



The Command Pattern

**Human Computer Interaction Research
University of Nevada, Reno**



Command

★ Behavioral Patterns

- » strategy
- » observer
- » decorator
- » **command**

★ Creational Patterns

- » factory method
- » abstract factory
- » singleton

Problem

“your program has many different actions it can perform, implementing these would lead to huge if-elseif or switch blocks”.

Dog Training



Roll



Bark



Sit

Naive solution

```
public static void main(String[] args) {  
    Dog charles = new Dog("Snoop");  
    if (args[0].equals("bark")) {  
        charles.bark(); }  
    else if (args[0].equals("bite")) {  
        charles.bite(); }  
    else if (args[0].equals("roll")) {  
        charles.rollover();  
    }  
}
```

should this even be in
main?

Works but gets
messy when you
add lots of
commands (bite,
attack, eat
homework, barf...)

Command Pattern

The Command Pattern: encapsulates a request as an object, thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable operations

Command Pattern

- ★ Move the code for each individual action into its **own** separate class.
- ★ Each of these classes implements the same **Interface**, allowing the code that uses them to interact solely with the Interface and not know or care about the individual classes.
- ★ This increases **Cohesion** because each class is responsible for one discrete set of logic.
- ★ This decreases **Coupling** because the code calling the command only deals with one type, the Interface.

Command Interface

```
public interface DogCommand {  
    public void execute();  
}
```

★ You can put any “generic” method in here:

★ log()

★ undo()

★ delete()

★ load()

Specific Commands

```
public class BarkCommand implements DogCommand {  
    public void execute() {  
        System.out.println("Bow wow!!");  
    }  
}
```

```
public class BiteCommand implements DogCommand {  
    public void execute() {  
        System.out.println("Munch..Munch");  
    }  
}
```

Dog

```
public class Dog {  
    String name;  
    DogCommand commands[] = new DogCommand[2];  
  
    public Dog(String name) {  
        this.name=name;  
        commands[0] = new SitCommand();  
        commands[1] = new BarkCommand();  
    }  
}
```

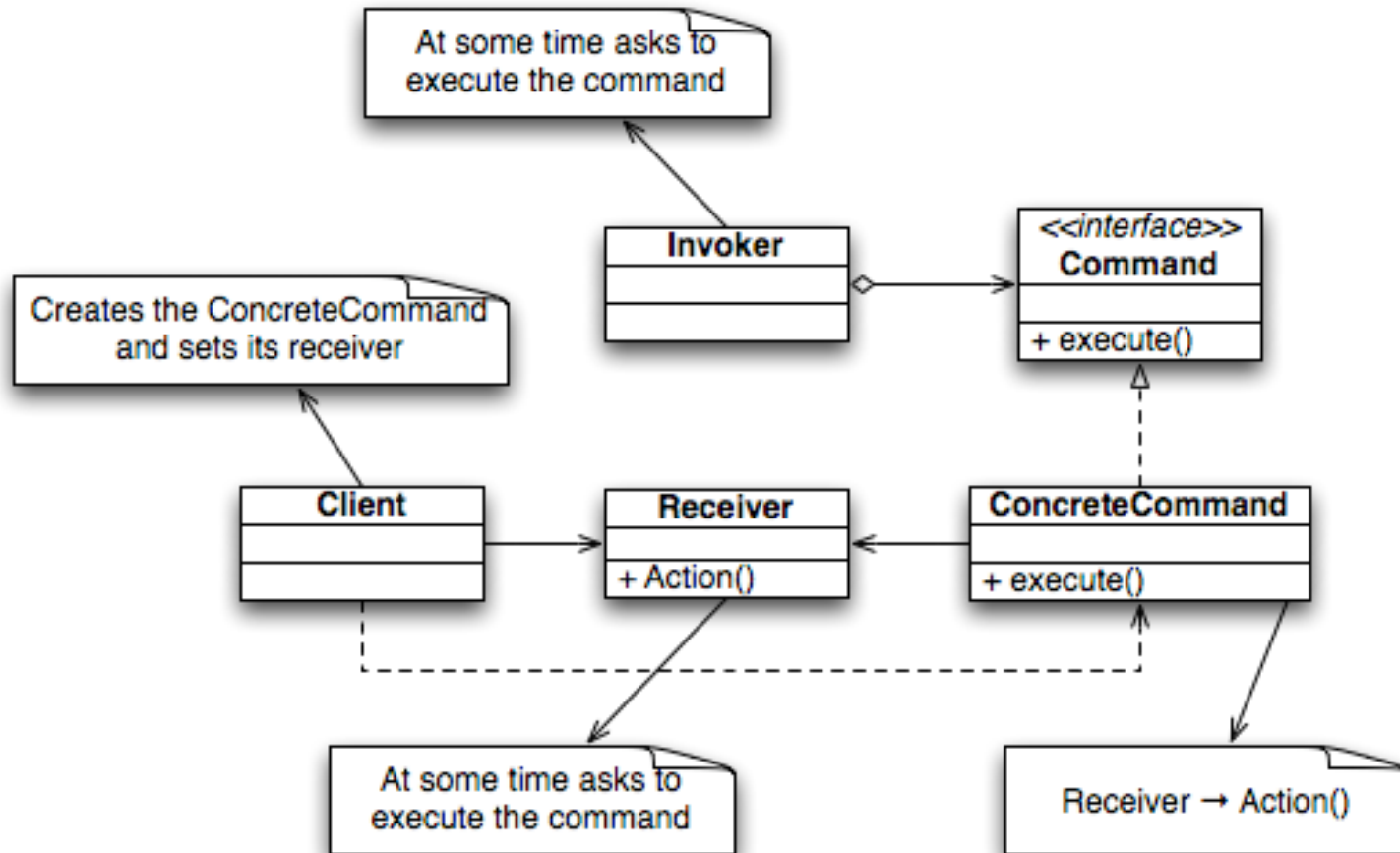
Main class

```
public class TestDogCommand {
    public static void main(String[] args) {
        DogCommand cmd;
        try {
            cmd = (DogCommand) Class.forName(args[0]).newInstance();
            cmd.execute();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

OK this is ugly, but it scales up a lot better

```
Eelke-Folmers-MacBook-Pro:bin eelke$ java TestDogCommand BarkCommand
Woof!!
```

Class Diagram



Exercise I



Chess

**Human Computer Interaction Research
University of Nevada, Reno**

Command interface

```
public interface Command {  
    public void execute();  
    public void undo();  
    public char toChar(int i);  
}
```

Jump Command

```
public class JumpCommand implements Command {  
  
    private ChessPiece cp;  
  
    public JumpCommand(ChessPiece cp) {  
        this.cp=cp;  
    }  
  
    public void execute() {  
        int x=cp.getX();  
        int y=cp.getY();  
        char c = toChar(x);  
        cp.setPos(x+1,y+2);  
        System.out.println(cp.toString()+" jumps from "+c+y+" to  
"+toChar(cp.getX()) +cp.getY());  
    }  
}
```

Move Command

```
public class MoveCommand implements Command {  
  
    private ChessPiece cp;  
  
    public JumpCommand(ChessPiece cp) {  
        this.cp=cp;  
    }  
  
    public void execute() {  
        int x=cp.getX();  
        int y=cp.getY();  
        char c = toChar(x);  
        cp.setPos(x,y+1);  
        System.out.println(cp.toString()+" moves from "+c+y+" to  
"+toChar(cp.getX())    +cp.getY());  
    }  
}
```

Chess piece

```
public abstract class ChessPiece {
    public int Xpos;
    public int Ypos;
    String name;
    Command move;

    int getX(){ return Xpos;}

    int getY() { return Ypos; }

    void setPos(int x, int y){
        Xpos=x;
        Ypos=y;
    }
    public String toString() {
        return name;
    }
}
```

Knight & Pawn

```
public class Pawn extends ChessPiece {  
    public Pawn(int x, int y){  
        this.setPos(x, y);  
        this.move = new MoveCommand(this);  
        this.name="Pawn";  
    }  
}
```

```
public class Knight extends ChessPiece {  
    public Knight(int x, int y){  
        this.setPos(x, y);  
        this.move = new JumpCommand(this);  
        this.name="Knight";  
    }  
}
```

Chess Board

```
public class ChessBoard {
    public int XMAX=8;
    public int YMAX=8;

    public ChessPiece white[] = new ChessPiece[2];
    public ChessPiece black[] = new ChessPiece[2];

    public ChessBoard() {
        white[0]=new Knight(0,0);
        white[1]=new Pawn(1,0);
        black[0]=new Knight(7,7);
        black[1]=new Pawn(6,7);
    }
}
```

player

```
public class Player {
    private ChessBoard cb;
    private int id;
    private int lastplayer;
    private int lastpiece;

    public Player(ChessBoard cb, int id){
        this.cb=cb;
        this.id=id;
    }

    public void move(int i) {
        if (id==0) System.out.print("White:");
        else System.out.print("Black:");
        if (i== -1) {
            if (id==0) cb.white[lastpiece].move.undo();
            else cb.black[lastpiece].move.undo();
        }
        else if (id==0) {
            cb.white[i].move.execute();
        }
        else {
            cb.black[i].move.execute();
        }
        lastplayer=id;
        lastpiece=i;
    }
}
```

Game

```
public class Game {  
  
    public Game() {  
        ChessBoard gb = new ChessBoard();  
        Player one = new Player(gb,0);  
        Player two = new Player(gb,1);  
        one.move(0, 0);  
        two.move(1,1);  
        one.move(1,0);  
        one.move(-1,0);  
        one.move(2,0);  
    }  
}
```