chain of responsibility

```
// Handler
abstract class Handler {
    protected Handler successor;

    public void setSuccessor(Handler successor) {
        this.successor = successor;
    }

    public abstract void handleRequest(int request);
}

// Concrete Handlers
class ConcreteHandler1 extends Handler {
    public void handleRequest(int request) {
        if (request <= 10) {
            System.out.println("ConcreteHandler1 handles request " + request);
        } else if (successor != null) {
            successor.handleRequest(request);
        }
    }
}

class ConcreteHandler2 extends Handler {
    public void handleRequest(int request) {
        if (request > 10 && request <= 20) {
            System.out.println("ConcreteHandler2 handles request " + request);
        } else if (successor != null) {
            successor.handleRequest(request);
        }
    }
}

// Client
public class ChainOfResponsibilityClient {
    public static void main(String[] args) {
        Handler handler1 = new ConcreteHandler1();
        Handler handler2 = new ConcreteHandler2();

        handler1.setSuccessor(handler2);

        handler1.handleRequest(5);
```

```
        handler1.handleRequest(15);
        handler1.handleRequest(25);
    }
}
```

Command

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

// Receiver
class Receiver {
    void action() {
        System.out.println("Receiver's action method");
    }
}

// Concrete Command
class ConcreteCommand implements Command {
    private Receiver receiver;

    public ConcreteCommand(Receiver receiver) {
        this.receiver = receiver;
    }

    public void execute() {
        receiver.action();
    }
}

// Invoker
class Invoker {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void executeCommand() {
        command.execute();
```

```
    }
}

// Client
public class CommandClient {
    public static void main(String[] args) {
        Receiver receiver = new Receiver();
        Command command = new ConcreteCommand(receiver);

        Invoker invoker = new Invoker();
        invoker.setCommand(command);
        invoker.executeCommand();
    }
}
```

Interpreter

```
 import java.util.Map;
import java.util.HashMap;

// Abstract Expression
interface Expression {
    int interpret(Map<String, Integer> context);
}

// Terminal Expression
class TerminalExpression implements Expression {
    private String variable;

    public TerminalExpression(String variable) {
        this.variable = variable;
    }

    public int interpret(Map<String, Integer> context) {
        return context.getOrDefault(variable, 0);
    }
}

// Nonterminal Expression
class AddExpression implements Expression {
    private Expression left;
    private Expression right;
```

```
    public AddExpression(Expression left, Expression right) {
        this.left = left;
        this.right = right;
    }

    public int interpret(Map<String, Integer> context) {
        return left.interpret(context) + right.interpret(context);
    }
}

// Client
public class InterpreterClient {
    public static void main(String[] args) {
        Expression x = new TerminalExpression("x");
        Expression y = new TerminalExpression("y");
        Expression sum = new AddExpression(x, y);

        Map<String, Integer> context = new HashMap<>();
        context.put("x", 5);
        context.put("y", 10);

        int result = sum.interpret(context);
        System.out.println("Result: " + result);
    }
}
```

Iterator

```
import java.util.ArrayList;
import java.util.List;

// Iterator
interface Iterator<T> {
    boolean hasNext();
    T next();
}

// Aggregate
interface Aggregate<T> {
```

```
    Iterator<T> createIterator();
}

// Concrete Iterator
class ConcreteIterator<T> implements Iterator<T> {
    private List<T> items;
    private int index = 0;

    public ConcreteIterator(List<T> items) {
        this.items = items;
    }

    public boolean hasNext() {
        return index < items.size();
    }

    public T next() {
        return items.get(index++);
    }
}

// Concrete Aggregate
class ConcreteAggregate<T> implements Aggregate<T> {
    private List<T> items = new ArrayList<>();

    public void addItem(T item) {
        items.add(item);
    }

    public Iterator<T> createIterator() {
        return new ConcreteIterator<>(items);
    }
}

// Client
public class IteratorClient {
    public static void main(String[] args) {
        Aggregate<String> aggregate = new ConcreteAggregate<>();
        aggregate.addItem("Item 1");
        aggregate.addItem("Item 2");
        aggregate.addItem("Item 3");

        Iterator<String> iterator = aggregate.createIterator();
        while (iterator.hasNext()) {
```

```
            System.out.println(iterator.next());
        }
    }
}
```

Mediator

```
import java.util.ArrayList;
import java.util.List;

// Mediator
interface Mediator {
    void sendMessage(String message, Colleague colleague);
}

// Colleague
abstract class Colleague {
    protected Mediator mediator;

    public Colleague(Mediator mediator) {
        this.mediator = mediator;
    }

    public abstract void receiveMessage(String message);
}

// Concrete Colleague
class ConcreteColleague extends Colleague {
    public ConcreteColleague(Mediator mediator) {
        super(mediator);
    }

    public void receiveMessage(String message) {
        System.out.println("ConcreteColleague received: " + message);
    }
}

// Concrete Mediator
class ConcreteMediator implements Mediator {
    private List<Colleague> colleagues = new ArrayList<>();

    public void addColleague(Colleague colleague) {
```

```
        colleagues.add(colleague);
    }

    public void sendMessage(String message, Colleague sender) {
        for (Colleague colleague : colleagues) {
            if (colleague != sender) {
                colleague.receiveMessage(message);
            }
        }
    }
}

// Client
public class MediatorClient {
    public static void main(String[] args) {
        ConcreteMediator mediator = new ConcreteMediator();

        Colleague colleague1 = new ConcreteColleague(mediator);
        Colleague colleague2 = new ConcreteColleague(mediator);

        mediator.addColleague(colleague1);
        mediator.addColleague(colleague2);

        colleague1.receiveMessage("Hello from Colleague 1");
        colleague2.receiveMessage("Hi from Colleague 2");
    }
}
```

Memento

```
// Memento
class Memento {
    private String state;

    public Memento(String state) {
        this.state = state;
    }

    public String getState() {
        return state;
    }
}
```

```
// Originator
class Originator {
    private String state;

    public void setState(String state) {
        this.state = state;
    }

    public Memento saveStateToMemento() {
        return new Memento(state);
    }

    public void restoreStateFromMemento(Memento memento) {
        state = memento.getState();
    }

    public void printState() {
        System.out.println("Current state: " + state);
    }
}

// Caretaker
class Caretaker {
    private Memento memento;

    public void saveMemento(Memento memento) {
        this.memento = memento;
    }

    public Memento retrieveMemento() {
        return memento;
```

Observer

```
import java.util.ArrayList;
import java.util.List;

// Observer
interface Observer {
    void update(String message);
```

```
}

// Subject
class Subject {
    private List<Observer> observers = new ArrayList<>();
    private String state;

    public void attach(Observer observer) {
        observers.add(observer);
    }

    public void detach(Observer observer) {
        observers.remove(observer);
    }

    public void setState(String state) {
        this.state = state;
        notifyObservers();
    }

    private void notifyObservers() {
        for (Observer observer : observers) {
            observer.update(state);
        }
    }
}

// Concrete Observer
class ConcreteObserver implements Observer {
    private String name;

    public ConcreteObserver(String name) {
        this.name = name;
    }

    public void update(String message) {
        System.out.println(name + " received update: " + message);
    }
}

// Client
public class ObserverClient {
    public static void main(String[] args) {
        Subject subject = new Subject();
```

```
        Observer observer1 = new ConcreteObserver("Observer 1");
        Observer observer2 = new ConcreteObserver("Observer 2");

        subject.attach(observer1);
        subject.attach(observer2);

        subject.setState("New state!");
    }
}
```

State

```
// State
interface State {
    void handle();
}

// Concrete State
class ConcreteStateA implements State {
    public void handle() {
        System.out.println("ConcreteStateA handles the request.");
    }
}

class ConcreteStateB implements State {
    public void handle() {
        System.out.println("ConcreteStateB handles the request.");
    }
}

// Context
class Context {
    private State state;

    public void setState(State state) {
        this.state = state;
    }

    public void request() {
        state.handle();
    }
}
```

```
// Client
public class StateClient {
    public static void main(String[] args) {
        Context context = new Context();
        State stateA = new ConcreteStateA();
        State stateB = new ConcreteStateB();

        context.setState(stateA);
        context.request();

        context.setState(stateB);
        context.request();
    }
}
```

Strategy

```
// Strategy
interface Strategy {
    void doOperation();
}

// Concrete Strategy
class ConcreteStrategyA implements Strategy {
    public void doOperation() {
        System.out.println("ConcreteStrategyA's doOperation method");
    }
}

class ConcreteStrategyB implements Strategy {
    public void doOperation() {
        System.out.println("ConcreteStrategyB's doOperation method");
    }
}

// Context
class Context {
    private Strategy strategy;

    public Context(Strategy strategy) {
        this.strategy = strategy;
```

```
    }

    public void executeStrategy() {
        strategy.doOperation();
    }
}

// Client
public class StrategyClient {
    public static void main(String[] args) {
        Strategy strategyA = new ConcreteStrategyA();
        Context contextA = new Context(strategyA);
        contextA.executeStrategy();

        Strategy strategyB = new ConcreteStrategyB();
        Context contextB = new Context(strategyB);
        contextB.executeStrategy();
    }
}
```

Template method

```
// Abstract Class
abstract class AbstractClass {
    public void templateMethod() {
        step1();
        step2();
    }

    abstract void step1();

    abstract void step2();
}

// Concrete Class
class ConcreteClassA extends AbstractClass {
    void step1() {
        System.out.println("ConcreteClassA's step1 method");
    }

    void step2() {
        System.out.println("ConcreteClassA's step2 method");
```

```
    }
}

class ConcreteClassB extends AbstractClass {
    void step1() {
        System.out.println("ConcreteClassB's step1 method");
    }

    void step2() {
        System.out.println("ConcreteClassB's step2 method");
    }
}

// Client
public class TemplateMethodClient {
    public static void main(String[] args) {
        AbstractClass classA = new ConcreteClassA();
        classA.templateMethod();

        AbstractClass classB = new ConcreteClassB();
        classB.templateMethod();
    }
}
```

Visitor

```
// Visitor
interface Visitor {
    void visit(ConcreteElementA element);
    void visit(ConcreteElementB element);
}

// Concrete Visitor
class ConcreteVisitor implements Visitor {
    public void visit(ConcreteElementA element) {
        System.out.println("ConcreteVisitor visits ConcreteElementA: " + element.operationA());
    }

    public void visit(ConcreteElementB element) {
        System.out.println("ConcreteVisitor visits ConcreteElementB: " + element.operationB());
    }
```

```
}

// Element
interface Element {
    void accept(Visitor visitor);
}

// Concrete Element
class ConcreteElementA implements Element {
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }

    public String operationA() {
        return "OperationA of ConcreteElementA";
    }
}

class ConcreteElementB implements Element {
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }

    public String operationB() {
        return "OperationB of ConcreteElementB";
    }
}

// Object Structure
class ObjectStructure {
    private Element[] elements;

    public ObjectStructure() {
        elements = new Element[]{new ConcreteElementA(), new ConcreteElementB()};
    }

    public void accept(Visitor visitor) {
        for (Element element : elements) {
            element.accept(visitor);
        }
    }
}

// Client
```

```
public class VisitorClient {
    public static void main(String[] args) {
        Visitor visitor = new ConcreteVisitor();
        ObjectStructure objectStructure = new ObjectStructure();
        objectStructure.accept(visitor);
    }
}


//
```