Đại Học Quốc Gia Tp. Hồ Chí Minh
Trường ĐH Khoa Học Tự Nhiên
Khoa Công Nghệ Thông Tin

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CHƯƠNG 8
KHUÔN MẪU (TEMPLATE)
MẪU THIẾT KẾ (DESIGN PATTERNS)

ThS: Phạm Nguyễn Sơn Tùng
Email: pnstung@fit.hcmus.edu.vn

NỘI DUNG BÀI HỌC

1. Giới thiệu về Template
2. Function & Class Template
3. Giới thiệu về mẫu thiết kế Design Patterns
4. Các mẫu thiết kế thông dụng
5. Demo

ĐỊNH NGHĨA VỀ TEMPLATE

- Định nghĩa: Template (khuôn mẫu) là một từ khóa được sử dụng để tạo ra các phương thức các lớp với nhiều loại kiểu dữ liệu khác nhau mà không cần phải viết lại nhiều lần.

CÁC LOẠI TEMPLATE

- Class Template: Khuôn mẫu lớp cho phép tạo các lớp khuôn mẫu tổng quát, bao gói dữ liệu và xử lý dữ liệu vào cùng một nơi, sử dụng một cách tổng quát.
- Function Template: Khuôn mẫu hàm cho phép định nghĩa các hàm/phương thức tổng quát dùng cho các kiểu dữ liệu tùy ý.
- Other Template: Template specialization (Khuông mẫu chuyên môn hóa).

CÁC LOẠI TEMPLATE CÓ SẴN

- Vector: Sử dụng như mảng động có thể dùng cho nhiều kiểu dữ liệu khác nhau.
- Stack/Queue: Ngăn xếp và hàng đợi.
- Array: Dùng tạo ra mảng với các phần tử có kiểu tuỳ ý.
- List: Dùng tạo ra danh sách liên kết có kiểu tuỳ ý.

CLASS TEMPLATE

template <class indetifier> class {……..}

từ khoá từ khoá Tên biến Tên lớp

CLASS TEMPLATE

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Stack
{
    int* list;
    int max;
    int size;
};
```

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

template <class T>
class Stack
{
private:
    T* list;
    int size;
    int max;
```

CLASS TEMPLATE

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Stack
{
    int* list;
    int max;
    int size;
};
```

```
public:
    Stack()
    {
        size = 0;
        max = 0;
        list = new T[size];
    }
    bool InitStack(int);
    bool IsEmpty();
    void Push(T);
    bool Pop();
    T Top();
};
```

FUNCTION TEMPLATE

template <class indetifier> function_declaration;

từ khoá từ khoá Tên biến Tên hàm

FUNCTION TEMPLATE

```
bool initStack(Stack& s, int n)
{
    s.list = new int[n];
    if (s.list == NULL)
        return false;
    s.max = n;
    s.size = 0;
    return true;
}
```

```
template <class T>
bool
Stack<T>::InitStack(int n)
{
    list = new int[n];
    if (list == NULL)
        return false;
    max = n;
    size = 0;
    return true;
}
```

FUNCTION TEMPLATE

```
bool IsEmpty(const Stack& s)
{
    if (s.size == 0)
        return true;
    return false;
}
```

```
template <class T>
bool Stack<T>::IsEmpty()
{
    if (size == 0)
        return true;
    return false;
}
```

FUNCTION TEMPLATE

```
bool IsFull(const Stack& s)
{
    if (s.size == s.max)
        return true;
    return false;
}
```

```
template <class T>
bool Stack<T>::IsFull()
{
    if (size == max)
        return true;
    return false;
}
```

FUNCTION TEMPLATE

```
bool Push(Stack& s, int newitem)
{
    if (IsFull(s) == true)
        return false;
    s.list[s.size] = newitem;
    s.size++;
    return true;
}
```

```
template <class T>
bool Stack<T>::Push(T newitem)
{
    if (IsFull() == true)
        return false;
    list[size] = newitem;
    size++;
    return true;
}
```

FUNCTION TEMPLATE

```
bool Pop(Stack& s)
{
    if (IsEmpty(s) == true)
        return false;
    s.size--;
    return true;
}
```

```
template <class T>
bool Stack<T>::Pop()
{
    if (IsEmpty() == true)
        return false;
    size--;
    return true;
}
```

FUNCTION TEMPLATE

```
int Top(const Stack s)
{
    if (IsEmpty(s)==true)
        return INT_MIN;
    int topitem = s.list[s.size-1];
    return topitem;
}
```

```
template <class T>
T Stack<T>::Top()
{
    if (Empty() == true)
        return NULL;
    T newitem = list[size - 1];
    return newitem;
}
```

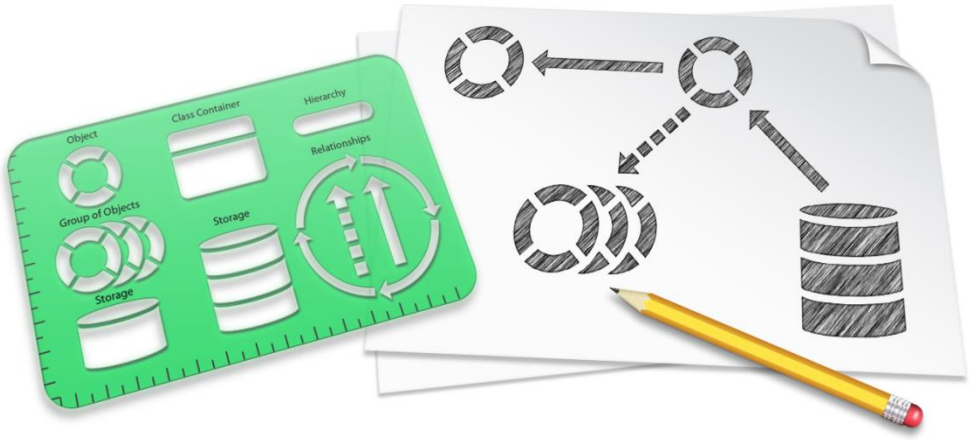
FUNCTION TEMPLATE

```
int main()
{
    Stack<int> s;
    InitStack(s, 5);
    Push(s, 5);
    Push(s, 6);
    Push(s, -1);
    Push(s, 2);
    Push(s, 3);
    Pop(s);
    int ketqua = Top(s);
    return 1;
}
```

```
int main()
{
    Stack<int> s;
    s.InitStack(5);
    s.Push(5);
    s.Push(6);
    s.Push(-1);
    s.Push(2);
    s.Push(3);
    s.Pop();
    int ketqua = s.Top();
    return 1;
}
```

GIỚI THIỆU TỔNG QUAN

- Mẫu thiết kế (Design Patterns) là một giải pháp tổng thể cho các vấn đề trong thiết kế phần mềm theo hướng đối tượng. Nó không phải là chuẩn được quy định của bất kỳ ai.

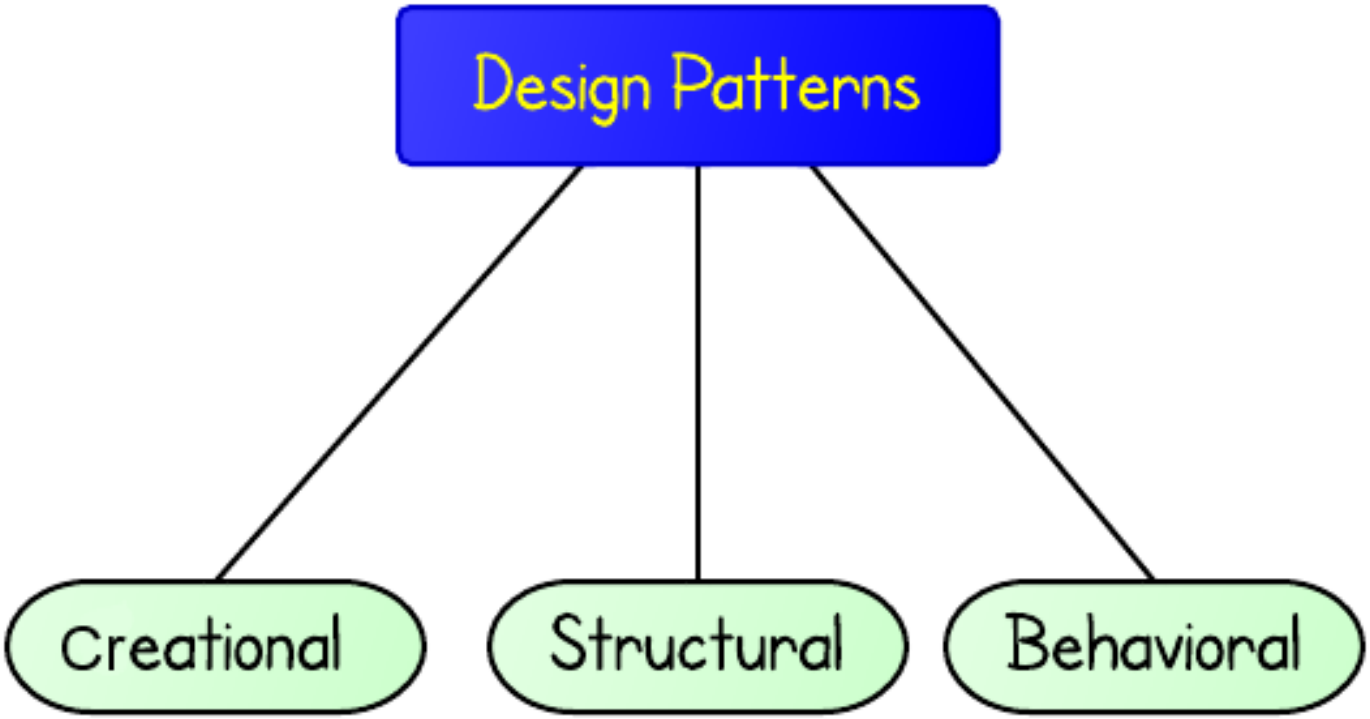


GIỚI THIỆU TỔNG QUAN

- Tính uyển chuyển: 1 khuôn mẫu dùng để giải quyết một vấn đề nào đó. Nó không phải là thiết kế cuối cùng.
- Dễ nâng cấp: Nâng cấp giao diện, tính năng phục vụ một công việc nào đó.
- Dễ thay đổi: Thay đổi các theo yêu cầu của khách hàng.

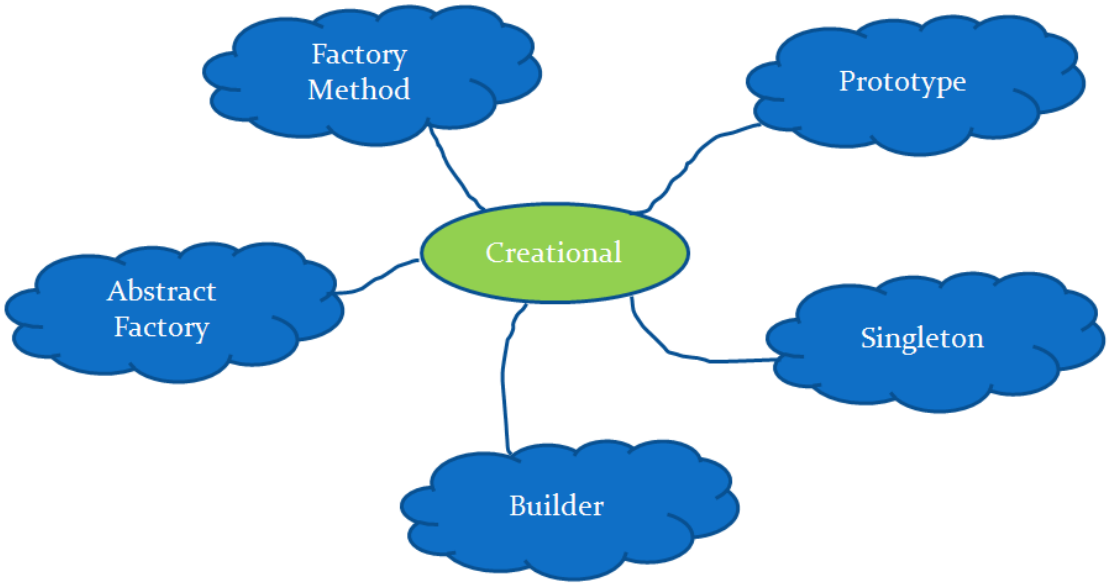
GIỚI THIỆU TỔNG QUAN

- Design Patterns gồm 3 loại chính như sau:



MẪU CREATIONAL PATTERNS

- Creational Patterns: Khắc phục các vấn đề về khởi tạo đối tượng, hạn chế sự phụ thuộc nền tảng (gồm 5 mẫu Factory, Abstract Factory, Singleton, Prototype, Builder)

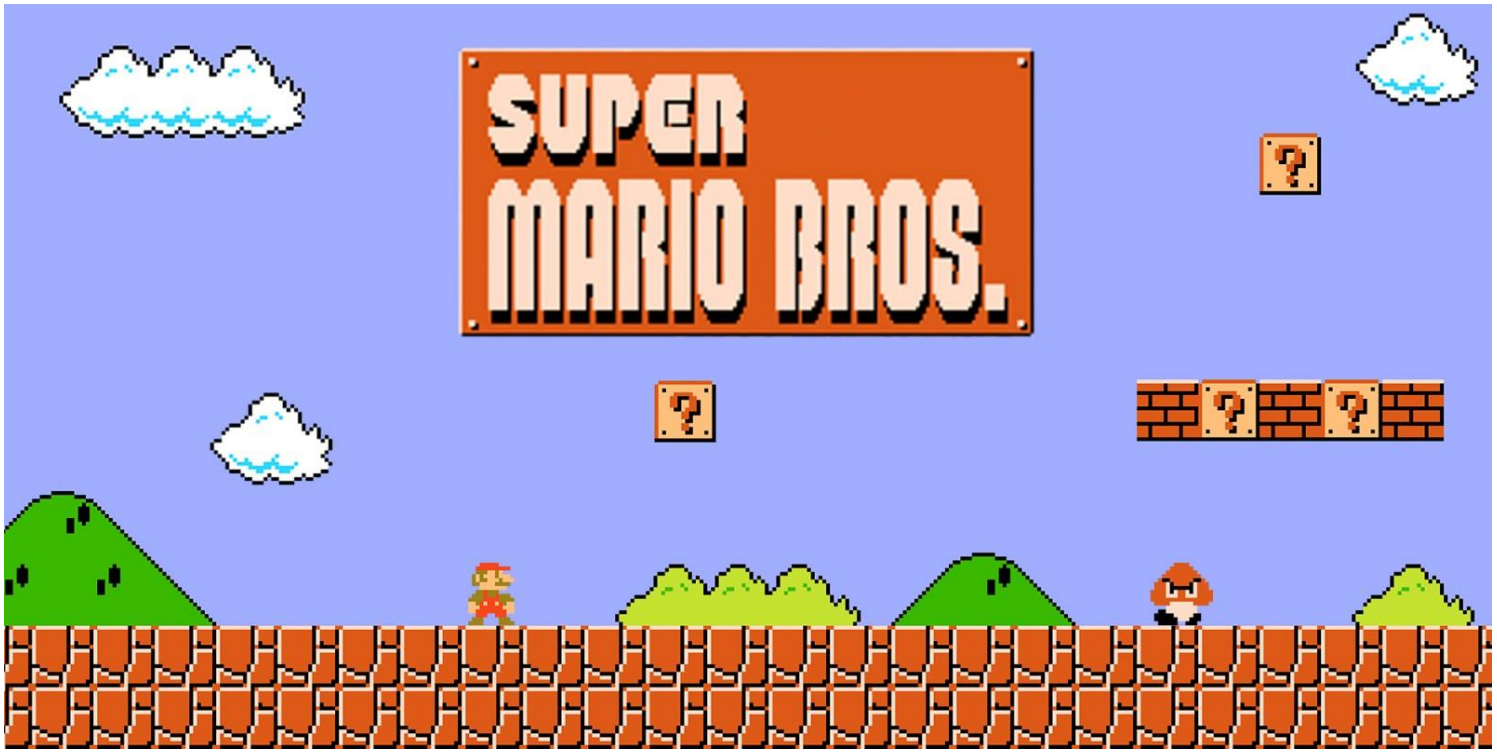


MẪU SINGLETON

- Creational Patterns: Demo mẫu Singleton. Đây là mẫu thường gặp ở các bài toán khởi tạo đối tượng.
- Sử dụng Singleton khi chúng ta muốn:
  - Đảm bảo rằng chỉ có một thể hiện của đối tượng riêng biệt tại một thời điểm bất kỳ.
  - Có thể quản lý số lượng thể hiện của một lớp trong giới hạn chỉ định.
  - Khi thể hiện duy nhất khả mở thông qua việc kế thừa, người dùng có thể sử dụng thể hiện kế thừa đó mà không cần thay đổi các đoạn mã của chương trình.

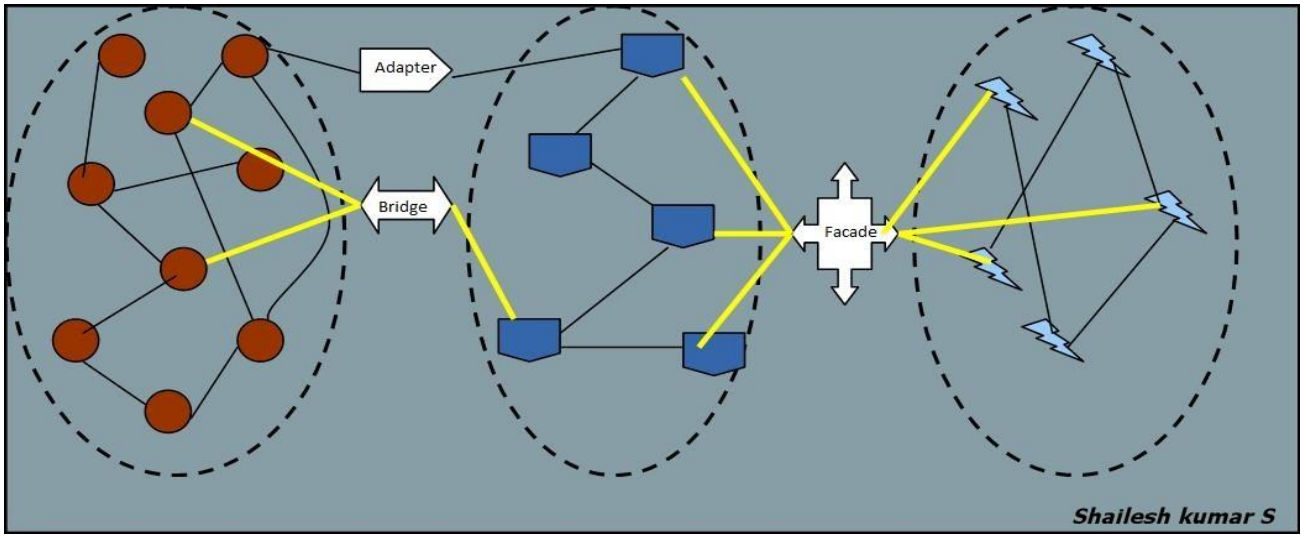
MẪU SINGLETON

➢ **Ví dụ:** Xây dựng nhân vật Mario



MẪU STRUCTURAL PATTERNS

- Structural Patterns: Liên quan đến các vấn đề làm thế nào để các lớp và đối tượng kết hợp với nhau tạo thành các cấu trúc lớn hơn (gồm 7 mẫu Adapter, Bridge, Composite, Decorator, Facade, Flyweight và Proxy)

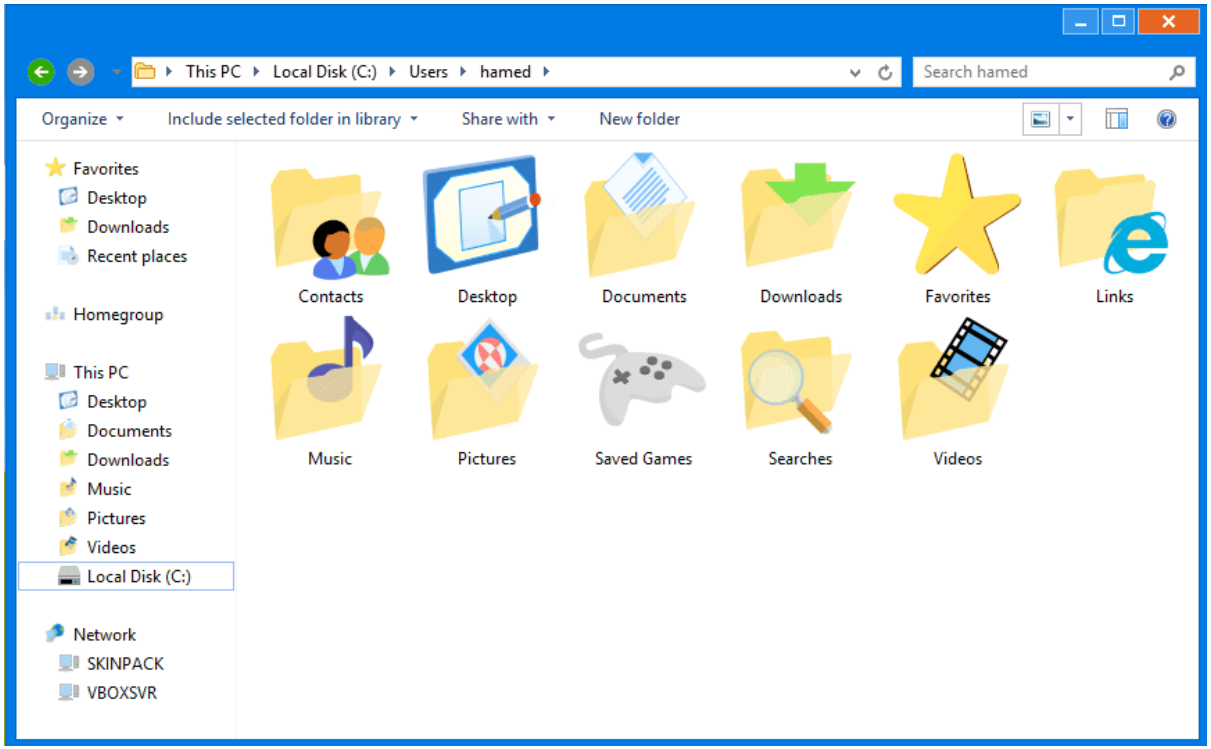


MẪU COMPOSITE

- Structural Patterns: Demo mẫu Composite. Đây là mẫu thường gặp ở các bài toán trong lớp có lớp lòng nhau.
- Sử dụng Composite khi chúng ta muốn:
  - Composite Pattern có thể được sử dụng để tạo ra một cấu trúc giống như cấu trúc cây.
  - Tất cả các đối tượng trong cấu trúc được thao tác một cách thuần nhất như nhau.

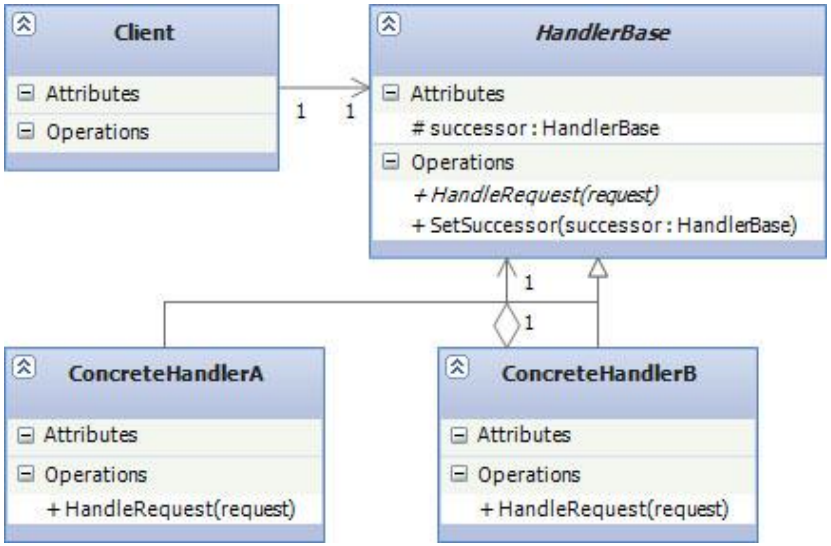
MẪU COMPOSITE

➢ **Ví dụ:** Bài toán cây thư mục và tập tin (word)



MẪU BEHAVIORAL PATTERNS

- Behavioral Patterns: Mô tả hành vi của đối tượng, cách thức để các lớp hoặc đối tượng có thể giao tiếp với nhau (gồm 11 mẫu Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy và Visitor)



MẪU STATE

- Behavioral Patterns: Demo mẫu State. Đây là mẫu thường gặp ở các dạng thiết kế game.
- Sử dụng State khi chúng ta muốn:
  - Hành vi của đối tượng có thể phụ thuộc vào trạng thái hiện hành của đối tượng đó. Cách tốt nhất để giúp đối tượng thay đổi linh động và dễ dàng 1 hành vi phù hợp theo từng trạng thái là dùng mẫu State.

MẪU STATE

Xây dựng trạng thái của cầu thủ bóng đá.

TÀI LIỆU THAM KHẢO

- 1. Lập trình hướng đối tượng, Trần Đan Thư, Đinh Bá Tiến, Nguyễn Tấn Trần Minh Khang, NXB Khoa Học Kỹ Thuật, 2010.
- 2. Lập trình hướng đối tượng, Trần Văn Lăng, NXB Thống Kê, 2004.
- 3. Object-oriented Programming in c++, 4th Edition, Robert Lafore, SAMS, 1997.
- 4. C++ Primer, Fifth Edition, Stephen Prata, SAMS, 2004.
- 5. Slide bài giảng của: Thầy Nguyễn Minh Huy, Thầy Hồ Tuấn Thanh, Thầy Đinh Bá Tiến, Thầy Trần Văn Lăng, Thầy Đặng Bình Phương, Cô Đặng Thị Thanh Nguyên.
- 6. Các website về lập trình:
  - http://www.cplusplus.com/
  - http://stackoverflow.com/
  - http://www.codeproject.com/