NGUYỄN XUÂN BÁCH

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

UỶ BAN NHÂN DÂN TỈNH QUẢNG NINH
TRƯỜNG ĐẠI HỌC HẠ LONG

NGUYỄN XUÂN BÁCH

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

(Tài liệu lưu hành nội bộ)

Dùng cho ngành đào tạo: Khoa học máy tính

Quảng Ninh, 2021

NGUYỄN XUÂN BÁCH

--ᏒᏨ--

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

(Tài liệu lưu hành nội bộ)

Dùng cho ngành đào tạo: Khoa học máy tính

Quảng Ninh, năm 2021

MỤC LỤC

Lời nói đầu 5

BÀI 1. TỔNG QUAN VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG 7

1.1. Lịch sử phát triển các phương pháp lập trình 7

1.1.1. Lập trình tuyến tính 7

1.1.2. Lập trình hướng cấu trúc 7

1.1.3. Lập trình hướng đối tượng 8

1.2. Phương pháp lập trình hướng đối tượng 9

1.2.1. Khái niệm 9

1.2.2. Các thuật ngữ cơ bản 9

1.2.3. Ưu điểm của lập trình hướng đối tượng 10

1.3. Các tính chất cơ bản của lập trình hướng đối tượng 11

Câu hỏi và bài tập 13

BÀI 2. CĂN BẢN VỀ NGÔN NGỮ LẬP TRÌNH C# 13

2.1. Các ngôn ngữ lập trình hướng đối tượng hiện nay 13

2.2. Ngôn ngữ lập trình hướng đối tượng c# 15

2.2.1. Cấu trúc dự án c# 15

2.2.2. Câu lệnh trong c# 16

2.2.3. Khối lệnh trong c# 17

2.2.4. Định danh 19

2.2.5. Lớp program và phương thức main() 20

2.2.6. Khai báo biến trong c# 21

2.2.7. Hằng trong c# 23

2.2.8. Các kiểu dữ liệu cơ sở của c# 24

2.2.9. Toán tử trong c# 26

2.2.10. Các cấu trúc điều khiển trong c# 27

2.2.11. Quản lí lỗi và ngoại lệ 32

2.2.12. Truyền tham số 37

Câu hỏi và bài tập 40

BÀI 3. LỚP VÀ ĐỐI TƯỢNG 41
3.1. Lớp 41
3.2. Đối tượng 43
3.3. Đóng gói dữ liệu 44
Câu hỏi và bài tập 48
BÀI 4. PHƯƠNG THỨC KHỞI TẠO, HỦY TẠO, CÁC THÀNH PHẦN TĨNH .. 49
4.1. Phương thức khởi tạo 49
4.2. Phương thức hủy tạo 52
4.3. Các thành phần tĩnh 53
Câu hỏi và bài tập 57
BÀI 5. THỰC HÀNH 1 59
BÀI 6. KẾ THỪA 67
6.1. Các kiểu kế thừa 67
6.2. Các kĩ thuật trong kế thừa 68
6.2.1. Kế thừa phương thức khởi tạo, phương thức hủy 68
6.2.2. Hàm trùng tên và cách gọi phương thức của lớp cha 70
6.2.3. Cấp phát vùng nhớ cho đối tượng 71
Câu hỏi và bài tập 70
BÀI 7. THỰC HÀNH 2 73
BÀI 8. LỚP TRỪU TƯỢNG, LỚP KHÔNG CHO PHÉP KẾ THỪA 83
8.1. Lớp trừu tượng 83
8.2. Ghi đè phương thức 86
8.3. Lớp không cho phép kế thừa 89
8.4. Phương thức không cho phép ghi đè 90
Câu hỏi và bài tập 89
BÀI 9. THỰC HÀNH 3 93
BÀI 10. TÍNH ĐA HÌNH 103
10.1. Khái niệm 103
10.2. Đa hình tĩnh 103

10.2.1. Nạp chồng hàm trong c# 103
10.2.2. Nạp chồng toán tử..105
10.3. Đa hình động 111
Câu hỏi và bài tập.. 110
BÀI 11. THỰC HÀNH 4 115
BÀI 12. GIAO DIỆN VÀ ĐA KẾ THỪA 125
12.1. Giao diện 125
12.2. Đa kế thừa 127
12.3. So sánh lớp trừu tượng và giao diện..129
Câu hỏi và bài tập.. 127
BÀI 13. THỰC HÀNH 5 133
BÀI 14: BỘ LẬP CHỈ MỤC, CƠ CHẾ ỦY QUYỀN, SỰ KIỆN..141
14.1. Bộ lập chỉ mục 141
14.2. Cơ chế ủy quyền..142
14.3. Sự kiện 144
Câu hỏi và bài tập.. 142
BÀI 15. THỰC HÀNH 6 149
TÀI LIỆU THAM KHẢO 157

LỜI NÓI ĐẦU

MỤC ĐÍCH

Tài liệu ''Lập trình hướng đối tượng'' gồm những nội dung nhằm cung cấp các kiến thức nền tảng về phương pháp lập trình hướng đối tượng, bổ sung tài liệu dạy học cho học phần Lập trình Cơ sở dữ liệu, Lập trình Web, Lập trình trên thiết bị di động và sử dụng ngôn ngữ lập trình C# để cài đặt chương trình.

CẤU TRÚC

Tài liệu được biên soạn thành 15 bài. Nội dung từng bài như sau :

Bài 1: Tổng quan về lập trình hướng đối tượng.

Bài 2: Căn bản về ngôn ngữ lập trình C#.

Bài 3: Lớp và đối tượng trong C#.

Bài 4: Phương thức khởi tạo, hủy tạo, các thành phần tĩnh.

Bài 5: Thực hành 1.

Bài 6: Kế thừa. Trình bày các kiểu, kĩ thuật trong kế thừa

Bài 7: Thực hành 2.

Bài 8: Lớp trừu tượng, lớp không cho phép kế thừa.

Bài 9: Thực hành 3.

Bài 10: Tính đa hình.

Bài 11: Thực hành 4.

Bài 12: Giao diện và đa kế thừa.

Bài 13: Thực hành 5.

Bài 14: Bộ lập chỉ mục, cơ chế ủy quyền, sự kiện.

Bài 15. Thực hành 6.

YÊU CẦU ĐỐI VỚI NGƯỜI HỌC

Khi tham gia học học phần ''Lập trình hướng đối tượng'', người học cần đạt được các yêu cầu sau :

Phân biệt phương pháp lập trình hướng đối tượng với các phương pháp lập trình khác.

Nắm vững các kiến thức về lớp, đối tượng, tính đóng gói, kế thừa, đa hình, giao diện.

Sử dụng ngôn ngữ lập trình C# để phát triển chương trình theo phương pháp hướng đối tượng.

CÁCH TỰ HỌC ĐỐI VỚI CUỐN TÀI LIỆU NÀY

- Khi xem xét một vấn đề, người học phải xuất phát từ định nghĩa, khái niệm và đặt vấn đề đó trong mối liên hệ với các vấn đề khác.

- Để tạo được sự hứng thú người học nên bắt đầu từ dễ đến khó, từ đơn giản đến phức tạp, từ xa đến gần, từ cụ thể đến khái quát, trừu tượng.

- Người học phải rèn luyện kỹ năng đọc và nghiên cứu tài liệu. Ngoài ra cần tìm hiểu địa chỉ các cuốn tài liệu khác liên quan để tham khảo.

THÔNG TIN LIÊN HỆ

Nguyễn Xuân Bách; ĐT: 0987735788; Email: NguyenXuanBach@daihochalong.edu.vn

Chủ biên

Nguyễn Xuân Bách

BÀI 1 TỔNG QUAN VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

NỘI DUNG CHÍNH

- Lịch sử phát triển các phương pháp lập trình
- Khái niệm, thuật ngữ trong lập trình hướng đối tượng
- Các tính chất nền tảng của lập trình hướng đối tượng
- Mô hình hóa một vấn đề dưới dạng đối tượng

1.1. LỊCH SỬ PHÁT TRIỂN CÁC PHƯƠNG PHÁP LẬP TRÌNH

1.1.1. Lập trình tuyến tính

Lập trình tuyến tính là một phương pháp, kỹ thuật lập trình truyền thống. Chương trình sẽ chỉ có một hàm và được thực hiện tuần tự từ đầu tới cuối.

- Ưu điểm :
    - Chương trình đơn giản, dễ hiểu
- Nhược điểm:
    - Mọi dữ liệu trong chương trình đều là biến toàn cục do đó dễ dàng bị thay đổi bởi một phần nào đó trong chương trình.
    - Khó khăn trong việc gỡ lỗi chương trình.
    - Để thực hiện lại một đoạn lệnh thì bắt buộc phải sao chép (copy) đoạn lệnh đó đến vị trí trong chương trình mà chúng ta muốn thực hiện.

1.1.2. Lập trình hướng cấu trúc

Lập trình hướng cấu trúc là một kỹ thuật lập trình truyền thống, trong đó chương trình được chia thành các hàm (chương trình con).

Các đặc điểm chính của phương pháp lập trình hướng cấu trúc :

- Tập chung vào công việc cần thực hiện (thuật toán)
- Chương trình lớn được chia thành các chương trình con, mỗi chương trình con có thể gọi tới một hoặc nhiều lần theo thứ tự bất kỳ.
- Phần lớn các hàm sử dụng dữ liệu chung.
- Dữ liệu trong hệ thống được chuyển từ hàm này sang hàm khác.
- Hàm biến đổi dữ liệu từ dạng này sang dạng khác.
- Sử dụng cách tiếp cận từ trên xuống (top-down) trong thiết kế chương trình

1.1.3. Lập trình hướng đối tượng

Lập trình hướng đối tượng cho phép chúng ta phân tích các bài toán thành các thực thể được gọi là các đối tượng và xây dựng các dữ liệu, các hàm xung quanh các đối tượng này. Dữ liệu được liên kết với các hàm thành các vùng riêng mà chỉ có các hàm đó tác động lên, các hàm bên ngoài không được truy cập vào. Các đối tượng có thể tác động và trao đổi thông tin với nhau thông qua các thông điệp.

Các đặc điểm của phương pháp lập trình hướng đối tượng :

- Tập trung vào dữ liệu thay cho các hàm
- Chương trình được chia thành các đối tượng
- Cấu trúc dữ liệu được thiết kế sao cho đặc tả được đối tượng gắn với cấu trúc dữ liệu đó.
- Dữ liệu được đóng gói lại, được che giấu và không cho phép các hàm ngoại lai truy nhập tự do.
- Các đối tượng tác động và trao đổi thông tin với nhau qua các hàm.
- Có thể dễ dàng bổ sung dữ liệu và các hàm mới vào đối tượng nào đó khi cần thiết.
- Chương trình được thiết kế theo cách tiếp cận từ dưới lên (bottom - up).

1.2. PHƯƠNG PHÁP LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

1.2.1. Khái niệm

Lập trình hướng đối tượng là cách lập trình mà lập trình viên không chỉ định nghĩa cấu trúc kiểu dữ liệu mà còn khai báo ra các phương thức tính toán để áp dụng cho cấu trúc dữ liệu đó.

Bằng cách này, cấu trúc kiểu dữ liệu đó trở thành một đối tượng bao gồm dữ liệu và các phương thức thực thi. Nói cách khác, lập trình viên tạo ra quan hệ giữa những đối tượng với nhau và có thể thừa kế lại các đối tượng khác.

Khi viết chương trình theo phương pháp hướng đối tượng ta phải trả lời các câu hỏi:

Chương trình liên quan tới những lớp đối tượng (*khái niệm của mục 1.2.2/b*) nào?

Mỗi đối tượng cần có những dữ liệu và thao tác nào?

Các đối tượng quan hệ với nhau như thế nào trong chương trình?

Từ đó ta thiết kế các lớp đối tượng và tổ chức trao đổi thông tin giữa các đối tượng, ra lệnh để đối tượng thực hiện các nhiệm vụ thích hợp.

Ví dụ 1.1 :

Với đối tượng chuỗi kí tự, chúng ta có các thông tin sau :

- Dữ liệu: mảng các kí tự.
- Thao tác: tính chiều dài, nối hai chuỗi...

Với đối tượng *stack* :

- Dữ liệu: số nguyên hay kí tự , hay một kiểu dữ liệu đã định nghĩa.
- Thao tác: khởi tạo *stack*, đưa một phần tử vào đỉnh, loại bỏ phần tử ở đỉnh…

1.2.2. Các thuật ngữ cơ bản

a. Đối tượng (Object)

Đối tượng là sự kết hợp giữa dữ liệu và thủ tục thao tác trên dữ liệu đó.

Đối tượng = Dữ liệu + Phương thức

Trong thế giới hiện nay, đối tượng là những thực thể tồn tại có trạng thái và hành vi.

Ví dụ 1.2: Quyển vở, cái bút, sinh viên, học sinh là những đối tượng

Đối tượng trong lập trình hướng đối tượng bao gồm 2 thành phần chính:

- Thuộc tính (*Attribute*): là những thông tin, đặc điểm của đối tượng
- Phương thức (*Method*): là những hành vi mà đối tượng có thể thực hiện

Để dễ hình dung, ta có một ví dụ thực tế về đối tượng là *smartphone*. Đối tượng này sẽ có:

- Thuộc tính: màu sắc, bộ nhớ, hệ điều hành…
- Phương thức: gọi điện, chụp ảnh, nhắn tin, ghi âm…

Tiếp cận hướng đối tượng, đây là kĩ thuật cho phép biểu diễn tự nhiên các đối tượng thực tế với đối tượng bên trong chương trình.

b. Lớp đối tượng (Class)

Những thuộc tính và những hành động chung của một thực thể được nhóm lại để tạo nên một đơn vị duy nhất gọi là một lớp (*class*).

Một lớp là một mô hình khái niệm về một thực thể. Nó mang tính cách tổng quát chứ không mang tính cách đặc thù. Khi định nghĩa một lớp, chúng ta muốn phát biểu rằng một lớp sẽ phải có một tập hợp các thuộc tính và các hành động riêng.

Tiếp nối ví dụ ở phần đối tượng (object) phía trên, ta có lớp (*class*) smartphone gồm 2 thành phần:

Thuộc tính: màu sắc, bộ nhớ, hệ điều hành…

Phương thức: gọi điện, chụp ảnh, nhắn tin, ghi âm…

Các đối tượng của lớp này có thể là: iPhone, Samsung, Oppo, Huawei…

1.2.3. Ưu điểm của lập trình hướng đối tượng

- Tính đóng gói làm giới hạn phạm vi sử dụng của các biến, nhờ đó việcquản lý giá trị của biến dễ dàng hơn, việc sử dụng mã an toàn hơn.

- Phương pháp này làm cho tốc độ phát triển các chương trình mới nhanh hơn vì mã được tái sử dụng và cải tiến dễ dàng.

- Phương pháp này tiến hành tiến trình phân tích, thiết kế chương trình thông qua việc xây dựng các đối tượng có sự tương hợp với các đối tuợng thực tế. Điều này làm cho việc sửa đổi dễ dàng hơn khi cần thay đổi chương trình.

1.3. CÁC TÍNH CHẤT CƠ BẢN CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

a. Tính đóng gói (Encapsulation)

Tính đóng gói cho phép che giấu thông tin và những tính chất xử lý bên trong của đối tượng. Các đối tượng khác không thể tác động trực tiếp đến dữ liệu bên trong và làm thay đổi trạng thái của đối tượng mà bắt buộc phải thông qua các phương thức công khai do đối tượng đó cung cấp.

Tính chất này giúp tăng tính bảo mật cho đối tượng và tránh tình trạng dữ liệu bị hư hỏng ngoài ý muốn.

b. Tính kế thừa (Inheritance)

Đây là tính chất được sử dụng khá nhiều. Tính kế thừa cho phép xây dựng một lớp mới (lớp Con), kế thừa và tái sử dụng các thuộc tính, phương thức dựa trên lớp cũ (lớp Cha) đã có trước đó.

Các lớp Con kế thừa toàn bộ thành phần của lớp Cha và không cần phải định nghĩa lại. Lớp Con có thể mở rộng các thành phần kế thừa hoặc bổ sung những thành phần mới.

Ví dụ 1.3:

Lớp Cha là smartphone, có các thuộc tính: màu sắc, bộ nhớ, hệ điều hành…

Các lớp Con là iPhone, Samsung, Oppo cũng có các thuộc tính: màu sắc, bộ nhớ, hệ điều hành…

c. Tính đa hình (Polymorphism)

Tính đa hình trong lập trình lập trình hướng đối tượng cho phép các đối tượng khác nhau thực thi chức năng giống nhau theo những cách khác nhau.

Ví dụ 1.4:

Ở lớp smartphone, mỗi một dòng máy đều kế thừa các thành phần của lớp cha nhưng iPhone chạy trên hệ điều hành iOS, còn Samsung lại chạy trên hệ điều hành Android.

d. Tính trừu tượng (Abstraction)

Tính trừu tượng là một tiến trình ẩn các chi tiết trình triển khai và chỉ hiển thị tính năng tới người dùng. Tính trừu tượng cho phép bạn loại bỏ tính chất phức tạp của đối tượng bằng cách chỉ đưa ra các thuộc tính và phương thức cần thiết của đối tượng trong lập trình.

Ví dụ 1.5: Để gửi tin nhắn, chúng ta sử dụng điện thoại, soạn thảo nội dung rồi nhấn gửi đi. Còn quy trình xử lý tin nhắn gửi như thế nào thì ta chưa đề cập đến …

Như vậy, tính trừu tượng là che giấu thông tin thực hiện từ người dùng, họ chỉ biết tính năng được cung cấp: chỉ biết thông tin đối tượng thay vì cách nó sử dụng như thế nào.

CÂU HỎI VÀ BÀI TẬP

Câu 1: Trình bày lịch sử phát triển của các phương pháp lập trình.

Câu 2: Nêu các đặc điểm của lập trình hướng đối tượng, so sánh sự khác nhau với các phương pháp lập trình khác.

TÀI LIỆU THAM KHẢO BÀI 1

[1] P*rogramming language evolution*, 2017, https://herbertograca.com/2017/07/10/programming-language-evolution/

BÀI 2 CĂN BẢN VỀ NGÔN NGỮ LẬP TRÌNH C#

NỘI DUNG CHÍNH

- Giới thiệu các ngôn ngữ lập trình hướng đối tượng
- Khái niệm, thuật ngữ, cú pháp câu lệnh trong ngôn ngữ lập trình C#

2.1. CÁC NGÔN NGỮ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG HIỆN NAY

2.1.1. Java

Java là một trong những ngôn ngữ lập trình hướng đối tượng. Nó được sử dụng trong phát triển phần mềm, trang web, trò chơi hay ứng dụng trên các thiết bị di động.

Java được khởi đầu bởi James Gosling và đồng nghiệp ở công ty Sun MicroSystem năm 1991. Ban đầu Java được tạo ra nhằm mục đích viết phần mềm cho các sản phẩm gia dụng, và có tên là *Oak*.

Java được phát hành năm 1994, đến năm 2010 được công ty Oracle mua lại từ công ty Sun MicroSystem.

Java được tạo ra với tiêu chí "Viết (code) một lần, thực thi khắp nơi" (Write Once, Run Anywhere – WORA). Chương trình phần mềm viết bằng Java có thể chạy trên mọi nền tảng (*platform*) khác nhau thông qua một môi trường thực thi với điều kiện có môi trường thực thi thích hợp hỗ trợ nền tảng đó.

2.1.2. Python

Python là ngôn ngữ lập trình cấp cao, đa mục đích được sử dụng rộng rãi. Ban đầu được thiết kế bởi Guido van Rossum vào năm 1991 và được duy trì, phát triển bởi tổ chức Python Software Foundation.

Python là ngôn ngữ có hình thức rất trong sáng, cấu trúc rõ ràng, rất thuận tiện cho người mới học.

2.1.3. C++

Ngôn ngữ C++ được Bjarne Stroustrup phát triển từ ngôn ngữ C từ cuối thập niên 1970.

C++ là một phiên bản mở rộng của ngôn ngữ C, kết hợp tất cả các tính năng đã có của C.

C++ được coi như là ngôn ngữ bậc trung (middle-level), kết hợp các đặc điểm và tính năng của ngôn ngữ bậc cao và bậc thấp.

C++ có thể dùng để lập trình nhúng, lập trình hệ thống, hoặc những ứng dụng, game…

2.1.4. PHP

PHP **(**Hypertext Preprocessor) là một ngôn ngữ lập trình kịch bản được chạy ở phía máy chủ (*server)* nhằm sinh ra mã html trên máy khách (*client*). PHP đã trải qua rất nhiều phiên bản và được tối ưu hóa cho các ứng dụng web, với cách viết mã rõ ràng, tốc độ nhanh, dễ học nên PHP đã trở thành một ngôn ngữ lập trình web rất phổ biến và được ưa chuộng.

PHP chạy trên môi trường Webserver và lưu trữ dữ liệu thông qua hệ quản trị cơ sở dữ liệu nên PHP thường đi kèm với Apache, MySQL và hệ điều hành Linux (LAMP).

Apache là một phần mềm web server có nhiệm vụ tiếp nhận yêu cầu từ trình duyệt người dùng sau đó chuyển giao cho PHP xử lý và gửi trả lại cho trình duyệt.

MySQL cũng tương tự như các hệ quản trị cơ sở dữ liệu khác (Postgress, Oracle, SQL server...) đóng vai trò là nơi lưu trữ và truy vấn dữ liệu.

Linux: Hệ điều hành mã nguồn mở được sử dụng rất rộng rãi cho các webserver. Thông thường các phiên bản được sử dụng nhiều nhất là RedHat Enterprise Linux, Ubuntu...

2.1.5. JavaScript

Ngôn ngữ lập trình Javascript được giới thiệu đầu tiên vào năm 1995. Mục đích là để đưa những chương trình vào trang web ở trình duyệt Netscape Navigator - một trình duyệt web phổ biến những năm 1990.

JavaScript được phát triển bởi Brendan Eich tại Hãng truyền thông Netscape với cái tên đầu tiên là Mocha, rồi sau đó đổi tên thành LiveScript, và cuối cùng thành JavaScript

2.2. NGÔN NGỮ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG C#

C# (hay C sharp) là một ngôn ngữ lập trình đơn giản, được phát triển bởi đội ngũ kỹ sư của Microsoft vào năm 2000. C# là ngôn ngữ lập trình hiện đại, hướng đối tượng và được xây dựng trên nền tảng của hai ngôn ngữ mạnh nhất là C++ và Java.

2.2.1. Cấu trúc dự án C#

Solution và *project* là hai cấp độ quản lý file mã nguồn của C# và các thành phần hỗ trợ khác. Trong mỗi file mã nguồn, *namespace* là cấp độ quản lý code cao nhất. Tất cả những khái niệm này và cách làm việc với chúng đóng vai trò xương sống để có thể làm việc với C#.

C# quản lý mã nguồn theo cấu trúc cây gần giống với cấu trúc thư mục và bao gồm hai cấp độ cơ bản: Project và Solution.

Project (dự án) là cấp độ quản lý mã nguồn quan trọng nhất của C#. Mỗi project sau khi biên dịch sẽ tạo ra một chương trình.

Mỗi project mặc định đều chứa:

- Các file mã nguồn: là các file văn bản có phần mở rộng .cs (viết tắt của C sharp);
- Các file cấu hình của chương trình: là file xml có phần mở rộng .config;
- Các thư viện được tham chiếu tới (References): là danh sách các file thư viện chuẩn của .NET framework, hoặc thư viện từ các hãng thứ ba, hoặc chính các project khác, chứa các class được sử dụng bởi các class trong project này.
- Các thuộc tính (Properties): bao gồm nhiều loại thông tin khác nhau quyết định những tính chất quan trọng của project, như phiên bản của .NET framework được sử dụng, loại chương trình mà dự án này sẽ được dịch thành, các tài nguyên được sử dụng trong project, cấu hình của ứng dụng, v.v.. Visual Studio cung cấp giao diện đồ họa để có thể dễ dàng quản lý các thông tin này. Giao diện này mở ra khi vào mục Properties của project.

Tất cả các thành phần của một project đều đặt chung trong một thư mục cùng tên với project.

giải pháp (*solution*) là cấp độ quản lý mã nguồn cao nhất trong C# cho phép quản lý tập trung nhiều *project*.

Mỗi *solution* trong C# có thể chứa nhiều *project*. Nếu *solution* không chứa *project* nào, nó gọi là *Empty Solution*. Tại mỗi thời điểm Visual Studio chỉ có thể mở một solution.

Trên giao diện Visual Studio, solution và các project của nó được hiển thị trong một cửa sổ riêng gọi là Solution Explorer. Cửa sổ này hiển thị tất cả các thành phần trong dự án C# theo cấu trúc cây, với solution làm gốc, các project là các nhánh trực tiếp xuất phát từ gốc này.

Ngoài ra, để tiện lợi trong việc quản lý các project thành viên, solution cho phép tạo thêm các thư mục, gọi là *Solution Folder*, trong đó lại có thể chứa các *project* khác. Cấu trúc quản lý này cho phép quản lý một số lượng lớn *project* một cách dễ dàng. Các *project* trong cùng một solution thường có quan hệ nhất định với nhau.

2.2.2. Câu lệnh trong C#

Bên trong phương thức Main chúng ta đã viết ba **câu lệnh** (statement), hai lệnh viết ra màn hình và một lệnh đọc từ bàn phím.

```
Console.WriteLine("Hello world from C#");
Console.WriteLine("Press any key to quit");
Console.ReadKey();
```

Câu lệnh là một hành động chúng ta yêu cầu chương trình thực hiện, ví dụ: khai báo biến, gán giá trị, gọi phương thức, duyệt danh sách, xử lý theo điều kiện giá trị, v.v..

Trình tự thực hiện các lệnh được gọi là luồng điều khiển (*flow of control*) hay luồng thực thi (*flow of execution*).

Một câu lệnh có thể chứa một dòng code duy nhất, gọi là câu lệnh đơn (*single line statement*). C# bắt buộc câu lệnh đơn phải kết thúc bằng dấu chấm phẩy.

Ví dụ: Trong thân của phương thức Main() có 3 câu lệnh đơn.

```
Console.WriteLine("Hello world from C#");
Console.WriteLine("Press any key to quit");
Console.ReadKey();
```

Một câu lệnh nếu chỉ chứa dấu chấm phẩy được gọi là một lệnh rỗng (*empty statement*).

Dưới đây là một số loại câu lệnh trong C#:

- *Lệnh khai báo (Declaration statements)*: dùng để khai báo các biến và hằng;
- *Lệnh tính toán (Expression statements):* thường gọi là *biểu thức*, dùng để thực hiện các tính toán trên dữ liệu và phải trả về giá trị có thể gắn cho biến;
- *Lệnh lựa chọn (Selection statements)* dùng trong các *cấu trúc rẽ nhánh* như if, else, switch, case;
- *Lệnh lặp* (*Iteration statements*): dùng để thực hiện nhiều lần một lệnh/khối lệnh, bao gồm do, for, foreach, in, while.

2.2.3. Khối lệnh trong C#

Một chuỗi câu lệnh đơn có thể được nhóm lại với nhau tạo thành một **khối lệnh** (*code block* hoặc *statement block*).

Một khối lệnh là một danh sách các lệnh được đặt chung trong một cặp dấu ngoặc kép {}.

Các khối lệnh có thể lồng nhau. Như trong tệp mã nguồn dưới đây, toàn bộ thân của phương thức Main() ở trên là một khối lệnh. Thân của cả lớp *Program* cũng là một khối lệnh. Thân của *namespace* là một khối lệnh. Ba khối lệnh này lồng nhau.

Từ khóa (*keyword*) là những từ được ngôn ngữ gán cho ý nghĩa riêng xác định, là nòng cốt của cú pháp ngôn ngữ. Chúng ta không được sử dụng từ khóa cho mục đích gì khác ngoài những gì đã được ngôn ngữ quy định. Ví dụ, không được sử dụng từ khóa làm định danh (tên) của biến, hằng, phương thức.

Dưới đây là danh sách từ khóa trong C#.

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	in (generic modifier)	int	interface
internal	is	lock	long
namespace	new	null	object
operator	out	out (generic modifier)	override
params	private	protected	public
readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc
static	string	struct	switch
this	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	using static
void	volatile	while	

Từ khóa của C# phân chia làm 2 loại: từ khóa dành riêng (*reserved keyword*) và từ khóa theo ngữ cảnh (*contextual keyword*). C# có 79 từ khóa dành riêng và 25 từ khóa ngữ cảnh (*contextual keyword*).

Dưới đây là danh sách từ khóa ngữ cảnh của C#:

add	alias	ascending
async	await	descending
dynamic	from	get
global	group	into
join	let	orderby
partial (loại)	partial (phương thức)	remove
select	set	value
var	when (filter condition)	where (hạn chế kiểu dữ liệu Generic)
yield		

2.2.4. Định danh

Định danh (*identifier*) là chuỗi ký tự của ngôn ngữ dùng để đặt tên cho các thành phần như kiểu, biến, hằng, phương thức, tham số, v.v..

Quy tắc đặt định danh

Định danh trong C# chỉ được tạo ra từ một nhóm ký tự xác định chứ không được chứa mọi loại ký tự. Nhìn chung, định danh trong C# có thể chứa các chữ cái a-z, A-Z, chữ số 0-9, dấu gạch chân _ và ký tự @.

Tuy vậy, có những giới hạn nhất định.

Ký tự a-z, A-Z và _ có thể có mặt ở mọi vị trí trong định danh

Chữ số không được phép đứng đầu định danh.

Ký tự @ chỉ được phép đứng đầu định danh (và cũng không được khuyến khích sử dụng).

Khi đặt định danh có sự phân biệt giữa ký tự hoa và thường. Ví dụ *write* và *Write* là hai định danh (của phương thức) hoàn toàn khác nhau. Điều này khác biệt với ngôn ngữ như Pascal vốn không phân biệt ký tự hoa và thường.

2.2.5. Lớp Program và phương thức Main()

C# là một ngôn ngữ lập trình hướng đối tượng, trong mỗi dự án (*project*) bắt buộc phải có ít nhất một lớp. Lớp là đơn vị quan trọng bậc nhất trong C#, và cả quá trình học ngôn ngữ sau này hầu đều tập trung vào các kỹ thuật xây dựng lớp.

Program là lớp được C# tự động sinh ra khi tạo *project*. Tên của lớp này không bắt buộc là *Program*. Chúng ta có thể thay đổi thành bất kỳ tên gọi nào, miễn phù hợp với quy tắc đặt tên. [1]

Trong lớp Program có một phương thức (method) đặc biệt :

```
static void Main(string[] args)
{
Console.WriteLine("Hello world from C#");
Console.WriteLine("Press any key to quit");
Console.ReadKey();
}
```

```
static void Main(string[] args) { Console.WriteLine("Hello world from C#"); Console.WriteLine("Press any key to quit"); Console.ReadKey(); }
```

```
static void Main(string[] args)
{
    Console.WriteLine("Hello world from C#");
    Console.WriteLine("Press any key to quit");
```

```
    Console.ReadKey();
}
```

Phương thức này có tên gọi riêng là "entry point". Main() là phương thức đầu tiên được gọi khi chạy chương trình C#. Đây là điểm khởi đầu trong hoạt động của các chương trình C#.

Lưu ý rằng, ngôn ngữ C# phân biệt chữ hoa – thường, giống như C/C++. Do đó, Main và main không giống nhau, Writeline khác với WriteLine.

Phương thức entry point của C# bắt buộc phải là *static* void Main() hoặc static int Main(). Phần tham số không bắt buộc. Nói chung chúng ta có thể gặp các biến thể sau của phương thức Main():

```
// int return type, array of strings as the parameter.
static int Main(string[] args)
{
// Must return a value before exiting!
return 0;
}
// No return type, no parameters.
static void Main()
{}
// int return type, no parameters.
static int Main()
{
// Must return a value before exiting!
return 0;
}
```

2.2.6. Khai báo biến trong C#

Biến trong C# được **khai báo** với cú pháp sau:

datatype identifier;

Trong đó *datatype (kiểu dữ liệu)* là tên kiểu dữ liệu mà biến đó có thể lưu giữ, *identifier* là định danh (tên) của biến.

Ví dụ 2.1: Khai báo biến có tên là i, lưu giữ được các giá trị số nguyên (int).

int i;

Tuy nhiên, trong C#, lệnh khai báo biến trên mặc dù đúng cú pháp nhưng compiler lại không cho phép dùng ngay biến i trong các biểu thức. C# bắt buộc biến phải được gán giá trị trước khi sử dụng.

Để gán giá trị cho biến ta dùng phép gán (assignment operator):

i = 10;

Phép gán trong C# đơn giản chỉ là một dấu bằng (=).

Có thể kết hợp cả khai báo biến và gán giá trị vào cùng một lệnh:

int i = 10;

Có thể khai báo và gán giá trị cho nhiều biến cùng kiểu trong cùng một lệnh:

int x = 10, y = 20; // x và y có cùng kiểu int

Nếu các biến khác kiểu, bắt buộc phải khai báo trong các lệnh khác nhau:

int x = 10;

bool y = true; // biến chứa giá trị logic true/false

// lệnh khai báo dưới đây là sai

int x = 10, bool y = true; // trình biên dịch sẽ báo lỗi ở dòng này

Biến cục bộ trong C# được đặt tên theo quy tắc đặt định danh.

Trình biên dịch C# bắt buộc mọi biến phải được khởi tạo với một giá trị nào đó trước khi sử dụng trong biểu thức.

C# có hai phương pháp để đảm bảo biến được khởi tạo trước khi sử dụng.

Nếu biến là một trường dữ liệu trong *class* hoặc *struct*. Nếu không được lập trình viên trực tiếp gán giá trị, trình biên dịch sẽ tự động gán cho biến một giá trị mặc định tùy từng kiểu dữ liệu (ví dụ 0 cho số nguyên).

Nếu biến nằm trong thân phương thức (gọi là biến cục bộ – *local variable*): bắt buộc lập trình viên phải trực tiếp gán giá trị trước khi sử dụng nó trong biểu thức. Trong trường hợp này, biến không nhất thiết phải được khởi tạo ngay khi khai báo. Miễn sao nó phải có giá trị trước khi sử dụng là được. Vi phạm này bị C# coi là lỗi và trình biên dịch sẽ dừng lại.

Dưới đây là một ví dụ lỗi về sử dụng biến không khởi tạo:

```
static void Main()
{
    int d;
    Console.WriteLine(d); // Sẽ báo lỗi ở đây. Biến d chưa có giá trị.
}
```

Nếu biên dịch đoạn lệnh trên sẽ gặp lỗi ‘’Use of unassigned local variable 'd'’’.

Phạm vi của biến trong C#

Phạm vi (*scope*) là vùng lệnh mà các lệnh trong đó có thể truy xuất biến. Phạm vi được xác định theo quy tắc sau:

Biến (trong thân phương thức) có thể truy xuất từ vị trí khai báo đến khi gặp dấu } báo hiệu kết thúc của nhóm lệnh. Một nhóm lệnh đặt trong cặp dấu { } như vậy gọi là một khối *code* (*code block*). Nói cách khác, biến có phạm vi tác dụng là khối lệnh mà nó được khai báo. Ra khỏi khối lệnh này, biến không sử dụng được nữa.

Các khối lệnh có thể nằm lồng nhau. Biến khai báo ở khối code lớn (nằm ngoài) sẽ có phạm là cả khối lệnh lớn, tức là bao trùm cả các khối lệnh con bên trong. Biến khai báo ở khối lệnh con (bên trong) thì có phạm vi là khối lệnh đó thôi. Khối lệnh bên ngoài không thuộc phạm vi của biến đó.

2.2.7. Hằng trong C#

Hằng có thể xem tương tự như biến về khía cạnh lưu trữ dữ liệu. Khác biệt duy nhất là giá trị của hằng không thể thay đổi.

Hằng được khai báo và khởi tạo với cú pháp như sau:

const int a = 1000; // giá trị của a sẽ không thể thay đổi được sau khai báo này

Tuy nhiên có một số sự khác biệt sau giữa hằng và biến: Hằng bắt buộc phải được khởi tạo ngay lúc khai báo. Sau khi gán giá trị (lúc khai báo), giá trị này sẽ không thể thay đổi.

Giá trị của hằng phải tính toán được ở giai đoạn biên dịch chương trình. Do vậy, chúng ta không thể khởi tạo một hằng nhưng sử dụng giá trị lấy từ một biến.

2.2.8. Các kiểu dữ liệu cơ sở của C#

Kiểu dữ liệu (data type, hay đơn giản là type) trong C# (cũng như các ngôn ngữ khác) là một đặc tính của dữ liệu nhằm thông báo cho C# compiler biết về ý định sử dụng dữ liệu của lập trình viên. Một trong những việc đầu tiên cần sử dụng đến kiểu dữ liệu là khai báo biến và hằng mà chúng ta đã biết

e. Các kiểu số nguyên

Kiểu số nguyên (*int)* của C# luôn luôn chiếm 4 byte (32 bit). Trong C++, số bit của kiểu int thay đổi phụ thuộc vào platform.

Kiểu *byte* là 8 bit, có dải giá trị từ 0 đến 255, và không thể chuyển đổi qua lại với kiểu char như trong C. Kiểu *byte* luôn luôn không dấu (khác với C). Nếu muốn sử dụng số nguyên 8 bit có dấu, chúng ta dùng kiểu sbyte.

Tất cả các kiểu số nguyên đều có thể nhận giá trị biểu diễn ở nhiều cơ số (base) khác nhau: cơ số 10 (decimal), 16 (hex), 8 (octal), 2 (binary). Giá trị biểu diễn ở các cơ số khác 10 phải sử dụng thêm tiếp tố (prefix) tương ứng.

long x = 0x12ab; // số hexa, prefix là 0x hoặc 0X

byte y = 0b1100; // số nhị phân, prefix là 0b hoặc 0B

int z = 01234; // số hệ cơ số 8, prefix là 0

C# cho phép sử dụng dấu _ giữa các chữ số để tách các chữ số cho dễ đọc hơn với các giá trị lớn. Dấu _ gọi là digit separator.

long l1 = 0x123_456_789_abc_def; // dấu _ giúp tách các chữ số cho dễ đọc

long l2 = 0x123456789abcdef; // cách viết thông thường

int bin = 0b1111_1110_1101; // viết tách các bit thế này dễ đọc hơn

Khi dùng từ khóa *var* để khai báo biến thuộc kiểu số nguyên, C# mặc định sẽ hiểu nó là kiểu int. Nếu muốn chỉ định giá trị nguyên thuộc một kiểu nào đó khác, chúng ta phải sử dụng một cách viết riêng gọi là integer literal.

Integer literal là các ký tự viết vào cuối giá trị số (postfix) để báo hiệu kiểu dữ liệu, bao gồm: U (hoặc u) báo hiệu số nguyên không dấu; L (hoặc l) báo hiệu giá trị thuộc kiểu long; UL (hoặc ul) cho kiểu ulong. Có thể sử dụng các ký tự này khi viết ở hệ cơ số khác 10. Ví dụ:

var i0 = 123; // c# mặc định coi đây là kiểu int

var i1 = 123u; // giá trị này thuộc kiểu uint

var i2 = 123l; // giá trị này thuộc kiểu long

var i3 = 123ul; // giá trị này thuộc kiểu ulong

var i4 = 0x123L; // giá trị kiểu long ở hệ hexa

3. *Các kiểu số thực*

C# (và .NET) chỉ cung cấp 2 loại số thực: float (System.Single) và double (System.Double)

Khi dùng từ khóa *var* với giá trị số thực, C# sẽ mặc định hiểu nó thuộc về kiểu double. Để chỉ định một giá trị thực thuộc kiểu float, cần dùng hậu tố F (hoặc f) sau giá trị. F (hoặc f) được gọi là **float literal**.

var r1 = 1.234; // r1 thuộc kiểu double

var r2 = 1.234f; // r2 thuộc kiểu float

decimal (System.Decimal) là một dạng số thực đặc biệt chuyên dùng trong tính toán tài chính.

Literal cho decimal là M (hoặc m).

var d = 12.30M; // biến này thuộc kiểu decimal

Các kiểu số thực cũng hỗ trợ cách viết dạng khoa học (và có thể kết hợp với float decimal):

*var d1 = 1.5E-20; // cách viết khoa học bình thường là 1,5*10^-20, kiểu double*

*var f1 = 1.5E-10F; // số 1,5*10^-10, kiểu float*

*var m1 = 1.5E-20M; // 1,5*10$-20, kiểu decimal*

4. *Kiểu Boolean*

Boolean (.NET) hay bool (C#) chỉ nhận đúng hai giá trị: true và false. Đây cũng được gọi là **literal** của kiểu bool.

Trong C# không thể tự do chuyển đổi giữa bool và số nguyên như trong C/C++. Có nghĩa là không thể sử dụng 0 thay cho false, giá trị khác 0 thay cho true như trong C. Biến khai báo thuộc kiểu bool chỉ có thể gán giá trị true hoặc false.

5. *Kiểu ký tự*

Kiểu char (C#) hay System.Char (.NET) dùng để biểu diễn ký tự đơn, mặc định là các ký tự Unicode 16 bit.

6. *Kiểu chuỗi ký tự*

Chuỗi (xâu) ký tự (string hoặc System.String), khác biệt với các kiểu dữ liệu bên trên, là một kiểu dữ liệu tham chiếu (reference type)

2.2.9. Toán tử trong C#

C# có khá nhiều toán tử (phép toán). Qua mỗi phiên bản C# lại đưa thêm vào những phép toán mới sử dụng với kiểu dữ liệu mới.

Phần lớn các toán tử cơ bản đều tương tự như các ngôn ngữ kiểu C, bao gồm các phép toán số học, phép toán logic, phép toán tăng giảm, phép toán nhị phân, phép toán index (truy xuất mảng), hay phép toán điều kiện.

Tuy nhiên, C# có nhiều phép toán hoàn toàn khác với C/C++, ví dụ phép toán kiểm tra giá trị null (null coalescing), kiểm tra kiểu, định danh, các phép toán cho delegate, v.v..

Thậm chí cho cùng một công việc nhưng phép toán của C# không giống như C/C++. Ví dụ phép toán truy xuất thành viên (object và struct).

Dưới đây là danh sách tất cả các phép toán hiện có trong C#.

NHÓM	TOÁN TỬ
Phép toán số học	+– * / %
Phép toán logic và nhị phân	& | ^ ~ && || !
Phép toán ghép xâu	+
Phép toán tăng giảm	++– –

Phép toán dịch bit	<< >>
Phép toán so sánh	== != < > <= >=
Phép gán	= += -= *= /= %= &= |= ^= <<= >>=
Phép toán truy xuất thành viên (object và struct)	.
Phép toán indexer (cho mảng)	[]
Ép kiểu (type casting)	()
Phép toán điều kiện	?:
Phép toán cho delegate (thêm/bớt)	+ –
Khởi tạo object	new
Lấy thông tin về kiểu dữ liệu	sizeof is typeof as
Kiểm soát lỗi tràn bộ đệm	checked unchecked
Phép toán liên kết null	??
Phép toán kiểm tra điều kiện null	?. ?[]
Lấy tên của phần tử	nameof()

2.2.10. Các cấu trúc điều khiển trong C#

a. Cấu trúc rẽ nhánh if else

Cấu trúc if-else của C# hoàn toàn thừa kế từ C/C++. Cú pháp của cấu trúc if-else như sau:

```
if (condition)
{
    statements 1
}
else
{
    statements 2
}
```

Trong đó, *condition* là *biểu thức logic* – biểu thức mà giá trị trả về là true hoặc false. Đây là kết quả thực hiện các phép so sánh, hoặc kết quả trả về của một số phương thức.

Statements 1 là danh sách các lệnh sẽ thực thi nếu *condition* có giá trị true; **Statement 2** là danh sách lệnh sẽ thực thi nếu *condition* có giá trị false.

Có một số lưu ý sau khi dùng if-else:

- Nếu statements 1 hoặc statements 2 chỉ có một lệnh duy nhất thì có thể không cần dùng cặp dấu {}.
- Nhánh else{} là không bắt buộc; if thì bắt buộc phải có.
- Bình thường chỉ có thể tạo ra 2 nhánh rẽ: 1 nhánh if, 1 nhánh else.
- Để tạo thêm nhiều nhánh rẽ nữa chúng ta có thể kết hợp thêm các nhánh else if vào cấu trúc trên. Số lượng nhánh else if không giới hạn.
- Có thể lồng nhiều if-else với nhau.

7. Cấu trúc rẽ nhiều nhánh switch-case

C# cung cấp một cấu trúc khác để thực hiện rẽ nhiều nhánh thay cho việc lồng ghép nhiều if-else: cấu trúc switch-case. Cú pháp như sau:

```
switch(expression)
{
  case <value1>
    // code block
  break;
  case <value2>
    // code block
  break;
  case <valueN>
    // code block
  break;
  default
    // code block
  break;
```

}

Cấu trúc này yêu cầu phải cung cấp một biểu thức "expression" (lệnh tính toán). Giá trị của expression sẽ được tính ra và lần lượt so sánh với value1, value2, .., valueN. Các value này bắt buộc phải là các hằng số hoặc biểu thức tính ra hằng số, KHÔNG được sử dụng biến.

Nếu trùng với value nào, khối lệnh tương ứng sẽ được thực hiện, sau đó sẽ bỏ qua tất cả các kiểm tra còn lại. Vì lý do này, C# bắt buộc mỗi "case" phải được kết thúc bằng lệnh "break" hoặc "return". Quy định này khiến cấu trúc switch-case của C# an toàn hơn một chút so với trong C/C++ (vốn không bắt buộc dùng break).

Khi một case được thực hiện, có thể tiếp tục nhảy sang một case khác bằng lệnh nhảy goto case. Cách sử dụng switch-case mà thực hiện được nhiều case cùng lúc như vậy có tên gọi là *fall-through* (thực hiện qua nhiều case).

Nếu giá trị của biểu thức tính ra không trùng với bất kỳ "case" nào, khối lệnh default sẽ được thực hiện. Khối "default" không bắt buộc. Trong trường hợp không có khối default và giá trị của expression không trùng với bất kỳ case nào, cấu trúc switch đơn giản là không thực hiện bất kỳ khối lệnh nào.

Các lệnh đi sau mỗi case không cần viết trong cặp {}, kể cả khi có nhiều lệnh.

Cấu trúc này yêu cầu phải cung cấp một biểu thức "expression" (lệnh tính toán). Giá trị của expression sẽ được tính ra và lần lượt so sánh với value1, value2, .., valueN. Các value này bắt buộc phải là các hằng số hoặc biểu thức tính ra hằng số, KHÔNG được sử dụng biến.

Nếu trùng với value nào, khối lệnh tương ứng sẽ được thực hiện, sau đó sẽ bỏ qua tất cả các kiểm tra còn lại. Vì lý do này, C# bắt buộc mỗi "case" phải được kết thúc bằng lệnh "break" hoặc "return". Quy định này khiến cấu trúc switch-case của C# an toàn hơn một chút so với trong C/C++ (vốn không bắt buộc dùng break).

Khi một case được thực hiện, có thể tiếp tục nhảy sang một case khác bằng lệnh nhảy goto case. Cách sử dụng switch-case mà thực hiện được nhiều case cùng lúc như vậy có tên gọi là *fall-through*.

Nếu giá trị của biểu thức tính ra không trùng với bất kỳ "case" nào, khối lệnh default sẽ được thực hiện. Khối "default" không bắt buộc. Trong trường hợp không có khối default và giá trị của expression không trùng với bất kỳ case nào, cấu trúc switch đơn giản là không thực hiện bất kỳ khối lệnh nào.

Các lệnh đi sau mỗi case không cần viết trong cặp {}, kể cả khi có nhiều lệnh.

8. Cấu trúc while

***while** (<biểu_thức_logic>) { [<danh_sách_lệnh>] }*

Chừng nào biểu thức logic còn nhận giá trị true thì danh sách lệnh sẽ được thực hiện. Cấu trúc này sẽ luôn kiểm tra biểu thức logic trước, sau đó mới thực hiện danh sách lệnh.

```
var i = 0;
while(i < 10)
{
   Console.Write($"{i}\t");
   i++;
}
```

Trong cấu trúc while, danh sách lệnh có thể không được thực hiện lần nào. Tình huống này xảy ra khi biểu thức logic nhận giá trị false ngay từ đầu.

Lưu ý rằng trong thân của while phải có lệnh làm thay đổi giá trị của biểu thức logic. Nếu không sẽ tạo ra vòng lặp vô hạn.

9. Cấu trúc do-while

***do** { [<danh_sách_lệnh>] } **while** (<biểu_thức_logic>);*

Thực hiện danh sách lệnh rồi mới kiểm tra giá trị của biểu thức logic. Nếu biểu thức logic vẫn nhận giá trị true, danh sách lệnh sẽ lại được thực hiện.

```
i = 0;
 do
 {
```

```
    Console.Write($"{i}\t");
    i++;
} while (i < 10);
```

Cấu trúc do-while khác biệt với while ở chỗ, danh sách lệnh sẽ được thực hiện trước, sau đó mới kiểm tra giá trị của biểu thức logic. Do đó, khi sử dụng cấu trúc do-while, danh sách lệnh luôn luôn thực hiện ít nhất một lần.

Lưu ý rằng, sau while(<biểu_thức_logic>) phải có dấu chấm phẩy.

```
do {...}
while(i < 10);
```

Giống như đối với while, phải có lệnh làm thay đổi giá trị của biểu thức logic trong khối code của do. Nếu không sẽ tạo ra vòng lặp vô hạn.

10. *Cấu trúc for*

for *(<khởi tạo giá trị đầu của biến điều khiển>; <kiểm tra điều kiện dừng của biến điều khiển>; <bước nhảy> { [<danh_sách_lệnh>] }*

Cấu trúc này sẽ thực hiện danh sách lệnh một số lần xác định (trong khi hai cấu trúc trên không xác định được số lần thực hiện).

```
for (i = 0; i < 10; i++)
{
    Console.Write($"{i}\t");
}
```

Trong cấu trúc for, biến điều khiển, cách thay đổi giá trị của biến điều khiển cũng như điều kiện kiểm tra biến điều khiển đều viết chung trong khai báo. C# sẽ tự thay đổi giá trị biến điều khiển theo công thức chúng ta cung cấp.

Chúng ta có thể thực hiện đồng thời khai báo và khởi tạo giá trị của biến điều khiển ngay trong cấu trúc for, thay vì phải khai báo biến riêng.

```
for (var i = 0; i < 10; i++) { ... }
```

Chúng ta có thể lồng nhiều vòng for với nhau, ví dụ, để duyệt một ma trận.

```
// duyệt qua các hàng
```

```
for (int i = 0; i < 100; i += 10)
{
    // duyệt qua các cột trong một hàng
    for (int j = i; j < i + 10; j++)
    {
        Console.Write($" {j}");
    }
    Console.WriteLine();
}
```

11. *Điều khiển vòng lặp*

Trong vòng lặp có thể sử dụng lệnh *break* hoặc *continue* để điều khiển hoạt động của vòng lặp. Cụ thể như sau:

- Lệnh break: phá vỡ vòng lặp. Khi gặp lệnh break, tất cả các lệnh đứng sau break sẽ không thực hiện nữa, vòng lặp kết thúc.

- Lệnh continue: phá vỡ chu kỳ hiện tại của vòng lặp. Khi gặp lệnh continue, tất cả lệnh đứng sau continue không thực hiện nữa, vòng lặp sẽ chuyển sang chu kỳ tiếp theo.

2.2.11. Quản lí lỗi và ngoại lệ

a. *Ngoại lệ*

Ngoại lệ (*exception*) là vấn đề lỗi phát sinh trong quá trình thực thi chương trình. Thường khi chương trình đang chạy mà phát sinh ngoại lệ (lỗi) thì dẫn đến chương trình kết thúc ngay lập tức. Có nhiều nguyên nhân để chương trình đang chạy mà phát sinh ngoại lệ, ví dụ:

- Dữ liệu người dùng nhập sai, mà chương trình không kiểm soát được
- Thực hiện các phép toán không được phép (như chia một số cho 0)
- Thao tác với tài nguyên không tồn tại (như mở file không có trên đĩa, kết nối đến cơ sở dữ liệu không tồn tại ...)
- Thiếu bộ nhớ…

Trong C# khi có một lỗi phát sinh hầu hết các lỗi đều có thể quản lý bởi thư viện C# thì nó sẽ phát sinh ra một đối tượng lớp Exeption (System.System) hoặc đối tượng lớp nào đó kế thừa từ Exception

Khi một đối tượng lớp *Exception* sinh ra mà chương trình không chủ động xử lý đối tượng này thì chương trình sẽ kết thúc. Đối tượng lớp *Exception* chứa trong nó các thông tin về lỗi (dòng thông báo, nguyên nhân lỗi, nơi phát sinh lỗi ...)

b. Quản lí lỗi

Nếu ngoại lệ (lỗi thực thi) phát sinh mà không xử lý thì chương trình sẽ dừng đột ngột do đó nếu muốn xử lý ngoại lệ thì ta cần bắt (catch) lấy nó và điều hướng chương trình một cách thích hợp. Để bắt ngoại lệ ta sử dụng câu lệnh try ... catch ... như sau:

```
try {
  // Các khối code được giám sát để bắt lỗi nếu có
  // nếu có lỗi sẽ phát sinh ngoại lệ Exception
  // Ngoại lệ này bắt lại được ở khối catch
}
catch (Exception loi)
{
  // Khối này thực thi khi có lỗi - đối tượng Exception bắt được lưu ở biến loi
}
```

Ví dụ 2.2 :

```
static void Main(string[] args)
{
   try {
      // khối này được giám sát để bắt lỗi - khi nó phát sinh
      int[] mynumbers = new int[] {1,2,3};
      int i = mynumbers[10];            // dòng này phát sinh lỗi
```

```
        Console.WriteLine(i);           // dòng này không được thực thi vì lỗi trên
    }
    catch (Exception loi)
    {
        // khối này thực thi khi bắt được lỗi
        Console.WriteLine("Có lỗi rồi");
        Console.WriteLine(loi.Message);
    }
}
```

Trong .NET từ lớp cơ sở Exception xây dựng nên rất nhiều loại ngoại lệ khác phục vụ chi tiết cho từng loại lỗi phát sinh khác nhau như: FileNotFoundException, FormatException, OutOfMemoryException, ArgumentException, NullReferenceException, IndexOutOfRangeException, DivideByZeroException...

Để bắt cụ thể một loại ngoại lệ nào đó chỉ việc thêm một khối *catch* tương ứng với ngoại lệ đó.

Ví dụ 2.3:

```
try {
    int x = 10;
    int y = 0;
    int z = x / y;


}
catch (DivideByZeroException e1) {
    Console.WriteLine(e1.Message);
}
catch (Exception e2) {
```

```
    Console.WriteLine(e2.Message);
}
```

Trong lệnh *try ... catch*, chúng ta có thể thêm một tùy chọn là khối *finally*, các lệnh trong khối này được thực thi ngay cả khi có phát sinh ngoại lệ hay không. Khối này cơ bản để giải phóng các tài nguyên chiếm giữ.

Ví dụ 2.4 :

```
int x = 10;
int y = 0;
int z = 0;
try {
    z = x / y;
}
catch (DivideByZeroException e1) {
    Console.WriteLine(e1.Message);
}
finally {
    // Luôn được thi hành dù có phát sinh ngoại lệ hay không
    Console.WriteLine(z);
}
```

Nếu chương trình muốn phát sinh ngoại lệ cho biết có một lỗi nào đó vừa xảy ra thì chúng ta cần tạo ra một đối tượng lớp *Exception* hoặc đối tượng thuộc lớp nào đó kế thừa từ *Exception*, sau đó phát sinh bằng lệnh *throw*. Ném lỗi (throw) dùng để ra hiệu lỗi lạ thường xảy ra trong quá trình thực thi chương trình.

Ví dụ 2.5 : Xây dựng hàm Thuong nếu tham số thứ 2 bằng không thì phát sinh ngoại lệ.

```
public static double Thuong(double x, double y) {
    if (y == 0) {
```

```
        // Khởi tạo ngoại lệ, tham số là thông báo lỗi
        Exception myexception = new Exception("Số chia không được bằng 0");
        // phát sinh ngoại lệ, code phía sau throw không được thực thi
        throw myexception;
    }
    return x / y;
}
static void Main(string[] args)
{
    double z = Thuong(100,0);
}
```

Chúng ta có thể khai báo một lớp ngoại lệ tùy ý kế thừa từ lớp ngoại lệ như : *System.Exception* rồi sử dụng chúng trong cấu trúc kiểm soát *try..catch*

Ví dụ 2.6 : Nhập vào một dữ liệu chuỗi *string*, chuỗi này phải có độ dài dưới 10 ký tự, nếu dài hơn thì phát sinh *Exception* riêng.

```
namespace Error_Exception
{
    public class DataTooLongExeption : Exception
    {
        const string erroMessage = "Dữ liệu quá dài";
        public DataTooLongExeption() : base(erroMessage) {
        }
    }
class Program
    {
        public static void UserInput(string  s) {
            if (s.Length > 10)
```

```
            {
                Exception e = new DataTooLongExeption();
                throw e;    // lỗi văng ra
            }
            //Other code - no exeption
        }
        static void Main(string[] args)a
        {
            try {
                UserInput("Đây là một chuỗi rất dài ...");
            }
            catch (DataTooLongExeption e) {
                Console.WriteLine(e.Message);
            }
            catch (Exception otherExeption) {
                Console.WriteLine(otherExeption.Message);
            }
        }
    }
}
```

2.2.12. Truyền tham số

Trong C#, ta có thể truyền tham số cho phương thức theo kiểu tham chiếu hoặc tham trị. Khi truyền theo kiểu tham trị sẽ xảy ra việc sao chép giá trị từ đối số (tham số thực) sang tham số (tham số hình thức). Còn khi truyền theo kiểu tham chiếu thì đối số và tham số đều là một.

C# cung cấp từ khóa *ref* để truyền đối số theo kiểu tham chiếu và từ khóa out để truyền đối số vào trong phương thức theo kiểu tham chiếu mà không cần

khởi gán giá trị đầu cho đối số. Tuy nhiên, khi dùng từ khóa out thì trong phương thức phải có lệnh gán giá trị cho tham chiếu này.

Đối với những dữ liệu kiểu giá trị (*int, long, float, char,...*), muốn thay đổi giá trị của chúng thông qua việc truyền tham số cho hàm, phương thức ta phải truyền theo kiểu tham chiếu một cách tường minh bằng từ khóa ref hoặc out.

Đối với những dữ liệu kiểu tham chiếu (đối tượng, mảng), khi dùng chúng để truyền đối số mà không có từ khóa *ref* hoặc *out*, ta chỉ có thể làm thay đổi giá trị của vùng nhớ trong *heap* mà chúng trỏ tới nhưng bản thân tham chiếu (địa chỉ vùng nhớ) không bị thay đổi. Nếu muốn thay đổi vùng nhớ mà chúng trỏ tới ta phải dùng từ khóa *ref* hoặc *out* một cách tường minh.

Ví dụ 2.7 : Sử dụng truyền tham trị bằng tham số kiểu giá trị trong bài toán đổi số như sau :

```
class PassingByVal
  {
    static void Swap(int a, int b)
    {
      int Temp = a;
      a = b;
      b = Temp;
      Console.WriteLine("Trong phuong thuc Swap: a = { 0}, b = { 1}", a, b);
    }
    static void Main(string[] args)
    {
      int x = 3, y = 4;
      Console.WriteLine("Truoc khi goi phuong Swap: x = { 0}, y = { 1}", x, y);
      Swap(x, y);
      Console.WriteLine("Sau khi goi phuong thuc Swap: x = { 0}, y = { 1} ", x,
y);
```

```
        }
    }
```

CÂU HỎI VÀ BÀI TẬP

1. Câu hỏi :

   Câu 1: Nêu các ngôn ngữ lập trình phổ biến hiện nay.

   Câu 2: Trình bày ưu và nhược điểm của ngôn ngữ C#.

2. Bài tập

   Bài 1: Viết chương trình C# giải phương trình bậc 2: $ax^2 + bx + c = 0$.

   Bài 2: Viết chương trình C# liệt kê tất cả các số nguyên tố nhỏ hơn n.

   Bài 3: Viết chương trình C# tìm các số thuận nghịch có 6 chữ số. Một số được gọi là số thuận nghịch nếu ta đọc từ trái sang phải hay từ phải sang trái số đó ta vẫn nhận được một số giống nhau. Ví dụ 123321 là một số thuận nghịch.

TÀI LIỆU THAM KHẢO BÀI 2

[1] P.H.Khang, *C# 2005 Tập 1 Lập trình cơ bản*, NXB Lao động xã hội, 2013.

BÀI 3 LỚP VÀ ĐỐI TƯỢNG

NỘI DUNG CHÍNH

- Cách khai báo và sử dụng lớp
- Khái niệm, cách sử dụng đối tượng trong lập trình C#
- Phạm vi sử dụng của lớp và đối tượng

3.1. LỚP

Khai báo lớp (*class)* trong C# sử dụng cấu trúc :

[Mức độ truy cập] class <tên class>{ [thân class] }

trong đó:

- *class* là từ khóa của C# dùng để khai báo lớp;
- Tên *class* do người lập trình lựa chọn và phải tuân thủ quy tắc đặt định danh.

Các phần này viết tách nhau bởi dấu cách.

Dấu cách trong C# chỉ có tác dụng phân tách các thành phần của một lệnh, khác với một số ngôn ngữ dùng dấu cách như một phần của cú pháp. Số lượng dấu cách không ảnh hưởng tới ý nghĩa của dòng lệnh. Trình biên dịch sẽ tự bỏ qua những dấu cách thừa. [2]

Từ khóa *"class"* là bắt buộc khi khai báo lớp. Mặc định, Visual Studio thể hiện từ khóa bằng màu xanh da trời. Từ khóa là những cụm ký tự được C# lựa chọn cho những mục đích riêng để diễn đạt cú pháp của ngôn ngữ. Một số kiểu

dữ liệu dựng sẵn của C# cũng được đặt thành từ khóa (sẽ xem xét trong các bài sau). [2]

Từ khóa điều khiển truy cập quyết định **phạm vi sử dụng** của lớp. Mỗi lớp trong C# có thể chỉ được sử dụng nội bộ trong phạm vi của dự án (*project*), hoặc có thể được sử dụng bởi các *project* khác. Mặc định, mỗi lớp trong C# chỉ được sử dụng trong phạm vi của *project* (sử dụng bởi các lớp khác trong cùng *project*). Do đó, nếu không thấy điều khiển truy cập nào thì sẽ hiểu là "*internal*". Nếu muốn sử dụng lớp này trong các *project* khác, trước từ khóa lớp cần bổ sung từ khóa "*public*".

Ví dụ 3.1 : Tạo một lớp có tên Motorbike có các thuộc tính như: tên, kiểu, màu sắc…

```
class Motorbike
{
 // Khai báo thuộc tính
 private string name;
 private string type;
 private string color;
 private int guarantee;
 // Khai báo và định nghĩa phương thức
 public void Input()
 {
  Console.Write("Enter motorbike name: ");
  name = Console.ReadLine();
  Console.Write("Enter motorbike type: ");
  type = Console.ReadLine();
  Console.Write("Enter motorbike color: ");
  color = Console.ReadLine();
  Console.Write("Enter motorbike guarantee: ");
  guarantee = Convert.ToInt32(Console.ReadLine());
```

```
}
public void Display()
{
 Console.WriteLine("Motorbike name: " + name);
 Console.WriteLine("Motorbike color: " + color);
 Console.WriteLine("Motorbike type: " + type);
 Console.WriteLine("Motorbike guarantee: " + guarantee);
}
```

3.2. ĐỐI TƯỢNG

Lớp mô tả cấu trúc chung của một nhóm đối tượng nào đó, ngược lại, một đối tượng là một trường hợp cụ thể của một lớp.

Vì đối tượng là một kiểu tham chiếu nên dữ liệu thực sự được tạo trên vùng nhớ Heap và ta phải dùng toán tử ***new*** để cấp phát cho đối tượng. Kể từ lúc đối tượng được cấp phát bộ nhớ, ta có thể gán các giá trị cho các biến thành viên, gọi thihành các phương thức của đối tượng này.

Thường thì ta chỉ việc khai báo và cấp phát đối tượng, việc hủy vùng nhớ mà đối tượng chiếm giữ khi đối tượng đó mất hiệu lực sẽ do bộ dọn rác của trình biêndịch đảm nhiệm.

Cú pháp khai báo đối tượng và cấp phát vùng nhớ cho đối tượng:

TênLớp TênBiếnĐốiTượng;

TênBiếnĐốiTượng = new TênLớp(DanhSáchĐốiSố);

hoặc

TênLớp TênBiếnĐốiTượng = new TênLớp(DanhSáchĐốiSố);

Chú ý:

- Sau khi khai báo biến đối tượng thì biến đó chỉ là một con trỏ.
- Sau khi cấp phát bằng từ khóa new thì biến trỏ tới một đối tượng thực sự.

Ví dụ 3.2: Tạo một đối tượng có tên là exciter, cú pháp như sau:

// Tạo đối tượng là exciter

```
Motorbike exciter = new Motorbike();
// Truy cập phương thức
exciter.Input();
exciter.Display();
```

3.3. ĐÓNG GÓI DỮ LIỆU

Tính đóng gói (Encapsulation) là khả năng che giấu thông tin của đối tượng với môi trường bên ngoài. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của đối tượng hoàn toàn tùy thuộc vào người viết mã.[2]

Tính đóng gói được thực hiện bằng cách sử dụng các chỉ thị truy cập. Một chỉ thị truy cập xác định phạm vi và khả năng truy cập của các thành viên lớp (trường, thuộc tính, phương thức). C# hỗ trợ các chỉ thị truy cập sau:

- public
- private
- protected
- internal
- protected internal

Chỉ thị truy cập public

Chỉ thị truy cập public cho phép một lớp đưa ra các thuộc tính và các phương thức thành viên của nó cho các phương thức và đối tượng khác. Bất kỳ thành viên nào cũng có thể được truy cập từ bên ngoài lớp.

Chỉ thị truy cập private

Chỉ thị truy cập private cho phép lớp ẩn các trường và các phương thức thành viên khỏi các phương thức và đối tượng khác.

Chỉ các thành viên trong cùng một lớp mới có thể truy cập các thành viên private của nó. Ngay cả thể hiện của lớp cũng không thể truy cập các thành viên private của nó.

Nếu một trường hoặc phương thức không có chỉ thị truy cập thì nó sẽ được gắn chỉ thị truy cập mặc định là private.

Chỉ thị truy cập protected

Chỉ thị truy cập protected cho phép lớp ẩn các trường và các phương thức thành viên khỏi các phương thức và đối tượng khác. Nó chỉ cho phép một lớp con truy cập các trường và các phương thức thành viên này của lớp cha (hoặc lớp cơ sở).

Chỉ các thành viên trong class cha và class con mới có thể truy cập các thành viên protected của lớp cha. Ngay cả thể hiện của lớp con hay lớp cha cũng không thể truy cập các thành viên protected của class cha.

Chỉ thị truy cập protected tương tự chỉ thị truy cập private, chỉ khác là nó cho phép class con truy cập các trường và phương thức thành viên của class cha có chỉ thị protected.

Chỉ thị truy cập internal

Chỉ thị truy cập internal cho phép một lớp đưa ra các thuộc tính và phương thức thành viên của nó cho các phương thức và đối tượng khác trong cùng assembly.

Nói cách khác, bất kỳ thành viên nào có chỉ thị truy cập internal đều có thể được truy cập từ bất kỳ lớp hoặc phương thức nào được định nghĩa trong cùng một assembly.

Chỉ thị truy cập internal tương tự như chỉ thị public, nó chỉ khác ở chỗ chỉ thị internal giới hạn phạm vi truy cập trong cùng một assembly còn chỉ thị public thì không có bất kỳ giới hạn nào.

Nếu một class không có chỉ thị truy cập thì chỉ thị truy cập mặc định sẽ là internal.

Chỉ thị truy cập protected internal

Chỉ thị truy cập protected internal cho phép lớp ẩn các trường và các phương thức thành viên khỏi các phương thức và đối tượng khác. Nó chỉ cho phép một lớp con cùng assembly với lớp cha truy cập các trường và các phương thức thành viên này của lớp cha (hoặc lớp cơ sở).

Do có chỉ thị truy cập internal nên các trường và phương thức thành viên của lớp cha có thể truy cập thông qua thể hiện của lớp cha hoặc lớp con.

3.4. CHƯƠNG TRÌNH MINH HỌA

Chương trình nhập chiều dài, chiều rộng của hình chữ nhật và xuất ra diện tích, chu vi của hình chữ nhật.

```
using System;
namespace LopDoiTuongHCN
{
class HCN
{
protected float Dai, Rong;
public float ChuVi()
{
return (Dai + Rong )*2;
}
public float DienTich()
{
return Dai* Rong;
}
public void Nhap()
{
Console.WriteLine("Nhap chieu dai: "); Dai = float.Parse(Console.ReadLine());
Console.WriteLine("Nhap chieu rong: "); Rong = float.Parse(Console.ReadLine());
}
public void Xuat()
{
Console.WriteLine("Hinh chu nhat: Dai = {0}, Rong = {1}", Dai, Rong);
}
}
class Application
```

```
{
static void Main(string[] args)
{
HCN h;
h = new HCN();
h.Nhap();
h.Xuat();
Console.WriteLine("Chu vi   hinh    chu nhat:
{0}", h.ChuVi());
Console.WriteLine("Dien tich hinh chu nhat:
{0}", h.DienTich()); Console.ReadLine();
}
}
}
```

Trong ví dụ trên, ta định nghĩa một lớp các hình chữ nhật (HCN), mỗi đối tượng thuộc lớp này có thành phần dữ liệu là chiều dài và chiều rộng và có các phương thức như: nhap(), xuat(), DienTich(), ChuVi(). Sau đó, trong hàm Main() ta khai báo một đối tượng hình chữ nhật tên là h, cấp phát vùng nhớ cho đối tượng này và gọi thực hiện các phương thức của nó.

Chú ý:

Nếu ta bỏ đi từ khóa public đứng trước mỗi phương thức của lớp HCN thì hàm Main() sẽ không thể truy cập đến các phương thức của đối tượng h và trình biên địch sẽ báo lỗi vì khi đó các phương thức này có mức độ truy cập là private.

Bài tập 1: xây dựng lớp hình chữ nhật với thành phần dữ liệu là tọa độ góc trên bên trái (x1, y1), tọa độ góc dưới bên phải (x2, y2) và các phương thức tính chiều dài, chiều rộng, diện tích, chu vi của hình chữ nhật và phương thức vẽ hình chữ nhật bằng các ký tự '*' ra màn hình.

CÂU HỎI VÀ BÀI TẬP

Bài 1: Xây dựng lớp diem với các thuộc tính tung độ, hoành độ của điểm đó, phương thức đổi tọa độ giữa dương và âm, phương thức di chuyển theo một giá trị nhập vào từ bàn phím, phương thức hiện điểm lên màn hình.

Bài 2: Xây dựng lớp diem như bài 1 sau đó viết chương trình nhập tọa độ của điểm từ bàn phím, di chuyển một tọa độ, lấy tọa độ đối xứng, hiện tọa độ của điểm lên màn hình.

Bài 3: Xây dựng chương trình nhập tam giác, tính chu vi, diện tích và in ra màn hình đó là loại tam giác nào: cân, vuông, vuông cân, đều hay thường.

Bài 4: Viết chương trình C# để nhập một mảng, sau đó sắp xếp mảng theo thứ tự giảm dần và in mảng đã sắp xếp trên màn hình.

Bài 5: Viết chương trình C# để nhập một mảng, sau đó tìm phần tử nhỏ thứ hai rồi in kết quả trên màn hình.

BÀI 4 PHƯƠNG THỨC KHỞI TẠO, HỦY TẠO, CÁC THÀNH PHẦN TĨNH

NỘI DUNG CHÍNH

- Phương thức khởi tạo, hủy tạo
- Các thành phần tĩnh trong C#

4.1. PHƯƠNG THỨC KHỞI TẠO

Phương thức khởi tạo của một đối tượng có các tính chất sau:

- Được gọi đến một cách tự động khi một đối tượng của lớp được tạo ra. Dùng để khởi động các giá trị đầu cho các thành phần dữ liệu của đối tượng thuộc lớp.
- Tên phương thức giống với tên lớp và có mức độ truy cập là public.
- Không có giá trị trả về.

Trước khi phương thức khởi tạo chạy, đối tượng chưa thực sự tồn tại trong bộ nhớ, sau khi khởi tạo hoàn thành, bộ nhớ lưu trữ một thể hiện hợp lệ của lớp.

Khi ta không định nghĩa một phương thức khởi tạo nào cho lớp, trình biên dịch sẽ tự động tạo một phương thức khởi tạo mặc định cho lớp đó và khởi tạo các biến bằng các giá trị mặc định.

Thông thường ta nên định nghĩa một phương thức khởi tạo cho lớp và cung cấp tham số cho phương thức khởi tạo để khởi tạo các biến cho đối tượng của lớp.

Ví dụ 4.1: Sử dụng phương thức khởi tạo cho lớp SanPham

class SanPham

```
{
    private string ten;
    private decimal gia;
    // Khai báo phương thức khởi tạo với 2 tham số
    public SanPham(string tenSanPham, decimal giaSanPham)
    {
        ten = tenSanPham;
        gia = giaSanPham;
    }
    // Khai báo phương thức khởi tạo không tham số
    public SanPham()
    {
        ten = "Không tên";
        gia = 0;
    }
    // Thuộc tính Ten lấy hoặc thiết lập tên sản phẩm
    public string Ten
    {
        set { ten = value;}
        get { return ten;}
    }
}
```

Chú ý rằng, nếu lớp có phương thức khởi tạo có tham số thì khi khởi tạo đối tượng (bằng toán tử new) ta phải truyền tham số cho phương thức khởi tạo theo cú pháp:

TênBiếnĐốiTượng = new TênLớp(DanhSáchĐốiSố)

Ví dụ 4.2: Chương trình nhập chiều dài, chiều rộng của hình chữ nhật và xuất ra diện tích, chu vi của hình chữ nhật.

```
class HCN
    {
        protected float Dai, Rong;
        public float ChuVi()
        {
            return (Dai + Rong) * 2;
        }
        public float DienTich()
        {
            return Dai * Rong;
        }
        public void Nhap()
        {
            Console.WriteLine("Nhap chieu dai: ");
            Dai = float.Parse(Console.ReadLine());
            Console.WriteLine("Nhap chieu rong: ");
            Rong = float.Parse(Console.ReadLine());
        }
        public void Xuat()
        {
            Console.WriteLine("Hinh chu nhat: Dai = {0},Rong = { 1}", Dai, Rong);
        }
        static void Main(string[] args)
        {
            HCN h;
```

```
        h = new HCN();
        h.Nhap();
        h.Xuat();
        Console.WriteLine("Chu vi hinh chu nhat: {0}", h.ChuVi());
        Console.WriteLine("Dien tich hinh chu nhat: {0}", h.DienTich());
        Console.ReadLine();
    }
```

4.2. PHƯƠNG THỨC HỦY TẠO

Phương thức hủy tạo (destructor) hoạt động ngược lại với phương thức khởi tạo, dùng để thu hồi đối tượng.

Đặc điểm:

- Có tên trùng với tên lớp nhưng để dễ phân biệt với phương thức khởi tạo thì thêm dấu "~" vào trước tên hàm
- Không có kiểu trả về
- Được tự động gọi khi một đối tượng thuộc lớp kết thúc "vòng đời" của nó thông qua bộ thu dọn rác tự động GC (Garbage Collection)
- Nếu không khai báo phương thức hủy tạo thì C# sẽ tự động tạo ra 1 destructor mặc định và không có nội dung gì.
- Phương thức hủy tạo không chấp nhận bất kỳ tham số nào và không được sửa đổi phương thức hủy tạo
- Phương thức hủy tạo không thể được định nghĩa trong cấu trúc. Phương thức hủy tạo chỉ được sử dụng với các lớp
- Phương thức hủy tạo không thể bị *overload* hoặc kế thừa
- Chỉ có một phương thức hủy tạo duy nhất trong một lớp

Ví dụ 4.3: Khai báo phương thức hủy tạo cho Lớp People

```
using System;
namespace ConsoleApp1
```

```
{
    class People
    {
        public People()
        {
            Console.WriteLine("\n---Goi phuong thuc khoi tao---");
        }
        ~People()
        {
            Console.WriteLine("\n---Goi phuong thuc huy tao---");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            People p = new People();
            Console.ReadKey();
        }
    }
}
```

4.3. CÁC THÀNH PHẦN TĨNH

Dữ liệu và phương thức của một lớp có thể là thành viên thuộc thể hiện của lớp (đối tượng) hoặc thành viên tĩnh (có từ khóa *static* đứng trước). Thành viên thể hiện được kết hợp riêng với từng đối tượng của lớp. Thành viên tĩnh (biến, phương thức) được coi là phần chung của các đối tượng trong cùng một lớp. Mọi đối tượng thuộc lớp đều có thể truy cập thành viên tĩnh. Nói cách khác,

các thành viên thể hiện được xem là toàn cục trong phạm vi từng đối tượng còn thành viên tĩnh được xem là toàn cục trong phạm vi một lớp.[2]

Việc truy cập đến thành viên tĩnh phải thực hiện thông qua tên lớp (không được truy cập thành viên tĩnh thông qua đối tượng) theo cú pháp:

TênLớp.TênThànhViênTĩnh

Chú ý: Phương thức tĩnh thao tác trên các dữ liệu tĩnh và không thể truy cập trực tiếp các thành viên không tĩnh.

Ngoài ra, ta có thể định nghĩa một phương thức khởi tạo tĩnh, phương thức này dùng để khởi gán giá trị cho biến tĩnh của lớp và sẽ chạy trước khi thể hiện của đầu tiên lớp được tạo. Phương thức khởi tạo tĩnh hữu dụng khi chúng ta cần cài đặt một số công việc mà không thể thực hiện được thông qua chức năng khởi dựng và công việc cài đặt này chỉ được thực hiện duy nhất một lần.

Ví dụ 4.4 : Nhập vào một danh sách Nhân viên và cho biết tổng lương của các nhân viên vừa nhập.

```
using System;
public class Nhanvien
{
string hoten;
double luong;
public static double tl=0;
public void nhap()
{
Console.Write("Ho ten:"); hoten = Console.ReadLine();
Console.Write("Luong:"); luong = Convert.ToDouble(Console.ReadLine());
tl = tl + luong;
}
public void hien(int i)
{
```

```
Console.WriteLine("{0}\t{1}\t{2}", i, hoten, luong);
}
}
public class DsNhanvien
{
int n;
Nhanvien[] ds;
public void nhap()
{
Console.Write("Nhap so nhan vien:");
n = Convert.ToInt16(Console.ReadLine());
ds=new Nhanvien[n];
for (int i = 0; i < n; ++i)
ds[i] = new Nhanvien();
Console.WriteLine("Nhap thong tin cho cac nhan vien");
for (int i = 0; i < n; ++i)
ds[i].nhap();
}
public void hien()
{
for (int i = 0; i < n; ++i)
ds[i].hien(i + 1);
}
}
public class ThanhPhanTinh
{
```

```
static void Main()
{
DsNhanvien c = new DsNhanvien();
c.nhap();
Console.WriteLine("\t\t\tDanh sach cac nhan vien la\n");
Console.WriteLine("STT\tHo va ten\tLuong");
c.hien();
Console.WriteLine("Tong luong cua cac nhan vien la:{0}", Nhanvien.tl);
}
}
```

CÂU HỎI VÀ BÀI TẬP

Bài 1: Tạo một lớp phân số có tử số và mẫu số. Yêu cầu:

- Khai báo lớp Phân số
- Cài đặt các constructor
- Viết hàm cộng 2 phân số

Bài 2: Xây dựng lớp Nhanvien để quản lý nhân viên bao gồm mã nhân viên, chức vụ của nhân viên và họ tên nhân viên. Trong lớp sử dụng thành phần tĩnh như biến toàn cục trong phạm vi lớp theo dõi khi một đối tượng được tạo ra (bằng cách cho biết số thứ tự của nhân viên mới đó).

Bài 3: Viết chương trình xây dựng đối tượng Sinh viên gồm có các thuộc tính sau:

- Mã sinh viên là số nguyên.
- Họ tên: chuỗi ký tự.
- Địa chỉ: chuỗi ký tự.
- Số điện thoại: là số bao gồm 7 chữ số.

Các thuộc tính khai báo *private*, định nghĩa các phương thức *get/set* cho từng thuộc tính

Viết các *constructor* để khởi tạo đối tượng (*constructor* mặc định, *constructor* có tham số).

BÀI 5 THỰC HÀNH 1

NỘI DUNG CHÍNH

- Xây dựng các lớp, đối tượng
- Sử dụng được các phương thức khởi tạo, hủy tạo và các thành phần tĩnh

Bài 5.1 : Viết chương trình tạo một lớp SinhVien có các thuộc tính: masv, ten, diemTk (mã sinh viên, họ tên, điểm tổng kết). Xây dựng hàm khởi tạo, hàm hủy, nhập xuất cho đối tượng thuộc lớp SinhVien.

a. Hướng dẫn

- Các thuộc tính gồm có:
  - String masv; //Mã sinh viên
  - String hoten; //Họ tên
  - Float diemTK; //Điểm tổng kết
- Các phương thức gồm có:
  - Khởi tạo
  - Hủy tạo
  - Nhập thông tin sinh viên
  - Xuất thông tin sinh viên

b. Bài giải tham khảo

```
class SinhVien
  {
```

```
        private string masv;
        private string hoten;
        private float diemTK; //Diem Tong Ket
        public SinhVien()
        {
            masv = "";
            hoten = "";
            diemTK = 0;
        }
~SinhVien()
{
Console.Writeline(''Doi tuong da duoc huy'') ;
Console.ReadKey() ;
}
        public SinhVien(string msv, string hoten, float diem)
        {
            this.masv = msv;
            this.hoten = hoten;
            this.diemTK = diem;
        }
        public void Nhap()
        {
            Console.Write("Nhap ma sinh vien: ");
            masv = Console.ReadLine();
            Console.Write("Nhap ho ten sinh vien: ");
            hoten = Console.ReadLine();
```

```
        Console.Write("Nhap diem: ");
        diemTK =float.Parse( Console.ReadLine());
    }
        public void Xuat()
    {
        Console.WriteLine(masv);
        Console.WriteLine(hoten);
        Console.WriteLine(diemTK);
    }
    class program
    {
        static void Main(string[] args)
        {
            SinhVien sv = new SinhVien();
            sv.Nhap();
            sv.Xuat();
            Console.ReadKey();
        }
    }
```

Bài 5.2 : Xây dựng lớp diem với các thuộc tính: tung độ, hoành độ của điểm đó, và các phương thức sau: phương thức đổi tọa độ giữa dương và âm, phương thức di chuyển theo một giá trị nhập vào từ bàn phím, phương thức hiện điểm lên màn hình.

a. Hướng dẫn

- Các thuộc tính gồm có:
  - int x; //*hoành độ*
  - int y; //*tung độ*

- Các phương thức gồm có:
  - Nhập thông tin
  - Đổi tọa độ
  - Di chuyển điểm
  - Hiển thị thông tin

b. Bài giải tham khảo

```
class Diem
    {
      private int x, y;
    public void Nhap()
    {
       Console.WriteLine("Nhap toa do cua diem:");
       x = int.Parse(Console.ReadLine());
       y = int.Parse(Console.ReadLine());
    }
    public void Move(int dx, int dy)
    {
       x += dx;
       y += dy;
    }
    public void Chuyen()
    {
       x = -x;
       y = -y;
    }
    public void Xuat()
```

```
{
    Console.Write("toa do :(");
    Console.Write("{0},{1}", x, y);
    Console.WriteLine(")");
}
class program
{
    static void Main(string[] args)
    {
        Diem diem = new Diem();
        diem.Nhap();
        diem.Xuat();
        Console.ReadKey();
    }
}
}
```

Bài 5.3: Khai báo mảng một chiều với n nhập từ bàn phím, xây dựng các phương thức tĩnh nhập, xuất, tính tổng cho mảng đó.

a. Hướng dẫn

- Các thuộc tính gồm có:
    - static int[]a;
    - static int n;
- Các phương thức gồm có:
    - Nhập mảng
    - Xuất mảng
    - Tính tổng mảng

b. Bài giải tham khảo

```
class Mang1Chieu
 {
   static int[] a = new int[100];
   static int n = 0;
   static void Nhap()
   {
     Console.Write("Nhap vao chieu dai cua mang n = ");
     n = int.Parse(Console.ReadLine());
     for (int i = 0; i < n; i++)
     {
       Console.Write("a[{0}] = ", i);
       a[i] = int.Parse(Console.ReadLine());
     }
   }
   static int TinhTong()
   {
     int sum = 0;
     int i = 0;
     while (i < n)
       sum += a[i++];
     return sum;
   }
   static void Xuat()
   {
     Console.Write("Mang vua nhap la ");
     for (int i = 0; i < n; i++)
```

```
            Console.Write("{0} ", a[i]);
        }
        static void Main(string[] args)
        {
            Nhap();
            //Chúng ta cũng có thể gọi phương thức tĩnh như sau
            Mang1Chieu.Xuat();
            Console.WriteLine("Tong cac phan tu mang la " +
TinhTong().ToString());
            Console.ReadLine();
        }
    }
```

BÀI 6 KẾ THỪA

NỘI DUNG CHÍNH

- Khái niệm kế thừa, các kiểu kế thừa
- Các kĩ thuật trong kế thừa

6.1. CÁC KIỂU KẾ THỪA

Kế thừa là cơ chế cho phép định nghĩa một lớp mới - còn gọi là lớp dẫn xuất (*drived class*) dựa trên một lớp đã có sẵn (còn gọi là lớp cơ sở, *base class*). Lớp dẫn xuất có hầu hết các thành phần giống như lớp cơ sở (bao gồm tất cả các phương thức và biến thành viên của lớp cơ sở, trừ các phương thức *private*, phương thức khởi tạo, phương thức hủy và phương thức tĩnh). Nói cách khác, lớp dẫn xuất sẽ kế thừa hầu hết các thành viên của lớp cơ sở. [2]

Một điều cần chú ý rằng, lớp dẫn xuất vẫn được kế thừa các thành phần dữ liệu *private* của lớp cơ sở nhưng không được phép truy cập trực tiếp (truy cập gián tiếp thông qua các phương thức của lớp cơ sở).

Cú pháp định nghĩa lớp dẫn xuất:

```
class TênLớpCon : TênLớpCơSở
{
// Thân lớp dẫn xuất
}
```

Có 2 loại kế thừa:

- Kế thừa đơn (*Single Inheritance*): Mỗi lớp chỉ được kế thừa từ một lớp cơ sở.

- Đa kế thừa (*Multi Inheritance*): Cho phép kế thừa nhiều giao diện (*Interface*).

6.2. CÁC KĨ THUẬT TRONG KẾ THỪA

6.2.1. Kế thừa phương thức khởi tạo, phương thức hủy

- Lớp dẫn xuất không thừa kế phương thức khởi tạo của lớp cơ sở bởi vì chúng ta không thể xác định chính xác cách mà các đối tượng của lớp dẫn xuất được khởi tạo như nào (giả sử phương thức khởi tạo được thừa kế và phương thức khởi tạo đó thực hiện xử lý hay khởi tạo trên lớp cha, mà phương thức khởi tạo dùng để tạo đối tượng nên lớp thừa kế mà không định nghĩa lại phương thức khởi tạo thì đối tượng tạo ra có thể lỗi, nên kế thừa phương thức khởi tạo có nghĩa là lớp dẫn xuất được phép gọi đến phương thức khởi tạo của lớp cơ sở).

- Cả lớp cơ sở và lớp dẫn xuất đều chứa phương thức khởi tạo mặc định. Tạo đối tượng lớp dẫn xuất thì cả hai phương thức khởi tạo đó ở cả hai lớp đều được thực thi và phương thức khởi tạo ở lớp cơ sở được thực thi trước.

- Nếu cả hai lớp có phương thức khởi tạo có tham số mà ở lớp cơ sở chúng ta không chỉ định phương thức khởi tạo mặc định thì trình biên dịch sẽ báo lỗi vì C# nó coi lớp đó có thể tự tạo đối tượng mới nên sẽ không tạo phương thức khởi tạo mặc định mà khi tạo đối tượng ở lớp dẫn xuất thì phương thức khởi tạo mặc định của lớp cơ sở luôn được gọi nên sẽ báo lỗi. Để khắc phục lỗi đó chúng ta sử dụng từ khóa **base.**

Đối với các phương thức hủy, khi đối tượng hủy nó sẽ thi hành phương thức hủy của lớp kế thừa trước, rồi mới đến phương thức hủy của lớp cơ sở (ngược với phương thức khởi tạo).

```
class A {
   public A() {
      Console.WriteLine("A Init");
   }
}
class B : A {
   public B()
```

```
    {
        Console.WriteLine("B Init");
    }
}
```

Khi sử dụng:

```
new B();
// KẾT QUẢ
// A Init
// B Ini
```

Tuy nhiên, khi phương thức khởi tạo lớp cơ sở có tham số, hoặc ấn định một phương thức khởi tạo của lớp cơ sở (nếu lớp cơ sở có quá tải nhiều phương thức khởi tạo), thì phương thức khởi tạo của lớp kế thừa phải chỉ định sẽ khởi chạy phương thức khởi tạo (và truyền tham số) nào của lớp cơ sở.

```
class A {
    public A(string mgs) {
        Console.WriteLine("A Init" + mgs);
    }
}
class B : A {
    public B(string abc) : base(abc)
    {
        Console.WriteLine("B Init");
    }
}
```

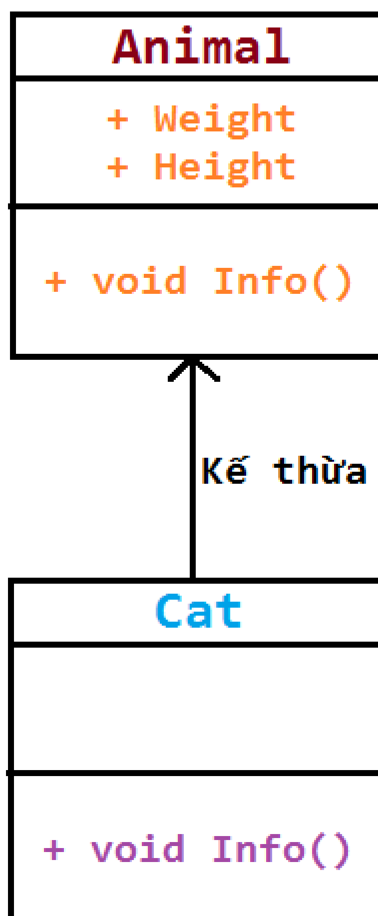
Sau phương thức tạo lớp kế thừa thấy có *base(abc)* đây chính là chỉ ra phương thức khởi tạo lớp cơ sở sẽ chạy, đó là hàm có một tham số - và giá trị tham số được truyền vào.

```
new B("!ABC");
// KẾT QUẢ
```

// A Init!ABC

// B Ini

6.2.2. Hàm trùng tên và cách gọi phương thức của lớp cha



Giả sử lớp *Animal* có phương thức Info(). Lớp *Cat* kế thừa lớp *Animal* nên cũng sẽ nhận được phương thức này.

Trong lớp *Cat* ta cũng định nghĩa một phương thức tên *Info*(), có kiểu trả về là *void* và không có tham số truyền vào. Vậy câu lệnh sau sẽ gọi phương thức Info() nào?

Cat BlackCat = new Cat();

BlackCat.Info();

Câu trả lời là C# sẽ gọi phương thức Info() của lớp Cat định nghĩa. Đồng thời cũng đưa ra một cảnh báo khi biên dịch.

Trong C# có hỗ trợ từ khoá *new* nhằm đánh dấu đây là 1 hàm mới và hàm kế thừa từ lớp cha sẽ bị che đi khiến bên ngoài không thể gọi được.

Cụ thể sẽ thêm từ khoá new vào trước khai báo hàm Info() trong lớp Cat.

```
public new void Info()
{
}
```

Khi đó hàm *Info*() của lớp cha sẽ bị che giấu đi. Và mọi đối tượng bên ngoài chỉ gọi được hàm *Info*() của lớp Cat. Từ khoá này làm tường minh khai báo của hàm *Info*(), và kết quả khi chạy chương trình sẽ không có thay đổi.

Chúng ta có thể sử dụng từ khoá *base* để đại diện cho lớp cha và gọi đến các thành phần của lớp cha.

Ví dụ 6.1: Gọi hàm Info() của lớp cha:

```
/* Từ khoá new chỉ định đây là 1 hàm Info mới của lớp Cat */
    public new void Info()
    {
        Console.WriteLine(" Info of Cat: ");
        base.Info(); // gọi đến hàm Info() của lớp cha
    }
```

Kết quả khi gọi hàm **Info()** của lớp Cat là:

```
file:///C:/Users/HT/Documents/Visual Studio 201
Info of Cat:
Weight: 500 Height: 20 Legs: 2
```

6.2.3. Cấp phát vùng nhớ cho đối tượng

Bình thường nếu như 1 đối tượng kiểu Animal không thể khởi tạo vùng nhớ có kiểu Cat được.

```
Animal cat = new Cat();
```

Câu lệnh này sẽ báo lỗi: "không thể chuyển từ kiểu Cat sang kiểu Animal".

Nhưng nếu như 2 lớp này có quan hệ kế thừa thì điều này hoàn toàn được.

Tính chất thể hiện tính đa hình trong lập trình hướng đối tượng được phát biểu như sau:

"Một đối tượng thuộc lớp cha có thể tham chiếu đến vùng nhớ của đối tượng thuộc lớp con nhưng ngược lại thì không".

Có nghĩa là nếu lớp Cat kế thừa từ lớp Animal thì câu lệnh Animal cat = new Cat(); hoàn toàn đúng nhưng ngược lại Cat cat = new Animal (); sẽ báo lỗi.

CÂU HỎI VÀ BÀI TẬP

Bài 1: Xây dựng phương trình bậc hai sau đó kế thừa giải phương trình trùng phương

Bài 2: Xây dựng lớp hình vuông, tính diện tích cho lớp đó. Dùng kĩ thuật thừa kế để tính thể tích cho một hình lập phương.

Bài 3: Xây dựng lớp Stack với các thuộc tính và phương thức cần thiết. Sau đó kế thừa để chuyển một số nguyên dương bất kì sang hệ đếm cơ số 2,8.

BÀI 7 THỰC HÀNH 2

NỘI DUNG CHÍNH

- Xây dựng được các lớp, đối tượng
- Thực hiện các kĩ thuật kế thừa dữ liệu

Bài 7.1 : Xây dựng lớp có tên là *luong* để tính lương cho cán bộ với các thông tin sau: Họ tên, lương cơ bản, hệ số lương, với các phương thức: khởi tạo không tham số dùng để khởi tạo lương cơ bản là 450, hệ số lương là 2,34 ; phương thức thiết lập hai tham số dùng để khởi tạo lương cơ bản, hệ số lương. Sau đó kế thừa lớp có tên là lương được xây dựng ở trên dùng để tính lương mới cho các cán bộ với việc bổ sung thêm hệ số phụ cấp, lương được tính lại như sau: *Lương=lương cơ bản*hệ số lương*hệ số phụ cấp*.

a. Hướng dẫn

- Các lớp đối tượng gồm có:
  - Lớp lương
  - Lớp lương mới (kế thừa từ lớp lương)
- Các thuộc tính gồm có:
  - string hoten; //*lớp lương*
  - static int luongcoban; //*lớp lương*
  - double hesoluong; //*lớp lương*
  - double hesophucap; // *thuộc lớp lương mới*
- Các phương thức gồm có:
  - Khởi tạo
  - Tính lương
  - Hiển thị dữ liệu

b. Bài giải tham khảo

```
class luong
 {
   private string hoten;
   private double hesoluong;
   private static int luongcoban;
   public void nhap()
   {
      Console.Write("Nhap ho ten: ");
      hoten = Console.ReadLine();
      Console.Write("He so luong: ");
      hesoluong=double.Parse(Console.ReadLine());
   }
   public luong()
   {
      luongcoban = 450;
      hesoluong = 2.34;
   }
   public luong(int lcb, float hsl)
   {
      hesoluong = hsl;
      luongcoban = lcb;
   }
   public void hien()
   {
      Console.WriteLine("Ho ten : {0}", hoten);
```

```
            Console.WriteLine("He so luong: {0}", hesoluong);
        }
        public double tinhluong()
        {
            return luongcoban * hesoluong;
        }
}
    class luongmoi : luong
    {
        private double hesophucap;
        public luongmoi(): base() // Gọi phương thức thiết lập tham số của lớp cơ sở
        {
            hesophucap=0.4;
        }
        public luongmoi(int lcb, float hsl, double hspc):base(lcb, hsl)
        {
            hesophucap = hspc;
        }
        public new void nhap()
        {
            base.nhap();
            Console.Write("He so phu cap:");
            hesophucap = double.Parse(Console.ReadLine());
        }
        public new void hien()
        {
```

```
            base.hien();
            Console.WriteLine("He so phu cap= {0}", hesophucap);
        }
        public new double tinhluong()
        {
            return base.tinhluong() * hesophucap;
        }
}
    class program
    {
        static void Main()
        {
            luongmoi A = new luongmoi();
            A.nhap();
            A.hien();
            Console.WriteLine("Luong moi:{0}", A.tinhluong());
            Console.ReadLine();
        }
    }
```

Bài 7.2 : Xây dựng lớp hình vuông, tính diện tích cho lớp đó. Dùng kĩ thuật thừa kế để tính thể tích cho một hình lập phương.

a. *Hướng dẫn*

- Các lớp đối tượng gồm có:
    - Lớp hình vuông
    - Lớp hình lập phương; *//kế thừa từ lớp hình vuông*
- Các thuộc tính gồm có:
    - protected float a; //cạnh hình vuông

- Các phương thức gồm có:
    - Khởi tạo hình vuông
    - Nhập hình vuông
    - Tính diện tích hình vuông
    - Hiển thị
    - Tính thể tích hình lập phương

b. Bài giải tham khảo

```
class hinhvuong
 {
   protected float a;
   public hinhvuong()
   {
     a = 2;
   }
   public hinhvuong(float x)
   {
     a = x;
   }
   public void nhap()
   {
     Console.Write("Nhap chieu dai canh cua hinh vuong:");
     a = float.Parse(Console.ReadLine());
   }
   public float dientich()
   {
     return a * a;
```

```
        }
        public void hien()
        {
            Console.WriteLine("Thong tin can hien thi");
            Console.WriteLine("Canh hinh vuong = {0}", a);
            Console.WriteLine("Dien tich cua hinh vuong= {0}", dientich());
        }
    }
    class hinhlp : hinhvuong
    {
        public hinhlp() : base()
        { }
        public hinhlp(float x) : base(x)
        { }
        public float thetich()
        {
            return base.dientich() * a;
        }
        public new void hien()
        {
            base.hien();
            Console.WriteLine("The tich cua hinh lap phuong = {0}", thetich());
        }
    }
    class program
    {
```

```
        static void Main()
        {
            hinhlp B = new hinhlp();
            B.nhap();
            B.hien();
            Console.ReadKey();
        }
    }
```

Bài 7.3 : Xây dựng các lớp, phương thức để tính chi phí xây dựng của một mảnh đất. Biết rằng mảnh đất có các thông tin : chiều dài, chiều rộng ; chi phí xây dựng là 80.

a. Hướng dẫn

- Các lớp đối tượng bao gồm:
    - Lớp hình chữ nhật;
    - Lớp tính chi phí xây dựng;
- Các thuộc tính gồm có:
    - Protected double chieudai;
    - Protected double chieurong;
    - Private double chiphi; //thuộc lớp chi phí xây dựng
- Các phương thức gồm có:
    - Khởi tạo
    - Tính diện tích
    - Hiển thị
    - Tính chi phí xây dựng

b. Bài giải tham khảo

```
class HinhChuNhat
    {
```

```
    //cac bien thanh vien
    protected double chieu_dai;
    protected double chieu_rong;
    // constructor
    public HinhChuNhat(double l, double w)
    {
        chieu_dai = l;
        chieu_rong = w;
    }
    //phuong thuc
    public double tinhDienTich()
    {
        return chieu_dai * chieu_rong;
    }
    public void HienThi()
    {
        Console.WriteLine("Chieu dai: {0}", chieu_dai);
        Console.WriteLine("Chieu rong: {0}", chieu_rong);
        Console.WriteLine("Dien tich: {0}", tinhDienTich());
    }
}
class ChiPhiXayDung : HinhChuNhat
{
    private double chiphi = 80;
    public ChiPhiXayDung(double l, double w) : base(l, w)
    { }
```

```
    public double tinhChiPhi()
    {
        double chi_phi;
        chi_phi = tinhDienTich() * chiphi;
        return chi_phi;
    }
    public void hienThiThongTin()
    {
        base.HienThi();
        Console.WriteLine("Chi phi: {0}", tinhChiPhi());
    }
}
class program
{
    static void Main()
    {
        Console.WriteLine("Tinh ke thua trong C#");
        Console.WriteLine("Khoi tao lop co so");
        Console.WriteLine("-------------------------\n");
        //tao doi tuong ChiPhiXayDung
        ChiPhiXayDung t = new ChiPhiXayDung(6, 9);
        t.hienThiThongTin();
        Console.ReadLine();
        Console.ReadKey();
    }}
```

BÀI 8 LỚP TRỪU TƯỢNG, LỚP KHÔNG CHO PHÉP KẾ THỪA

NỘI DUNG CHÍNH

- Lớp trừu tượng
- Cách ghi đè phương thức
- Lớp không cho phép kế thừa

8.1. LỚP TRỪU TƯỢNG

Lớp trừu tượng (*abstract class*) là lớp có mức độ trừu tượng cao dùng làm khuôn mẫu để tạo ra các lớp khác. [2]

Như vậy, *lớp* bình thường là khuôn mẫu để tạo ra các đối tượng (object), còn *lớp* trừu tượng lại dùng làm khuôn mẫu để tạo ra *lớp* khác. Sự khác biệt này dẫn đến tình huống là lớp trừu tượng không được sử dụng để tạo ra đối tượng như *lớp* bình thường.

Trong C#, lớp trừu tượng được xây dựng bằng cách thêm từ khóa ***abstract*** vào trước từ khóa *class* khi khai báo. Lớp trừu tượng không thể dùng để khởi tạo đối tượng mà chỉ đóng vai trò lớp cơ sở để tạo ra các lớp dẫn xuất, là những trường hợp cụ thể hơn.

Ví dụ 8.1: Khai báo lớp trừu tượng SinhVien như sau:

```
abstract class SinhVien // đây là một lớp trừu tượng
{
}
```

Lớp SinhVien không cho phép tạo đối tượng. Do đó, lệnh khởi tạo sau sẽ báo lỗi:

```
SinhVien sv = new SinhVien(); // lỗi! Lớp SinhVien không cho phép khởi tạo đối tượng
```

Ví dụ 8.2: Xây dựng lớp HinhHoc với phương thức tính chu vi, diện tích là phương thức trừu tượng hoặc phương thức ảo. Sau đó định nghĩa các lớp HinhChuNhat (hình chữ nhật), HinhTron (hình tròn) kế thừa từ lớp HinhHọc với các thành phần dữ liệu và phương thức tính chu vi, diện tích cụ thể của từng loại đối tượng.

```
// lop hinh hoc (truu tuong)
abstract public class HinhHoc
{
abstract public double DienTich();
virtual public double ChuVi() { return 0;}
}
// lop hinh tron ke thua tu lop hinh hoc
public class HinhTron : HinhHoc
{
double _bankinh;
public double BanKinh
{
get{ return _bankinh;} set{ _bankinh=value;}
}
public override double DienTich()
{
return  _bankinh*_bankinh*3.14;
}
public override double ChuVi()
{
return  _bankinh*2*3.14;
}
}
```

```
// lop hinh chu nhat ke thua tu lop hinh hoc
public class HinhChuNhat : HinhHoc
{
double _dai, _rong;
public double ChieuDai
{
get{ return _dai;}set{ _dai=value;}
}
public double ChieuRong
{
get{ return _rong;}set{ _rong=value;
public override double DienTich() { return _dai*_rong;}
public override double ChuVi() {return (_dai +_rong)*2 ;}
class Tester
{
static void Main(string[] args)
{
HinhHoc h;
HinhTron t=new HinhTron();t.BanKinh = 5;
Console.WriteLine("Thong tin ve hinh tron");h = t;
Console.WriteLine("Chu vi hinh tron: {0} ",h.ChuVi());
Console.WriteLine("Dien tich hinh tron:{0} ",h.DienTich());
HinhChuNhat n=new HinhChuNhat();n.ChieuDai = 4;
n.ChieuRong=3;h = n;
Console.WriteLine("Thong tin ve hinh chu nhat ");
Console.WriteLine("Chu vi hinh chu nhat:{0}",
h.ChuVi());
Console.WriteLine("Dien tich hinh chu nhat:{0}",h.DienTich());
```

```
Console.ReadLine();
}}
```

Trong lớp ***HinhHoc*** ở ví dụ trên, ta khai báo phương thức tính diện tích là một phương thức trừu tượng (không có phần cài đặt mã của phương thức vì chưa biết đối tượng hình thuộc dạng nào nên không thể tính diện tích của nó).

```
abstract public double DienTich();
```

Trong lớp trên, ta cũng khai báo một phương thức tính chu vi là phương thức ảo với mục đích minh họa rằng trong lớp trừu tượng có thể có phương thức ảo (hoặc bất cứ một thành phần nào khác của một lớp). Vì đây là một phương thức không trừu tượng nên phải có phần cài đặt mã bên trong thân phương thức.

Các lớp ***HinhChuNhat***, ***HinhTron*** kế thừa từ lớp ***HinhHoc*** nên chúng buộc phải cài đặt lại phương thức tính diện tích.

Trong hàm *Main()*, ta tạo ra đối tượng ***n*** thuộc lớp ***HinhChuNhat*** và đối tượng ***t*** thuộc lớp ***HinhTron*. Khi tham chiếu ***h*** thuộc lớp ***HinhHoc*** trỏ tới đối tượng ***n*** (tương tự với đối tượng ***t***), nó có thể gọi được phương thức tính diện tích của lớp HinhChuNhat (tương tự lớp HinhTron), điều này chứng tỏ phương thức trừu tượng cũng có thể dùng với mục đích đa hình.

Chú ý: Phân biệt giữa từ khóa new và override

- Từ khóa **override** dùng để định nghĩa lại (ghi đè) phương thức ảo (**virtual**) hoặc phương thức trừu tượng (**abstract)** của lớp cơ sở, nó được dùng với mục đích đa hình.
- Từ khóa **new** để che dấu thành viên của lớp cơ sở trùng tên với thành viên của lớp dẫn xuất.

8.2. GHI ĐÈ PHƯƠNG THỨC

- Trong C#, nếu lớp dẫn xuất định nghĩa cùng một phương thức đã được định nghĩa trong lớp cơ sở thì được gọi là phương thức ghi đè. Phương thức ghi đè được sử dụng để đạt được tính đa hình. Phương thức ghi đè cho phép cung cấp việc thực hiện cụ thể chức năng đã được lớp cơ sở của nó cung cấp.

- Để thực hiện ghi đè phương thức trong C #, chúng ta sử dụng từ khóa *virtual* với phương thức lớp cơ sở và từ khóa *override* cho phương thức lớp dẫn xuất.

- Override (ghi đè): được sử dụng khi muốn thay đổi hành vi (behavior) của một phương thức (method) ở lớp cha (base class) trong lớp con (derived class).

- *Override* chỉ xảy ra giữa các lớp có quan hệ kế thừa.

- Phương thức được *override* (ở lớp cha) và phương thức *override* (ở lớp con) phải giống hệt nhau ở cả 3 phần: kiểu dữ liệu trả về, tên phương thức và danh sách tham số.

- Chỉ có thể *override* các phương thức có quyền truy cập (access modifier) là *public* hoặc *protected*. Và nó không thể thay đổi được quyền truy cập của phương thức mà nó ghi đè.

- Từ khóa *override* được sử dụng để thực thi một *inherited method, property, indexer* hoặc *event* khi nó được dùng kèm với từ khóa *abstract* hoặc *virtual*.

- Không thể ghi đè một phương thức tĩnh, hoặc phương thức không phải là phương thức ảo.

- Không thể sử dụng các bổ từ truy cập như *new, static, virtual* cho phương thức ghi đè.

- Không thể ghi đè phương thức khởi tạo.

Ví dụ 8.3: Có một lớp động vật DongVat, có phương thức là TiengKeu() như sau:

```
class DongVat
{
  public virtual void TiengKeu()
  {
    Console.WriteLine( "Tiếng kêu động vật!!!" );
  }
```

```
}
```

– Giả sử chúng ta có một con thú nuôi, cụ thể chẳng hạn như là **ConCho** *kế thừa từ lớp động vật* DongVat, với tiếng kêu cụ thể là *''gâu gâu''* thì lúc này sẽ *override* (ghi đè) lại phương thức TiengKeu**()** của lớp DongVat:

```
class ConCho : DongVat
{
   public override void TiengKeu()
   {
      Console.WriteLine( "Gâu gâu." );
   }
}
```

– Và, có một con thú nuôi khác là **ConMeo** cũng *kế thừa từ lớp động vật* DongVat, nhưng không *override* (ghi đè) lại phương thức TiengKeu**()** của lớp DongVat:

```
class Cat : Animal
{
}
```

– Lúc này, nếu khởi tạo 2 đối tượng Cho và Meo thực hiện thông báo ra tiếng kêu của mỗi con:

```
DongVat cho = new ConCho();
DongVat meo = new ConMeo();
cho.TiengKeu();
meo. TiengKeu();
```

– Thì chúng ta sẽ được kết quả là:

Gâu gâu.

Tiếng kêu động vật!!!

– Như vậy, có thể thấy con *Cho* "ghi đè" lên hành động tiếng kêu của động vật thì sẽ được tiếng kêu của nó. Còn con *meo* không làm gì thì tiếng kêu của nó sẽ là tiếng kêu chung của lớp cha (lớp DongVat).

8.3. LỚP KHÔNG CHO PHÉP KẾ THỪA

Một lớp được chỉ định với từ khoá *sealed* là một lớp không cho phép kế thừa. Một phương thức được chỉ định với từ khoá *sealed* là một phương thức không cho ghi đè (*overriding*).[2]

Cú pháp khai báo lớp sealed

```
sealed class Class_Name
{
  //body of the class
}
```

Cú pháp khai báo phương thức sealed

```
access_modifier sealed override return_type Method_Name
{
  //body of the method
}
```

Ví dụ 8.4 : Khai báo phương thức sealed

```
using System;
namespace DemoSealed
{
   class Nguoi
   {
      public string maso;
      public string hoten;
      public string gioitinh;
      public Nguoi(string maso, string hoten, string gioitinh)
      {
         this.maso = maso;
         this.hoten = hoten;
         this.gioitinh = gioitinh;
      }
      public virtual void Nhap()
      {
         Console.Write("Nhap ma so");
```

```
            maso = Console.ReadLine();
            Console.Write("Nhap ho ten: ");
            hoten = Console.ReadLine();
            Console.Write("Nhap gioi tinh: ");
            gioitinh = Console.ReadLine();
        }
        public virtual void HienThi()
        {
            Console.WriteLine("Ma so: {0}, Ho ten: {1}, Gioi tinh: {2}", maso, hoten,
gioitinh);
        }
    }
    class NhanVien : Nguoi
    {
        private string bangcap;
        public NhanVien(string maso, string hoten, string gioitinh, string bangcap)
         : base(maso, hoten, gioitinh)
        {
            this.bangcap = bangcap;
        }
        // Phương thức này không được phép overriding ở lớp con
        public sealed override void Nhap()
        {
            base.Nhap(); // Gọi phương thức của lớp Nguoi
                        // Thêm xử lý cho thuộc tính bằng cấp
            Console.Write("Nhap thong tin bang cap: ");
            bangcap = Console.ReadLine();
        }
        public override void HienThi()
        {
            base.HienThi();
            Console.WriteLine("Bang cap: {0}", bangcap);
        }
    }
}
```

8.4. PHƯƠNG THỨC KHÔNG CHO PHÉP GHI ĐÈ

Các phương thức tĩnh và phương thức *non-virtual* sẽ không cho phép được ghi đè ở lớp con.

Ví dụ 8.5 :

```
class Animal
{
  public  static void Speak()
   {
      Console.WriteLine("Tieng keu dong vat");
   }
}
class Dog : Animal
{
   public override void Speak()
   {
      Console.WriteLine("Go go");
   }
}
class program
{
   static void Main()
   {
      Animal dog = new Dog();
      dog.Speak();
      Console.ReadKey();
   }
}
```

Ở ví dụ trên, phương thức *public static void Speak()* của lớp *Animal* sẽ không cho phép ghi đè ở lớp *Dog*.

CÂU HỎI VÀ BÀI TẬP

Bài 1: Xây dựng lớp Stack với các thao tác cần thiết. Từ đó hãy dẫn xuất từ lớp Stack để đổi một số nguyên dương n sang hệ đếm 16.

Bài 2: Viết chương trình quản lí nhân sự và tính lương cho nhân viên trong công ty.

1. Quản lí thông tin nhân viên (Họ tên, ngày sinh, địa chỉ)

2. Tính lương cho nhân viên.

Biết trong công ty có ba loại nhân viên và cách tính lương như sau:

- Nhân viên sản xuất: số sản phẩm * 20000 đ
- Nhân viên công nhật: số ngày công * 50000 đ
- Nhân viên quản lí: hệ số lương * lương cơ bản

Bài 3: Viết chương trình tính giá điện của một căn hộ :

Phương thức thiết lập 2 tham số dung khởi tạo giá điện (=750 vnd) và số điện đã dùng.

Phương thức nhập cho từng lớp

Phương thức hiện cho từng lớp

Sau đó kế thừa lớp cơ sở để tính tiền điện mới cho hợp với việc nếu số điện dùng quá 100 số điện thì giá điện kể từ số 100 trở đi là 1000 vnd. Còn nếu số điện sử dụng lớn hơn 50 và nhỏ hơn 100 thì giá điện là 800, Số diện sử dụng nhỏ hơn 50 thì giá điện là 750.

BÀI 9 THỰC HÀNH 3

NỘI DUNG CHÍNH

- Xây dựng các lớp, các đối tượng
- Giải các bài toán theo hướng trừu tượng

Bài 9.1: Xây dựng một lớp học sinh bao gồm các thuộc tính: họ tên, điểm toán, điểm lí, phương thức nhập, hiển thị. Lớp học sinh được khai báo là lớp không cho phép kế thừa. Sau đó xây dựng một lớp sinh viên kế thừa các thuộc tính và phương thức của lớp học sinh, ngoài ra lớp sinh viên còn có thêm thuộc tính điểm hoá. Hãy thực hiện chương trình trên và xem chương trình sẽ xảy ra điều gì?

c. Hướng dẫn

- Các lớp đối tượng gồm có:
    - Lớp học sinh; *// là lớp không cho phép kế thừa*
    - Lớp sinh viên;
- Các thuộc tính gồm có:
    - string hoten;
    - float diemtoan, diemly;
    - float diemhoa; // thuộc lớp sinh viên
- Các phương thức gồm có:
    - Nhập
    - Xuất

d. Bài giải tham khảo

sealed class hocsinh

```
{
    private string hoten;
    private float diemtoan, diemly;
    public void nhap()
    {
        Console.Write("Nhap ho ten hoc sinh: ");
        hoten = Console.ReadLine();
        Console.Write("Diem toan: ");
        diemtoan = float.Parse(Console.ReadLine());
        Console.Write("Diem li: ");
        diemly = float.Parse(Console.ReadLine());
    }
    public void hien()
    {
        Console.WriteLine("Thong tin can hien thi");
        Console.WriteLine("Ho ten :{0}", hoten);
        Console.WriteLine("Diem toan: {0}", diemtoan);
        Console.WriteLine("Diem li: {0}", diemly);
    }
}
class sinhvien : hocsinh
{
    private float dhoa;
    public new void nhap()
    {
        base.nhap();
```

```
            Console.Write("Nhap diem hoa:");
            dhoa = float.Parse(Console.ReadLine());
        }
        public new void hien()
        {
            base.hien();
            Console.WriteLine("Diem hoa: {0}", dhoa);
        }
    }
    class program
    {
        static void Main()
        {
            sinhvien a = new sinhvien();
            a.nhap();
            a.hien();
            Console.ReadLine();
        }
    }
```

Bài 9.2: Công ty Honda có 5 phân xưởng sản xuất oto và xe máy (mỗi phân xưởng chỉ sản xuất một dòng sản phẩm là oto hoặc xe máy). Viết chương trình thực hiện các công việc sau: nhập vị trí, tên của phân xưởng; sau quá trình nhập dữ liệu, người sử dụng nhập vị trí và chương trình đưa ra vị trí và tên của phân xưởng. Xây dựng bài toán trên theo lớp trừu tượng.

a. Hướng dẫn

- Các lớp đối tượng gồm có:
    - Lớp phân xưởng; // sử dụng *abstract* để tạo lớp trừu tượng

  - Lớp ô tô
  - Lớp xe máy
- Các thuộc tính gồm có:
  - Vị trí của phân xưởng
  - Tên phân xưởng
- Các phương thức gồm có:
  - Khởi tạo
  - Phương thức nhập, xuất cho các phân xưởng

b. Bài giải tham khảo

```
public abstract  class PhanXuong
 {
   protected int vitri;
   protected string tenPX;
   public abstract void nhap(int v);
   public abstract void xuat();
 }
 class oto : PhanXuong
 {
   public override void nhap(int v)
   {
     vitri = v;
     Console.WriteLine("\nNhap thong tin cho phan xuong san xuat o to");
     Console.WriteLine("Vi tri cua phan xuong: {0}", vitri);
     Console.Write("Nhap ten cua phan xuong: ");
     tenPX = Console.ReadLine();
   }
```

```
    public override void xuat()
    {
        Console.WriteLine("Day la phan xuong san xuat o to");
        Console.WriteLine("Vi tri cua phan xuong: {0}", vitri);
        Console.WriteLine("Ten cua phan xuong: {0}", tenPX);
    }
}
class xemay : PhanXuong
{
    public override void nhap(int v)
    {
        vitri = v;
        Console.WriteLine("\nNhap thong tin cho phan xuong xe may");
        Console.WriteLine("Vi tri cua phan xuong: {0}", vitri);
        Console.Write("Nhap ten cua phan xuong: ");
        tenPX = Console.ReadLine();
    }
    public override void xuat()
    {
        Console.WriteLine("Day la phan xuong san xuat xe may");
        Console.WriteLine("Vi tri cua phan xuong: {0}", vitri);
        Console.WriteLine("Ten cua phan xuong: {0}", tenPX);
    }
}
class program
{
```

```
static void Main(string[] args)
{
    PhanXuong[] ds = new PhanXuong[5];
    int i;
    for (i = 0; i < 5; ++i)
        ds[i] = null;
    ConsoleKeyInfo kt;
    i = 1;
    do
    {
     Console.Write("Phan xuong thu {0} san xuat oto (o) hay xe may(x): ", i);
        kt = Console.ReadKey();
        switch (char.ToUpper(kt.KeyChar))
        {
            case 'O':
                ds[i] = new oto();
                ds[i].nhap(i);
                break;
            case 'X':
                ds[i] = new xemay();
                ds[i].nhap(i);
                break;
        }
        i++;
    }
    while (i < 6);
```

```
        int vitri;
        Console.Write("Nhap vao vi tri phan xuong can kiem tra");
        vitri = int.Parse(Console.ReadLine());
        ds[vitri].xuat();
        Console.ReadKey();
      }
  }
```

Bài 9.3 : Xây dựng lớp Stack với các thuộc tính và phương thức cần thiết. Sau đó kế thừa để chuyển một số nguyên dương bất kì sang hệ đếm cơ số 2 hoặc 8.

a. Hướng dẫn

- Các lớp đối tượng gồm có:
  - Lớp Stack
  - Lớp đổi số
- Các thuộc tính gồm có:
  - Int top;
  - Int []s;
  - Int a,n; *// a là hệ đếm, n là số cần đổi*
- Các phương thức gồm có:
  - Khởi tạo stack
  - Phương thức push, pop của stack
  - Nhập số
  - Đổi số
  - Hiển thị kết quả

b. Bài giải tham khảo

```
class Stack
  {
```

```
private int top;
private int[] s;
public bool empty()
{
    return (top == -1);
}
public bool full()
{
    return (top >= s.Length);
}
public Stack()
{
    s = new int[20];
    top = -1;
}
public void push(int x)
{
    if (!full())
    {
        top = top + 1;
        s[top] = x;
    }
    else
        Console.Write("Stack tran");
}
public int pop()
```

```
    {
        if (empty())
        {
            Console.Write("Stack can");
            return 0;
        }
        else
            return s[top--];
    }
}
class doiso : Stack
{
    private int a, n; public void nhap()
    {
        Console.Write("Nhap vao so can doi:");
        n = Convert.ToInt32(Console.ReadLine());
        Console.Write("Nhap vao he can doi:");
        a = Convert.ToInt32(Console.ReadLine());
    }
    public void doi()
    {
        int du;
        while (n != 0)
        {
            du = n % a;
            push(du);
```

```
            n = n / a;
        }
    }
    public void hien()
    {
        while (!empty())
        {
            Console.Write("{0}", pop());
        }
    }
}
class program
{
    static void Main(string[] args)
    {
            doiso d = new doiso();
            d.nhap();
            d.doi();
            d.hien();
            Console.ReadKey();
        }
}
```

BÀI 10 TÍNH ĐA HÌNH

NỘI DUNG CHÍNH

- Khái niệm về tính đa hình
- Cách sử dụng đa hình tĩnh và đa hình động trong lập trình hướng đối tượng

10.1. KHÁI NIỆM

Tính đa hình cho phép một phương thức có các cách thể hiện khác nhau trên nhiều loại đối tượng khác nhau. Tính đa hình trong C# có thể là tĩnh (*static*) hoặc động (*dynamic*). [2]

10.2. ĐA HÌNH TĨNH

Trong đa hình tĩnh, phần phản hồi tới một hàm được xác định tại thời điểm biên dịch. Trong khi đó với đa hình động, nó được quyết định tại thời điểm chạy chương trình.

Kỹ thuật liên kết một hàm với một đối tượng trong thời gian biên dịch được gọi là ràng buộc sớm (*Early Binding*). C# cung cấp hai kỹ thuật để triển khai đa hình tĩnh:

- Nạp chồng hàm (*Function overloading*)
- Nạp chồng toán tử (*Operator overloading*)

10.2.1. Nạp chồng hàm trong C#

Ta có thể có nhiều định nghĩa cho cùng tên hàm trong cùng một phạm vi. Các định nghĩa này của hàm phải khác nhau: như kiểu và/hoặc số tham số trong danh sách tham số. Trong C#, ta không thể nạp chồng các khai báo hàm mà chỉ khác nhau ở kiểu trả về.

Ví dụ 10.1 : Minh họa cách sử dụng hàm **print()** để in các kiểu dữ liệu khác nhau trong C#:

```
public class TestCsharp
{
   void print(int i)
   {
      Console.WriteLine("In so nguyen: {0}", i);
   }
   void print(double f)
   {
      Console.WriteLine("In so thuc: {0}", f);
   }
   void print(string s)
   {
      Console.WriteLine("In chuoi: {0}", s);
   }
   static void Main(string[] args)
   {
      Console.WriteLine("Tinh da hinh trong C#");
      Console.WriteLine("--------------------------");
      //tao doi tuong TestCsharp
      TestCsharp p = new TestCsharp();
      // goi ham print()
      p.print(5);
      p.print(100.69);
      p.print("Huong dan hoc lap trinh C#");
      Console.ReadKey();
   }
```

}

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:

```
D:\Csharp\workspace\AppDemo\AppDemo\bin\Debug\...
Tinh da hinh trong C#
--------------------------
In so nguyen: 5
In so thuc: 100.69
In chuoi: Huong dan hoc lap trinh C#
```

10.2.2.Nạp chồng toán tử

Nạp chồng toán tử (*operator overloading*) trong C# là khả năng định nghĩa lại hoạt động của một số toán tử để có thể áp dụng nó với các đối tượng của lớp chúng ta tự định nghĩa. [2]

Đối với các kiểu dữ liệu số, C# định nghĩa sẵn một số phép toán như các phép toán số học, phép toán so sánh, phép toán tăng giảm. Đối với kiểu xâu (string), như chúng ta đã biết, được định sẵn phép toán cộng xâu.

Tuy nhiên, các kiểu dữ liệu (class) do người dùng định nghĩa lại không thể sử dụng ngay các phép toán đó được.

Ví dụ, nếu người dùng định nghĩa kiểu số phức, các phép toán cơ bản trên kiểu số phức lại không thể thực hiện được ngay, mặc dù về mặt toán học các phép toán đối với số phức không có gì khác biệt với kiểu số được C# định nghĩa.

Để giải quyết những vấn đề tương tự, C# cho phép *nạp chồng toán tử*, tức là cho phép định nghĩa lại những phép toán đã có với các kiểu dữ liệu do người dùng xây dựng.

Nạp chồng phương thức (*method overloading*) cùng với nạp chồng toán tử (*operator overloading*) là hai hiện tượng thuộc về nguyên lý đa hình tĩnh (*static polymorphism*).

Cách nạp chồng toán tử trong C#

Hãy cùng thực hiện và phân tích ví dụ sau để hiểu cách nạp chồng toán tử. Chú ý xem xét cú pháp nạp chồng đối với mỗi toán tử.

using *System;*

```
namespace P01_OperatorOverload
{
/// <summary>
/// lớp biểu diễn hình hộp
/// </summary>
internal class Box
{
public double Length { get; set; }
public double Breadth { get; set; }
public double Height { get; set; }
public Box() { }
public Box(double length, double breadth, double height)
{
Length = length;
Breadth = breadth;
Height = height;
}
/// <summary>
/// tính thể tích khối hộp
/// </summary>
public double Volume => Length * Breadth * Height;
// nạp chồng phép cộng
public static Box operator +(Box b, Box c)
{
Box box = new Box
{
Length = b.Length + c.Length,
Breadth = b.Breadth + c.Breadth,
```

```
Height = b.Height + c.Height
};
return box;
}
// nạp chồng phép so sánh bằng
public static bool operator ==(Box lhs, Box rhs)
{
bool status = false;
if (lhs.Length == rhs.Length && lhs.Height == rhs.Height
&& lhs.Breadth == rhs.Breadth)
{
status = true;
}
return status;
}
// nạp chồng phép so sánh khác
public static bool operator !=(Box lhs, Box rhs)
{
bool status = false;
if (lhs.Length != rhs.Length || lhs.Height != rhs.Height ||
lhs.Breadth != rhs.Breadth)
{
status = true;
}
return status;
}
// nạp chồng phép so sánh nhỏ hơn
public static bool operator <(Box lhs, Box rhs)
```

```
{
bool status = false;
if (lhs.Length < rhs.Length && lhs.Height < rhs.Height
&& lhs.Breadth < rhs.Breadth)
{
status = true;
}
return status;
}
// nạp chồng phép so sánh lớn hơn
public static bool operator >(Box lhs, Box rhs)
{
bool status = false;
if (lhs.Length > rhs.Length && lhs.Height >
rhs.Height && lhs.Breadth > rhs.Breadth)
{
status = true;
}
return status;
}
public override string ToString()
{
return string.Format("({0}, {1}, {2})", Length, Breadth, Height);
}
}
internal class Program
{
private static void Main(string[] args)
```

```
{
Box Box1 = new Box(6, 7, 5);
Box Box2 = new Box(12, 13, 10);
Box Box3 = new Box();
Box Box4 = new Box();
/* phép cộng hai hình hộp cho ra hình hộp khác có kích thước
* bằng tổng kích thước của hai hộp */
Box3 = Box1 + Box2;
Console.WriteLine("Box 3: {0}", Box3.ToString());
Console.WriteLine("Volume of Box3 : {0}", Box3.Volume);
// so sánh hai hình hộp
if (Box1 > Box2)
Console.WriteLine("Box1 lớn hơn Box2");
else
Console.WriteLine("Box1 không lớn hơn Box2");
if (Box3 == Box4)
Console.WriteLine("Box3 bằng Box4");
else
Console.WriteLine("Box3 không bằng Box4");
Console.ReadKey();
}
}
}
```

trong ví dụ trên chúng ta đã thực hiện nạp chồng cho phép toán cộng (+), các phép so sánh (bằng ==, khác !=, lớn hơn >, nhỏ hơn <).

Cú pháp khai báo này được tổng hợp lại dưới đây:

```
public static Box operator +(Box b, Box c) {...}
public static bool operator ==(Box lhs, Box rhs) {...}
```

```
public static bool operator !=(Box lhs, Box rhs) {...}
public static bool operator <(Box lhs, Box rhs) {...}
public static bool operator >(Box lhs, Box rhs) {...}
```

Đây đều là các phép toán nhị phân. Cách nạp chồng các phép toán này có cùng một cú pháp.

Mỗi loại phép toán sẽ có cách nạp chồng riêng. Tuy nhiên, cú pháp chung là

```
public static <return_type> operator <operator>(<parameters>) { ... }
```

Các toán tử có thể nạp chồng: +, -, !, ~, ++, –, +, -, *, /, %, ==, !=, <, >, <=, >=

Ngoài ra phép toán indexer cũng là một phép toán có thể nạp chồng.

Một số lưu ý khi nạp thực hiện chồng toán tử :

Các phép toán chia làm ba loại: unary (chỉ cần một toán hạng, như phép toán tăng ++, phép toán giảm –), binary (cần hai toán hạng, như các phép toán +,-,*,/), ternary (cần ba toán hạng, như phép toán điều kiện ?). Do đó, khi nạp chồng phép toán nào thì phải cung cấp đủ lượng tham số phù hợp. Ví dụ, khi nạp chồng phép toán binary (như +, -) thì phải cấp 2 tham số như đã làm ở trên.

Phép toán tăng giảm (++, –) thuộc loại unary nên trong danh sách tham số chỉ cần 1 tham số. Các phép toán này cũng không có giới hạn gì khi nạp chồng. Cùng ví dụ với lớp Box trên:

```
public static Box operator ++ (Box b)
{
return new Box(b.Length++, b.Breadth++, b.Height++);
}
```

Các phép toán số học (+, – *, /, %) không đặt ra giới hạn gì khi nạp chồng.

Có thể nạp chồng cùng một phép toán nhiều lần. Ví dụ, hoàn toàn có thể thêm nạp chồng phép + một lần nữa như sau:

```
public static Box operator +(Box b, double size)
{
return new Box(b.Length += size, b.Breadth += size, b.Height + size);
```

```
}
```

Ở đây đã nạp chồng phép cộng Box với một số thực. Điều kiện để nạp chồng phép toán nhiều lần là danh sách tham số của mỗi lần nạp chồng phải khác nhau.

Đối với các phép toán so sánh, phải thực hiện nạp chồng cả cặp. Nghĩa là, nếu nạp chồng phép so sánh bằng == thì đồng thời phải nạp chồng cả phép so sánh khác !=; nếu nạp chồng phép so sánh hơn > thì phải nạp chồng cả phép so sánh kém <.

Các phép gán (+=, -=, v.v.) không cho phép nạp chồng trực tiếp. Tuy nhiên, nếu đã nạp chồng phép +,-, v.v. thì các phép toán này tự nhiên sẽ được nạp chồng. Ví dụ, nếu đã nạp chồng phép cộng Box với 1 số như trên thì hoàn toàn có thể gọi lệnh :

```
var Box5 = Box4 += 5; // phép cộng gán với số
```

10.3. ĐA HÌNH ĐỘNG

C# cho phép tạo các lớp trừu tượng (*abstract*) mà được sử dụng để cung cấp trình triển khai cục bộ lớp của một giao diện (*Interface*). Trình triển khai (*Implementation*) được hoàn thành khi một lớp kế thừa kế thừa từ nó. Các lớp trừu tượng chứa các phương thức trừu tượng. [2]

Dưới đây là một số qui tắc về các lớp trừu tượng trong C#:

- Không thể tạo một sự thể hiện (*Instance*) của một lớp trừu tượng.
- Không thể khai báo một phương thức trừu tượng (*abstract*) ở bên ngoài một lớp trừu tượng.
- Khi một lớp được khai báo là *sealed*, nó không thể được kế thừa, các lớp trừu tượng không thể được khai báo là *sealed*.

Ví dụ 10.2: Minh họa một lớp trừu tượng: tạo 3 lớp có tên lần lượt là **Shape, HinhChuNhat, TestCsharp** như sau:

Lớp **Shape**: là một lớp *abstract*

```
abstract class Shape
 {
    public abstract int tinhDienTich();
```

```
    }
}
```

Lớp **HinhChuNhat**: là một lớp kế thừa lớp Shape

```
class HinhChuNhat : Shape
  {
    private int chieu_dai;
    private int chieu_rong;
    public HinhChuNhat(int a = 0, int b = 0)
    {
      chieu_dai = a;
      chieu_rong = b;
    }
    public override int tinhDienTich()
    {
      Console.WriteLine("Dien tich hinh chu nhat:");
      return (chieu_rong * chieu_dai);
    }
  }
}
```

Lớp **TestCsharp**: chứa phương thức **main()** để thao tác trên đối tượng HinhChuNhat

```
public class program
{
  static void Main(string[] args)
  {
    HinhChuNhat r = new HinhChuNhat(10, 8);
    double a = r.tinhDienTich();
    Console.WriteLine("Dien tich: {0}", a);
```

```
            Console.ReadKey();
        }
    }
```

Khi có một hàm được định nghĩa trong một lớp mà muốn được triển khai một lớp được kế thừa, chúng ta sử dụng hàm ***virtual*** trong C#. Các hàm *virtual* có thể được triển khai một cách khác nhau trong lớp được kế thừa khác nhau và việc gọi những hàm này sẽ được quyết định tại thời điểm biên dịch.

Đa hình động trong C# được triển khai bởi các lớp ***abstract*** và các hàm ***virtual***.

CÂU HỎI VÀ BÀI TẬP

Bài 1: Xây dựng một lớp hình, kế thừa lớp hình đó để tính diện tích cho các hình: Hình vuông, hình tam giác.

Bài 2: Viết chương trình tính diện tích và chu vi các hình: Hình chữ nhật biết hai cạnh, hình tam giác biết ba cạnh, hình tròn biết bán kính. Chương trình có giao diện như sau:

1. Chọn hình để tính (1-hình chữ nhật, 2-hình tam giac, 3-hình tròn);

2. Hiện thị diện tích và chu vi các hình đã nhập;

3. Kết thúc chương trình.

Lựa chọn công việc (1, 2, 3): Khi người sử dụng chọn 1 thì hỏi người sử dụng nhập kích thước cho hình nào, nhập xong hỏi người sử dụng có nhập tiếp không. Yêu cầu trong chương trình có cài đặt sự đa hình.

Bài 3: Xây dựng các loại đối tượng sách giáo khoa, tiểu thuyết, tạp chí. Viết chương trình cho phép quản lý một danh sách các loại đối tượng kể trên.

BÀI 11 THỰC HÀNH 4

NỘI DUNG CHÍNH

- Xây dựng các lớp, đối tượng
- Sử dụng tính đa hình để giải các bài toán

Bài 11.1: Xây dựng một lớp hình, kế thừa lớp hình đó để tính diện tích cho các hình: Hình tròn, hình chữ nhật.

a. Hướng dẫn

- Các lớp đối tượng gồm có:
  - Lớp hình
  - Lớp hình tròn
  - Lớp hình chữ nhật
- Các thuộc tính gồm có:
  - Bán kính r của hình tròn;
  - Chiều dài, chiều rộng của hình chữ nhật
- Các phương thức gồm có:
  - Khởi tạo
  - Tính diện tích; *// thuộc lớp hình là phương thức đa hình*
  - Phương thức ghi đè tính diện tích hình tròn;
  - Phương thức ghi đè tính diện tích hình chữ nhật;

b. Bài giải tham khảo

```
class hinh
  {
```

```
    public virtual double getArea()
    {
        return getArea();
    }
}
class hinhtron : hinh
{
    private float r;
    public hinhtron(float r)
    {
        this.r = r;
    }
    public override double getArea()
    {
        return r * r * 3.14;
    }
}
class hinhcn : hinh
{
    private float cd, cr;
    public hinhcn(float cd, float cr)
    {
        this.cd = cd;
        this.cr = cr;
    }
    public override double getArea()
```

```
        {
            return cd * cr;
        }
    }

    class program
    {
        static void Main(string[] args)
        {
            hinh H = new hinh();
            hinhcn hcn = new hinhcn(2, 3);
            hinhtron ht = new hinhtron(3);
            H = ht;
            Console.WriteLine("Dien tich hinh tron= {0}",
            H.getArea());
            H = hcn;
            Console.WriteLine("Dien tich hinh chu nhat= {0}",
            H.getArea());
            Console.ReadLine();
        }
    }
```

Bài 11.2: Xây dựng các lớp để quản lý thu nhập hàng tháng của một cơ quan, biết rằng:

Cơ quan có hai loại nhân viên được hưởng lương: biên chế thì hưởng lương theo quỹ lương của nhà nước, và hợp đồng thì hưởng lương theo số giờ làm việc.

Mỗi nhân viên trong công ty đều có các thông tin sau: Họ tên, số CMND, Phòng ban.

Biên chế: có thông tin riêng là Bậc lương.

Hợp đồng: có thông tin riêng là Số giờ, Tiền công một giờ.

Chương trình sử dụng tính đa hình cho phép nhập vào một danh sách các nhân viên theo biên chế, hợp đồng, tính tổng lương của các nhân viên thuộc dạng biên chế, hiển thị.

a. Hướng dẫn

- Các thuộc tính gồm có:
  - Họ tên;
  - Số CMND;
  - Phòng ban
- Các phương thức gồm có:
  - Khởi tạo
  - Nhập;
  - Xuất;

b. Bài giải tham khảo

```
public class NhanVien
{
    protected string HoTen, PhongBan;
    protected int SCMND;
    public NhanVien()
    {
        HoTen = "";
        SCMND = 0;
        PhongBan = "";
    }
    public NhanVien(string HoTen, int SCMND, string PhongBan)
    {
```

```
            this.HoTen = HoTen;
            this.SCMND = SCMND;
            this.PhongBan = PhongBan;
        }
        public void Nhap()
        {
            Console.Write("Nhap Ho va ten :");
            HoTen = Console.ReadLine();
            Console.Write("Nhap So CMND: ");
            SCMND = Int32.Parse(Console.ReadLine());
            Console.Write("Nhap Phong Ban: ");
            PhongBan = Console.ReadLine();
        }
        public void Xuat()
        {
            Console.WriteLine("Ho va ten: {0}", HoTen);
            Console.WriteLine("So CMND: {0}", SCMND);
            Console.WriteLine("Phong Ban: {0}", PhongBan);
        }
    }
    public class BienChe : NhanVien
    {
        double BacLuong, LuongCoBan;
        public BienChe() : base()
        {
            BacLuong = 0; LuongCoBan = 0;
```

```
        }
        public BienChe(string HoTen, int SCMND, string PhongBan, double
BacLuong,
        double LuongCoBan) : base(HoTen, SCMND, PhongBan)
        {
            this.BacLuong = BacLuong;
            this.LuongCoBan = LuongCoBan;
        }
        public new void Nhap()
        {
            base.Nhap();
            Console.Write("Nhap Bac luong: "); BacLuong =
Double.Parse(Console.ReadLine());
            Console.Write("Nhap Luong co ban: ");
            LuongCoBan = Double.Parse(Console.ReadLine());
        }
        public new void Xuat()
        {
            double Luong;
            Luong = (BacLuong * LuongCoBan);
            Console.WriteLine("Luong Bien Che cua nhan vien {0} la : {1}", HoTen,
            Luong);
        }
        public double tong()
        {
            double s;
            s = (LuongCoBan * BacLuong);
```

```
            return (s);
        }
    }
    public class HopDong : NhanVien
    {
        double SoGio, TienCong;
        public HopDong() : base()
        {
            SoGio = 0;
            TienCong = 0;
        }
        public HopDong(string HoTen, int SCMND, string PhongBan, double
SoGio, double
        TienCong) : base(HoTen, SCMND, PhongBan)
        {
            this.SoGio = SoGio;
            this.TienCong = TienCong;
        }
        public new void Nhap()
        {
            base.Nhap();
            Console.Write("Nhap so gio lam: ");
            SoGio = Double.Parse(Console.ReadLine());
            Console.Write("Nhap tien cong: ");
            TienCong = Double.Parse(Console.ReadLine());
        }
        public new void Xuat()
```

```
        {
            double LuongHD;
            LuongHD = SoGio * TienCong;
            Console.WriteLine("Luong Hop Dong cua nhan vien {0} la : {1}", HoTen,
            LuongHD);
        }
        public double TongHD()
        {
            double r;
            r = (SoGio * TienCong);
            return (r);
        }
    }
    class Cau2
    {
        static void Main(string[] args)
        {
            int n, p;
            Console.Write("Nhap vao so nhan vien bien che: ");
            n = Int32.Parse(Console.ReadLine()); Console.Write("Nhap vao so nhan
vien hop dong:");
            p = Int32.Parse(Console.ReadLine());
            Console.WriteLine();
            BienChe[] bc = new BienChe[n];
            HopDong[] hd = new HopDong[p];
            Console.WriteLine("Moi nhap du lieu cho nhan vien Bien Che");
            for (int i = 0; i < n; i++)
```

```
        {
            bc[i] = new BienChe();
            bc[i].Nhap();
        }
        Console.WriteLine();
        Console.WriteLine("Moi nhap du lieu cho nhan vien Hop Dong");
        for (int j = 0; j < p; j++)
        {
            hd[j] = new HopDong();
            hd[j].Nhap();
        }
        Console.WriteLine();
        double m, q;
        m = 0; q = 0;
        for (int i = 0; i < n; i++) bc[i].Xuat();
        for (int j = 0; j < p; j++) hd[j].Xuat();
        for (int i = 0; i < n; i++) m = m + bc[i].tong();
        for (int j = 0; j < p; j++) q = q + hd[j].TongHD();
        Console.WriteLine("Tong luong cua nhan vien bien che {0}", m);
        Console.WriteLine("Tong luong cua nhan vien Hop Dong {0}", q);
        Console.ReadKey();
    }
}
```

BÀI 12 GIAO DIỆN VÀ ĐA KẾ THỪA

NỘI DUNG CHÍNH

- Thiết lập một bản thiết kế của một lớp sử dụng giao diện (interface).
- Tính đa kế thừa

12.1. GIAO DIỆN

Ta dùng từ khóa ***interface*** để khai báo một giao diện với cú pháp sau:

[MứcĐộTruyCập] interface TênGiaoDiện [:CácGiaoDiệnCơSở]

{

Nội dung

}

MứcĐộTruyCập: public hoặc internal.

CácGiaoDiệnCơSở: danh sách các interface khác mà nó kế thừa.

Về mặt cú pháp, một giao diện giống như một lớp chỉ có phương thức trừu tượng. Nó có thể chứa khai báo của phương thức, thuộc tính, chỉ mục, sự kiện nhưng không được chứa biến dữ liệu. Khi kế thừa một giao diện ta phải thực thi mọi phương thức, thuộc tính,… của giao diện.

Chú ý:

- Mặc định tất cả các thành phần khai báo trong giao diện đều là *public*. Nếu có từ khóa *public* đứng trước sẽ bị báo lỗi. Các thành phần trong giao diện chỉ là các khai báo, không có phần cài đặt mã.
- Một lớp có thể kế thừa một lớp khác đồng thời kế thừa nhiều giao diện.

Ví dụ 12.1: Mọi chiếc xe hơi hoặc xe máy đều có hành động (phương thức) khởi động và dừng. Ta có thể dùng định nghĩa một giao diện kèm thêm một thuộc tính để cho biết chiếc xe đã khởi động hay chưa:

```
public interface IDrivable
{
void KhoiDong();void Dung(); bool DaKhoiDong
{
get;
}
}
```

Thuộc tính đã khởi động (*DaKhoiDong*) chỉ có phương thức ***get*** vì khi gọi phương thức ***KhoiDong()*** thì thuộc tính này sẽ có giá trị ***true***, khi gọi phương thức ***Dung()*** thì thuộc tính này sẽ có giá trị ***false***.

Khi đó, một lớp *Car* thực thi giao diện này phải cài đặt các phương thức và thuộc tính đã khai báo trong giao diện ***Idrivable*** trên:

```
public interface IDrivable
{
void KhoiDong();void Dung(); bool DaKhoiDong
{
get;
}
}
public class Car:IDrivable
{
public bool Started = false;public void KhoiDong()
{
Console.WriteLine("Xe ca khoi dong");Started = true;
}
public void Dung()
```

```
{
Console.WriteLine("Xe  ca  dung");Started = false;
}
public bool DaKhoiDong
{
get{return Started;}
}
}
public class Tester
{
public static void Main()
{
Car c = new Car();c.KhoiDong();
Console.WriteLine("c.DaKhoiDong=          "          +c.DaKhoiDong);
c.Dung();
Console.WriteLine("c.DaKhoiDong=          "          +c.DaKhoiDong);
Console.ReadLine();
}
}
```

12.2. ĐA KẾ THỪA

C# không hỗ trợ đa kế thừa lớp. Tuy nhiên, chúng ta có thể sử dụng giao diện (Interface) để thực hiện đa kế thừa.

Ví dụ 12.2 : Sử dụng *interface* để tính chi phí sơn cho diện tích của một phòng.

```
class Shape
   {
       protected int chieu_rong;
       protected int chieu_cao;
```

```
        public void setChieuRong(int w)
        {
            chieu_rong = w;
        }
        public void setChieuCao(int h)
        {
            chieu_cao = h;
        }
    }
public interface IChiPhiSon
    {
        int tinhChiPhi(int dien_tich);
    }
class HinhChuNhat : Shape, IChiPhiSon
    {
        public int tinhDienTich()
        {
            return (chieu_rong * chieu_cao);
        }
        public int tinhChiPhi(int dien_tich)
        {
            return dien_tich * 80;
        }
    }
public class program
    {
```

```
public static void Main(string[] args)
{
    Console.WriteLine("Vi du minh hoa Da ke thua");
    //tao doi tuong HinhChuNhat
    HinhChuNhat hcn = new HinhChuNhat();
    int dien_tich;
    hcn.setChieuRong(6);
    hcn.setChieuCao(9);
    dien_tich = hcn.tinhDienTich();
    // in dien tich va chi phi.
    Console.WriteLine("Tong dien tich: {0}",
        hcn.tinhDienTich());
    Console.WriteLine("Tong chi phi son: {0}",
        hcn.tinhChiPhi(dien_tich));
    Console.ReadLine();
    Console.ReadKey();
}
}
```

12.3. SO SÁNH LỚP TRỪU TƯỢNG VÀ GIAO DIỆN

Lớp trừu tượng	Giao diện
1) *Abstract class* có phương thức *abstract* (không có thân hàm) và phương thức *non-abstract* (có thân hàm).	*Interface* chỉ có phương thức *abstract*.
2) *Abstract class không hỗ trợ đa kế thừa.*	*Interface* có hỗ trợ đa kế thừa

3) Abstract class có các biến *final, non-final, static and non-static.*	Interface chỉ có các biến *static* và *final*.
4) Lớp trừu tượng có thể cung cấp nội dung cài đặt cho phương thức của giao diện	Giao diện không thể cung cấp nội dung cài đặt cho phương thức của lớp trừu tượng.
5) Từ khóa *abstract* được sử dụng để khai báo lớp trừu tượng.	Từ khóa *interface* được sử dụng để khai báo giao diện.

6) Ví dụ:
public abstract class Shape {
public abstract void draw();
}
	Ví dụ:
	public interface Drawable {
	void draw();
	}

Ví dụ 12.3 :

```
//tạo interface có 4 phương thức
        interface A {
           void a();
           abstract void b();
           public void c();
           public abstract void d();
        }
        // tạo abstract class cung cấp cài đặt cho một phương thức của interface A
        abstract class B implements A {
           public void c() {
              System.out.println("I am C");
           }
        }

        // tạo subclass của abstract class B, cung cấp cài đặt cho các phương thức còn lại
```

```
class M extends B {
    public void a() {
        System.out.println("I am a");
    }
    public void b() {
        System.out.println("I am b");
    }
    public void d() {
        System.out.println("I am d");
    }
}
// tạo lớp Test5 để gọi các phương thức của interface A
public class Test5 {
    public static void main(String args[]) {
        A a = new M();
        a.a();
        a.b();
        a.c();
        a.d();
    }
}
```

CÂU HỎI VÀ BÀI TẬP

Bài 1: Xây dựng giao diện nhân viên bao gồm hai phương thức: Nhập và hiển thị thông tin của cán bộ. Ta có thể xây dựng thêm một giao diện nhân viên mới mở rộng từ giao diện nhân viên. Giao diện nhân viên mới này xây phương thức cách tính lương cho mỗi loại nhân viên. Hãy xây dựng lớp nhân viên sản

xuất thực thi hai giao diện trên. Biết cách tính lương nhân viên như sau: Lương=Hsl*Lương cơ bản

Bài 2 : Xây dựng lớp "DaySo" để mô tả một dãy số

Gồm các phương thức sau:

- Phương thức "nhap" dùng để nhập dãy số từ bàn phím

- Phương thức "print" dùng để in dãy số ra màn hình

- Phương thức khởi tạo DaySo(int n) dùng để khởi tạo một mảng gồm n phần tử

Xây dựng giao diện Sort như sau: interface Sort { public void Sort(); }

Xây dựng các lớp "QuickSort", "SelectionSort", "InsertSort" bằng cách kế thừa từ lớp DaySo và triển khai giao diện Sort để thực hiệp việc sắp xếp: nổi bọt, chọn trực tiếp, chèn trực tiếp

BÀI 13 THỰC HÀNH 5

NỘI DUNG CHÍNH

- Xây dựng giao diện các lớp
- Đa kế thừa

Bài 13.1 :

a) Xây dựng lớp "DaySo" để mô tả một dãy số, gồm các phương thức sau:

- Phương thức "nhap" dùng để nhập dãy số từ bàn phím

- Phương thức "xuat" dùng để in dãy số ra màn hình

- Hàm tạo DaySo(int n) dùng để khởi tạo một mảng gồm n phần tử Hướng dẫn

b) Xây dựng giao diện Sort như sau:

```
interface Sort
{
public void Sort();
}
```

c) Xây dựng các lớp "QuickSort", "SelectionSort", "InsertSort" bằng cách kế thừa từ lớp DaySo và triển khai giao diện Sort để thực hiệp việc sắp xếp: nổi bọt, chọn trực tiếp, chèn trực tiếp.

a. Hướng dẫn

- Các lớp đối tượng gồm có:
    - Lớp dãy số
    - Lớp giao diện sort

  - Lớp Quicksort, SelectionSort, InsertSort;
- Các thuộc tính gồm có:
  - Mảng a[];
  - int n; //*số phần tử của mảng*
- Các phương thức gồm có:
  - Nhập mảng
  - Sắp xếp

b. Bài giải tham khảo

```
class dayso
 {
    public int[] a;
    protected int n;
    public dayso()
    {
       a = new int[5];
    }
    public dayso(int n)
    {
       a = new int[n];
    }
    public void nhap()
    {
       Console.Write("Nhap vao so phan tu cua day so:");
       n = int.Parse(Console.ReadLine());
       Console.WriteLine("Nhap vao thong tin cua day so:");
       for (int i = 0; i < n; i++)
       {
```

```
            Console.Write("a[{0}]=", i);
            a[i] = int.Parse(Console.ReadLine());
        }
    }
    public void print()
    {
        for (int i = 0; i < n; i++)
            Console.Write("{0}\t", a[i]);
        Console.WriteLine();
    }
}
interface sort
{
    void sapxep();
}
class quicksort : dayso, sort
{
    public void sapxep()
    {
        int i, j, x, tmp;
        x = a[(n) / 2];
        i = n - 1; j = n;
        do
        {
            while (a[i] < x) i++;
            while (a[j] > x) j--;
```

```
            if (i <= j)
            {
                tmp = a[i];
                a[i] = a[j];
                a[j] = tmp;
                i++; j--;
            }
        } while (i < j);
    }
}
class selectsort : dayso, sort
{
    public void sapxep()
    {
        int min, tmp, j;
        for (int i = 0; i < n - 1; i++)
        {
            min = i;
            for (j = i + 1; j < n; j++)
                if (a[j] < a[min])
                    min = j;
            tmp = a[min];
            a[min] = a[i];
            a[i] = tmp;
        }
    }
```

```
}
class inserttionsort : dayso, sort
{
    public void sapxep()
    {
        int i, pos, k;
        for (i = 0; i < n; i++)
        {
            k = a[i]; pos = i - 1;
            while (pos >= 0 && a[pos] > k)
            {
                a[pos + 1] = a[pos];
                pos--;
            }
            a[pos + 1] = k;
        }
    }
}
class Tester
{
    static void Main()
    {
        //sort a = new quicksort();
        //selectsort a = new selectsort();
        inserttionsort a = new inserttionsort();
        a.nhap(); Console.Clear();
```

```
        Console.WriteLine("Day truoc khi sap xep :");
        a.print();
        Console.WriteLine("Day sau khi sap xep :");
        a.sapxep();
        a.print();
        Console.ReadKey();
    }
}
```

Bài 13.2 : Một ngôi nhà có diện tích S (chiều dài x chiều rộng). Xây dựng tính đa kế thừa để tính chi phí sơn và làm sạch cho ngôi nhà đó.

a. Hướng dẫn

- Các lớp đối tượng gồm có:
  - Lớp ngôi nhà;
  - Lớp giao diện giá sơn và làm sạch;
  - Lớp chi phí;
- Các thuộc tính gồm có:
  - Chiều dài;
  - Chiều rộng;
- Các phương thức gồm có:
  - Khởi tạo
  - Tính chi phí giá sơn;
  - Tính chi phí giá làm sạch;

b. Bài giải tham khảo

```
class House
{
    protected int width;
    protected int height;
```

```
        public void SetWidth(int w)
        {
            width = w;
        }
        public void SetHeight(int h)
        {
            height = h;
        }
    }
    public interface IPaintCost
    {
        int GetPaintCost(int area);
    }
    public interface ICleanCost
    {
        int GetCleanCost(int area);
    }
class Cost : House, IPaintCost, ICleanCost
    {
        public int GetArea()
        {
            return (width * height);
        }
        public int GetPaintCost(int area)
        {
            return area * 70;
```

```
    }
    public int GetCleanCost(int area)
    {
        return area * 20;
    }
}
class Program
{
    static void Main(string[] args)
    {
        Cost house1 = new Cost();
        house1.SetWidth(5);
        house1.SetHeight(7);
        int area = house1.GetArea();
        Console.WriteLine("Total area: {0}", area);
        Console.WriteLine("Total paint cost: ${0}", house1.GetPaintCost(area));
        Console.WriteLine("Total clean cost: ${0}", house1.GetCleanCost(area));
        Console.ReadKey();
    }
}
```

BÀI 14 BỘ LẬP CHỈ MỤC, CƠ CHẾ ỦY QUYỀN, SỰ KIỆN

NỘI DUNG CHÍNH

- Cách lập chỉ mục cho các đối tượng của một lớp
- Cách gọi một phương thức thông qua cơ chế uỷ quyền
- Cách sử dụng sự kiện (event)

14.1. LẬP CHỈ MỤC

Việc định nghĩa chỉ mục cho phép tạo các đối tượng của lớp hoạt động giống như một mảng ảo. Tức là các đối tượng có thể sử dụng toán tử [] để truy cập đến một thành phần dữ liệu nào đó. Việc định nghĩa chỉ mục tương tự như việc định nghĩa một thuộc tính. [2]

Cú pháp tạo chỉ mục:

```
public  KiểuTraVề  this[DanhSáchThamSố]
{
//Hàm đọcget
{
//thân hàm đọc
}
// Hàm ghiset
{
//thân hàm ghi
}
}
```

Ví dụ 14.1: Tạo một lớp sinh viên sử dụng Indexer như sau :

```
class Student
{
    private string[] namelist = new string[size];
    static public int size = 10;
    public Student() {
        for (int i = 0; i = 0 && index = 0 && index <= size-1 ) {
            namelist[index] = value;
        }
    }
}
```

14.2. CƠ CHẾ ỦY QUYỀN

Giả sử chúng ta cần xây dựng một *class*, trong *class* này sẽ phải gọi một phương thức để thực hiện một hành động nào đó. Tuy nhiên chúng ta lại không biết được phương thức này khi xây dựng *class*. Phương thức này chỉ xuất hiện khi người khác sử dụng *class* để khởi tạo *object*.

Những tình huống khi không biết trước được phải gọi một phương thức cụ thể nào dẫn đến việc phải sử dụng một loại công cụ đặc biệt trong C#: *delegate*.

Delegate là những kiểu dữ liệu trong C# mà biến tạo ra từ nó chứa **tham chiếu tới phương thức**, thay vì chứa giá trị hoặc chứa tham chiếu tới *object* của các *class* bình thường. [2]

Một biến được tạo ra từ một kiểu *delegate* được gọi là một *biến delegate*.

Mỗi kiểu *delegate* khi được định nghĩa chỉ cho phép biến của nó chứa tham chiếu tới những phương thức phù hợp với quy định của delegate này.

Vai trò của *delegate* trong C#

Delegate cho phép một *class* linh động hơn trong việc sử dụng phương thức. Theo đó, nội dung cụ thể của một phương thức không được định nghĩa sẵn trong *class* mà sẽ do người dùng *class* đó tự định nghĩa trong quá trình khởi tạo *object*.

Điều này giúp phân chia logic của một *class* ra các phần khác nhau và do những người khác nhau xây dựng.

Delegate được sử dụng để giúp một *class object* tương tác ngược trở lại với thực thể tạo ra và sử dụng *class* đó. Điều này giúp *class* không bị "cô lập" bên trong thực thể đó. Ví dụ, *delegate* giúp gọi các phương thức của thực tế tạo ra và chứa *class object*.

Với khả năng tạo ra tương tác ngược như vậy, *delegate* trở thành hạt nhân của mô hình lập trình hướng sự kiện, được sử dụng trong công nghệ *winforms* và WPF.

Ví dụ 14.2 : Khi xây dựng các điều khiển của windows form (nút bấm, nút chọn, menu, v.v.), người lập trình ra các lớp này không thể xác định được người dùng muốn làm gì khi nút được bấm, khi menu được chọn. Do đó, bắt buộc phải sử dụng cơ chế của delegate để chuyển logic này sang cho người sử dụng các lớp đó viết code. Như vậy, Button khi được tạo ra trong một Form có khả năng tương tác với code của Form, chứ không cô lập chính mình.

Ví dụ 14.3 :

```
class Program
    {
    delegate int MyDelegate(string s);
    static void Main(string[] args)
    {
        Console.OutputEncoding = Encoding.Unicode;
        MyDelegate convertToInt = new MyDelegate(ConvertStringToInt);
        string numberSTR = "35";
        int valueConverted = convertToInt(numberSTR);
Console.WriteLine("Giá trị đã convert thành int: " + valueConverted);
        Console.ReadLine();
    }
    static int ConvertStringToInt(string stringValue)
```

```
{
    int valueInt = 0;
    Int32.TryParse(stringValue, out valueInt);
Console.WriteLine("Đã ép kiểu dữ liệu thành công");
    return valueInt;
}
  }
```

14.3. SỰ KIỆN

Sự kiện (*event*) là một khái niệm rất phổ biến trong lập trình và được sử dụng với mô hình *publisher/subscriber*. Trong .NET, *event* được sử dụng trong các mô hình lập trình cho giao diện đồ họa như Windows Forms hoặc Windows Presentation Foundation. Event và Delegate trong .NET có quan hệ rất gần gũi và thường gây nhầm lẫn. [2]

Khả năng hỗ trợ tương tác của delegate được vận dụng trong một mô hình lập trình gọi là publisher/subscriber, thường gọi tắt là mô hình pub/sub.

Khi sử dụng delegate chúng ta hoàn toàn có thể thực hiện mô hình pub/sub.

Tuy nhiên mô hình pub/sub đưa ra thêm hai yêu cầu quan trọng:

(1) các subscriber không được biết và không ảnh hưởng lẫn nhau;

(2) Việc raise/invoke một event chỉ được phép thực hiện bởi broadcaster.

Delegate không đáp ứng được các yêu cầu này do:

(1) delegate cho phép subscriber sử dụng phép gán =. Khi đó tất cả các subscriber sẵn có sẽ bị hủy.

(2) client code cũng có thể kích hoạt delegate.

Vì vậy .NET đưa thêm vào từ khóa *event* để dễ dàng vận dụng *delegate* vào mô hình pub/sub.

Như vậy, event trong .NET thực chất chỉ là một dạng hạn chế của delegate để phù hợp với mô hình pub/sub. Hoặc cũng có thể nói rằng *event* được xây dựng bên trên *delegate*.

Ví dụ 14.4: Tạo ra một class HocSinh để quản lý Tên của học sinh. Chúng ta muốn biết khi nào tên của học sinh thay đổi sẽ ghi lại thành log (thao tác ghi nhận lại quá trình sử dụng chương trình) để đối chiếu lịch sử thay đổi. Chúng ta sẽ tạo một Event NameChanged với Delegate là EventHandler có sẵn của .Net và ủy thác việc ghi log lại.

```
class Program
 {
   static void Main(string[] args)
   {
     Console.OutputEncoding = Encoding.Unicode;
     HocSinh hs = new HocSinh();
     hs.NameChanged += Hs_NameChanged;
     hs.Name = "Tên lần 1";
     hs.Name = "Tên lần 2";
     hs.Name = "Tên cuối";
     Console.ReadLine();
   }
   private static void Hs_NameChanged(object sender, EventArgs e)
   {
     Console.WriteLine("Tên có thay đổi.");
   }
 }
 public class HocSinh
 {
   private string _Name;
   public string Name
   {
```

```
    get => _Name;
    set
    {
        _Name = value;
        OnNameChanged();
    }
}
private event EventHandler _NameChanged;
public event EventHandler NameChanged
{
    add
    {
        _NameChanged += value;
    }
    remove
    {
        _NameChanged -= value;
    }
}
void OnNameChanged()
{
    if(_NameChanged != null)
    {
        _NameChanged(this, new EventArgs());
    }
}
```

```
}
```

CÂU HỎI VÀ BÀI TẬP

1. Lý thuyết

Câu 1: Indexer là gì? Cách sử dụng Indexer trong ngôn ngữ C#.

Câu 2: Nêu cách sử dụng delegate trong C#.

2. Bài tập

Bài 1: Viết chương trình tính tổng, hiệu, tích và thương sử dụng cơ chế ủy quyền (delegate).

Bài 2: Kết hợp delegate, event và ArrayList để viết chương trình cho phép nhập và hiển thị nhiều số nguyên dương (số nguyên lớn hơn).

BÀI 15 THỰC HÀNH 6

NỘI DUNG CHÍNH

- Cài đặt và sử dụng được bộ lập chỉ mục
- Cài đặt và sử dụng được cơ chế ủy quyền
- Cài đặt và sử dụng được sự kiện

Bài 15.1 : Sử dụng bộ lập chỉ mục (indexer) để xây dựng một lớp học viên, có thuộc tính : họ tên, số lượng học viên.

a. Hướng dẫn

- Các thuộc tính gồm có:
  - Mảng danh sách học viên;
  - Số lượng học viên;
- Các phương thức gồm có:
  - Khởi tạo
  - Bộ lập chỉ mục cho danh sách học viên

b. Bài giải tham khảo

```
class HocVien
 {
    private string[] namelist = new string[size];
    static public int size = 10;
    public HocVien()
    {
      for (int i = 0; i < size; i++)
```

```
        namelist[i] = "N/A";
}
public string this[int index]
{
    get
    {
        string tmp;
        if (index >= 0 && index <= size - 1)
        {
            tmp = namelist[index];
        }
        else
        {
            tmp = "";
        }
        return (tmp);
    }
    set
    {
        if (index >= 0 && index <= size - 1)
        {
            namelist[index] = value;
        }
    }
}
static void Main(string[] args)
```

```
        {
            HocVien names = new HocVien();
            names[0] = "Nam";
            names[1] = "Hoang";
            names[2] = "Dung";
            names[3] = "Hue";
            names[4] = "Huong";
            names[5] = "Phuc";
            names[6] = "Trung";
            for (int i = 0; i < HocVien.size; i++)
            {
                Console.WriteLine(names[i]);
            }
            Console.ReadKey();
        }
    }
```

Bài 15.2 : Chạy đoạn chương trình dưới đây, cho biết cách hoạt động của phương thức *delegate*, và các phương thức trong đó.

```
class baitapDelegate
    {
        delegate int NumberChanger(int n);
        static int num = 10;
        public static int AddNum(int p)
        {
            num += p;
            return num;
```

```
        }

        public static int MultNum(int q)
        {
            num *= q;
            return num;
        }
        public static int getNum()
        {
            return num;
        }
        static void Main(string[] args)
        {
            //tao cac doi tuong delegate
            NumberChanger nc1 = new NumberChanger(AddNum);
            NumberChanger nc2 = new NumberChanger(MultNum);
            //goi cac phuong thuc boi su dung cac doi tuong delegate
            nc1(25);
            Console.WriteLine("Gia tri cua num la: {0}", getNum());
            nc2(5);
            Console.WriteLine("Gia tri cua num la: {0}", getNum());
            Console.ReadKey();
        }
    }
```

Bài 15.3 : Viết chương trình thông báo nhiệt độ, áp suất của nồi hơi và ghi vào file log của hệ thống sử dụng *event*.

Bài giải tham khảo

```
using system ;
using system.IO ;
class Boiler
 {
    private int nhietdo;
    private int apsuat;
    public Boiler(int t, int p)
    {
       nhietdo = t;
       apsuat = p;
    }
    public int getNhietdo()
    {
       return nhietdo;
    }
    public int getApsuat()
    {
       return apsuat;
    }
 }
 class DelegateBoilerEvent
 {
    public delegate void BoilerLogHandler(string status);
    //dinh nghia su kien dua vao delegate tren
    public event BoilerLogHandler BoilerEventLog;
    public void LogProcess()
```

```
        {
            string remarks = "OK!";
            Boiler b = new Boiler(100, 12);
            int t = b.getNhietdo();
            int p = b.getApsuat();
            if (t > 150 || t < 80 || p < 12 || p > 15)
            {
                remarks = "Can duy tri";
            }
            OnBoilerEventLog("Thong tin log:\n");
            OnBoilerEventLog("Nhiet do: " + t + "\nAp suat: " + p);
            OnBoilerEventLog("\nThong bao: " + remarks);
        }
        protected void OnBoilerEventLog(string message)
        {
            if (BoilerEventLog != null)
            {
                BoilerEventLog(message);
            }
        }
    }
    class BoilerInfoLogger
    {
        FileStream fs;
        StreamWriter sw;
        public BoilerInfoLogger(string filename)
```

```
        {
            fs = new FileStream(filename, FileMode.Append, FileAccess.Write);
            sw = new StreamWriter(fs);
        }
        public void Logger(string info)
        {
            sw.WriteLine(info);
        }
        public void Close()
        {
            sw.Close();
            fs.Close();
        }
    }
    class program
    {
        static void Logger(string info)
        {
            Console.WriteLine(info);
        }
        static void Main(string[] args)
        {
            BoilerInfoLogger filelog = new BoilerInfoLogger("d:\\boiler.txt");
            DelegateBoilerEvent boilerEvent = new DelegateBoilerEvent();
            boilerEvent.BoilerEventLog += new
            DelegateBoilerEvent.BoilerLogHandler(Logger);
```

```
            boilerEvent.BoilerEventLog += new
            DelegateBoilerEvent.BoilerLogHandler(filelog.Logger);
            boilerEvent.LogProcess();
            Console.ReadLine();
            Console.ReadKey();
            filelog.Close();
        }
    }
```

TÀI LIỆU THAM KHẢO

[1] P.H.Khang, *C# 2005 Tập 1 Lập trình cơ bản*, NXB Lao động xã hội, 2013.

[2] P.H.Khang, *C# 2005 Tập 3 Lập trình hướng đối tượng*, NXB Lao động xã hội, 2013.

[3] P.V.Ất, *Giáo trình C++ và lập trình hướng đối tượng*, NXB Hồng Đức, 2009.