1- Builder (Padrão Criacional)

```
// Produto final
class Pizza {
    private List<String> ingredientes = new ArrayList<>();

    public void addIngrediente(String ingrediente) {
        ingredientes.add(ingrediente);
    }

    @Override
    public String toString() {
        return "Pizza com ingredientes: " + String.join(", ", ingredientes);
    }
}

// Builder
class PizzaBuilder {
    private Pizza pizza;

    public PizzaBuilder() {
        this.pizza = new Pizza();
    }

    public PizzaBuilder addQueijo() {
        pizza.addIngrediente("queijo");
        return this;
    }

    public PizzaBuilder addTomate() {
        pizza.addIngrediente("tom
```

2- **Adapter (Padrão Estrutural)**

```
class AudioPlayer {
    public void playMp3(String filename) {
        System.out.println("Tocando arquivo MP3: " + filename);
    }
}

// Adapter para suportar novos formatos
class AudioAdapter {
    private AudioPlayer player;

    public AudioAdapter(AudioPlayer player) {
        this.player = player;
    }

    public void play(String filename, String filetype) {
        if (filetype.equalsIgnoreCase("mp3")) {
            player.playMp3(filename);
        } else if (filetype.equalsIgnoreCase("wav")) {
            System.out.println("Convertendo " + filename + " para MP3 e tocando");
            player.playMp3(filename);
        } else if (filetype.equalsIgnoreCase("flac")) {
            System.out.println("Convertendo " + filename + " para MP3 e tocando");
            player.playMp3(filename);
        } else {
            System.out
```

3 - **Command (Padrão Comportamental)**

```
// Classe que representa o receptor do comando
class Luz {
    public void ligar() {
        System.out.println("A luz foi ligada.");
    }

    public void desligar() {
        System.out.println("A luz foi desligada.");
    }
}

// Interface do comando
interface Comando {
    void executar();
}

// Comandos Concretos
class ComandoLigarLuz implements Comando {
    private Luz luz;

    public ComandoLigarLuz(Luz luz) {
        this.luz = luz;
    }

    @Override
    public void executar() {
        luz.ligar();
    }
}

class ComandoDesligarLuz implements Comando {
    private Luz luz;

    public ComandoDesligarLuz(Luz luz) {
        this.luz = luz;
    }

    @Override
    public void executar() {
        luz.desligar();
    }
}

// Invocador que mantém e executa comandos
class ControleRemoto {
```

```
    private List<Comando> comandos = new ArrayList<>();

    public void adicionarComando(Comando comando) {
        comandos.add(comando);
    }

    public void executarComandos() {
        for (Comando comando : comandos) {
            comando.executar();
        }
        comandos.clear(); // Limpa os comandos após a execução
    }
}

// Uso
public class Main {
    public static void main(String[] args) {
        Luz luz = new Luz();
        Comando ligar = new Com
```