COMMAND DESSIGN PATTERN

WELCOME TO OUR GROUP

Hao Trinh The

10421017

Huy Vo Vuong Bao

10421021

TABLE OF CONTENT

The Remote Problem

How do we solve it with Command Pattern?

I want an API that operates like a **REMOTE** but it **CONTROLS** different household **DEVICES** that are assigned to an app. I'll email you the prototype of the app and a set of **C++ Classes** that control home automation devices. I'll look forward to seeing your design.

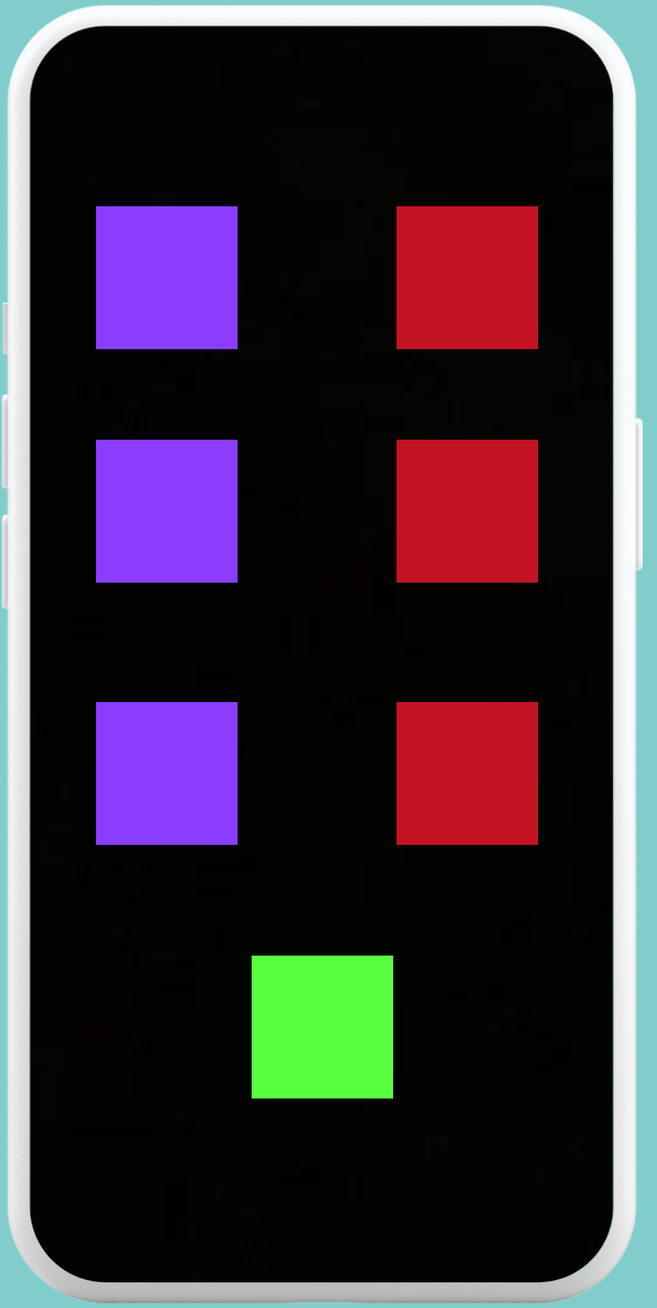
SPECIFICATIONS

- 3 SLOTS for 3 DEVICES
- Each slot contain 2 BUTTONS:"ON" and "OFF"
- An UNDO button

01

02

03



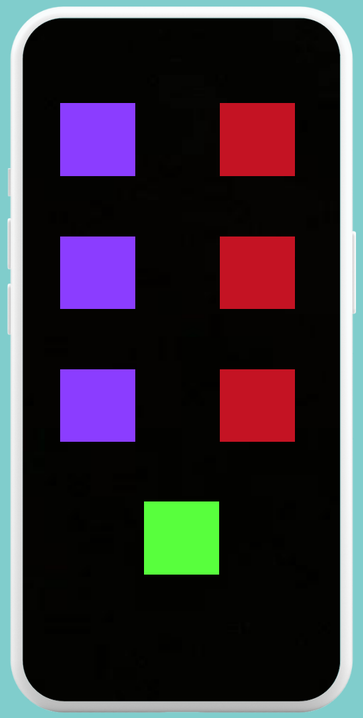
SPECIFICATIONS

- 3 SLOTS for 3 DEVICES
- Each slot contain 2 BUTTONS:"ON" and "OFF"
- An UNDO button at the end

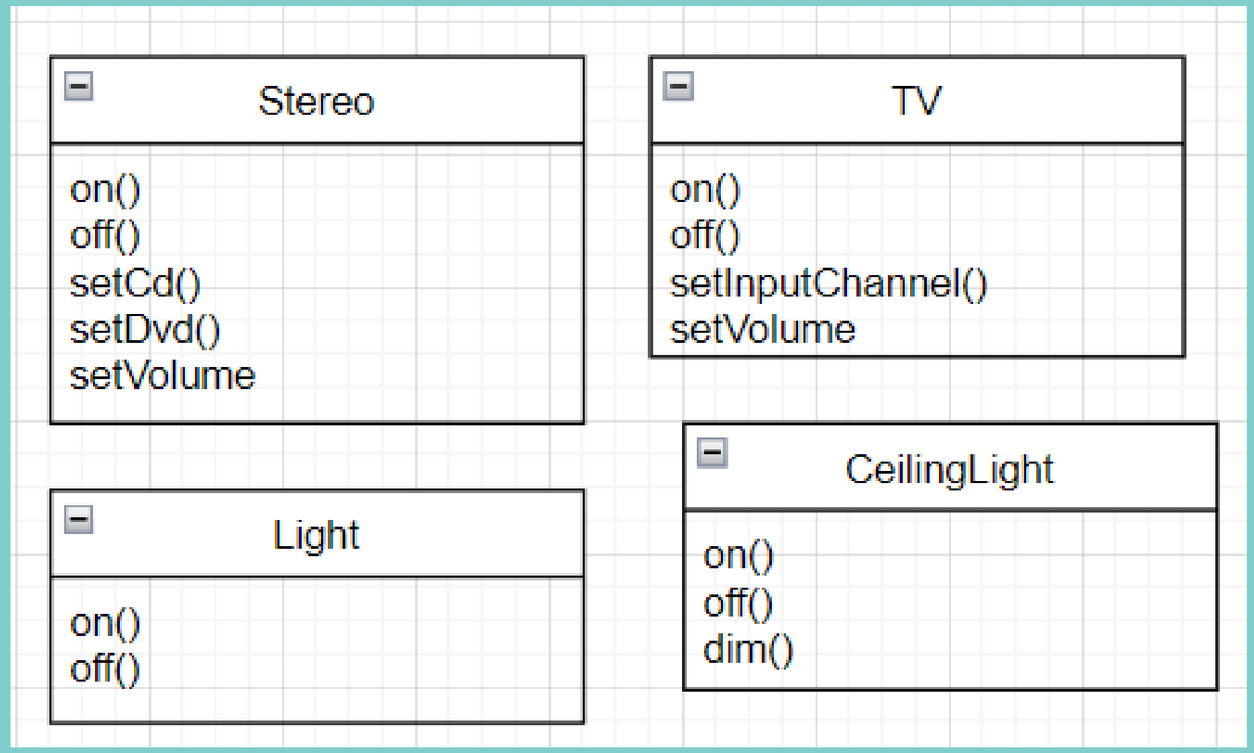
01

02

03



C++ CLASSES:



IMPLEMENTATION:

remote

Devices slot[3]

void onButtonPressed(int slot)
void offButtonPressed(int slot)
void setDevices(Devices &obj, int slot)

Devices

action1()
action2()
action3()

CODE:

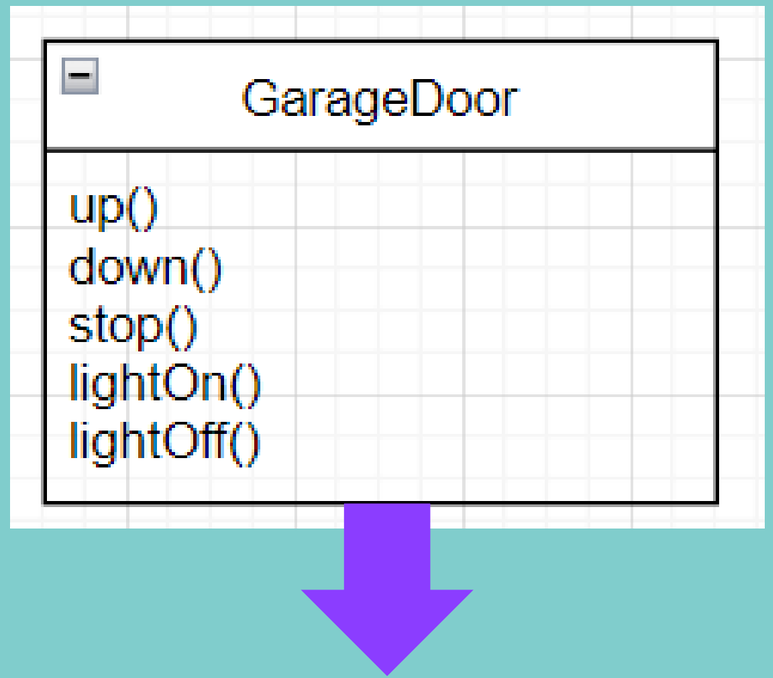
```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Devices{
public:
    virtual void on() = 0;
    virtual void off() = 0;
};


class Remote{
private:
    Devices slot[3];
public:
    void onButtonPressed(int number){
        this->slot[number].on();
    }

    void offButtonPressed(int number){
        this->slot[number].off();
    }

    void setDevices(Devices &obj, int number){
        this->slot[number] = obj;
    }
};
```

BUGS

Generalization

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Devices{
public:
    virtual void on() = 0;
    virtual void off() = 0;
};


class Remote{
private:
    Devices slot[3];
public:
    void onButtonPressed(int number){
        this->slot[number].on();
    }

    void offButtonPressed(int number){
        this->slot[number].off();
    }

    void setDevices(Devices &obj, int number){
        this->slot[number] = obj;
    }
};
```

```cpp
int main(){
    Remote myphone;
    GarageDoor garagedoor1;

    myphone setDevices(garagedoor1, 0);
    myphone onButtonPressed(0);
}
```



No function called on() in class GarageDoor

BUGS

Encapsulation

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Devices{
public:
    virtual void on() = 0;
    virtual void off() = 0;
};


class Remote{
private:
    Devices slot[3];
public:
    void onButtonPressed(int number){
        this->slot[number].on();
    }

    void offButtonPressed(int number){
        this->slot[number].off();
    }

    void setDevices(Devices &obj, int number){
        this->slot[number] = obj;
    }
};
```

POSSIBLE SOLUTION???

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Devices{
public:
    virtual void on() = 0;
    virtual void off() = 0;
};


class Remote{
private:
    Devices slot[3];
public:
    void onButtonPressed(int number){
        this->slot[number].on();
    }

    void offButtonPressed(int number){
        this->slot[number].off();
    }

    void setDevices(Devices &obj, int number){
        this->slot[number] = obj;
    }
};
```

Adding coditional statements:

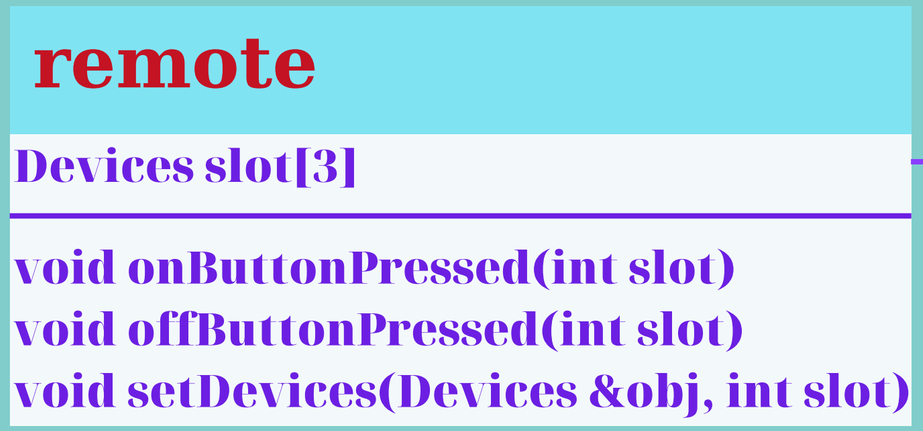
Pseudocode:

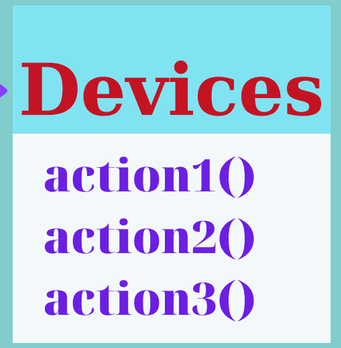
```
If(slot[number] == "Light"):
        slot[number].on();
else if(slot[number] == "Door"):
    slot[number].open(); .....
```

The Method

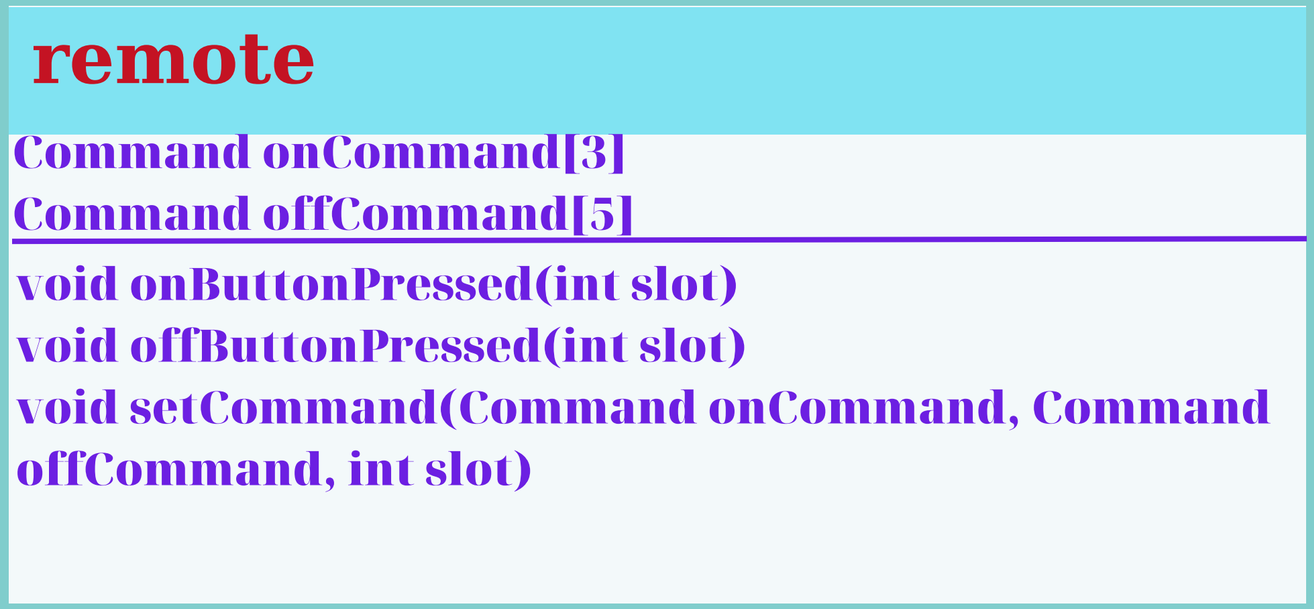
How?

OLD VERSION:





NEW VERSION:



CODE VISUALISAZTION

```
class Command{
public:
    virtual void execute() = 0;
};
class Remote{
private:
    vector<Command*>onCommand;
    vector<Command*>offCommand;
public:
    Remote(){
        this->onCommand.reserve(3);
        this->offCommand.reserve(3);
    }

    void onButtonPressed(int number){
        if(number<=2 && number>=0){
            this->onCommand[number]->execute();
        }
    }

    void offButtonPressed(int number){
        if(number<=2 && number>=0){
            this->offCommand[number]->execute();
        }
    }

    void setCommand(Command *onCommand, Command *offCommand, int number){
        if(number <= 2 && number >= 0){
            this->onCommand[number] = onCommand;
            this->offCommand[number] = offCommand;
        }
    }
};
```

NEW VERSION:

remote

Command onCommand[3]
Command offCommand[5]

void onButtonPressed(int slot)
void offButtonPressed(int slot)
void setCommand(Command onCommand, Command offCommand, int slot)

PARENT CLASS:

Command

virtual void excecute()

LightOnCommand

Light object

void excecute()

LightOffCommand

Light object

void excecute()

OpenDoorCommand

Door object

void excecute()

CODE

```
class LightOnCommand: public Command{
private:
    Light object;
public:
    LightOnCommand(const Light& object){
        this->object = object;
    }

    void excecut(){
        this->object.on();
    }
};
```

```
class LightOffCommand: public Command{
private:
    Light object;
public:
    LightOffCommand(const Light& object){
        this->object = object;
    }

    void excecut(){
        this->object.off();
    }
};
```

```
class OpenDoorCommand: public Command{
private:
    Door object;
public:
    OpenDoorCommand(const Door& object){
        this->object = object;
    }

    void excecut(){
        this->object.open();
    }
};
```

The Command Design Pattern

Is this the pattern?

DEFINTION:

a BEHAVIORAL pattern

Turns a REQUEST into a STAND-ALONE OBJECT that CONTAINS all INFORMATIN about the REQUEST

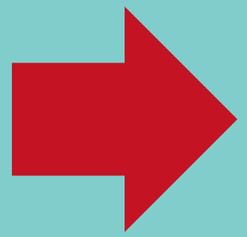
Encapsulates a request as an object

OLD VERSION:

remote

Devices slot[3]

void onButtonPressed(int slot)
void offButtonPressed(int slot)
void setDevices(Devices &obj, int slot)



Contains all information:
Devices, actions needed
to compete the requests

NEW VERSION:

remote

Command onCommand[3]
Command offCommand[5]

void onButtonPressed(int slot)
void offButtonPressed(int slot)
void setCommand(Command onCommand, Command offCommand, int slot)

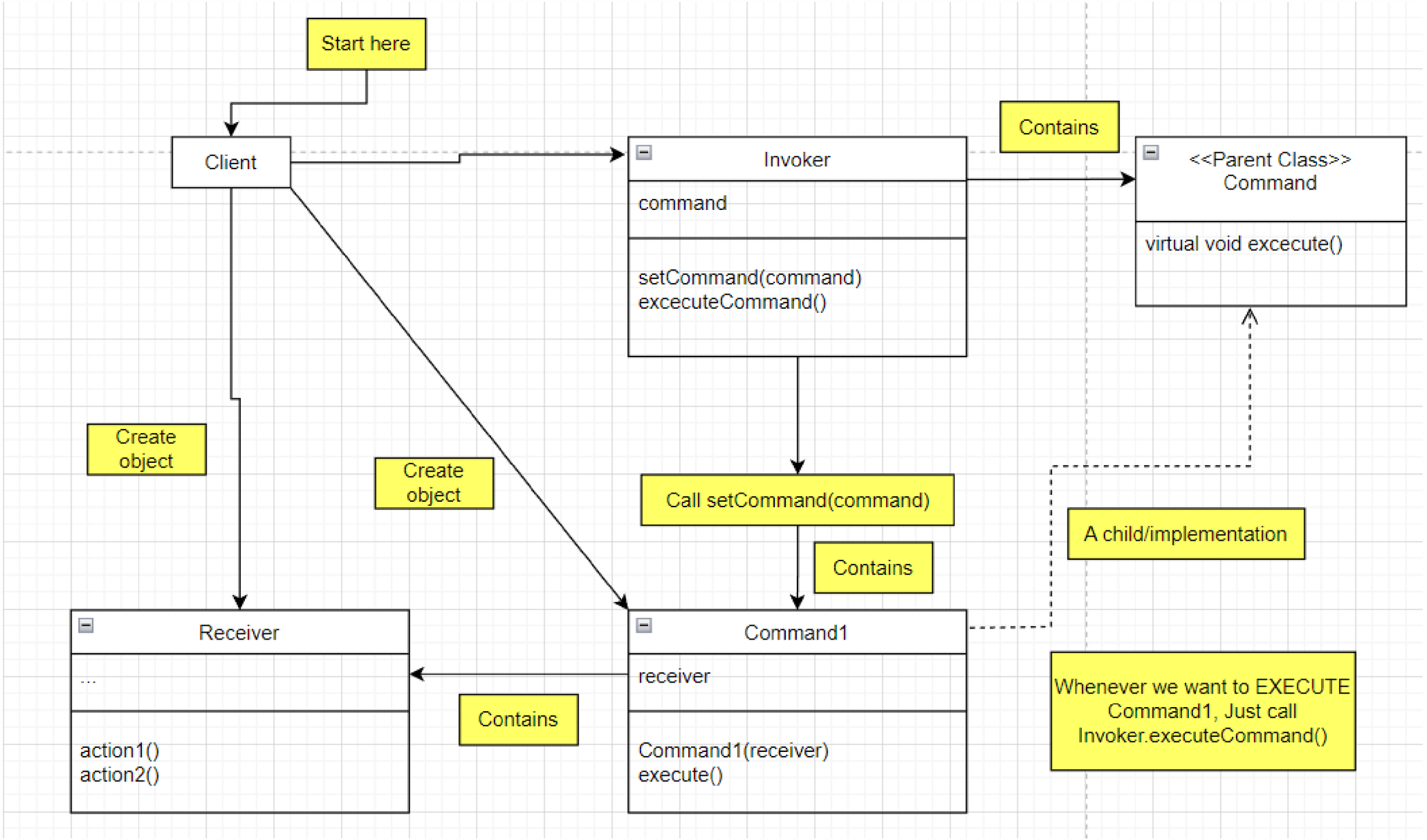
Contains requests

Command

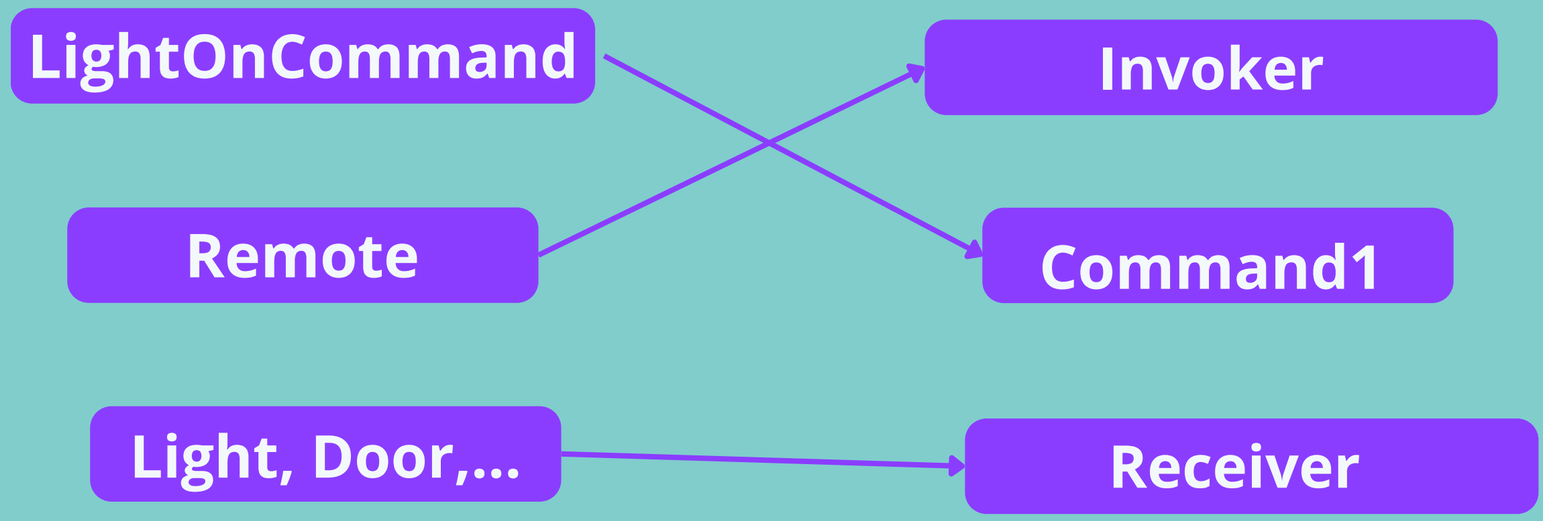
virtual void
excecute()

Contains all information:
Devices, actions needed to compete
the requests

COMMAND PATTERN



CONNECTION



Receiver such as Light, Door is the one that actually PERFORM the ACTION needed to complete the Command

Code in action

Time to implement some codes

CODE:

```
class Light{
    public:
        void on(){
            cout << "Light is on" << endl;
        }
        void off(){
            cout << "Light is off" << endl;
        }
};

class TV{
    public:
        void on(){
            cout << "TV is on" << endl;
        }
        void off(){
            cout << "TV is off" << endl;
        }
        void setInputChanel(int i){
            cout<<"Turn to chanel "<<i<<endl;
        }
        void setVolume(int i){
            cout<<"Turn volume to "<<i<<endl;
        }
};

class Command{
    public:
        virtual void execute() = 0;
};

class LightOnCommand: public Command{
private:
    Light object;
public:
    LightOnCommand(const Light &object){
        this->object = object;
    }
    void execute(){
        this->object.on();
    }
```

```
class LightOffCommand: public Command{
private:
    Light object;
public:
    LightOffCommand(const Light &object){
        this->object = object;
    }
    void execute(){
        this->object.off();
    }
};

class TVonCommand: public Command{
private:
    TV object;
public:
    TVonCommand(const TV &object){
        this->object = object;
    }
    void execute(){
        this->object.on();
    }
};

class VolumeTVHighCommand: public Command{
private:
    TV object;
public:
    VolumeTVHighCommand(const TV &object){
        this->object = object;
    }
    void execute(){
        this->object.setVolume(50);
    }
};
```

CODE:

```
class Remote{
private:
    vector<Command*>onCommand;
    vector<Command*>offCommand;
public:
    Remote(){
        this->onCommand.reserve(3);
        this->offCommand.reserve(3);
    }

    void onButtonPressed(int number){
        if(number<=2 && number>=0){
            this->onCommand[number]->execute();
        }
    }

    void offButtonPressed(int number){
        if(number<=2 && number>=0){
            this->offCommand[number]->execute();
        }
    }

    void setCommand(Command *onCommand, Command *offCommand, int number){
        if(number <= 2 && number >= 0){
            this->onCommand[number] = onCommand;
            this->offCommand[number] = offCommand;
        }
    }
};
```

```
int main(){
    Remote myPhone;

    Light bedroomLight;
    TV livingroomTV;

    LightOnCommand bedroomLightOn(bedroomLight);
    LightOffCommand bedroomLightOff(bedroomLight);

    TVonCommand livingroomTVOn(livingroomTV);
    VolumeTVHighCommand livingroomTVHigh(livingroomTV);

    myPhone.setCommand(&bedroomLightOn, &bedroomLightOff, 0);
    myPhone.setCommand(&livingroomTVOn, &livingroomTVHigh, 1);

    myPhone.onButtonPressed(1);
    myPhone.offButtonPressed(1);

    myPhone.onButtonPressed(0);
    myPhone.offButtonPressed(0);

    return 0;
}
```

```
TV is on
Turn volume to 50
Light is on
Light is off
```

SOLUTION IF BUTTON SLOT 3 IS PRESSED

+ Create a new Class named NothingCommand

```
class NothingCommand: public Command{
public:
    void execute(){}
};
```

+ We add these code in Remote class

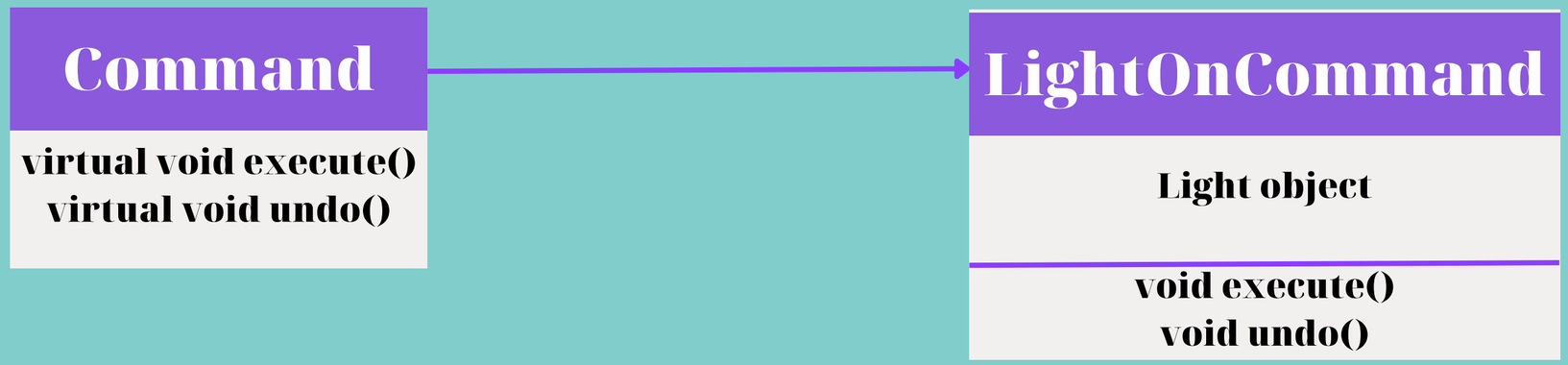
- A little adjust in the Constructor

```
Remote(){
    this->onCommand.reserve(3);
    this->offCommand.reserve(3);

    for(int i= 0; i<3; i++){
        onCommand[i] = new NothingCommand;
        offCommand[i] = new NothingCommand;
    }
}
```

- Implement a destructor

```
~Remote(){
    for(int i= 0; i<3; i++){
        delete onCommand[i];
        delete offCommand[i];
    }
}
```

HOW ABOUT THE UNDO BUTTON?




```
class Command{
    public:
        virtual void execute() = 0;
        virtual void undo() = 0;
};
```

```
class LightOnCommand: public Command{
private:
    Light object;
public:
    LightOnCommand(const Light &object){
        this->object = object;
    }
    void execute(){
        this->object.on();
    }
    void undo(){
        this->object.off();
    }
};
```

REMARK!!!

Advantages:

- Flexibility
- Easy for Update and mantenance
- Widely used and practical

Disadvantages:

- Not function or algorithms
- Higher abstraction




THANK YOU FOR LISTENING