



[Home](#) / [Design Patterns](#) / [Structural patterns](#) / [Bridge](#)

# Bridge in C++



[Back to Bridge description](#)

## Bridge design pattern demo

Discussion. The motivation is to decouple the Time interface from the Time implementation, while still allowing the abstraction and the realization to each be modelled with their own inheritance hierarchy. The implementation classes below are straight-forward. The interface classes are a little more subtle. Routinely, a Bridge pattern interface hierarchy "has a" implementation class. Here the interface base class "has a" a pointer to the implementation base class, and each class in the interface hierarchy is responsible for populating the base class pointer with the correct concrete implementation class. Then all requests from the client are simply delegated by the interface class to the encapsulated implementation class.

```
#include <iostream.h>
#include <iomanip.h>
#include <string.h>

class TimeImp {
public:
    TimeImp(int hr, int min) {
        hr_ = hr;
        min_ = min;
    }
    virtual void tell() {
        cout << "time is " << setw(2) << setfill(48) << hr_ << min_ << endl;
    }
protected:
    int hr_, min_;
};

class CivilianTimeImp: public TimeImp {
public:
    CivilianTimeImp(int hr, int min, int pm): TimeImp(hr, min) {
        if (pm)
            strcpy(whichM_, " PM");
        else
            strcpy(whichM_, " AM");
    }

    /* virtual */
    void tell() {
        cout << "time is " << hr_ << ":" << min_ << whichM_ << endl;
    }
protected:
    char whichM_[4];
};

class ZuluTimeImp: public TimeImp {
public:
    ZuluTimeImp(int hr, int min, int zone): TimeImp(hr, min) {
        if (zone == 5)
            strcpy(zone_, " Eastern Standard Time");
        else if (zone == 6)
            strcpy(zone_, " Central Standard Time");
    }

    /* virtual */
    void tell() {
        cout << "time is " << setw(2) << setfill(48) << hr_ << min_ << zone_ <<
            endl;
    }
}
```

```
protected:
    char zone_[30];
};

class Time {
public:
    Time(){}
    Time(int hr, int min) {
        imp_ = new TimeImp(hr, min);
    }
    virtual void tell() {
        imp_->tell();
    }
protected:
    TimeImp *imp_;
};

class CivilianTime: public Time {
public:
    CivilianTime(int hr, int min, int pm) {
        imp_ = new CivilianTimeImp(hr, min, pm);
    }
};

class ZuluTime: public Time {
public:
    ZuluTime(int hr, int min, int zone) {
        imp_ = new ZuluTimeImp(hr, min, zone);
    }
};

int main() {
    Time *times[3];
    times[0] = new Time(14, 30);
    times[1] = new CivilianTime(2, 30, 1);
    times[2] = new ZuluTime(14, 30, 6);
    for (int i = 0; i < 3; i++)
        times[i]->tell();
}
```

## Output

```
time is 1430
time is 2:30 PM
time is 1430 Central Standard Time
```

## 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	Bridge in Java	Bridge in Java	Bridge in Java
C++	Bridge in C++		
PHP	Bridge in PHP		

Design Patterns  
AntiPatterns  
Refactoring  
UML

My account  
Forum  
Contact us  
About us

