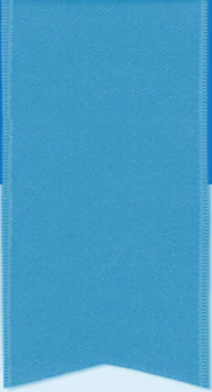


CÔNG NGHỆ PHẦN MỀM

Khoa Công nghệ Thông tin – Đại học Đà Lạt





CHƯƠNG 6. XÂY DỰNG PHẦN MỀM

Mục tiêu

- Sau khi hoàn thành bài học này, sinh viên cần:
 - Nắm được các khái niệm về kiến trúc phần mềm, mẫu thiết kế, đặc tả phần mềm
 - Phân biệt được các kiểu kiến trúc phần mềm
 - Nắm vững cách thức quản lý mã nguồn và tiến độ công việc
 - Hiểu rõ được khái niệm về mẫu thiết kế và các cách thức chuẩn hóa mã hóa
 - Hiểu được cách thực hiện đặc tả kỹ thuật phần mềm
 - Nắm được quy trình thực hiện xây dựng các chức năng của phần mềm

Nội dung

- Kiến trúc phần mềm
- Một số công cụ hỗ trợ xây dựng phần mềm
- Những nguyên tắc cơ bản trong xây dựng phần mềm
- Đặc tả kỹ thuật
- Ví dụ minh họa thực thi một số Use case

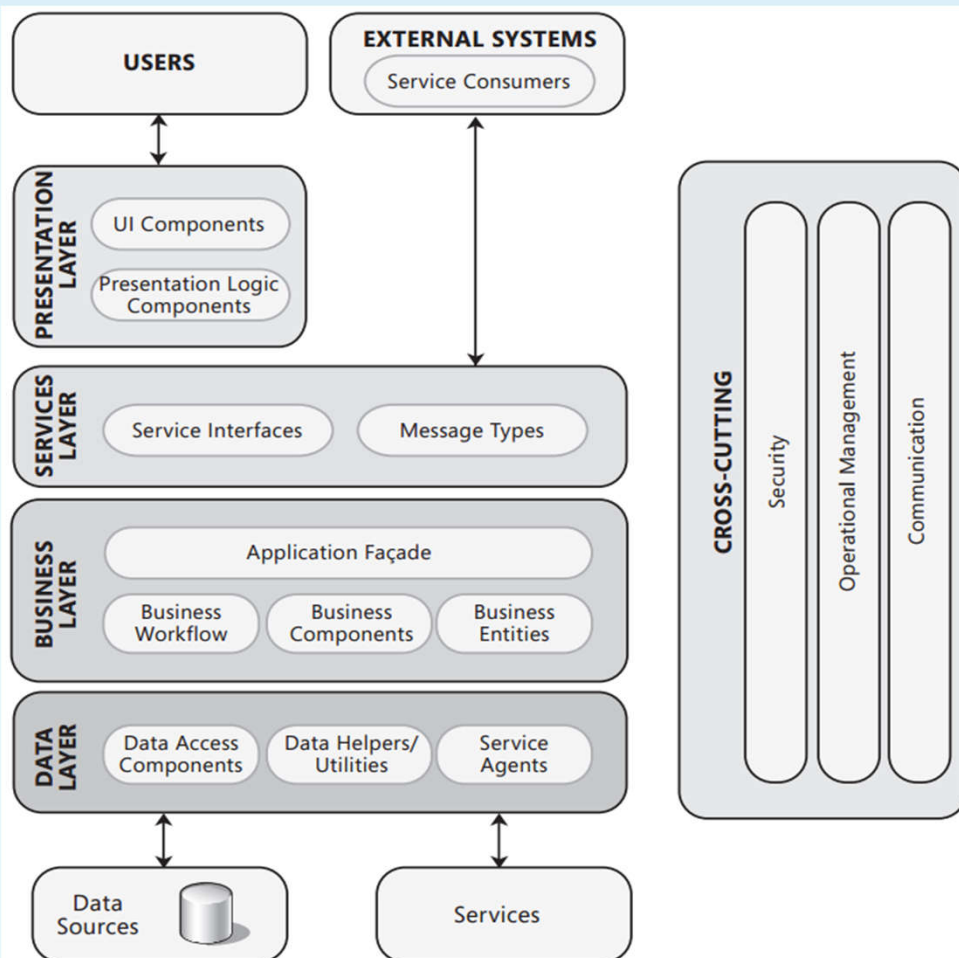
Kiến trúc phần mềm là gì ?

- Tập hợp các nguyên tắc để hình thành một framework (khung) chung cho các hệ thống phần mềm hay cấu trúc của các thành phần trong phần mềm
- Cung cấp các giải pháp để giải quyết các vấn đề xảy ra lặp lại nhiều lần trong quá trình phát triển giúp tái sử dụng các thành phần và mẫu thiết kế của các dự án
- Tầm quan trọng của kiến trúc phần mềm:
 - Hỗ trợ giao tiếp
 - Giúp ra quyết định sớm
 - Tính khả chuyển cho hệ thống

Các kiểu kiến trúc phần mềm

Danh mục	Kiểu kiến trúc
Truyền thông - Communication	Kiến trúc hướng dịch vụ (SOA), Message Bus
Triển khai - Deployment	Client/Server, N-Tier, 3-Tier
Lĩnh vực - Domain	Domain Driven Design
Cấu trúc - Structure	Kiến trúc dựa trên các thành phần, Kiến trúc hướng đối tượng, kiến trúc hướng dịch vụ

Phân tầng ứng dụng



- Tầng trình bày (Presentation Layer): Quản lý tương tác trực quan của người dùng với hệ thống
- Tầng dịch vụ (Service Layer): cung cấp các tính năng trong hệ thống cho các hệ thống khác thông qua các dịch vụ
- Tầng Business (Business Layer): Tầng xử lý chính với định nghĩa của các đối tượng dữ liệu và các thao tác xử lý
- Tầng dữ liệu (Data Layer): cung cấp các chức năng truy cập cơ sở dữ liệu hoặc các dịch vụ từ các hệ thống khác

Công nghệ lựa chọn cho phần mềm

- Ứng dụng di động (Mobile Applications): các phần mềm nhỏ sử dụng bởi cá nhân trên những màn hình hạn chế về kích thước.
- Ứng dụng Desktop (Desktop Applications): Sử dụng cho các thiết bị máy tính liên quan đến cá nhân hoặc các doanh nghiệp với các hệ quản lý cơ sở dữ liệu
- Ứng dụng Web (Web Applications): ứng dụng có thể chạy trên nhiều nền tảng khác nhau. Truy cập bởi nhiều người dùng cùng một thời điểm.
- Ứng dụng cung cấp dịch vụ (Service Applications): ứng dụng cung cấp các dịch vụ (các API tính toán, trích lọc, thống kê) cho các loại ứng dụng khác.

Một số công cụ hỗ trợ xây dựng phần mềm

- Quản lý mã nguồn
- Quản lý tiến độ công việc

Quản lý mã nguồn là gì

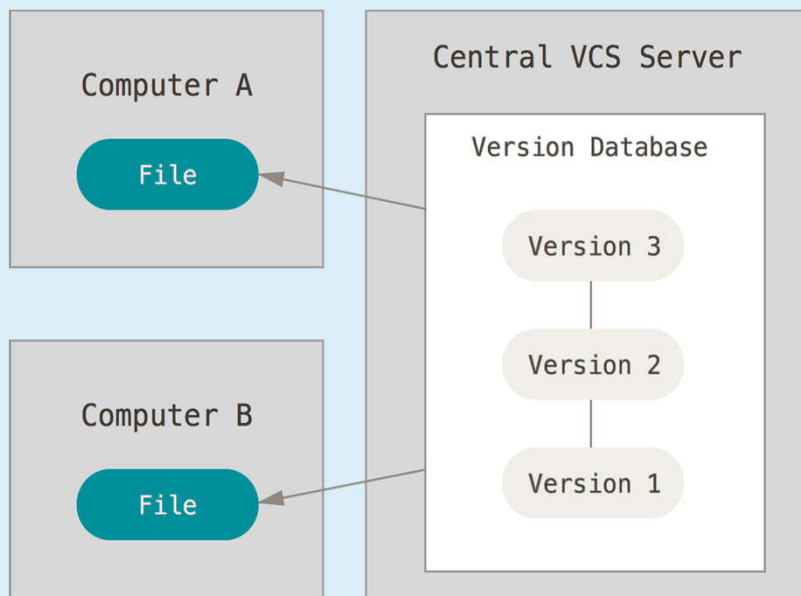
- Một số vấn đề gặp phải khi xây dựng phần mềm:
 - Ai đã chỉnh sửa tập tin X, trước đây nó hoạt động tốt không có lỗi nhưng hiện tại nó lại xuất hiện lỗi?
 - Salomé, bạn có thể giúp tôi làm việc trên tập tin X trong khi tôi làm việc trên tập tin Y? Hãy cẩn thận không can thiệp vào tập tin Y bởi vì nếu chúng ta làm việc trên cùng tập tin Y trong cùng một thời điểm có thể mã lệnh của tôi và bạn bị ghi đè lên nhau!
 - Ai đã thêm dòng mã này trong tập tin này? Nó không dùng để làm gì cả?
 - Những tập tin mới được thêm vào nhằm mục đích gì và ai đã thêm chúng vào dự án?
 - Ai đã thực hiện những thay đổi gì để giải quyết lỗi?

Quản lý mã nguồn

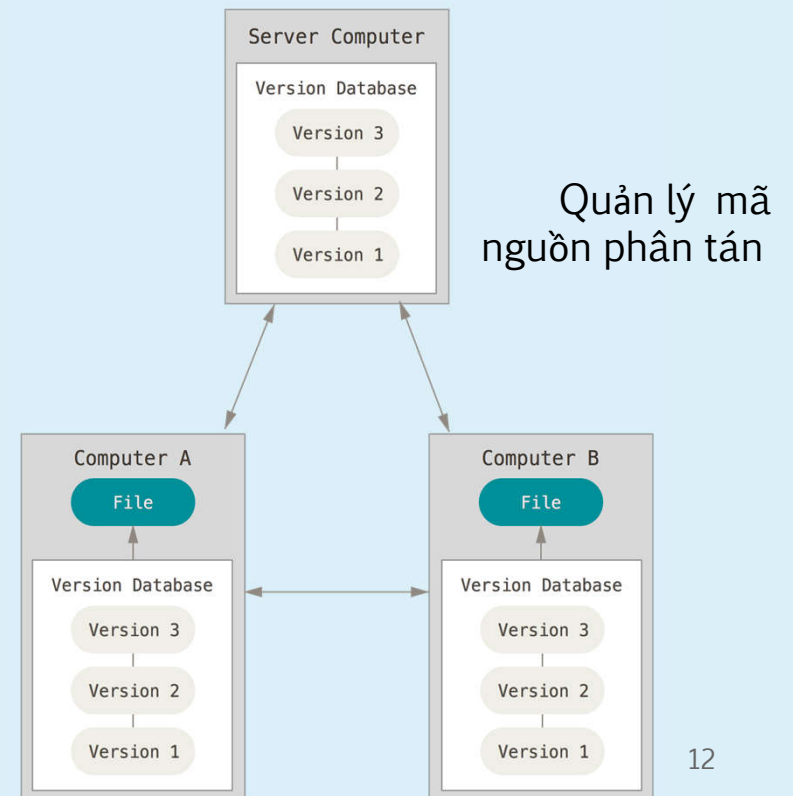
- Công cụ quản lý mã nguồn giúp các thành viên cùng làm việc với nhau trên cùng một dự án:
 - Theo dõi sự thay đổi của các tập tin
 - Lưu giữ lịch sử phiên bản các phiên bản tập tin
 - Hợp nhất các sửa đổi để tránh công việc của các thành viên khác nhau bị ghi đè hoặc trùng lặp

Quản lý mã nguồn

- Các công cụ loại này phân làm 2 nhóm: Tập trung và phân tán



Quản lý mã nguồn tập trung



Quản lý mã nguồn

Công cụ	Loại	Mô tả	Dự án sử dụng
CVS	Tập trung	Đây là ứng dụng quản lý mã nguồn lâu đời nhất. Vẫn còn được sử dụng bởi một vài dự án.	OpenBSD
SVN (Subversion)	Tập trung	Công cụ được sử dụng phổ biến hiện tại, đơn giản để sử dụng. Sử dụng trên window với Tortoise SVN, trước đây được sử dụng bởi google code.	Apache, Remine, Struts
Mercurial	Phân tán	Công cụ có các chức năng hoàn thiện và hỗ trợ mạnh mẽ, nó xuất hiện sau Git một thời gian ngắn và thường được so sánh với Git	Python, OpenOffice.org
Bazaar	Phân tán	Bazaar được phát triển bởi Canonical (nhà phát hành Ubuntu). Nó tập trung vào tính tiện dụng và tính mềm dẻo.	Ubuntu, MySQL, Inkscape
Git	Phân tán	Tạo ra bởi Linus Torvalds. Ưu điểm về tốc độ và khả năng quản lý về các nhánh của phần mềm.	Phát triển Kernal của Linux, VLC, Android

Quản lý tiến độ công việc

- Một số nguyên nhân làm cho các dự án phần mềm thường xuyên gặp phải tình trạng quá tải trong công việc đó là việc phân bổ, quản lý tiến độ công việc không phù hợp
- Mô hình được sử dụng phổ biến Kanban, được phát triển tại Nhật bản sau thế chiến II được áp dụng tại Toyota từ 1959

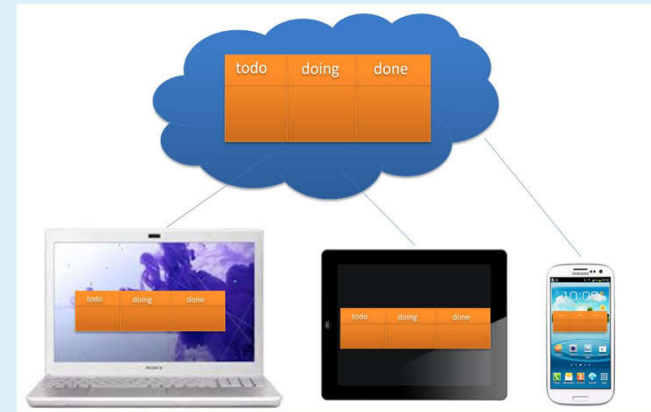


Quản lý tiến độ công việc

- Về nguyên tắc công việc thực hiện được phân chia và lên kế hoạch được đặt trên 3 cột: To do (kế hoạch), Doing – Work in process (Đang thực hiện) – Done (Hoàn thành)
- Đầu tiên, lập kế hoạch công việc trong ngày/tuần và đặt trên trạng thái to do
- Khi quyết định làm việc gì, sẽ chuyển công việc sang cột Doing và ghi thời gian lên trên từng công việc
- Khi làm xong việc gì thì chuyển sang cột Done, lưu ngày hoàn thành trên từng công việc để phục vụ công tác báo cáo và đánh giá về sau

Quản lý tiến độ công việc

- Sử dụng Kanban giúp:
 - Trực quan hóa công việc
 - Giới hạn công việc đang làm
 - Tập trung vào luồng công việc
 - Cải tiến liên tục hiệu quả công việc
- Hiện tại có nhiều công cụ cho phép sử dụng Kanban chạy trên nhiều nền tảng khác nhau từ Android, iOS, Window như: Trello, KanbanFlow, Pomodoro etc.



Chuẩn hóa mã hóa

- Việc mã hóa là công việc đòi hỏi tính logic cao và phức tạp. Một chương trình tốt phải bắt đầu từ những đoạn mã tốt, việc này đòi hỏi người lập trình mất nhiều thời gian và công sức.
- Những người mới bắt đầu viết mã thường viết theo kiểu miễn sao chương trình chạy được -> ít quan tâm đến phong cách viết mã và cách viết một chương trình tốt.
- Hậu quả thường thấy từ những việc viết mã tự do như vậy là sự lộn xộn của mã trong ứng dụng và nó có thể gây nhiều phần toái sau này ví dụ như khi muốn phải thay đổi và chỉnh sửa mã.
- Do vậy, Các nhóm phát triển luôn có những quy ước chung cho các thành viên trong nhóm để có thể dễ dàng xây dựng các chức năng của phần mềm

Chuẩn hóa mã hóa

- Các kiểu đặt tên trong .NET:

Kiểu	Ví dụ	Mô tả
Pascal	SinhVien	Chữ cái đầu tiên trong từ định danh và chữ cái đầu tiên của mỗi từ theo sau phải được viết hoa. Sử dụng Pascal Case để đặt tên cho một tên có từ 3 ký tự trở lên
Camel	sinhVien	Chữ cái đầu tiên trong từ định danh là chữ thường và chữ cái đầu tiên của mỗi từ nối theo sau phải được viết hoa
Upper case	SINHVIEN	Tất cả các ký tự trong từ định danh phải được viết hoa. Sử dụng quy tắc này đối với tên định danh có 2 ký tự trở xuống

Chuẩn hóa mã hóa

- Các trường hợp sử dụng:

Loại	Kiểu đặt tên	Ví dụ	Ghi chú
Tên dự án (Project file)	Pascal	QuanLySinhVien	
Tên file mã nguồn (Source File)	Pascal	SinhVien.cs	
Tên biến (Variable)	Camel	hoTen	
Hằng số (Constant)	Uppercase	SO_PI	Có gạch chân giữa các từ
Tên class, enum, Delegate	Pascal	SinhVien, XepLoai	
Tham số (Parameter)	Camel	tenKhachHang	
Thuộc tính (Property)	Pascal	NgaySinh	
Phương thức (method)	Pascal	TinhDiemTrungBinh()	
Sự kiện (event)	Pascal	ClickEventHandler	Thường có hậu tố EventHandler
Giao diện (Interface)	Pascal	IHinhHoc	Thường có tiền tố I

Chuẩn hóa mã hóa

- Ví dụ về quy tắt đặt tiền tố cho một số điều khiển

STT	Tên điều khiển	Tiền tố	Ví dụ
1	Panel	pnl	pnlGroup
2	CheckBox	chk	chkReadOnly
3	ComboBox, Drop-DownListBox	cbo	cboCountry
4	Command Button	btn	btnExit
5	Common dialog	dlg	dlgFileOpen
6	Data	dat	datBiblio
7	Data-Bound Combo Box	cbo	cboLanguage
8	Data-Bound grid	grd	grdQueryResult
9	Data-Bound List Box	lst	lstJobType

Xem thêm trong giáo trình về tất cả các điều khiển

Chuẩn hóa mã hóa

- Để viết comment, sự dụng ngôn ngữ quy ước (tiếng việt hoặc tiếng anh) phụ thuộc vào quy định của nhóm hoặc công ty về ngôn ngữ sử dụng.
- Comment cho module, class: mỗi module, class cần có mô tả ngắn về mục đích của module hay class đó. Nội dung gồm:
 - Mục đích: module hay class thực hiện những công việc gì
 - Người lập: người tạo module hay class
 - Những biến/hàm quan trọng (không bắt buộc): liệt kê tên các biến và hàm quan trọng trong module/class.

Chuẩn hóa mã hóa

- Comment cho method hoặc event: comment cho method/event gồm:
 - Phần 1 (không bắt buộc): mô tả mục đích và diễn giải ngắn gọn ý nghĩa các tham số đầu vào, đầu ra. Lưu ý: mô tả method đó làm gì (What), không mô tả method đó thực hiện thế nào (how). Người lập trình có thể không cần viết phần mô tả mục đích này với các method/event đơn giản.
 - Phần 2 (bắt buộc): ghi thông tin về lịch sử tạo và sử dụng method/event đó (người tạo/ngày tạo, người sửa/ngày sửa). Thông tin này bắt buộc phải có với mọi method/event.
- Comment cho đoạn code: những đoạn mã phức tạp cần có comment gắn liền bên trên để chú thích. Những đoạn mã sửa đổi (modified), bổ sung (added) hoặc xóa bỏ (removed) bởi người không phải là người tạo ra cần có comment rõ ngay tại nơi sửa đổi, bổ sung: người sửa, ngày sửa, mục đích

Chuẩn hóa mã hóa

- Ví dụ comment cho method/event đơn giản:

```
//Created by HoaiKhong - 31/05/2016:  
//Lấy danh sách sinh viên theo lớp  
//Modified by VoDanh - 01/06/2016:  
//Sửa tham số truy vào tên lớp thay vì mã lớp  
protected void DanhSachSinhVienTheoLop(string tenLop)  
{  
  
}
```

- Ví dụ comment cho method/event phức tạp:

```
/// <summary>  
/// Lấy bảng điểm sinh viên theo học kỳ  
/// </summary>  
/// <param name="maSinhVien">Mã sinh viên</param>  
/// <param name="hocKy">Mã học kỳ</param>  
/// <remarks>Nhận xét - nếu có</remarks>  
/// Created by HoaiKhong - 31/05/2016:  
/// Lấy bảng điểm sinh viên theo học kỳ  
/// Modified by VoDanh - 01/06/2016:  
/// Sửa tham số truy vấn SQL  
protected void XemBangDiem(string maSinhVien, string hocKy)  
{  
  
}
```

Chuẩn hóa mã hóa

- Quy tắt phân nhóm
 - Sử dụng region phân nhóm mã để thuận tiện cho việc sửa đổi, bảo trì. Phân nhóm mã theo cấu trúc như sau (theo thứ tự bắt buộc, nhưng không bắt buộc có đủ tất cả các region): Declaration, Constructor, Property, Method/Function, Event.
- Quy tắc xử lý biệt lệ
 - Bắt buộc bẫy lỗi bằng cách sử dụng khối lệnh try ... catch hoặc using trong tất cả các xử lý sự kiện của form và control trên form, trong các xử lý kết nối với cơ sở dữ liệu, trong việc đọc ghi thông tin vào các tập tin liên quan đến các đối tượng nhập xuất dữ liệu .vv. .
 - Không sử dụng khối lệnh try ... catch để che dấu lỗi tức là không xử lý gì sau từ khóa catch

Mẫu thiết kế là gì

- Mẫu thiết kế (design pattern) là một giải pháp tổng thể cho các vấn đề chung trong thiết kế phần mềm. một mô tả hay là mẫu (template) mô tả cách giải quyết một vấn đề mà có thể được dùng trong nhiều tình huống khác nhau.
- Các mẫu thiết kế có thể giúp tăng tốc quá trình phát triển phần mềm bằng cách cung cấp các mô hình phát triển đã được chứng thực và kiểm chứng.
- Các mẫu thiết kế cung cấp các giải pháp chung, được viết tài liệu dưới một định dạng mà không gắn liền với một vấn đề cụ thể nào.
- Việc vận dụng các mẫu thiết kế vào chức năng cụ thể của phần mềm sẽ phụ thuộc vào cách nhìn nhận và kinh nghiệm của người lập trình.

Phân loại mẫu thiết kế

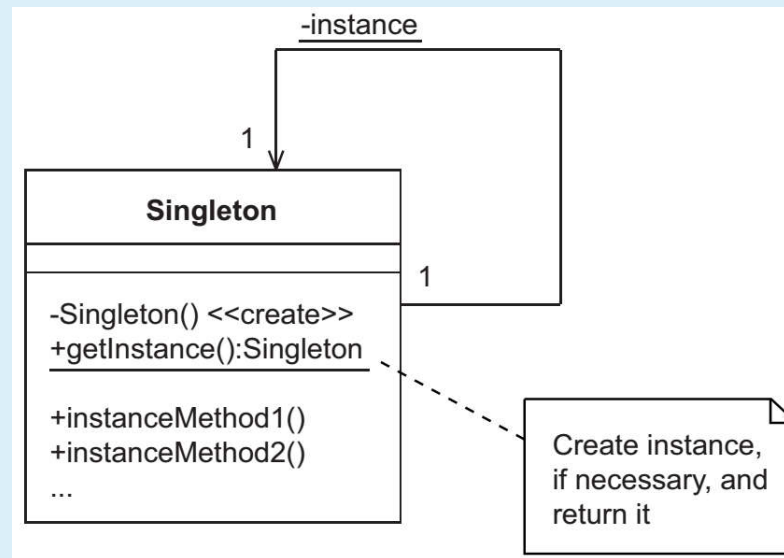
- Các mẫu cơ sở (Fundamental pattern): Interface, Abstract Parent Class, Private Method, Accessor Methods, Monitor
- Các mẫu tạo lập (Creational pattern): Factory, Singleton, Abstract, Prototype, Builder
- Các mẫu cấu trúc (Structural pattern): Decorator, Adapter, Chain of Responsibility, Façade, Proxy, Bridge, Virtual Proxy, Counting Proxy
- Các mẫu ứng xử (Behavioral pattern): Command, Mediator, Memento, Observer, Interpreter, State, Strategy, Null Object, Template Method,
- Các mẫu đồng thời (Concurrency pattern): Critical Section, Consistent Lock, Guard Suspension, Read-Write lock
- Các mẫu tập hợp (Collectional Patterns): Composite, Iterator, Flyweight

Sử dụng mẫu thiết kế

- Ví dụ mẫu thiết kế Singleton sử dụng khi một lớp yêu cầu chỉ có duy nhất một thể hiện (instance) được sử dụng xuyên suốt trong chương trình.
- Lợi ích: tiết kiệm không gian bộ nhớ sử dụng, tối ưu hóa thời gian thực thi bằng cách hạn chế việc khởi tạo và hủy đối tượng.
- Những tiêu chí được ra khi xem xét sử dụng mẫu thiết kế này:
 - Lớp sử dụng phải dễ dàng để tìm kiếm
 - Đảm bảo là một thể hiện duy nhất
 - Không được tạo ra cho đến khi thực sự cần thiết

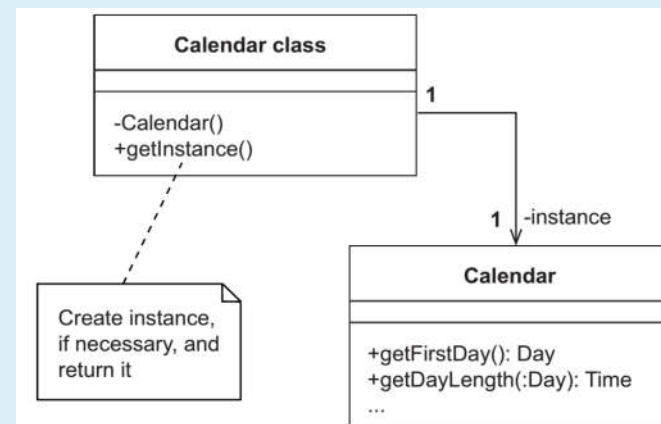
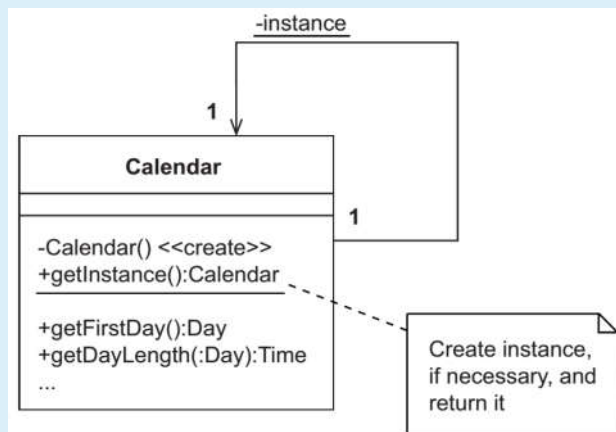
Sử dụng mẫu thiết kế

- Lớp sử dụng Singleton chỉ được khởi tạo thực sự khi cần thiết. Điều này nhằm mục đích hạn chế việc khởi tạo một đối tượng mà thực tế không bao giờ sử dụng.



Sử dụng mẫu thiết kế

- Ví dụ ta chỉ cần sử dụng một đối tượng Calendar duy nhất để trả lời cho tất cả những câu hỏi “Có bao nhiêu ngày trong tháng 2” thay cho việc tạo ra nhiều đối tượng Calendar khác nhau.
- Về lớp *Calendar* gồm các phương thức khởi tạo đặt ở chế độ private *Calendar()*. Khởi tạo đối tượng của lớp Calendar bằng cách gọi phương thức getInstance(). Việc khởi tạo và truy cập được quản lý trực tiếp bởi chính lớp đó.



Sử dụng mẫu thiết kế

```
public class Calendar
{
    private static Calendar instance; // static means "class field"
    private Calendar()
    { // Must declare a private constructor
      // Initialize any fields here
    }
    public static Calendar getInstance()
    {
        if (instance == null)
        {
            instance = new Calendar();
        }
        return instance;
    }
    // Declare any instance fields here...
    // And now for the instance methods:
    public DateTime getFirstDay() { return firstday; }
    public int getDayLength(DateTime aDay) { return aDay.Day; }

    DateTime firstday;
    int dayLength;
}
```

- Khởi tạo và sử dụng lớp Calendar như sau:

```
DateTime d = Calendar.getInstance().getFirstDay();
```

Đặc tả kỹ thuật phần mềm

- Đặc tả kỹ thuật là một phần trong sự hoàn thiện của phần mềm.
- Mô tả rõ ràng về các yêu cầu hành vi của phần mềm, ở đây có thể hiểu là toàn bộ hệ thống, một hệ thống con, một lớp hoặc một chức năng cụ thể.
- Đặc tả kỹ thuật sẽ mô tả các giới hạn của các phương thức lớp. Ví dụ: Giới hạn của một đối tượng gồm các thông điệp (messages) và các thuộc tính.

Tại sao phải đặc tả phần mềm

- Để loại bỏ sự mơ hồ còn lại sau giai đoạn phân tích.
- Để nâng cao sự am hiểu của chúng ta về công việc thực thi.
- Để tăng cường sự tin cậy vào việc hệ thống sẽ hoạt động.
- Để giúp chúng ta gỡ lỗi (debug) của phần mềm.
- Để giúp chúng ta kiểm tra (test) phần mềm.
- Để giúp sửa đổi phần mềm.
- Để người phát triển sau có thể tiếp nhận và hoàn thiện những công việc của những người phát triển khác.
- Để cung cấp tài liệu tốt hơn về phần mềm

Đặc tả phần mềm

- Về phân loại có hai loại đặc tả kỹ thuật:
 - Đặc tả chính quy (formal specification - đặc tả theo khuôn mẫu khoa học và chặt chẽ)
 - Đặc tả không chính quy (informal specification - đặc tả một cách theo thực tế và thường chỉ một phần)

Đặc tả phần mềm

- Đối với đặc tả chính quy, có một số ngôn ngữ như *Vienna Development Method* (VDM) được tạo bởi công ty IMB tại Vienna, ngôn ngữ *Z* từ đại học Oxford hoặc ngôn ngữ *Object Constraint language* (OCL) là một phần được định nghĩa trong ngôn ngữ UML.
- Ví dụ: nếu muốn viết một đặc tả chính quy cho các hoạt động của hàm tính căn bậc hai *SquareRoot()* ta có thể diễn tả về các hoạt động này như sau:
 - Nhập vào một giá trị dương
 - Bình phương của kết quả sẽ bằng giá trị nhập
 - Kết quả sẽ là số dương (căn bậc 2 của 4 có thể là 2 hoặc -2. Chúng ta lựa chọn kết quả là số dương ở phép căn)

Đặc tả phần mềm

- Đặc tả các điều kiện giới hạn của hàm *SquareRoot()* bằng ngôn ngữ VDM:
 - *SquareRoot*($x: \mathbb{R}$) $y: \mathbb{R}$
 - pre $x \geq 0$
 - post $(y^2 = x)$ and $(y \geq 0)$

Đặc tả phần mềm

- Đối với đặc tả không chính quy: việc sử dụng sẽ nằm trong một chừng mực nhất định. Ví dụ như thêm các bình luận (comment) cho một chức năng nào đó. Các comment cho các hàm thường sẽ mô tả một số hoặc tất cả các thông tin như:
 - Khi Client có thể gọi hàm
 - Khi các tham số cần được thông qua
 - Những gì hàm (function) thực hiện
 - Những loại kết quả được trả về (loại hoặc giá trị)
 - Những gì hiệu lực cho các hàm trên dữ liệu toàn cục (global data)
 - Những hành động cần có một khi có vấn đề gì đó (xử lý lỗi)

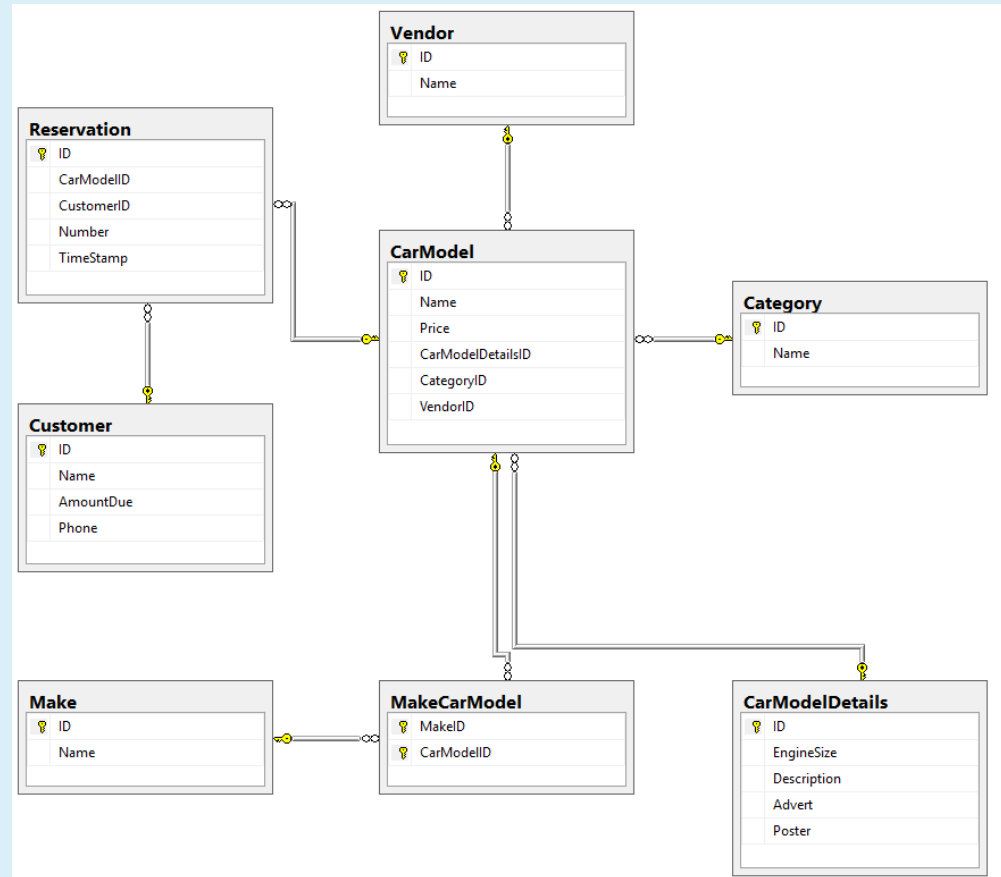
Đặc tả hướng đối tượng

- Sử dụng ngôn ngữ OCL trong UML để đặc tả cho các khái niệm về lớp, đối tượng.
- Ví dụ cho lớp Container ta có đặc tả sử dụng OCL như sau:

```
context Container::  
  add(o:Object)  
  pre: (o <> null) and !contains(o)  
  post: contains(o) and (size =size@pre +1)  
context Container::  
  inv: size >=0
```

Ví dụ minh họa

- Đến giai đoạn này mọi phân tích đã hoàn thành với sơ đồ use case, sơ đồ hoạt động, sơ đồ lớp, sơ đồ tuần tự, sơ đồ giao tiếp, các đặc tả kỹ thuật cũng như mô hình cơ sở dữ liệu của phần mềm iCoot:

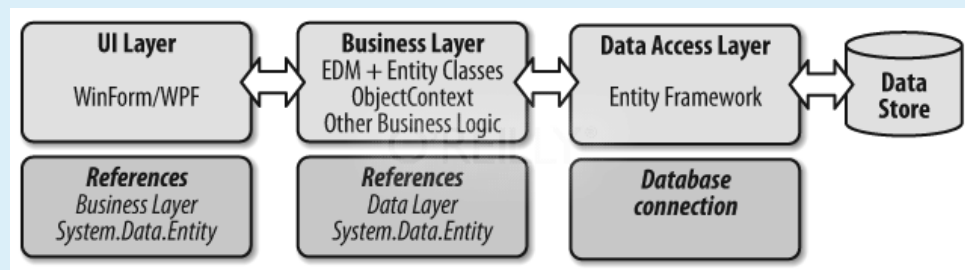


Ví dụ minh họa – Kiến trúc

- ORM Entity Framework được sử dụng cho việc thiết lập việc ánh xạ và thao tác dữ liệu.
- Đây là một nền tảng được phát triển bởi Microsoft giúp cho người phát triển làm việc các cơ sở dữ liệu như một mô hình ở tầng Business, có thể hiểu ở đây tầng truy cập dữ liệu bằng ADO.NET sẽ được tự động sinh ra khi ta sử dụng ORM này.
- Ưu điểm nổi trội là sẽ giảm thiểu khối lượng mã hóa mà người phát triển phải thực hiện khi xây dựng tầng truy cập dữ liệu.
- Phiên bản đầu tiên của Entity Framework ra đời vào năm 2008 dưới nền tảng .NET Framework 3.5 cho đến hiện nay là phiên bản 6.0.

Ví dụ minh họa – Kiến trúc

- Kiến trúc phần mềm khi sử dụng Entity Framework:

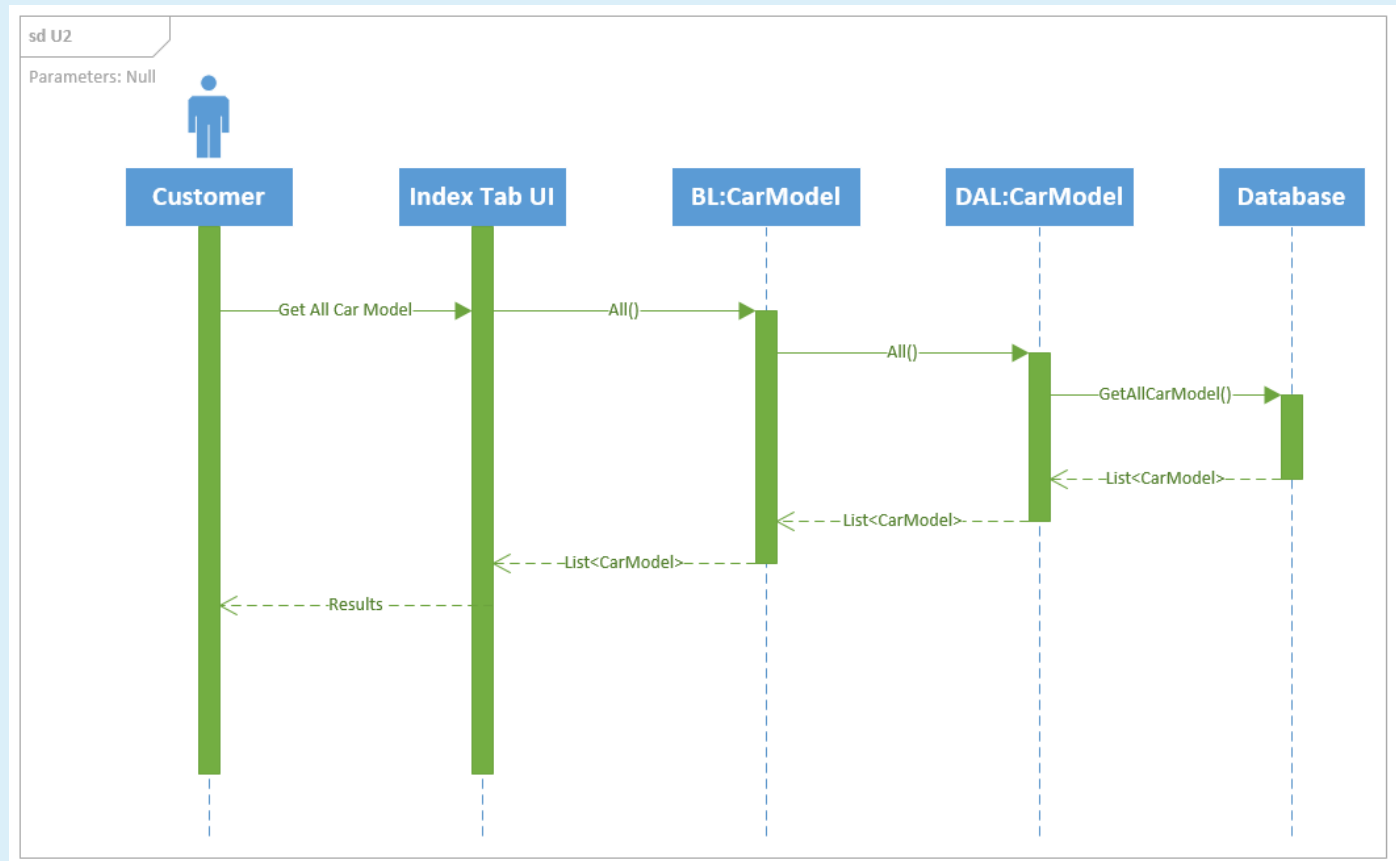


Ví dụ minh họa – Use case 02

- Use case 02: Xem danh sách mẫu xe
 - Preconditions: Không
 - Hệ thống trình bày cho khách hàng với các mẫu xe
 - Mở rộng sang use case 03
 - Postconditions: Không

Ví dụ minh họa – Use case 02

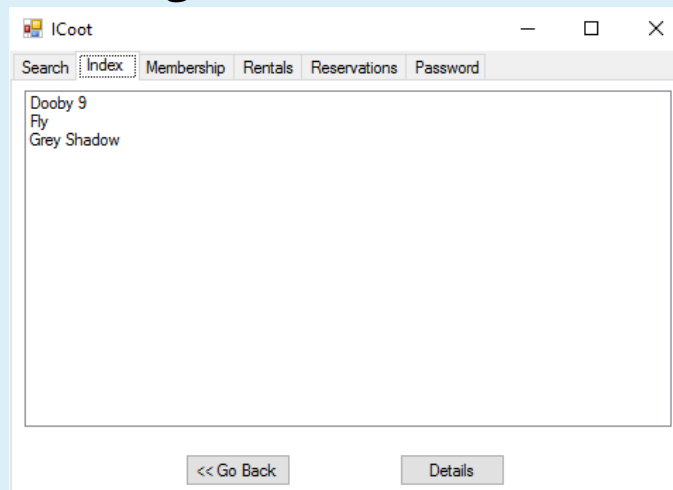
- Sơ đồ tuần tự:



Ví dụ minh họa – Use case 02

- Trong use case này, danh sách mẫu xe sẽ được tải lên khi người dùng chọn vào Tab Index trên giao diện chính. Dữ liệu đơn thuần được lưu trữ trên bảng CarModel. Do vậy, cần truy cập đối tượng CarModel ở các tầng Business và tầng truy cập dữ liệu để lấy dữ liệu.
- Mã thực thi ở tầng giao diện:
- Giao diện tương tác:

```
private void LoadData()
{
    CarModel carModel = new CarModel();
    List<CarModel> all = carModel.All();
    foreach (CarModel i in all)
        lbIndex.Items.Add(i.Name);
}
```

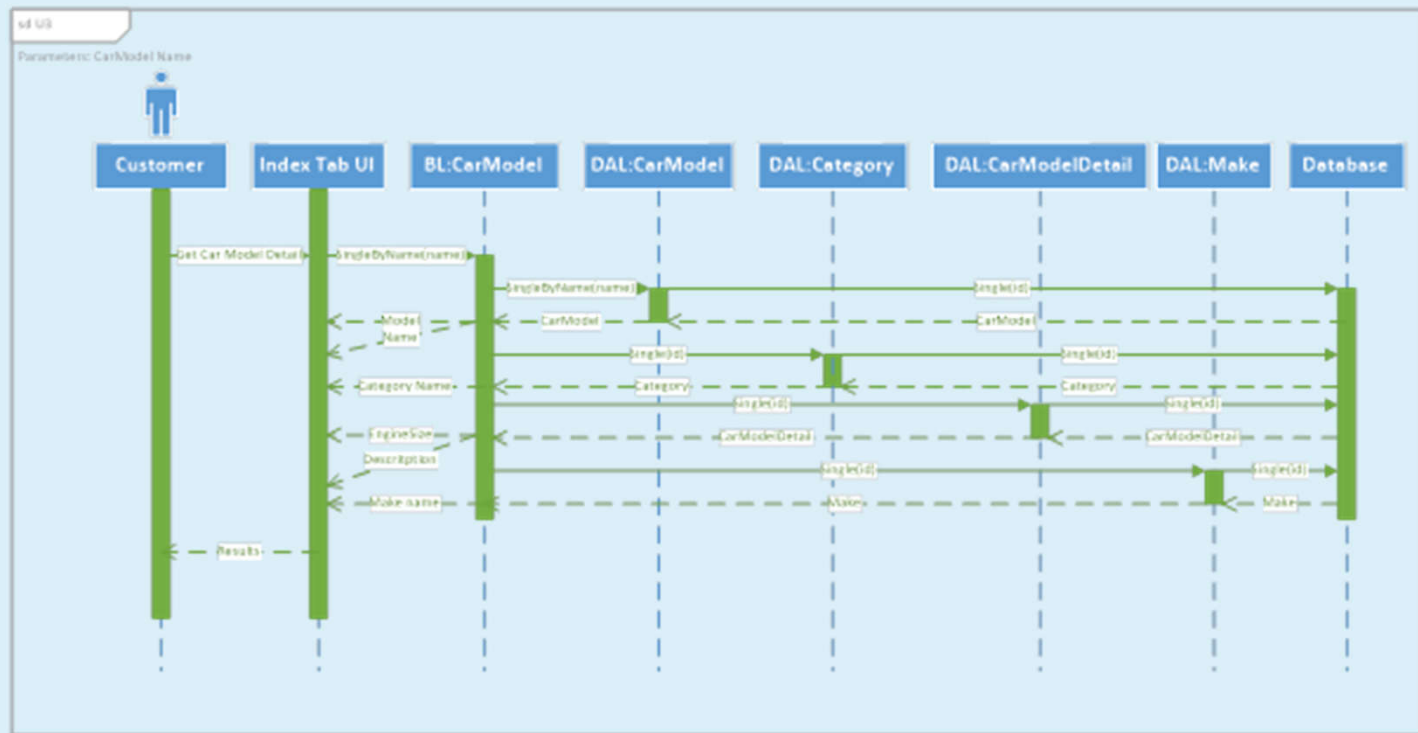


Ví dụ minh họa – Use case 03

- Use case 03: Xem chi tiết mẫu xe
 - Preconditions: không
 - Khách hàng chọn một trong các mẫu xe
 - Khách hàng yêu cầu xem chi tiết
 - Hệ thống hiển thị thông tin chi tiết
 - Postconditions: thông tin chi tiết của mẫu xe

Ví dụ minh họa – Use case 03

- Sơ đồ tuần tự:



Ví dụ minh họa – Use case 03

- Trong use case này, thông tin chi tiết của mẫu xe sẽ được lấy từ nhiều bảng dữ liệu khác nhau. Ở đây thông tin về tên danh mục lấy từ bảng danh mục, thông tin về kích thước máy và mô tả được lấy từ bảng chi tiết mẫu xe và thông tin về nhà sản xuất được lấy từ bảng Make.
- Mã ở tầng Business trong lớp CarModel

```
public CarModel SingleByName(string name)
{
    using (iCootDBEntities db = new iCootDBEntities())
    {
        CarModel carModel = db.CarModels.Where(p => p.Name == name).FirstOrDefault();
        Category category = new Category();
        category = category.Single(carModel.CategoryID.Value);
        carModel.Category = category;

        CarModelDetail carModelDetail = new CarModelDetail();
        carModelDetail = carModelDetail.Single(carModel.CarModelDetailsID.Value);
        carModel.CarModelDetail = carModelDetail;

        Make make = new Make();
        make = make.Find(carModel.ID);
        carModel.Makes.Add(make);
        return carModel;
    }
}
```

Ví dụ minh họa – Use case 03

- Mã thực thi ở tầng ứng
 - Giao diện tương tác người dùng:
- dùng:

```
private void btDetails_Click(object sender, EventArgs e)
{
    if (lbIndex.SelectedItem != null)
    {
        String name = lbIndex.SelectedItem.ToString();
        CarModel carModel = new CarModel();

        carModel = carModel.SingleByName(name);

        tbDailyPrice.Text = carModel.Price.ToString();
        tbModel.Text = carModel.Name;
        tbCategory.Text = carModel.Category.Name;
        tbEngineSize.Text = carModel.CarModelDetail.EngineSize.ToString();
        tbDescription.Text = carModel.CarModelDetail.Description;
        tbModelID.Text = carModel.ID.ToString();
        tbMakes.Text = carModel.Makes.FirstOrDefault().Name;

        pnDetails.Show();
        pnList.Hide();
        pnDetails.Refresh();
    }
}
```

The screenshot shows a window titled 'ICoot' with a tabbed interface. The 'Index' tab is active, displaying a search form with the following fields and values:

Field	Value
Category	Sedan
Make(s)	Astra Marten
Model	14
Engine Size	3000
Description	Power
Daily Price	35000

At the bottom of the form, there are four buttons: '<< Go Back', 'Reserve', 'Advert', and 'Poster'.



THẢO LUẬN





Question?