State in C++

Back to **State** description

```
        State design pattern – an FSM with two states and two events
(distributed transition logic – logic in the derived state classes).
```

```
#include <iostream>
using namespace std;
class Machine
{
  class State *current;
  public:
    Machine();
    void setCurrent(State *s)
    {
        current = s;
    }
    void on();
    void off();
};

class State
{
  public:
    virtual void on(Machine *m)
    {
        cout << "   already ON\n";
    }
    virtual void off(Machine *m)
    {
        cout << "   already OFF\n";
    }
};

void Machine::on()
{
  current->on(this);
}

void Machine::off()
{
  current->off(this);
}

class ON: public State
{
  public:
    ON()
    {
        cout << "   ON-ctor ";
    };
    ~ON()
    {
        cout << "   dtor-ON\n";
```

```
    };
    void off(Machine *m);
};

class OFF: public State
{
  public:
    OFF()
    {
        cout << "   OFF-ctor ";
    };
    ~OFF()
    {
        cout << "   dtor-OFF\n";
    };
    void on(Machine *m)
    {
        cout << "   going from OFF to ON";
        m->setCurrent(new ON());
        delete this;
    }
};

void ON::off(Machine *m)
{
  cout << "   going from ON to OFF";
  m->setCurrent(new OFF());
  delete this;
}

Machine::Machine()
{
  current = new OFF();
  cout << '\n';
}

int main()
{
  void(Machine:: *ptrs[])() =
  {
    Machine::off, Machine::on
  };
  Machine fsm;
  int num;
  while (1)
  {
    cout << "Enter 0/1: ";
    cin >> num;
    (fsm. *ptrs[num])();
```

```
  }
}
```

Output

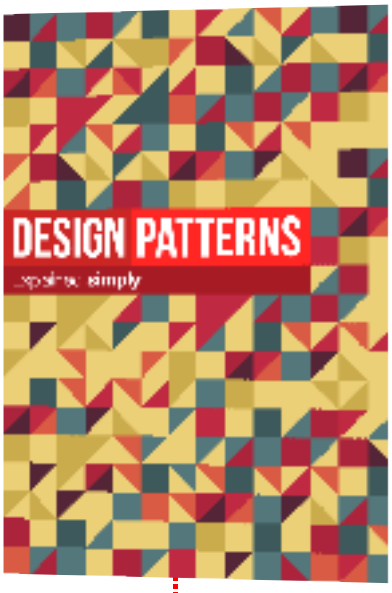
```
   OFF-ctor
Enter 0/1: 0
   already OFF
Enter 0/1: 1
   going from OFF to ON   ON-ctor    dtor-OFF
Enter 0/1: 1
   already ON
Enter 0/1: 0
   going from ON to OFF   OFF-ctor    dtor-ON
Enter 0/1: 1
   going from OFF to ON   ON-ctor    dtor-OFF
Enter 0/1: 0
   going from ON to OFF   OFF-ctor    dtor-ON
Enter 0/1: 0
   already OFF
Enter 0/1:
```

Read next

This article is taken from our book **Design Patterns Explained Simply**.

All of the design patterns are compiled there. The book is written in clear, simple language that makes it easy to read and understand (just like this article).

We distribute it in PDF & EPUB formats so you can get it onto your iPad, Kindle, or other portable device immediately after your purchase.



Learn more

Code examples

Java	State in Java: Before and after	State in Java: Case statement considered harmful	State in Java	State in Java	State in Java	State in Java: Distributed transition logic	State in Java
C++	State in C++						
PHP	State in PHP						
Delphi	State in Delphi	State in Delphi					
Python	State in						

Design Patterns
AntiPatterns
Refactoring
UML

My account
Forum
Contact us
About us