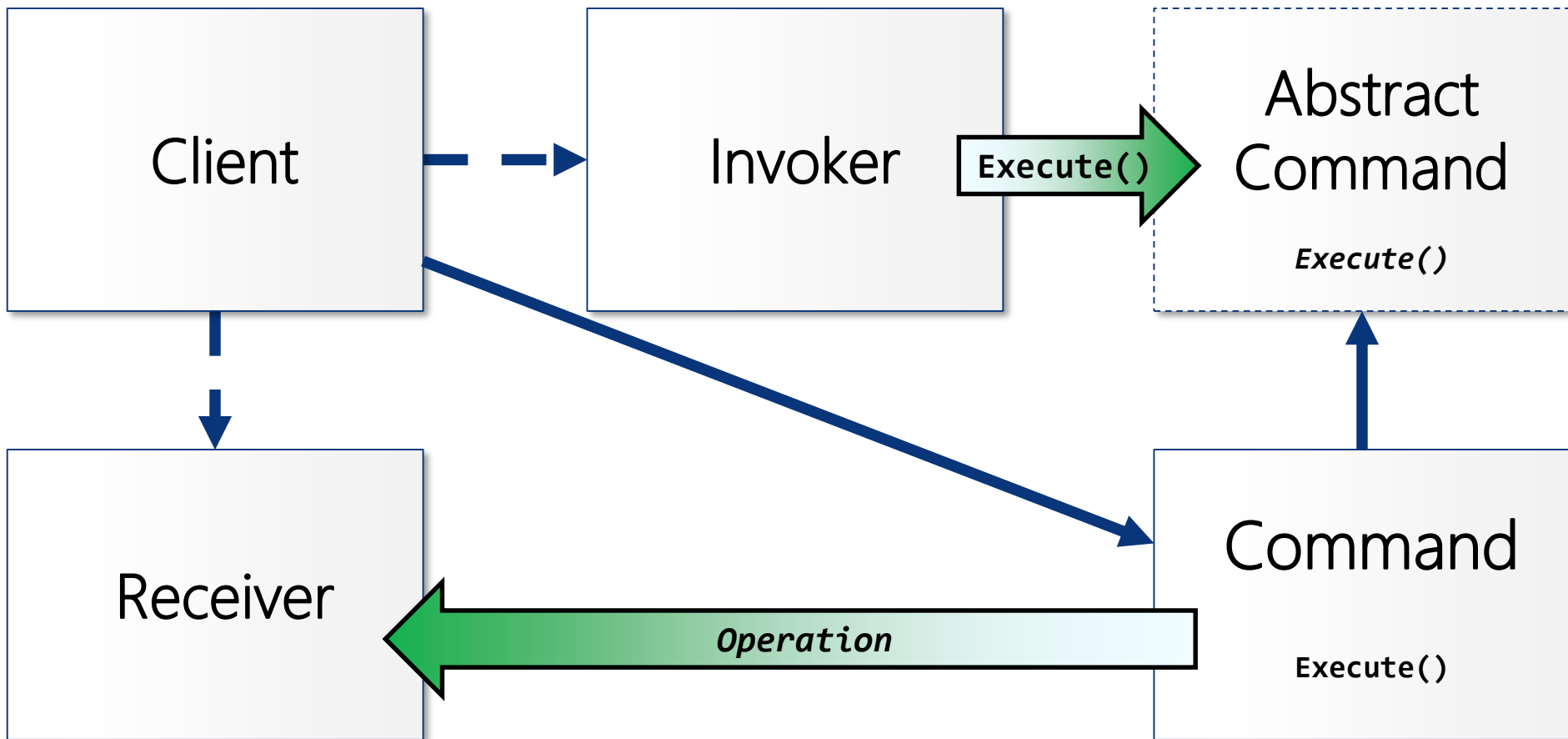
# Pattern: Command

- ▶ *Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.*
- ▶ Outline
  - A class delegates a request to command object instead of implementing directly
  - Decouples the invoker of a request from how it is executed.
- ▶ Origin: Gang of Four

# Overview of Command Pattern



# Overview of Command Pattern



# Overview of Command Pattern

- ▶ Client
  - Creates Command and Invoker and supplies Invoker with Command
- ▶ Abstract Command
  - Interface or abstract class defining general **Execute()** method
- ▶ Command
  - Concrete command class encapsulating action and parameter to **Execute()** at Receiver
- ▶ Invoker
  - Invokes Command without knowing contents of command object
- ▶ Receiver
  - Performs specified action when Invoker executes command object



# .NET Framework Example: WPF Commands

- ▶ Commands have already been built into Windows Presentation Foundation
  - See Module 18: "Memento" Example

```
<Button Command="{Binding AddGuestCommand}">Add</Button>  
<Button Command="{Binding UndoCommand}">Undo</Button>
```

```
public class RelayCommand : ICommand  
{  
    public RelayCommand( Action<object> execute,  
                        Predicate<object> canExecute )  
    { ... }  
}
```

# Command Extensions and Variations

- ▶ Invocations often use parameters
  - These are stored within the Command object
- ▶ Commands are often constructed by factories
- ▶ Possible extensions to the  **ICommand**  interface
  - `Validate()`
  - `Execute()`
  - `Undo()`



WINCUBATE

Jesper Gulmann Henriksen

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : [jgh@wincubate.net](mailto:jgh@wincubate.net)

WWW : <http://www.wincubate.net>

Ringgårdsvej 4A

8270 Højbjerg

Denmark