Command Design Pattern (Java) - Cheat Sheet

What is the Command Design Pattern?

A behavioral pattern that encapsulates a request as an object, allowing you to parameterize clients with queues, requests, or logs, and support undoable operations. It decouples the object that invokes the operation (Invoker) from the one that knows how to perform it (Receiver).

Why & When to Use

- Decouples sender and receiver
- Supports undo/redo functionality
- Enables queuing or logging requests
- Helps implement macro commands (batching actions)

Key Participants

1. Command interface: declares execute()
2. ConcreteCommand: implements Command, calls Receiver
3. Receiver: knows how to perform the action
4. Invoker: asks the command to execute()
5. Client: creates commands and sets receivers

Quick Example

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

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

public class TurnOnLightCommand implements Command {
    private Light light;
    public TurnOnLightCommand(Light light) { this.light = light; }
    public void execute() { light.turnOn(); }
}

public class RemoteControl {
    private Command command;
    public void setCommand(Command command) { this.command = command; }
    public void pressButton() { command.execute(); }
}

public class Main {
    public static void main(String[] args) {
        Light light = new Light();
        Command cmd = new TurnOnLightCommand(light);
        RemoteControl remote = new RemoteControl();
        remote.setCommand(cmd);
        remote.pressButton(); // Output: Light on
    }
}
```

UML (conceptual)

Client -> sets Command on Invoker
Invoker -> calls execute() on Command
Command -> calls action on Receiver

Real-world Use Cases

- GUI buttons and menu items
- Undo/redo systems
- Task scheduling and queues

Command Design Pattern (Java) - Cheat Sheet

- Macro recording

Big Idea

Encapsulate a request as an object so you can parameterize clients with different requests, queue or log requests, and support undoable operations.