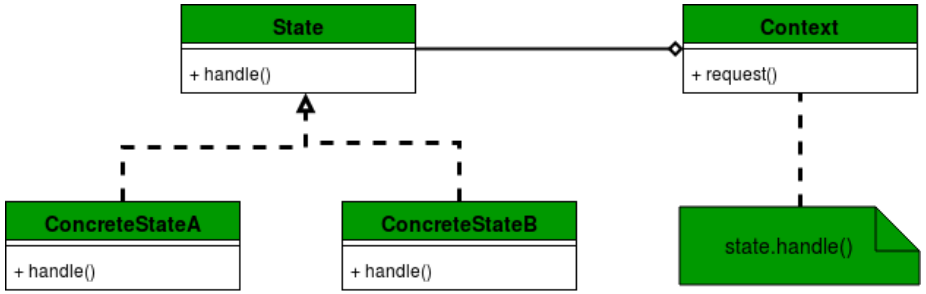
# State Design Pattern

State pattern is one of the behavioral design pattern. State design pattern is used when an Object changes its behavior based on its internal state.

If we have to change behavior of an object based on its state, we can have a state variable in the Object and use if-else condition block to perform different actions based on the state. State pattern is used to provide a systematic and lose-coupled way to achieve this through Context and State implementations.

**UML Diagram of State Design Pattern**




- **Context:** Defines an interface to client to interact. It maintains references to concrete state object which may be used to define current state of object.
- **State:** Defines interface for declaring what each concrete state should do.
- **ConcreteState:** Provides implementation for methods defined in State.

**Example of State Design Pattern**

In below example, we have implemented a mobile state scenario . With respect to alerts, a mobile can be in different states. For example, vibration and silent. Based on this alert state, behavior of the mobile changes when an alert is to be done.



```
// Java program to demonstrate working of
// State Design Pattern

interface MobileAlertState
{
    public void alert(AlertStateContext ctx);
}

class AlertStateContext
{
    private MobileAlertState currentState;

    public AlertStateContext()
    {
        currentState = new Vibration();
    }

    public void setState(MobileAlertState state)
    {
        currentState = state;
    }

    public void alert()
    {
        currentState.alert(this);
    }
}

class Vibration implements MobileAlertState
{
```

```
    @Override
    public void alert(AlertStateContext ctx)
    {
         System.out.println("vibration...");
    }

}

class Silent implements MobileAlertState
{
    @Override
    public void alert(AlertStateContext ctx)
    {
        System.out.println("silent...");
    }

}

class StatePattern
{
    public static void main(String[] args)
    {
        AlertStateContext stateContext = new AlertStateContext();
        stateContext.alert();
        stateContext.alert();
        stateContext.setState(new Silent());
        stateContext.alert();
        stateContext.alert();
        stateContext.alert();
    }
}
```

Output:

```
vibration...
vibration...
silent...
silent...
silent...
```

### Advantages of State Design Pattern

- With State pattern, the benefits of implementing polymorphic behavior are evident, and it is also easier to add states to support additional behavior.
- In the State design pattern, an object's behavior is the result of the function of its state, and the behavior gets changed at runtime depending on the state. This removes the dependency on the if/else or switch/case conditional logic. For example, in the TV remote scenario, we could have also implemented the behavior by simply writing one class and method that will ask for a parameter and perform an action (switch the TV on/off) with an if/else block.
- The State design pattern also improves Cohesion since state-specific behaviors are aggregated into the ConcreteState classes, which are placed in one location in the code.

### Disadvantages of State Design Pattern

- The State design pattern can be used when we need to change state of object at runtime by inputting in it different subclasses of some State base class. This circumstance is advantage and disadvantage in the same time, because we have a clear separate State classes with some logic and from the other hand the number of classes grows up.

This article is contributed by **Saket Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



## Recommended Posts:

Design Patterns | Set 1 (Introduction)

Design Patterns | Set 2 (Factory Method)

Command Pattern

Observer Pattern | Set 1 (Introduction)

Observer Pattern | Set 2 (Implementation)

Singleton Design Pattern | Implementation

Decorator Pattern | Set 1 (Background)

The Decorator Pattern | Set 2 (Introduction and Design)

Decorator Pattern | Set 3 (Coding the Design)

Strategy Pattern | Set 1 (Introduction)

Strategy Pattern | Set 2 (Implementation)

Adapter Pattern

Iterator Pattern

Curiously recurring template pattern (CRTP)

Flyweight Design Pattern



**Article Tags :** Design Pattern

Be the First to upvote.

To-do Done

1.5

Based on **2** vote(s)

Feedback/ Suggest Improvement | Add Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments | Share this post!