Twitter

Facebook 6148

Google+ 1257





★ / Design Patterns / Behavioral patterns / State

State in C++

4

Back to State description

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

Design Patterns AntiPatterns Refactoring UML My account Forum

Contact us

About us

© 2007-2017 SourceMaking.com / All rights reserved.

Privacy policy



★ Premium Stuff

♣ Design Patterns

Creational patterns

Structural patterns

Behavioral patterns

Chain of Responsibility

Command

Interpreter

Iterator

Mediator

Memento

Null Object

Observer

→ State

Strategy

Template Method

Visitor

★ AntiPatterns

≫ Refactoring

♣ UML


```
#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";</pre>
    }
    virtual void off(Machine *m)
        cout << " already OFF\n";</pre>
    }
};
void Machine::on()
  current->on(this);
}
void Machine::off()
  current->off(this);
}
class ON: public State
  public:
    ON()
        cout << " ON-ctor ";</pre>
    };
    ~ON()
        cout << " dtor-ON\n";</pre>
    };
    void off(Machine *m);
};
class OFF: public State
  public:
    OFF()
    {
        cout << " OFF-ctor ";
    };
    ~0FF()
    {
        cout << " dtor-OFF\n";</pre>
    };
    void on(Machine *m)
        cout << " going from OFF to ON";</pre>
        m->setCurrent(new ON());
        delete this;
    }
};
```

2 of 4 2017年05月09日 17:33



★ Premium Stuff

Creational patterns

Structural patterns

Behavioral patterns

Chain of Responsibility

Command

Interpreter

Iterator

Mediator

Memento

Null Object

Observer

→ State

Strategy

Template Method

Visitor

★ AntiPatterns

≫ Refactoring

♣ UML

```
void ON::off(Machine *m)
{
  cout << " going from ON to OFF";</pre>
  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: ";</pre>
    cin >> num;
    (fsm. *ptrs[num])();
  }
}
```

Output

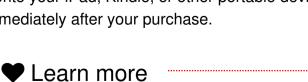
```
OFF-ctor
Enter 0/1: 0
  already OFF
Enter 0/1: 1
   going from OFF to ON ON-ctor
                                     dtor-OFF
Enter 0/1: 1
  already ON
Enter 0/1: 0
   going from ON to OFF
                         OFF-ctor
                                      dtor-ON
Enter 0/1: 1
             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

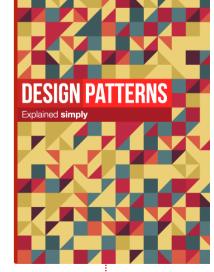
Read next

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.





Code examples

3 of 4 2017年05月09日 17:33



★ Premium Stuff

♣ Design Patterns

Creational patterns

Structural patterns

Behavioral patterns

Chain of Responsibility

Command

Interpreter

Iterator

Mediator

Memento

Null Object

Observer

→ State

Strategy

Template Method

Visitor

★ AntiPatterns

≫ Refactoring

♣ UML

▲ Login 🖂 Feedback

Java	State in Java: Before and after	State in Java	State in Java	State in Java	State in Java: Distribut transition logic	State in Java: ecCase n statemen consider harmful	I .
C++	State in C++						
PHP	State in PHP						
Delphi	State in Delphi	State in Delphi					
Python	State in Python						

4 of 4 2017年05月09日 17:33