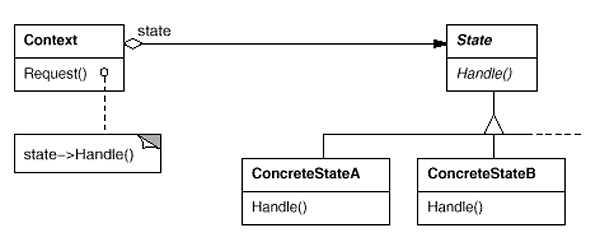
**State Design Pattern -Behavioural- 2022**

GOF : **Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.**

  
As per the above diagram, the context is going to delegate the behavior to the state implementation. In other words, all incoming requests will be handled by the concrete implementation of the state.

Current State + Some Action / Event = Another State

**When to use state pattern**

During the life cycle of an application an object can be in different states and how it processes incoming requests (or make state transitions) based on it’s present state – we can use the state pattern.

If we do not use the state pattern in such case, we will end up having lots of if-else statements which make the code base ugly, unnecessarily complex and hard to maintain. You should use the state pattern, if you have a different behaviour for each state.

It is mainly used for Workflow or Process Orchestration.

Best Example will be

1. Spring Batch Processing: It can be in various states like STARTED, STOPPED, FAILED, COMPLETED,
2. Atlassian Jira: Open, In Progress, Resolved, Closed
3. Traffic Light System : States: RED, YELLOW, GREEN
4. Supply Chain Order System” Ordered, Shipped, Out of Delivery, Delivered

Working example is given below.

**public** **interface** JiraTaskState {

**void** next(JiraTask task);

**void** prev(JiraTask task);

**void** printStatus();

}

**public** **class** JiraTask { 🡸 May be it is context

**private** JiraTaskState state = **new** OpenState(); get()/set() for above state field

**public** **void** previousState() {

state.prev(**this**);

}

**public** **void** nextState() {

state.next(**this**);

}

**public** **void** printStatus() {

state.printStatus();

}

}

**public** **class** OpenState **implements** JiraTaskState {

@Override

**public** **void** next(JiraTask task) {

task.setState(**new** InProgressState());

}

@Override

**public** **void** prev(JiraTask task) {

System.***out***.println("The Task is in Open State.");

}

@Override

**public** **void** printStatus() {

System.***out***.println("Task picked up, In Progress State.");

}

}

**public** **class** InProgressState **implements** JiraTaskState {

@Override

**public** **void** next(JiraTask task) {

task.setState(**new** ResolvedState());

}

@Override

**public** **void** prev(JiraTask task) {

task.setState(**new** OpenState());

}

@Override

**public** **void** printStatus() {

System.***out***.println("Task is in In Progress State.");

}

}

**public** **class** ResolvedState **implements** JiraTaskState {

@Override

**public** **void** next(JiraTask task) {

System.***out***.println("Task has been resolved.");

}

@Override

**public** **void** prev(JiraTask task) {

task.setState(**new** InProgressState());

}

@Override

**public** **void** printStatus() {

System.***out***.println("Task has been completed.");

}

}

**public** **class** StateDemo {

**public** **static** **void** main(String[] args) {

JiraTask pkg = **new** JiraTask();

pkg.printStatus();

pkg.nextState();

pkg.printStatus();

pkg.nextState();

pkg.printStatus();

pkg.nextState();

pkg.printStatus();

}

}