/ Design Patterns / Structural patterns / Facade

Facade in C++

Back to **Facade** description

Facade design pattern demo

Discussion. Structuring a system into subsystems helps reduce complexity. A common design goal is to minimize the communication and dependencies between subsystems.

One way to achieve this goal is to introduce a "facade" object that provides a single, simplified interface to the many, potentially complex, individual interfaces within the subsystem. In this example, the "subsystem" for responding to a networking service request has been modeled, and a facade (FacilitiesFacade) interposed. The facade "hides" the twisted and bizarre choreography necessary to satisfy even the most basic of requests. All the user of the facade object has to do is make one or two phone calls a week for 5 months, and a completed service request results.

```
#include <iostream.h>

class MisDepartment
{
  public:
    void submitNetworkRequest()
    {
        _state = 0;
    }
    bool checkOnStatus()
    {
        _state++;
        if (_state == Complete)
          return 1;
        return 0;
    }
  private:
    enum States
    {
        Received, DenyAllKnowledge, ReferClientToFacilities,
          FacilitiesHasNotSentPaperwork, ElectricianIsNotDone,
          ElectricianDidItWrong, DispatchTechnician, SignedOff, DoesNotWork,
          FixElectriciansWiring, Complete
    };
    int _state;
};

class ElectricianUnion
{
  public:
    void submitNetworkRequest()
    {
        _state = 0;
    }
    bool checkOnStatus()
    {
        _state++;
        if (_state == Complete)
          return 1;
        return 0;
    }
  private:
    enum States
    {
        Received, RejectTheForm, SizeTheJob, SmokeAndJokeBreak,
          WaitForAuthorization, DoTheWrongJob, BlameTheEngineer, WaitToPunchOut,
          DoHalfAJob, ComplainToEngineer, GetClarification, CompleteTheJob,
          TurnInThePaperwork, Complete
    };
    int _state;
};

class FacilitiesDepartment
{
  public:
    void submitNetworkRequest()
    {
        _state = 0;
    }
    bool checkOnStatus()
    {
        _state++;
        if (_state == Complete)
          return 1;
        return 0;
    }
  private:
    enum States
    {
        Received, AssignToEngineer, EngineerResearches, RequestIsNotPossible,
          EngineerLeavesCompany, AssignToNewEngineer, NewEngineerResearches,
          ReassignEngineer, EngineerReturns, EngineerResearchesAgain,
          EngineerFillsOutPaperWork, Complete
    };
    int _state;
};

class FacilitiesFacade
{
  public:
    FacilitiesFacade()
    {
        _count = 0;
    }
    void submitNetworkRequest()
    {
        _state = 0;
    }
    bool checkOnStatus()
    {
        _count++;
        /* Job request has just been received */
        if (_state == Received)
        {
            _state++;
            /* Forward the job request to the engineer */
            _engineer.submitNetworkRequest();
            cout << "submitted to Facilities - " << _count <<
              " phone calls so far" << endl;
        }
        else if (_state == SubmitToEngineer)
        {
            /* If engineer is complete, forward to electrician */
            if (_engineer.checkOnStatus())
            {
                _state++;
                _electrician.submitNetworkRequest();
                cout << "submitted to Electrician - " << _count <<
                  " phone calls so far" << endl;
            }
        }
        else if (_state == SubmitToElectrician)
        {
            /* If electrician is complete, forward to technician */
            if (_electrician.checkOnStatus())
            {
                _state++;
                _technician.submitNetworkRequest();
                cout << "submitted to MIS - " << _count <<
                  " phone calls so far" << endl;
            }
        }
        else if (_state == SubmitToTechnician)
        {
            /* If technician is complete, job is done */
            if (_technician.checkOnStatus())
              return 1;
        }
        /* The job is not entirely complete */
        return 0;
    }
    int getNumberOfCalls()

    {
        return _count;
    }
  private:
    enum States
    {
        Received, SubmitToEngineer, SubmitToElectrician, SubmitToTechnician
    };
    int _state;
    int _count;
    FacilitiesDepartment _engineer;
    ElectricianUnion _electrician;
    MisDepartment _technician;
```

```
};

int main()
{
  FacilitiesFacade facilities;

  facilities.submitNetworkRequest();
  /* Keep checking until job is complete */
  while (!facilities.checkOnStatus())
    ;
  cout << "job completed after only " << facilities.getNumberOfCalls() <<
    " phone calls" << endl;
}
```

Output

```
submitted to Facilities - 1 phone calls so far
submitted to Electrician - 12 phone calls so far
submitted to MIS - 25 phone calls so far
job completed after only 35 phone calls
```

Code examples

Java	Facade in Java
C++	Facade in C++
PHP	Facade in PHP
Delphi	Facade in Delphi

Design Patterns
AntiPatterns
Refactoring
UML

My account
Forum
Contact us
About us