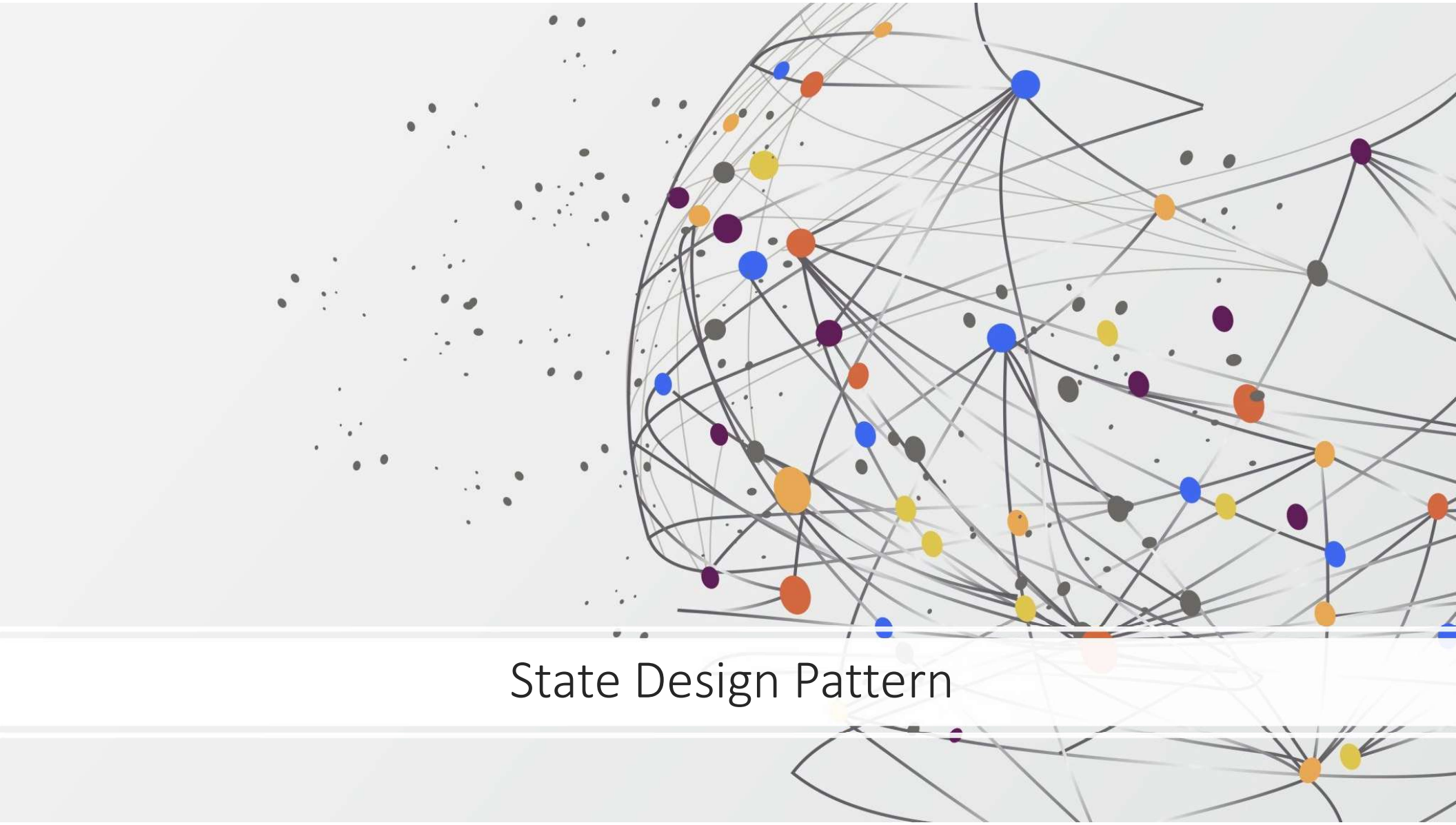
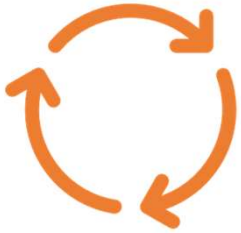


# State Design Pattern

# Definition



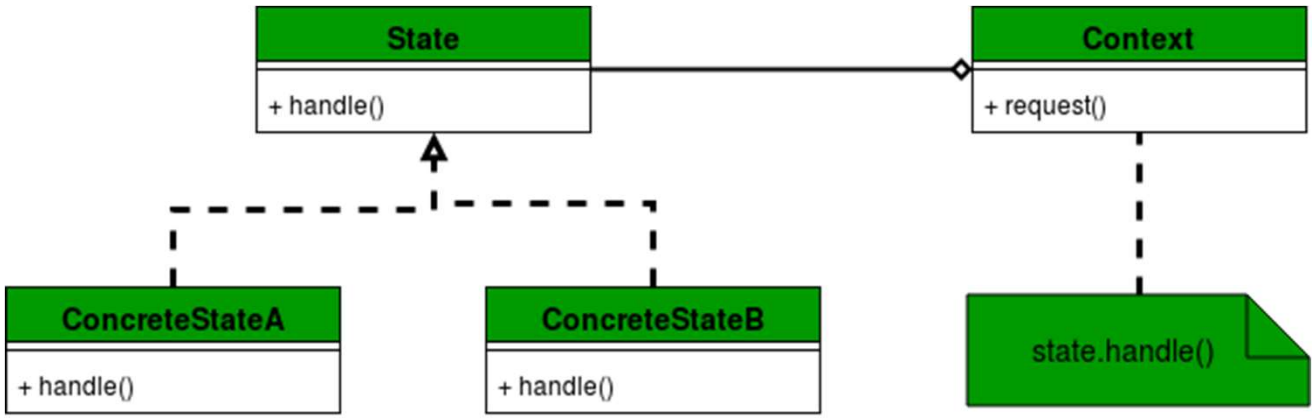
State pattern is to **allow the object for changing its behavior without changing its class.**

Also, by implementing it, the code should remain cleaner without many if/else statements.

# Major component

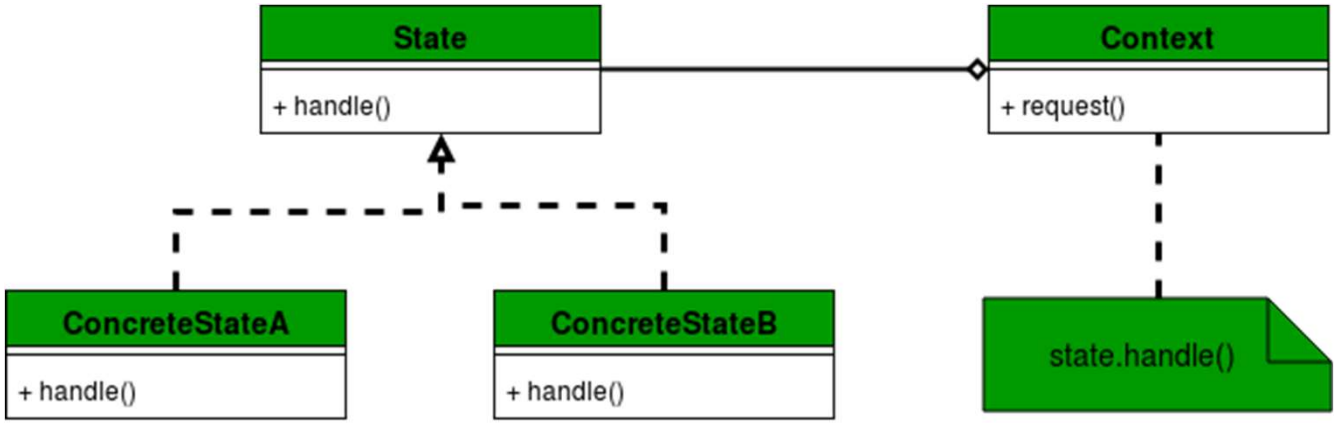
- **Context:** Defines an interface for clients to interact. It maintains references to concrete state objects which may be used to define the current state of objects.
- **State:** Defines interface for declaring what each concrete state should do.
- **ConcreteState:** Provides the implementation for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

# UML diagram

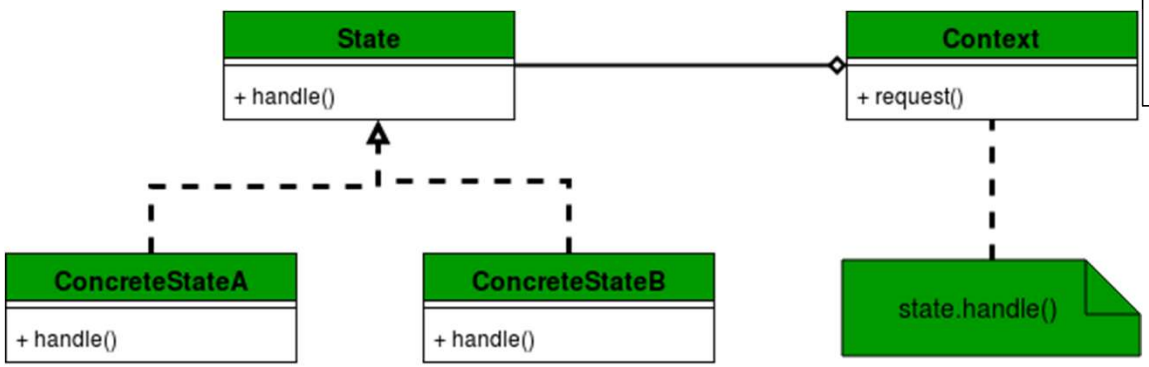

State
+ handle()
Context
+ request()
ConcreteStateA
+ handle()
ConcreteStateB
+ handle()
state.handle()

# Example:

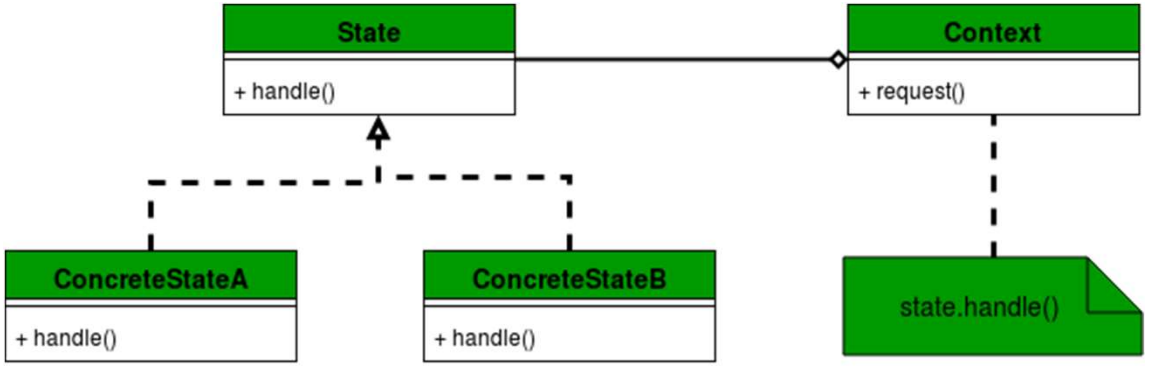
```
// Java program to demonstrate working of
// State Design Pattern

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); }
}
```

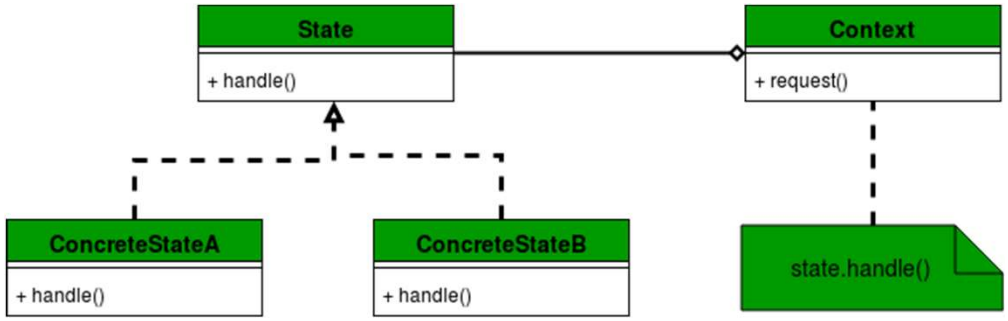


# Different state




```
class Vibration implements MobileAlertState {
@Override public void alert(AlertStateContext ctx)
{
System.out.println(" vibration... & quot;);
}
}

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

# Demo class




```
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();
}
}
```