|  |
| --- |
| ỦY BAN NHÂN DÂN TP. HỒ CHÍ MINH  **TRƯỜNG CAO ĐẲNG CÔNG NGHỆ THỦ ĐỨC**  **KHOA CÔNG NGHỆ THÔNG TIN**  **-----oOo-----**    **GIÁO TRÌNH**  **HỌC PHẦN: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA**  **NGÀNH: CÔNG NGHỆ THÔNG TIN**  **TRÌNH ĐỘ: CAO ĐẲNG**  *Ban hành kèm theo Quyết định số:…*  *.*    **TP. Hồ Chí Minh, năm 2024** |

**TUYÊN BỐ BẢN QUYỀN**

Tài liệu này thuộc loại sách giáo trình nên các nguồn thông tin có thể được phép dùng nguyên bản hoặc trích dùng cho các mục đích về đào tạo và tham khảo.

Mọi mục đích khác mang tính lệch lạc hoặc sử dụng với mục đích kinh doanh thiếu lành mạnh sẽ bị nghiêm cấm.

**LỜI GIỚI THIỆU**

Quyển giáo trình này được biên soạn dựa theo đề cương môn học “Lập trình hướng đối tượng” dành cho chương trình Nhật của Khoa Công nghệ thông tin Trường Cao đẳng Công nghệ Thủ Đức. Giáo trình biên soạn sẽ không tránh khỏi những sai sót về nội dung lẫn hình thức, rất mong nhận được sự góp ý chân thành từ quý Thầy, Cô và các em Sinh viên để giáo trình hoàn thiện hơn.

Xin chân thành cảm ơn Phòng Khoa học công nghệ, Khoa Công nghệ thông tin Trường Cao Đẳng Công Nghệ Thủ Đức đã hỗ trợ tôi hoàn thành giáo trình này.

Tp.HCM, ngày 14 tháng 10 năm 2024

Ths Nguyễn Thị Vân

**MỤC LỤC**

[Chương 1. GIỚI THIỆU VỀ PHƯƠNG PHÁP LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG 6](#_Toc93309568)

[1.1| Khái niệm lập trình Hướng đối tượng 7](#_Toc93309569)

[1.2| So sánh lập trình Hướng đối tượng và lập trình theo kiểu cấu trúc 8](#_Toc93309570)

[1.2.1| Phương pháp lập trình kiểu cấu trúc 8](#_Toc93309571)

[1.2.2| Phương pháp lập trình Hướng đối tượng 9](#_Toc93309572)

[1.3| Mục tiêu của lập trình hướng đối tượng 11](#_Toc93309573)

[1.4| Các đặc điểm của lập trình Hướng đối tượng 11](#_Toc93309574)

[1.5| Lớp và Đối tượng 16](#_Toc93309575)

[1.5.1| Khái niệm Đối tượng (Objects) 16](#_Toc93309576)

[1.5.2| Khái niệm lớp 17](#_Toc93309577)

[1.5.3| Thuộc tính, phương thức 17](#_Toc93309578)

[1.5.4| Phân biệt Lớp và Đối tượng 18](#_Toc93309579)

[1.6| Sơ Đồ lớp 18](#_Toc93309580)

[1.7| Một số ngôn ngữ hỗ trợ lập trình Hướng đối tượng 19](#_Toc93309581)

[1.8| Bài tập áp dụng 19](#_Toc93309582)

[Chương 2. XÂY DỰNG LỚP ĐỐI TƯỢNG 21](#_Toc93309583)

[2.1| Định nghĩa lớp 22](#_Toc93309584)

[2.1.1| Bổ từ truy xuất 24](#_Toc93309585)

[2.1.2| Phân loại lớp 26](#_Toc93309586)

[2.1.3| Trường, Thuộc tính 26](#_Toc93309587)

[2.1.3.1| Trường (Fields) 27](#_Toc93309588)

[2.1.3.2| Thuộc tính (Properties) 28](#_Toc93309589)

[2.1.4| Cài đặt các phương thức thành viên (Methods) 32](#_Toc93309590)

[2.1.4.1| Phương thức khởi tạo (constructors) 32](#_Toc93309591)

[2.1.4.2| Phương thức hủy đối tượng (destructors / Finalizers) 36](#_Toc93309592)

[2.1.4.3| Con trỏ this 38](#_Toc93309593)

[2.2| Từ khóa Static 38](#_Toc93309594)

[2.2.1| Static fields 38](#_Toc93309595)

[2.2.2| Static Methods 39](#_Toc93309596)

[2.2.3| Static class 41](#_Toc93309597)

[2.3| Sử dụng đối tượng làm tham số cho phương thức 43](#_Toc93309598)

[2.3.1| Truyền đối tượng dạng tham chiếu 43](#_Toc93309599)

[2.3.2| Truyền tham số dùng từ khóa params 45](#_Toc93309600)

[2.4| Mảng đối tượng 46](#_Toc93309601)

[2.5| Biểu diễn mối quan hệ giữa các lớp 50](#_Toc93309602)

[2.6| Bài tập áp dụng 53](#_Toc93309603)

[Chương 3. TÍNH KẾ THỪA 60](#_Toc93309604)

[3.1| Quan hệ giữa các lớp đối tượng 61](#_Toc93309605)

[3.2| Tổng quát hóa, đặc biệt hóa 61](#_Toc93309606)

[3.3| Khái niệm kế thừa 62](#_Toc93309607)

[3.4| Lợi ích của kế thừa 63](#_Toc93309608)

[3.5| Cú pháp khai báo kế thừa 63](#_Toc93309609)

[3.6| Tầm vực trong kế thừa 64](#_Toc93309610)

[3.7| Phân loại kế thừa 65](#_Toc93309611)

[3.8| Một số mô hình kế thừa 66](#_Toc93309612)

[3.9| Bài tập áp dụng 68](#_Toc93309613)

[Chương 4. TÍNH ĐA HÌNH 74](#_Toc93309614)

[4.1| Khái niệm Đa hình 75](#_Toc93309615)

[4.2| Phân loại Đa hình 76](#_Toc93309616)

[4.3| Đa hình tĩnh - Static Polymorphism 76](#_Toc93309617)

[4.3.1| Nạp chồng phương thức 76](#_Toc93309618)

[4.3.2| Nạp chồng toán tử 80](#_Toc93309619)

[4.4| Đa hình động - Dynamic Polymorphism 82](#_Toc93309620)

[4.5| Abstract class và Abstract methods 87](#_Toc93309621)

[4.6| Từ khóa virtual 89](#_Toc93309622)

[4.7| Từ khóa sealed 91](#_Toc93309623)

[4.8| Kiểu dữ liệu Object 93](#_Toc93309624)

[4.9| Bài tập áp dụng 96](#_Toc93309625)

[Chương 5. INTERFACE 101](#_Toc93309626)

[5.1| Khái niệm Interface 102](#_Toc93309627)

[5.2| Khai báo interface 103](#_Toc93309628)

[5.3| Cài đặt Interface 104](#_Toc93309629)

[5.4| Truy xuất phương thức của giao diện 106](#_Toc93309630)

[5.5| Một số trường hợp sử dụng interface 109](#_Toc93309631)

[5.6| Bài tập áp dụng 110](#_Toc93309632)

[Chương 6. LẬP TRÌNH KHÁI QUÁT 117](#_Toc93309633)

[6.1| Khái niệm Generic 118](#_Toc93309634)

[6.2| Một số đặc điểm của generic trong C# 118](#_Toc93309635)

[6.3| Generic được áp dụng cho các đối tượng nào trong C#? 119](#_Toc93309636)

[6.4| Generic fields 120](#_Toc93309637)

[6.5| Generic methods 120](#_Toc93309638)

[6.6| Generic class trong C# 125](#_Toc93309639)

[6.7| Bài tập áp dụng 128](#_Toc93309640)

[PHỤ LỤC 131](#_Toc93309641)

**GIÁO TRÌNH HỌC PHẦN**

**Tên học phần:** Lập trình Hướng đối tượng

**Mã học phần:**  CSC106320

**Vị trí, tính chất, ý nghĩa và vai trò của học phần:**

- Vị trí: Học phần này thuộc khối kiến thức cơ sở, được tổ chức giảng dạy cho sinh viên ở học kỳ 2 năm thứ nhất.

- Tính chất: Học phần nhằm trang bị cho sinh viên chuyên ngành Công nghệ thông tin các kiến thức và kỹ năng để giải quyết một số bài toán bằng phương pháp lập trình hướng đối tượng. Ngoài ra, sinh viên còn có thể thiết kế, cài đặt sơ đồ lớp cho một số bài toán cơ bản sử dụng ngôn ngữ lập trình C#. Thông qua các hoạt động học tập, sinh viên có thể hình thành thói quen tư duy hệ thống, luôn viết code theo chuẩn và thói quen tuân thủ các quy định làm việc trong môi trường chuyên nghiệp.

**Mục tiêu của học phần :**

- Về kiến thức:

* Trình bày các đặc điểm của lập trình hướng đối tượng.

- Về kỹ năng:

* Phân tích, thiết kế và cài đặt thành thạo sơ đồ lớp cho các chương trình vừa và nhỏ;
* Sử dụng thành thạo cơ chế kế thừa và đa hình để giải quyết một số bài toán vừa và nhỏ.
* Sử dụng được lập trình khái quát để giải quyết một số bài toán vừa và nhỏ.
* Sử dụng thành thạo Visual Studio và Debug chương trình.

- Về năng lực tự chủ và trách nhiệm:

* Viết code theo chuẩn
* Học tập chủ động và tích cực. Làm bài tập theo yêu cầu của Giảng viên và nộp bài đúng quy định.

**Nội dung của học phần**:

|  |  |
| --- | --- |
|  | GIỚI THIỆU VỀ PHƯƠNG PHÁP LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG |

Chương này nhằm giới thiệu cho sinh viên các khái niệm, các đặc điểm của phương pháp lập trình Hướng đối tượng, phân biệt phương pháp hướng cấu trúc và hướng đối tượng.

### Khái niệm lập trình Hướng đối tượng

Lập trình Hướng đối tượng có tên tiếng Anh là Object-Oriented Programming Languages, được viết tắt là OOP. Đây là một phương pháp lập trình sử dụng các đối tượng làm vai trò trung tâm để xây dựng chương trình. Lập trình hướng đối tượng giúp cho lập trình viên có thể tổ chức một chương trình lớn thành những đối tượng rời rạc, độc lập, ít phụ thuộc lẫn nhau, nhờ đó có thể dễ dàng kiểm soát, hiệu chỉnh, nâng cấp chương trình mà không ảnh hưởng đến các thành phần khác của chương trình.

Trong lập trình hướng đối tượng, một đối tượng được định nghĩa bao gồm hai thành phần chính là:

* Tập các thuộc tính dữ liệu (attributes): là những thông tin, đặc điểm của đối tượng. Ví dụ một người có những đặc điểm như họ tên, ngày sinh, giới tính nghề nghiệp, màu mắt, màu tóc, …
* Tập các phương thức (methods) là những thao tác, hành động mà đối tượng đó có thể thực hiện. Ví dụ một người sẽ có thể thực hiện các hành động như nói, đi, làm việc, ăn, uống, …

Hình 1. Ví dụ về thuộc tính và hành động của đối tượng Người

### So sánh lập trình Hướng đối tượng và lập trình theo kiểu cấu trúc

### Phương pháp lập trình kiểu cấu trúc

Phương pháp lập trình có cấu trúc (structured programming) hay còn gọi phương pháp lập trình hướng thủ tục (**P**rocedure-**O**riented **P**rogramming -**POP**) là phương pháp phân tích một nhiệm vụ lớn thành nhiều công việc nhỏ hơn – chia để trị. Cách thức phân tích và thiết kế theo nguyên lý lập trình từ trên xuống (top-down). Phương pháp này thể hiện quá trình suy diễn từ cái chung cho đến cái cụ thể, từ một chương trình lớn thành các chương trình con nhỏ hơn có chức năng độc lập. Chương trình con đóng vai trò trung tâm của việc lập trình hay còn gọi phương pháp Lập trình hướng thủ tục.

Chương trình có cấu trúc được tổ chức thành các chương trình con riêng lẻ (còn gọi là các module hay phương thức. Mỗi chương trình con đảm nhận xử lý một công việc nhỏ trong toàn bộ hệ thống.

Hình 2. Mô hình phân rã chức năng theo phương pháp lập trình có cấu trúc

Trong lập trình hướng thủ tục thường quan tâm đến việc phát triển các chức năng (phương thức) mà ít quan tâm tới dữ liệu, điều này khiến cho dữ liệu khó kiểm soát khi được dùng chung giữa các chức năng.

Ưu điểm của phương pháp lập trình hướng thủ tục: triển khai các phần mềm dễ dàng, chương trình dễ hiểu và dễ bảo trì. Các nhược điểm của phương pháp lập trình hướng thủ tục như sau:

* Không thích hợp khi xây dựng các chương trình lớn và phức tạp.
* Phần lớn các phương thức sử dụng dữ liệu chung nên khi có một sự thay đổi dữ liệu phải thực hiện thay đổi ở tất cả phương thức liên quan đến dữ liệu đó. Đây là công việc tốn thời gian và kém hiệu quả.
* Bảo mật dữ liệu kém.
* Cách tiếp cận đôi khi không phù hợp với thực tế, khó mô tả được các hoạt động của thế giới thực.

Để khắc phục các nhược điểm trên phương pháp lập trình hướng đối tượng ra đời.

### Phương pháp lập trình Hướng đối tượng

Lập trình hướng đối tượng **(O**bject **O**riented **P**rogramming – **OOP**) là phương pháp lập trình lấy đối tượng làm nền tảng để xây dựng các phương thức và xây dựng chương trình. Theo mô hình đối tượng, chương trình có cấu trúc là một tập các đối tượng độc lập, tương tác với nhau khi cần thiết để hoàn thành nhiệm vụ của chương trình.

Trong phương pháp lập trình hướng đối tượng thì ta thường tư duy theo hướng thực hiện thao tác gì với các đối tượng đã có để giải quyết bài toán đặt ra. Với cách tư duy này, đối tượng là trung tâm của việc lập trình, người ta gọi là nguyên lý lập trình từ dưới lên (Bottom-up).

Ưu điểm của chương trình hướng đối tượng là cấu trúc thuần nhất chỉ chứa một loại thành phần là đối tượng nên dễ dàng quản lý code hơn khi có sự thay đổi chương trình, dễ mở rộng chương trình, có tính bảo mật dữ liệu cao và có tính tái sử dụng lại phần mềm giúp giảm thiểu chi phí.

Nhược điểm của chương trình hướng đối tượng là không phù hợp với mọi loại chương trình, một số chương trình có thể sử dụng nhiều bộ nhớ hơn và thực thi nhiều lệnh hơn so với lựa chọn phương pháp lập trình hướng thủ tục.

|  |  |  |
| --- | --- | --- |
|  | **POP** (Lập trình hướng thủ tục) | **OOP** (Lập trình hướng đối tượng) |
| Hướng thiết kế chương trình | Từ trên xuống (Top-down) | Từ dưới lên (Bottom-up) |
| Cách phân chia chương trình | Chia nhỏ theo các chức năng (functions). Một phương thức có thể chứa nhiều dữ liệu khác nhau | Chia nhỏ theo các đối tượng (objects). Một đối tượng chỉ điều khiển dữ liệu của nó |
| Thiết kế thuật toán | Tập trung xây dựng các thuật toán theo cách có hệ thống | Tập trung vào bảo mật dữ liệu, không phân biệt các thuật toán |
| Tính bảo mật dữ liệu | Rất khó để che dấu dữ liệu | Dễ dàng cho phép hoặc giới hạn quyền truy cập đến các dữ liệu bằng các từ khóa: public, protected, private |
| Khả năng mở rộng chương trình | Rất khó chỉnh sửa, mở rộng khi chương trình có sự thay đổi về dữ liệu.  Tính tái sử dụng thấp | Dễ dàng chỉnh sửa, mở rộng khi chương trình có sự thay đổi về dữ liệu.  Tính tái sử dụng cao |
| Tính kế thừa  ( tái sử dụng) | Không cho phép kế thừa | Cho phép kế thừa các phương thức và thuộc tính có sẵn |
| Đa năng hóa các toán tử, phương thức | Không cho phép | Các toán tử, phương thức có thể thực hiện với nhiều kiểu dữ liệu khác nhau, theo nhiều cách khác nhau |

Bảng so sánh POP và OOP

### Mục tiêu của lập trình hướng đối tượng

Lập trình hướng đối tượng được thực hiện thông qua các ngôn ngữ lập trình hướng đối tượng để đạt được các mục tiêu sau:

* **Tính mạnh mẽ:**
* Cho phép các chương trình phức tạp có thể hoạt động chính xác.
* Nếu một thư viện có sẵn không phù hợp yêu cầu thì người lập trình có khả năng sửa đổi hoặc mở rộng một cách dễ dàng, không cần phải can thiệp đến mã nguồn thư viện.
* Đơn giản hoá việc xây dựng và sử dụng các thư viện.
* Nâng cao độ tin cậy và tính bền vững của phần mềm.
* **Khả năng thích nghi**
* Hỗ trợ mạnh các dự án phát triển phần mềm quy mô lớn, đòi hỏi nhiều người tham gia, phần mềm phát triển trong một thời gian dài.
* Có thể chạy trên nhiều kỹ thuật và phần cứng khác nhau
* **Khả năng tái sử dụng**
* Các đối tượng có thể sử dụng lại trong một ứng dụng hoặc nhiều ứng dụng khác nhau.

### Các đặc điểm của lập trình Hướng đối tượng

Lập trình Hướng đối tượng có bốn đặc điểm sau:

* **Tính trừu tượng (Abtraction)**

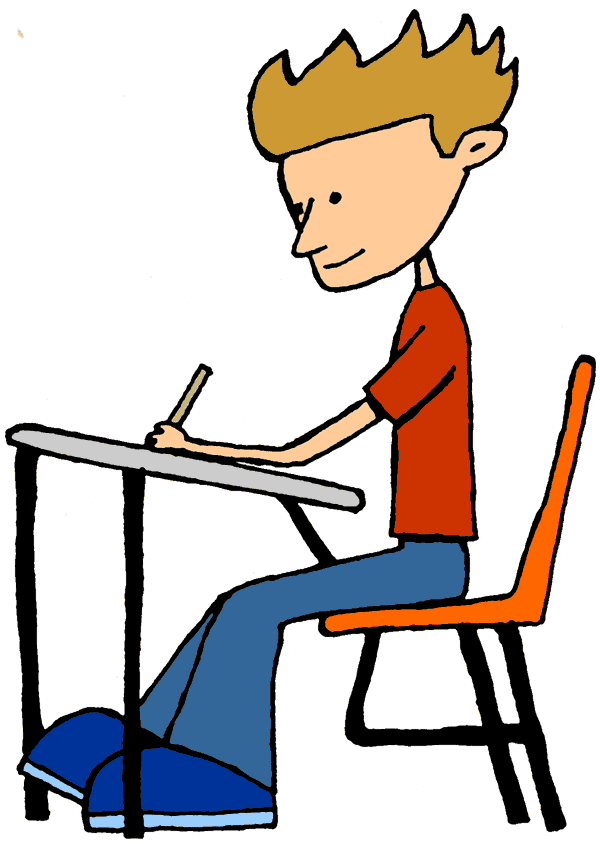
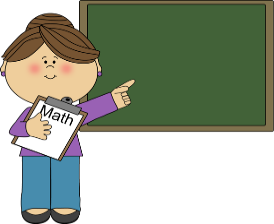
Tính trừu tượng là một đặc điểm cơ bản của lập trình hướng đối tượng. Mục tiêu của nó là làm giảm độ phức tạp cho chương trình bằng cách loại bỏ những chi tiết không cần thiết của đối tượng và chỉ tập trung vào những chi tiết cốt lõi, quan trọng.

Trừu tượng còn là một quá trình ẩn các chi tiết triển khai và chỉ hiển thị chức năng cho người dùng. Mỗi đối tượng có thể thực hiện các công việc, tương tác với các đối tượng khác mà không cần phải biết làm cách nào đối tượng tiến hành được các thao tác.

Nâng cao hơn nữa là tính trừu tượng còn thể hiện qua việc một đối tượng ban đầu có thể có một số đặc điểm chung cho nhiều đối tượng khác như là sự mở rộng của nó nhưng bản thân đối tượng ban đầu này có thể không được định nghĩa chi tiết cách thực hiện. Tính trừu tượng này thường được xác định trong khái niệm gọi là lớp cơ sở trừu tượng (abstract class) và giao diện (interface).

Giả sử bạn cần tạo một ứng dụng quản lý kết quả học tập của sinh viên trong một trường học. Sinh viên là một trong các đối tượng của hệ thống này. Các chức năng có thể thực hiện trên đối tượng sinh viên như truy cập, cập nhật, xóa các dữ liệu, tạo email tự động cho sinh viên từ thông tin họ tên và ngày sinh của sinh viên. Như vậy các hành động này được định nghĩa như là một hành động của đối tượng sinh viên. Khi người dùng gọi các phương thức này, ví dụ gọi phương thức tạo tạo email tự động, thì họ không cần quan tâm chi tiết phương thức được thực hiện bên trong như thế nào mà người dùng chỉ cần quan tâm đến việc sử dụng phương thức đó và kết quả trả về của nó.

Tuy nhiên, các đối tượng khác có tương tác với hệ thống như giáo viên, nhân viên cũng có các thông tin giống như đối tượng sinh viên như họ tên, ngày sinh, giới tính và phương thức tạo email nhưng việc thực hiện ở từng đối tượng khác nhau có thể khác nhau. Khi đó một lớp trừu tượng (abstract class) được tạo ra để chứa tất cả các thuộc tính và phương thức chung của các đối tượng giáo viên, sinh viên, nhân viên, có thể đặt tên lớp trừu tượng này là lớp “Con người”. Chúng ta cũng có thể sử dụng một giao diện (interface), interface này không chứa định nghĩa chi tiết phương hoạt động như thế nào mà chỉ chứa khai báo khuôn mẫu cho các phương thức chung.



**Con người**

Họ tên

Ngày sinh

Giới tính

Tạo email

Tính tuổi  
. . .

…

Sinh viên

Nhân viên

Giáo viên

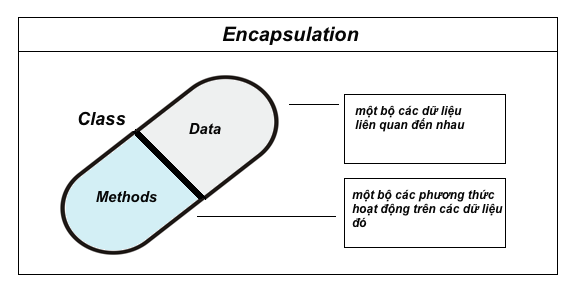
Lớp trừu tượng chứa các đặc điểm chung

Hình 3. Minh họa tính trừu tượng của OOP

* **Tính đóng gói (Encapsulation)**

Tính đóng gói (encapsulation) và che giấu thông tin (information hidding) là tính chất không cho phép người sử dụng thay đổi trạng thái nội tại của đối tượng. Chỉ có các chức năng nội tại của đối tượng cho phép thay đổi trạng thái của nó. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã nguồn nhằm đảm bảo sự toàn vẹn của đối tượng.

Sự đóng gói còn là cơ chế ràng buộc dữ liệu và thao tác trên dữ liệu đó thành một thể thống nhất, tránh được các tác động bất ngờ từ bên ngoài. Thể thống nhất này gọi là lớp. Cơ chế đóng gói là phương thức tốt để thực hiện cơ chế che dấu thông tin so với các ngôn ngữ lập trình cấu trúc.



Hình 4. Minh họa tính đóng gói của OOP

Khi khai báo lớp các thành phần dữ liệu thông thường sẽ được giới hạn quyền truy cập là riêng tư để tránh việc truy cập trực tiếp từ bên ngoài lớp vào các dữ liệu của lớp. Việc truy cập vào các dữ liệu của lớp chỉ được thông qua việc gọi các phương thức, nhờ vậy các thao tác khi truy cập đến các dữ liệu của lớp sẽ được thực hiện thông qua các đoạn lệnh kiểm soát trong các phương thức. Ngoài ra các chi tiết phức tạp cài đặt bên trong mã nguồn của lớp cũng được che dấu.

**Ví dụ**: Trong hệ thống quản lý kết quả học tập sinh viên, người dùng lớp Sinh Viên có thể đọc nhưng không được phép sửa các thông tin họ tên, ngày sinh của một đối tượng sinh viên đã được tạo ra trong hệ thống, nhưng người dùng có thể được toàn quyền thêm, xóa, sửa đối với dữ liệu địa chỉ của sinh viên. Hơn nữa, người dùng sẽ không thấy được các mã nguồn chi tiết cài đặt bên trong lớp Sinh viên cũng như các thuộc tính và phương thức được gán quyền riêng tư. Tính đóng gói này giúp tăng sự toàn vẹn dữ liệu và che dấu sự phức tạp mã nguồn bên trong lớp.

* **Tính kế thừa (Inheritance)**

Tính kế thừa trong OOP cho phép chúng ta xây dựng các lớp mới từ các lớp đã có. Một lớp mới gọi là lớp dẫn xuất (derived class) có thể thừa hưởng dữ liệu và các phương thức của lớp cơ sở (base class) ban đầu và cho phép định nghĩa thêm các dữ liệu, phương thức mới.

Mỗi lớp cơ sở không có giới hạn về số lượng các lớp dẫn xuất được tạo ra. Qua cơ chế kế thừa này, dạng hình cây của các lớp được hình thành. Dạng cây của các lớp trông giống như các cây gia phả vì thế các lớp cơ sở còn được gọi là lớp cha (parent class) và các lớp dẫn xuất được gọi là lớp con (child class).

**Ví dụ**: Trong hệ thống quản lý kết quả học tập của sinh viên ở trên cần mở rộng quản lý cho hai loại đối tượng là sinh viên chưa tốt nghiệp và sinh viên đã tốt nghiệp, Quan sát sơ đồ lớp sau:

Trong hệ thống có lớp cơ sở / lớp cha (base class / parent class) **Người** có hai lớp dẫn xuất / lớp con (derived class / child class) là lớp Sinh viên và lớp Giáo viên. Hai lớp con được thừa hưởng tất cả các thuộc tính và phương thức của lớp cha. Trong hệ thống này lớp Sinh viên cũng là lớp cơ sở của hai lớp dẫn xuất Sinh viên chưa tốt nghiệp và Sinh viên đã tốt nghiệp.

Hình 5. Minh họa tính kế thừa

* **Tính đa hình (Polymorphism)**

Tính đa hình thể hiện thông qua việc gửi các thông điệp (message). Việc gửi các thông điệp này có thể so sánh như việc gọi các phương thức bên trong của một đối tượng. Các phương thức dùng trả lời cho một thông điệp sẽ tùy theo đối tượng mà thông điệp đó được gửi tới, sẽ có phản ứng khác nhau. Vậy tính đa hình là khả năng một thông điệp có thể có cách thực hiện khác nhau tùy theo đối tượng nhận thông điệp.

**Ví dụ:** cùng một hành động in bảng điểm cho Sinh viên. Nhưng đối với mỗi loại sinh viên khác nhau thì thông tin chi tiết trong bảng điểm sẽ khác nhau. Như vậy cùng một đối tượng sinh viên, cùng một hành động là in bảng điểm nhưng tại các thời điểm khác nhau (đang học hoặc đã tốt nghiệp) thì cách thực hiện của hành động in bảng điểm cũng khác nhau. Người dùng chỉ cần truy cập một phương thức duy nhất nhưng kết quả trả về sẽ khác nhau tại các thời điểm khác nhau. Điều này thể hiện tính đa hình của các phương thức trong lập trình hướng đối tượng.

InBangDiem()

BẢNG ĐIỂM QUÁ TRÌNH

BẢNG ĐIỂM

TỐT NGHIỆP

InBangDiem()

InBangDiem()



Hình 6. Minh họa tính đa hình

### Lớp và Đối tượng

### Khái niệm Đối tượng (Objects)

Đối tượng là người, vật, hiện tượng mà con người tác động tới (trong suy nghĩ, hành động). Mỗi phần tử trong thế giới thực xung quanh chúng ta là một đối tượng, ví dụ: giáo viên A, sinh viên B, máy tính MT, xe X….

Một thực thể (instance) là một vật thể có thực bên trong bộ nhớ, thực chất đó là một đối tượng (nghĩa là một đối tượng được cấp phát vùng nhớ).

Trong lập trình hướng đối tượng, đối tượng được hiểu như là một thực thể (instance): người, vật hoặc một bảng dữ liệu, . . .Khi một phần tử dữ liệu được khai báo là từ một lớp thì nó được gọi là một đối tượng.

Một đối tượng bao gồm hai thông tin: thuộc tính và phương thức.

* Thuộc tính chính là những thông tin, đặc điểm của đối tượng. Ví dụ: một người sẽ có họ tên, ngày sinh, màu da, kiểu tóc, . . .
* Phương thức là những thao tác, hành động mà đối tượng đó có thể thực hiện. Ví dụ: một người sẽ có thể thực hiện hành động nói, đi, ăn, uống, . . .

### Khái niệm lớp

Lớp (còn được gọi là “lớp đối tượng”, class) được dùng để mô hình hóa một nhóm các thực thể cùng loại trong thế giới thực.

Ví dụ: lớp Sinh viên (là tập hợp các đối tượng sinh viên đều có dữ liệu mã sinh viên, họ tên, lớp, điểm trung bình, ăn, ngủ, đi học...).

### Thuộc tính, phương thức

Các đối tượng đều có hai thành phần chính đó là thành phần thuộc tính và thành phần phương thức. Thuộc tính (attributes) bao gồm những thông tin mô tả về đối tượng còn phương thức chính là các hành động của đối tượng hay các phương thức thao tác, tác động lên dữ liệu (thuộc tính) của đối tượng.

Dấu hiệu nhận diện (indentity): khi đọc mô tả của một bài toán, để xác định thuộc tính và phương thức của đối tượng chúng ta dựa vào thông tin sau:

*Tìm các danh từ* *🡪* Thuộc tính mô tả đối tượng, ta cần biết thông tin gì về đối tượng

*Tìm các động từ* 🡪 Hành vi (phương thức) của đối tượng, xét xem đối tượng cần có xử lý gì.

Tất cả các đối tượng cùng loại đều có thuộc tính và phương thức giống nhau. Do đó *lớp đối tượng* cũng có hai thành phần đó là thành phần thuộc tính và thành phần phương thức trong đó thuộc tính của các đối tượng này chính là thuộc tính của lớp và phương thức của các đối tượng cũng chính là phương thức của lớp. Mỗi phương thức của lớp thực chất là một phương thức được viết riêng cho các đối tượng của lớp, chỉ được phép gọi để tác động lên chính các đối tượng của lớp này.

Ví dụ: Lớp Sinh viên

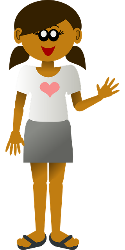
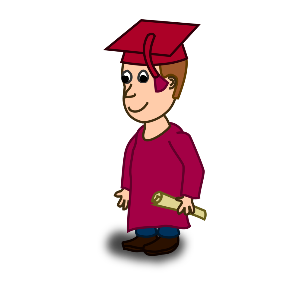
* Thuộc tính: mã sinh viên, tên sinh viên, điểm trung bình, xếp loại
* Phương thức: khởi tạo, gán giá trị cho các thành phần thông tin, cập nhật giá trị cho các dữ liệu, in thông tin, tạo email tự động, tính tuổi …

### Phân biệt Lớp và Đối tượng

Lớp là một khái niệm bao quát nói lên những đặc điểm chung nhất của các đối tượng, còn đối tượng chính là một phần tử tồn tại thực tế của lớp.

Ví dụ:

Sinh viên Lan, Tuấn, Hoa, ... đều là các đối tượng thuộc lớp Sinh viên nói chung.



Objects

Class

SinhVien

- hoTen

- ngaySinh

- diaChi

+GanDiaChi()

+TinhTuoi()

+TaoEmail()

…

Hình 7. Minh họa class và objects

Lớp là khuôn mẫu để tạo các đối tượng. Mỗi đối tượng có cấu trúc và hành vi giống như lớp đối tượng mà nó được tạo từ đó.

Lớp là cái chúng ta thiết kế và lập trình còn đối tượng được tạo (từ một lớp) tại thời gian chạy. Ví dụ lớp Student tạo ra nhiều đối tượng như *Jenna, John*, ...

### Một số lớp dựng sẵn trong Java

### String class

Trong java, string (chuỗi) là một đối tượng biểu diễn một chuỗi các kí tự (char).

String là một reference type, không thể thay đổi giá trị (immutable).

Có 2 cách để tạo đối tượng String :

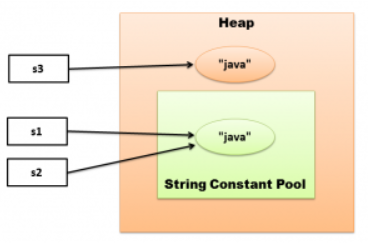
* Sử dụng string literal :

Ví dụ :

String s1 = "java";

String s2 = “java”;

Mỗi khi tạo một biến string literal, đầu tiên JVM sẽ kiểm tra xem giá trị đó đã tồn tại trong Pool[[1]](#footnote-1) chưa. Nếu chuỗi này đã tồn tại trong Pool, thì giá trị của biến sẽ được tham chiếu đến instance đã được tạo ra trong Pool. Nếu chuỗi này không tồn tại trong Pool, một instance mới được tạo ra và đặt vào trong Pool. Trong ví dụ trên s1 và s2 đều tham chiếu đến cùng một ô nhớ (có giá trị là “java”), chính vì vậy lúc này s1==s2.



Heap

Hình 8. Minh họa string literal

* Sử dụng từ khóa new : JVM sẽ luôn tạo ra ô nhớ mới trong Heap memory (cho dù đã tồn tại ô nhớ mới có cùng giá trị).

Ví dụ :

String s3 = new String(“java”);

String s4 = new String(“java”);

Khi khai báo s3 hay s4, JVM luôn luôn tạo ra 2 ô nhớ khác nhau trong Heap memory để tham chiếu đến. Vì thế khi so sánh s3 == s4 sẽ cho kết quả false.

Lớp java.lang.String trong java sử sụng để tạo đối tượng string, cung cấp rất nhiều các phương thức để thực hiện các thao tác với chuỗi như bảng sau:

|  |  |  |
| --- | --- | --- |
| **No.** | **Phương thức** | **Mô tả** |
| 1 | char charAt(int index) | Trả về giá trị char cho chỉ số cụ thể. |
| 2 | [int length()](https://viettuts.vn/java-string/phuong-thuc-length) | Trả về độ dài chuỗi. |
| 3 | [static String format(String format, Object... args)](https://viettuts.vn/java-string/phuong-thuc-format) | Trả về chuỗi được format. |
| 4 |  |  |
| 5 | [String substring(int beginIndex)](https://viettuts.vn/java-string/phuong-thuc-substring) | Trả về chuỗi con bắt đầu từ chỉ số index. |
| 6 | [String substring(int beginIndex, int endIndex)](https://viettuts.vn/java-string/phuong-thuc-substring) | Trả về chuỗi con từ chỉ số bắt đầu đến chỉ số kết thúc. |
| 7 | [boolean contains(CharSequence s)](https://viettuts.vn/java-string/phuong-thuc-contains) | Kiểm tra chuỗi chứa chuối không, kết quả trả về là giá trị boolean. |
| 8 | [static String join(CharSequence delimiter, CharSequence... elements)](https://viettuts.vn/java-string/phuong-thuc-join) | Trả về chuỗi được nối từ nhiều chuỗi. |
| 9 | [boolean equals(Object another)](https://viettuts.vn/java-string/phuong-thuc-equals) | kiểm tra sự tương đương của chuỗi với đối tượng. |
| 10 | [boolean isEmpty()](https://viettuts.vn/java-string/phuong-thuc-isempty) | Kiểm tra chuỗi rỗng. |
| 11 | [String concat(String str)](https://viettuts.vn/java-string/phuong-thuc-concat) | Nối chuỗi cụ thể. |
| 12 | [String replace(char old, char new)](https://viettuts.vn/java-string/phuong-thuc-replace) | Thay thế tất cả giá trị char cụ thể bằng một giá trị char mới. |
| 13 | [String replace(CharSequence old, CharSequence new)](https://viettuts.vn/java-string/phuong-thuc-replace) | Thay thế tất cả các chuỗi bằng một chuỗi mới. |
| 14 | [static String equalsIgnoreCase(String another)](https://viettuts.vn/java-string/phuong-thuc-equalsignorecase) | So sanh chuỗi, không phân biệt chữ hoa hay chữ thường. |
| 15 | [String[] split(String regex)](https://viettuts.vn/java-string/phuong-thuc-split) | Trả về mảng các chuỗi được tách ra theo giá trị regex. |
| 16 | [String[] split(String regex, int limit)](https://viettuts.vn/java-string/phuong-thuc-split) | Trả về mảng các chuỗi được tách ra theo giá trị regex và có giới hạn. |
| 17 | [int indexOf(int ch)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về vị trí của ký tự ch cụ thể. |
| 18 | [int indexOf(int ch, int fromIndex)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về vị trí của ký tự ch tính từ từ vị trí fromIndex. |
| 19 | [int indexOf(String substring)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về vị trí của chuỗi con substring. |
| 20 | [int indexOf(String substring, int fromIndex)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về chuỗi con bắt đầu từ vị trí substring đến vị trí fromIndex. |
| 21 | [String toLowerCase()](https://viettuts.vn/java-string/phuong-thuc-tolowercase) | Trả về chuỗi chữ thường. |
| 22 | [String toLowerCase(Locale l)](https://viettuts.vn/java-string/phuong-thuc-tolowercase) | Trả về chuỗi chữ thường bằng việc sử dụng locale cụ thể. |
| 23 | [String toUpperCase()](https://viettuts.vn/java-string/phuong-thuc-touppercase) | Trả về chuỗi chữ hoa. |
| 24 | [String toUpperCase(Locale l)](https://viettuts.vn/java-string/phuong-thuc-touppercase) | Trả về chuỗi chữ hoa bằng việc sử dụng locale cụ thể. |
| 25 | String trim() | Xóa khoảng trẳng ở đầu và cuối của chuỗi. |
| 26 | [String toUpperCase(Locale l)](https://viettuts.vn/java-string/phuong-thuc-touppercase) | Trả về chuỗi chữ hoa bằng việc sử dụng locale cụ thể. |
| 27 | int lastIndexOf(int ch)  int lastIndexOf(int ch, int fromIndex) | Trả vể chỉ số cuối của ký tự hoặc chuỗi con. Nếu không tìm thấy trả về -1 |
| 28 | int lastIndexOf(String substring)  int lastIndexOf(String substring, int fromIndex) |

Ví dụ: Tạo email từ chuỗi họ tên chuẩn.

|  |
| --- |
| **package** MyPackage;  **import** java.util.\*; // Scaner  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {    Scanner sc =**new** Scanner(System.***in***);  String hoTen = sc.nextLine();  sc.close();    hoTen = hoTen.toLowerCase();    // tim vi tri khoang trang dau tien tinh tu cuoi  **int** i = hoTen.length()-1;  **while** (hoTen.charAt(i) != ' ') i--;    // tao mail  String email = hoTen.substring(i+1); // lấy phần tên    email = email.concat(hoTen.substring(0, 1)); // lấy chữ cái đầu của họ  // lấy các chữ cái đầu lót trong họ tên nối vào email:  **int** j = 1;  **while** (j < i) { // vì i đánh dấu điểm bắt đầu phần tên    **if** (hoTen.charAt(j) == ' ') {  email = email.concat(hoTen.substring(j+1, j+2));  }  j++;  }    // nối đuôi "@mail.tdc.edu.vn" vào email  email = email.concat("@mail.tdc.edu.vn");    System.***out***.println(email);  }  } |

Kết quả:

Nguyen Thi Thuy Hong

[hongntt@mail.tdc.edu.vn](mailto:hongntt@mail.tdc.edu.vn)

### Calendar class

Lớp Calendar trong java là một lớp trừu tượng (abstract) cung cấp phương thức chuyển đổi ngày giữa một thời điểm cụ thể theo thời gian và một tập hợp các trường của calendar như MONTH, YEAR, HOUR, ... Nó kế thừa lớp Object và implements giao diện Comparable.

Các phương thức của lớp Calendar trong java:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| abstract void add(int field, int amount) | Nó được sử dụng để thêm hoặc trừ số lượng thời gian nhất định vào trường calendar đã cho, dựa trên các quy tắc của calendar. |
| int get(int field) | Nó được sử dụng để trả lại giá trị của trường calendar đã cho. |
| static Calendar getInstance() | Nó được dùng để lấy calendar sử dụng time zone và locale mặc định. |
| abstract int getMaximum(int field) | Nó được sử dụng để trả về giá trị MAX cho trường calendar đã cho của thể hiện Calendar hiện tại. |
| abstract int getMinimum(int field) | Nó được sử dụng để trả về giá trị MIN cho trường calendar đã cho của thể hiện Calendar hiện tại. |
| void set(int field, int value) | Nó được sử dụng để thiết lập trường cho trước vói giá trị đã cho. |
| void setTime(Date date) | Nó được sử dụng để thiết lập time của Calendar với Date đã cho. |
| Date getTime() | Nó được sử dụng để trả về đối tượng Date biểu diễn giá trị time của Calendar. |

Ví dụ về Calendar trong java:

|  |
| --- |
| **package** MyPackage;  **import** java.util.Calendar;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {    Calendar calendar = Calendar.*getInstance*();  System.***out***.println("Ngay gio hien tai: " + calendar.getTime());  calendar.add(Calendar.***DATE***, -15);  System.***out***.println("15 ngay sau: " + calendar.getTime());  calendar.add(Calendar.***MONTH***, 4);  System.***out***.println("4 thang sau: " + calendar.getTime());  calendar.add(Calendar.***YEAR***, 2);  System.***out***.println("2 nam sau: " + calendar.getTime());  }  } |

Kết quả:

Ngay gio hien tai: Tue Feb 20 11:27:03 ICT 2024

15 gay sau: Mon Feb 05 11:09:57 ICT 2024

4 thang sau: Wed Jun 05 11:09:57 ICT 2024

2 nam sau: Fri Jun 05 11:09:57 ICT 2026

Ví dụ về Calendar trong java - phương thức get():

|  |
| --- |
| **package** MyPackage;  **import** java.util.\*;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {    Calendar calendar = Calendar.*getInstance*();  System.***out***.println("Ngay gio hien tai: " + calendar.getTime());  System.***out***.println("YEAR: " + calendar.get(Calendar.***YEAR***));  System.***out***.println("MONTH: " + calendar.get(Calendar.***MONTH***));  System.***out***.println("DAY: " + calendar.get(Calendar.***DATE***));  System.***out***.println("DAY\_OF\_WEEK: " + calendar.get(Calendar.***DAY\_OF\_WEEK***));  }  } |

Kết quả:

Ngay gio hien tai: Tue Feb 20 11:30:52 ICT 2024

YEAR: 2024

MONTH: 1 // tháng January tương ứng với 0

DAY\_OF\_WEEK: 3

### Arrays class

Lớp *java.util.Arrays* chứa nhiều phương thức static đa dạng để xếp thứ tự và tìm kiếm các mảng, so sánh các mảng và điền các phần tử vào mảng.

|  |  |
| --- | --- |
| **STT** | **Phương thức và Miêu tả** |
| 1 | **public static int binarySearch(Object[] a, Object key)**  Tìm kiếm mảng của Object (byte, int, double, …) đã cho với giá trị đã xác định bởi sử dụng thuật toán tìm kiếm nhị phân. Mảng này phải được xếp thứ tự trước khi gọi phương thức này. Nó trả về chỉ mục của từ khóa tìm kiếm, nếu nó nằm trong danh sách, nếu không thì, bằng (-(điểm chèn + 1)). |
| 2 | **public static boolean equals(long[] a, long[] a2)**  Trả về true nếu hai mảng long đã cho là cân bằng nhau. Hai mảng này được cho là cân bằng nếu cả hai mảng chứa cùng số lượng phần tử, và tất cả các cặp phần tử tương ứng của hai mảng là cân bằng. Phương thức tương tự có thể được sử dụng bởi tất cả kiểu dữ liệu gốc khác (byte, short, int, …). |
| 3 | **public static void fill(int[] a, int val)**  Gán giá trị int đã cho tới mỗi phần tử của mảng int đã cho. Phương thức tương tự có thể được sử dụng bởi tất cả kiểu dữ liệu gốc khác (byte, short, int, …). |
| 4 | **public static void sort(Object[] a)**  Xếp thứ tự mảng các đối tượng đã cho theo thứ tự tăng dần. Phương thức tương tự có thể được sử dụng bởi tất cả kiểu dữ liệu gốc khác (byte, short, int, …). |
| 5 | **public static void sort( Integer[] a,Comparator.reverseOrder());**  Xếp thứ tự mảng các đối tượng đã cho theo thứ tự giảm dần. Phương thức tương tự có thể được sử dụng bởi tất cả kiểu dữ liệu gốc khác (chỉ sử dụng với các lớp wrapper của các kiểu dữ liệu cơ sở) |

### Exception Handling – Xử lý ngoại lệ

* Ngoại lệ (Exception) là gì?

*Exception* là một sự kiện xảy ra trong quá trình thực thi một chương trình Java, nó làm phá vỡ cái **flow**(luồng xử lý) bình thường của một chương trình, thậm chí chết chương trình.

Một ngoại lệ có thể xảy ra với nhiều lý do khác nhau, nó nằm ngoài dự tính của chương trình. Một vài ngoại lệ xảy ra bởi lỗi của người dùng, một số khác bởi lỗi của lập trình viên và số khác nữa đến từ lỗi của nguồn dữ liệu vật lý. Chẳng hạn như:

* Người dùng nhập dữ liệu không hợp lệ.
* Truy cập ngoài chỉ số mảng.
* Một file cần được mở nhưng không thể tìm thấy.
* Kết nối mạng bị ngắt trong quá trình thực hiện giao tiếp hoặc JVM hết bộ nhớ.
* ….

Tóm lại *Exception* có thể được sinh ra từ lỗi nào đó, như lỗi code ẩu của lập trình viên, lỗi nhập liệu ẩu bởi người dùng. Và *Exception* còn được sinh ra từ các tình huống không phải là lỗi nữa, như hết bộ nhớ, như rớt kết nối mạng, như sự can thiệp vô tình bởi người dùng,…

Ví dụ chương trình chia 2 số. Nếu ta cho mẫu số = 0 thì phát sinh lỗi và đó được coi là 1 ngoại lệ.

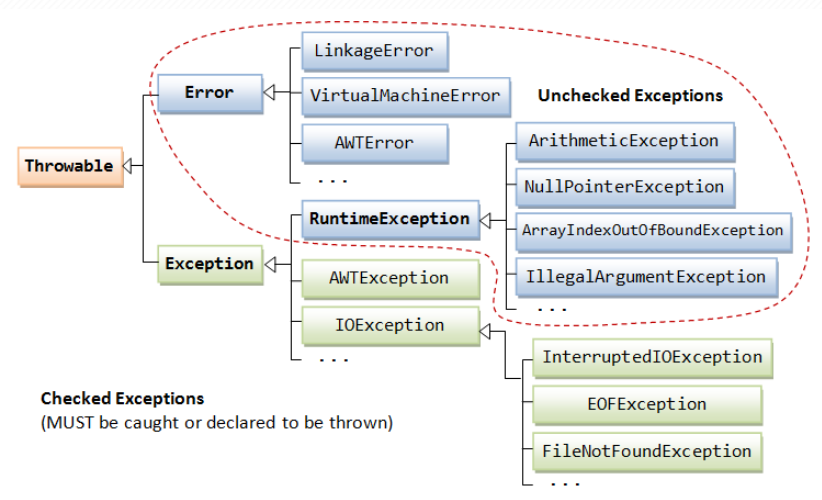
|  |
| --- |
| **package** MyPackage;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {  **int** zero = 0;  **int** average = 10 / zero;  System.***out***.println("Average = " + average);  }  } |

Lỗi ngoại lệ phát sinh:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at MyPackage.MyClass.main(MyClass.java:11)

Mô hình sơ đồ phân cấp của *Exception* trong java:



Hình 9. Sơ đồ phân cấp của Exception trong java

* Phân loại Exception

Trong Java có 2 loại exception: checked và unchecked. Tất cả các checked exception được kế thừa từ lớp Exception ngoại trừ lớp RuntimeException. RuntimeException là lớp cơ sở của tất cả các lớp unchecked exception. Điểm khác biệt giữa các lớp checked và unchecked expcetion chính là thời điểm xác định được expcetion có thể xảy ra.

**Checked exceptions:**

Là loại exception xảy ra trong lúc compile time, nó cũng có thể được gọi là **compile time exceptions**. Tức là bạn sẽ nhận được các thông báo lỗi từ trình biên dịch khi mà nó phát hiện ra rằng bạn đang code các dòng code nào đó mà có khả năng xảy ra *Exception.*

Ví dụ việc mở file, trình biên dịch nhận thấy lệnh này tiềm ẩn lỗi nên sẽ bắt xử lý lỗi:



**UnChecked exceptions:**

Là loại exception xảy ra tại thời điểm thực thi chương trình, nó cũng có thể gọi là **runtime exceptions** đó là programming bugs, lỗi logic của chương trình… Loại exception này được bỏ qua trong quá trình compile, không bắt buộc phải handle nó. *Exception* này khá nguy hiểm, khi mà trình biên dịch không thể nào kiểm tra lỗi. Nếu ứng dụng đến tay người dùng có thể chương trình sẽ bị crash (ngắt đột ngột) gây hoang mang cho người sử dụng.

Ví dụ một biến có giá trị null, thực hiện bất kỳ hoạt động nào bởi biến đó sẽ xảy ra ngoại lệ NullPointerException:

|  |
| --- |
| **package** MyPackage;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {  String obj = **null**;  System.***out***.print(obj.length());  }  } |

Kết quả:

Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "obj" is null

at MyPackage.MyClass.main(MyClass.java:7)

**Error:**

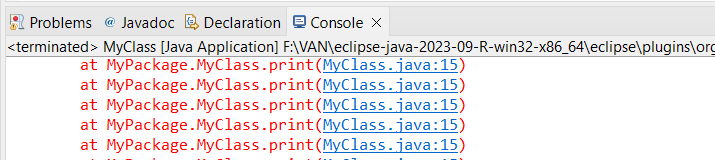
Error là những vấn đề nghiêm trọng liên quan đến môi trường thực thi của ứng dụng hoặc hệ thống mà lập trình viên không thể kiểm soát. Nó thường làm chết chương trình.

Lớp Error định nghĩa các ngoại lệ mà không thể bắt (catch) từ chương trình.

Ví dụ: OutOfMemoryError, VirtualMachineError, and StackOverflowError, …

Ví dụ chương trình đệ quy vô tận:

|  |
| --- |
| **package** MyPackage;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {  *print*();  }  **private** **static** **void** print() {  *print*();  }  } |



* Các kịch bản phổ biến nơi ngoại lệ có thể xảy ra
* ArithmeticException

Nếu chia bất kỳ số nào cho số 0, xảy ra ngoại lệ ArithmeticException.

Ví dụ:

int a = 10 / 0; // ArithmeticException

* NullPointerException

Nếu một bất kỳ biến nào có giá trị null, thực hiện bất kỳ hoạt động nào bởi biến đó sẽ xảy ra ngoại lệ NullPointerException.

Ví dụ:

String obj = null;

System.out.println(obj .length()); // NullPointerException

* NumberFormatException

Một biến String có giá trị là các ký tự, chuyển đổi biến này thành số sẽ xảy ra NumberFormatException.

Ví dụ:

String str = "abc";

int num = Integer.parseInt(str); // NumberFormatException

* ArrayIndexOutOfBoundsException

Nếu chèn bất kỳ giá trị nào vào index sai, sẽ xảy ra ngoại lệ ArrayIndexOutOfBoundsException.

Ví dụ:

int arr[] = new int[5];

arr[5] = 50; // ArrayIndexOutOfBoundsException

* ClassCastException

Nếu không thể chuyển kiểu object này sang kiểu object khác, sẽ xảy ra ngoại lệ ClassCastException.

Ví dụ:

Object dog = new Dog();

Rectangle rect = (Rectangle) dog;

* Xử lý ngoại lệ (Exception Handling) trong java:

Xử lý ngoại lệ (*Exception* *Handling*) trong java là một cơ chế xử lý các lỗi runtime để có thể duy trì luồng bình thường của ứng dụng.

Quá trình xử lý *Exception* được gọi là *Catch* *Exception*, nếu Runtime System không xử lý được ngoại lệ thì chương trình sẽ bị ngắt.

**Cú pháp xử lý ngoại lệ trong Java**:

Sử dụng *try* *catch* để bẻ luồng logic của chương trình, đảm bảo cho ứng dụng luôn “sống” do dù có bị các lỗi nào tác động đi chăng nữa.

Khối lệnh **try** trong java được sử dụng để chứa một đoạn code có thế xảy ra một ngoại lệ và phải được khai báo trong phương thức. Sau một khối lệnh try phải khai báo khối lệnh **catch** hoặc **finally** hoặc cả hai.

|  |
| --- |
| **try** {  // code có thể ném ra ngoại lệ (có khả năng gây ra Exception)  } **catch**(Exception\_class\_Name ex) {  // code xử lý ngoại lệ  } |

|  |
| --- |
| **try** {  // code có thể ném ra ngoại lệ  } **finally** {  // code trong khối này luôn được thực thi  } |

|  |
| --- |
| **try** {  // code có thể ném ra ngoại lệ  } **catch**(Exception\_class\_Name\_1 ex) {  // code xử lý ngoại lệ 1  } **catch**(Exception\_class\_Name\_2 ex) {  // code xử lý ngoại lệ 2  } **catch**(Exception\_class\_Name\_n ex) {  // code xử lý ngoại lệ n  } **finally** {  // code trong khối này luôn được thực thi  } |

Một số lưu ý:

* Tại một thời điểm chỉ xảy ra một ngoại lệ và tại một thời điểm **chỉ có một khối catch được thực thi**. Khi exception đã bị bắt ở một catch thì các catch tiếp theo sẽ không được bắt.
* Tất cả các khối catch phải được sắp xếp **từ cụ thể nhất đến chung nhất (từ exception con đến exception cha)**, tức là phải khai báo khối lệnh catch để xử lý lỗi NullPointerException, ArithmeticException, … trước khi khai báo catch để xử lý lỗi Exception.
* Khối lệnh finally luôn được thực thi dù chương trình có xảy ra ngoại lệ hay không (ngay cả sử dụng lệnh **return**).
* Đối với mỗi khối try, có thể không có hoặc nhiều khối catch, nhưng chỉ có một khối finally.
* Khối finally sẽ không được thực thi nếu chương trình bị thoát bằng cách gọi **System.exit()** hoặc xảy ra một lỗi (Error) không thể tránh khiến chương trình bị chết.

Sau đây là các ví dụ minh họa xử lý các ngoại lệ:

Ví dụ 1: Xử lý ngoại lệ thực hiện phép chia cho số 0

Trước khi xử lý ngoại lệ:

|  |
| --- |
| **package** MyPackage;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {  **int** zero = 0;  **int** average = 10 / zero;  System.***out***.println("Average = " + average)//(1);  System.***out***.println("Finished!"); // (2)  }  } |

Chương trình ở ví dụ trên sẽ bị bị lỗi chia cho 0 nên chương trình ngắt ngang. Đây là kết quả:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at MyPackage.MyClass.main(MyClass.java:8)

Xử lý ngoại lệ ArithmeticException cho chương trình trên:

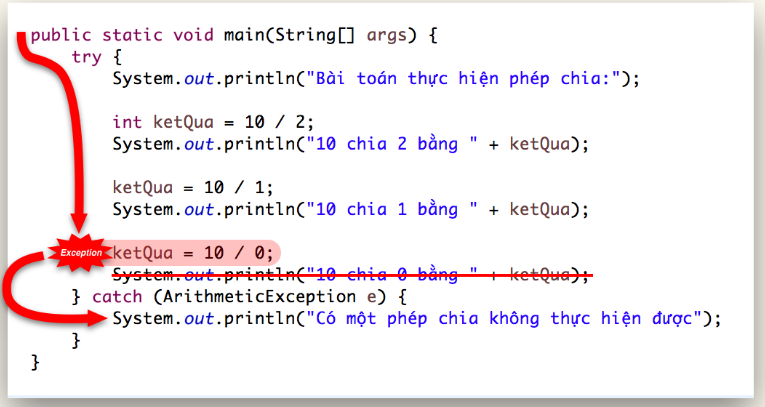
|  |
| --- |
| **package** MyPackage;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {  **try** {  **int** zero = 0;  **int** average = 10 / zero;  System.***out***.println("Average = " + average);  } **catch** (ArithmeticException ex) {  System.***out***.println(ex);  }  System.***out***.println("Finished!");  }  } |

Kết quả: Khối try bị lỗi, lỗi sẽ được xử lý ở khối catch, in ra lỗi và chương trình sẽ không bị ngắt.

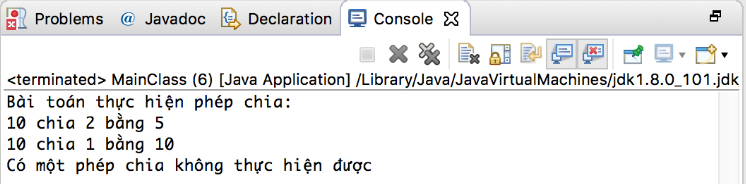
java.lang.ArithmeticException: / by zero

Finished!

Ví dụ 2:



Kết quả:



Ví dụ 3:

|  |
| --- |
| **public** **class** MyClass {  **public** **static** **void** main(String[] args) {  **try** {  **int** arr[] = **new** **int**[5];  arr[5] = 4;  System.***out***.println("arr[5] = " + arr[5]);  } **catch** (NullPointerException ex) {  System.***out***.println(ex);  } **finally** {  System.***out***.println("Khối lệnh finally luôn được thực thi");  }  System.***out***.println("Finished!");  }  } |

Kết quả:

Exception in thread "main" Khối lệnh finally luôn được thực thi

java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5

at MyPackage.MyClass.main(MyClass.java:8)

### Sơ Đồ lớp

Class được mô tả gồm tên Class, thuộc tính và phương thức.

|  |
| --- |
| Class name (tên lớp) |
| Attributes (Các thuộc tính / dữ liệu) |
| Methods (Các phương thức/ hành động) |

Ví dụ: class SinhVien

|  |
| --- |
| **Student** |
| - studentName: string  - idNumber: int  - birthday: DateTime  - address: string  + numOfStudent : static int |
| + Student()  + Student(string, int, DateTime, string)  + GetAge(): int  +CreateEmail(): string  + Print(): void |

### Một số ngôn ngữ hỗ trợ lập trình Hướng đối tượng

Xuất phát từ tư tưởng của ngôn ngữ SIMULA67, trung tâm nghiên cứu Palo Alto (PARC) của hãng XEROR đã tập trung nghiên cứu để hoàn thiện ngôn ngữ OOP đầu tiên với tên gọi là Smalltalk. Sau đó các ngôn ngữ OOP lần lượt ra đời như Eiffel, Clos, Loops, Flavors, Object Pascal, Object C, C++, Delphi, Java…

Chính XEROR trên cơ sở ngôn ngữ OOP đã đề ra tư tưởng “icon base screen interface”, kể từ đó Apple Macintosh cũng như Microsoft Windows phát triển giao diện đồ họa như ngày nay. Trong Microsoft Windows, tư tưởng OOP được thể hiện một cách rõ nét nhất đó là "click vào đối tượng", mỗi đối tượng có thể là control menu, control menu box, menu bar, scroll bar, button, minimize box, maximize box, … sẽ đáp ứng công việc tùy theo đặc tính của đối tượng. Turbo Vision của hãng Borland là một ứng dụng OOP tuyệt vời, giúp lập trình viên không quan tâm đến chi tiết của chương trình giao diện mà chỉ cần thực hiện các nội dung chính của vấn đề.

### Bài tập áp dụng

**BÀI TẬP THỰC HÀNH SỐ 1**

1. **Thông tin chung:**

* Mã số bài tập : HW1-LTHDT
* Hình thức nộp bài : Nộp qua hệ thống EL môn học
* Thời hạn nộp bài : … / … / ……
* Nội dung : Chương 1: Giới thiệu lập trình hướng đối tượng

**Chuẩn đầu ra cần đạt:**

|  |  |
| --- | --- |
| L.O.1 | Phân tích, thiết kế chương trình theo hướng đối tượng. Cài đặt được sơ đồ lớp cho các chương trình vừa và nhỏ; |
| L.O.8 | Làm bài tập và nộp bài đúng quy định. |

**Sinh viên thực hành các yêu cầu sau:**

**BÀI TẬP**

Bài 1: Sử dụng thư viện String để viết chương trình chứa các hàm sau:

Hàm đếm số ký tự là chữ số có trong chuỗi

Hàm đổi chuỗi số thành số.

Hàm đếm số từ trong một chuỗi

Hàm đảo ngược chuỗi

Hàm tạo email từ họ tên của một người.

Ví dụ hoTen = “Le Thi Lan Anh” 🡪 email: anhltl@gmail.com

Bài 2: Viết chương trình cho phép nhập vào ngày tháng năm sinh của một người. Hãy tính tuổi của người đó đến thời điểm hiện tại và cho biết người đó sinh vào ngày thứ mấy trong tuần.

Bài 3: Hãy bắt ngoại lệ cho đoạn code sau:

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // Tạo một mảng 10 các số nguyên ngẫu nhiên  **int** randomIntNumbers[] = **new** **int**[10];  Random rand = **new** Random();  **for**(**int** i = 0; i < 10; i++) {  randomIntNumbers[i] = rand.nextInt(100);  }  Scanner input = **new** Scanner(System.***in***);  System.***out***.println("Bạn muốn in ra phần tử mảng thứ mấy? ");  **int** index = input.nextInt();  System.***out***.println("OK, phần tử mảng thứ " + index + " mang giá trị " + randomIntNumbers[index]);  input.close();  } |

Bài 4. Hãy vẽ sơ đồ lớp cho các lớp đối tượng sau:

1. Sinh viên (Student)
2. Cuốn sách (Book)
3. Nhân viên (Employee)
4. Tài khoản ngân hàng (Bank Account)
5. Thời gian (Time)
6. Phân số (Fraction)
7. Điểm trên trục tọa độ (Point)

|  |  |
| --- | --- |
|  | XÂY DỰNG LỚP ĐỐI TƯỢNG |

Chương này nhằm giới thiệu cho sinh viên cách thiết kế một lớp đối tượng, xây dựng các thành phần của một lớp đối tượng và cách sử dụng đối tượng để giải quyết một số bài toán cơ bản theo yêu cầu.

### Định nghĩa lớp

Như chúng ta đã biết một lớp bao gồm ba thành phần sau:

|  |
| --- |
| Class name (tên lớp) |
| Fields (Các trường dữ liệu) |
| Methods (Các phương thức/ hành động) |

Ví dụ: ta có lớp Student như sau

|  |
| --- |
| **Student** |
| - studentName: string  - idNumber: int  - birthday: DateTime  - address: string  + numOfStudent : static int |
| + Student()  + Student(string, int, DateTime, string)  + GetAge(): int  +CreateEmail(): string  + Print(): void |

Ký hiệu: public (+), protected (#), internal (~), private (-).

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <Phạm vi truy cập> **class** <Tên lớp> {      // Khai báo thuộc tính (field) hay biến      <Phạm vi truy cập> <kiểu dữ liệu> <field\_1>;      <Phạm vi truy cập> <kiểu dữ liệu> <field\_2>;      // Hàm khởi tạo không có tham số      <Phạm vi truy cập> <Tên lớp>(){          // Khởi tạo dữ liệu      }      // Hàm khởi tạo có tham số      <Phạm vi truy cập> <Tên lớp>(<Kiểu dữ liệu> <tên biến>[, <Kiểu dữ liệu> <tên biến>]){          // Khởi tạo dữ liệu      }      // Phương thức xử lý (method)      <Kiểu trả về> method\_1() {          // Logic xử lý      }      <Phạm vi truy cập> <Kiểu trả về> method\_2(<Kiểu dữ liệu> <tên biến>) {          // Logic xử lý      }      <Phạm vi truy cập> <Từ khoá tuỳ chọn> <Kiểu trả về> method\_3(<Kiểu dữ liệu> <tên biến>) {          // Logic xử lý      }  } | |

Trong đó:

* class: là từ khóa để khai báo lớp trong Java.
* Tên lớp: là tên chúng ta đặt cho lớp (quy tắc đặt tên: viết hoa mỗi chữ cái đầu).
* field\_1, field\_2: các thuộc tính, các biến, hay các thành phần dữ liệu của lớp (quy tắc đặt tên: chữ cái đầu tiên viết thường, các chữ cái đầu tiên tiếp theo viết hoa).
* Hàm khởi tạo (constructor): hàm dùng để khởi tạo đối tượng lớp.
* method\_1, method\_2, method\_3: là các phương thức thể hiện các thao tác xử lý, tác động lên các thành phần dữ liệu của lớp.
* Kiểu trả về có thể là primitive type, Wrapper class, array, enum, Collection, Object, …
* Phạm vi truy cập: private, Default (Package-Private) Access, protected, public.
* Từ khoá tuỳ chọn bao gồm: static, abstract, final, synchronized, …

Ví dụ:

|  |
| --- |
| **public** **class** Student {  // Khai báo thuộc tính  **private** **int** id;  **private** String name;    // Hàm khởi tạo có 1 tham số là id  **public** Student(**int** id) {  // Sử dụng từ khóa this để truy cập thuộc tính trong lớp  // nếu không sử dụng từ khóa this thì đang truy cập vào tham số của hàm khởi tạo  **this**.id = id;  }    // Cung cấp phương thức lấy giá trị id  **public** **int** getId() {  **return** id;  }    // Cung cấp phương thức lấy giá trị name  **public** String getName() {  **return** name;  }    // Cung cấp phương thức gán giá trị name  **public** **void** setName(String name) {  **this**.name = name;  }  } |

### Bổ từ truy xuất

**Bổ từ truy xuất (Access modifiers)**: Từ khoá khai báo quyền truy cập của class. Trong Java có các từ khoá khai báo quyền truy cập của lớp như sau: ***public***, ***protected,*** ***private***, ***default.*** Trường hợp người dùng không khai báo quyền truy cập cho lớp thì quyền truy cập mặc định là ***default***.

Ký hiệu trong UML được sử dụng như sau: public (+), protected (#), internal (~), private (-).

Sau đây là một số nguyên tắc chung, nếu tuân thủ chương trình sẽ dễ dàng xây dựng và mở rộng hơn:

*Phạm vi truy cập private:*

Private Access Modifier chỉ được truy cập trong phạm vi lớp.

Ví dụ về private access modifier trong java:

|  |
| --- |
| **class** A {  **private** **int** data = 40;  **private** **void** msg() {  System.***out***.println("Hello java");  }  }  **public** **class** Simple {  **public** **static** **void** main(String args[]) {  A obj = **new** A();  System.***out***.println(obj.data);// Compile Time Error  obj.msg();// Compile Time Error  }  } |

Chú ý một lớp không thể là private, protected ngoại trừ lớp lồng.

*Phạm vi truy cập default:*

Default Access Modifier là chỉ được phép truy cập trong cùng package.

Ví dụ:

|  |
| --- |
| // Lưu file với tên A.java  **package** demo;  **class** A {  **void** msg() {  System.***out***.println("Hello");  }  } |
| // Lưu file với tên B.java  **package** mypack;  **import** demo.\*;  **public** **class** B {  **public** **static** **void** main(String args[]) {  A obj = **new** A(); // Compile Time Error  obj.msg(); // Compile Time Error  }  } |

*Phạm vi truy cập proteced:*

Protected access modifier được truy cập nội tại trong lớp khai báo và trong các lớp con trong và ngoài package. Phạm vi này có thể áp dụng cho thuộc tính, phương thức, contructor nhưng không áp dụng cho lớp.

Ví dụ:

|  |
| --- |
| // Lưu file với tên A.java  **package** vn.demo;  **public** **class** A {  **protected** **void** msg() {  System.***out***.println("Hello");  }  } |
| // Lưu file với tên B.java  **package** vn.mypack;  **import** vn.demo.\*;  **public** **class** B **extends** A {  **public** **static** **void** main(String args[]) {  B obj = **new** B();  obj.msg();  }  } |

*Phạm vi truy cập public:*

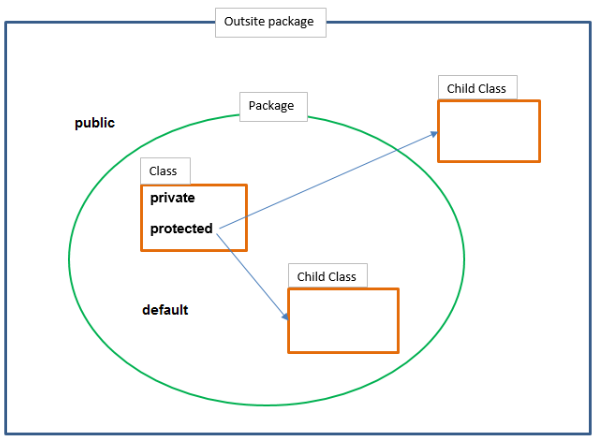
Public access modifier được truy cập ở mọi nơi.

Ví dụ:

|  |
| --- |
| // Lưu file với tên A.java  **package** vn.demo;  **public** **class** A {  **public** **void** msg() {  System.***out***.println("Hello");  }  } |
| // Lưu file với tên B.java  **package** vn.mypack;  **import** vn.demo.\*;  **public** **class** B {  **public** **static** **void** main(String args[]) {  A obj = **new** A();  obj.msg();  }  } |

Phạm vi truy cập được tổng kết qua bảng sau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Modifier | Trong lớp | Trong package | Ngoài package bởi lớp con | Ngoài package |
| Private | Y | N | N | N |
| Default | Y | Y | N | N |
| Protected | Y | Y | Y | N |
| Public | Y | Y | Y | Y |



Hình minh họa acccess modifier

Có thể xem đối tượng như là một cái hộp đen: người dùng không cần biết cách đối tượng làm việc mà chỉ quan tâm đối tượng có thể làm gì.

Một nguyên tắc trong thiết kế lớp là dữ liệu nên để thuộc tính truy xuất là private và các phương thức mà cung cấp dịch vụ cho khách hàng nên để là public. Mục tiêu của việc này là để:

* Giảm lỗi: tại nơi sử dụng không thể sử dụng các thành phần private của lớp vì nó đã bị ẩn với người dùng, từ đó giảm thiểu lỗi phát sinh khi thao tác với dữ liệu.
* Che dấu sự phức tạp: chi tiết cài đặt của các chức năng đối tượng đã được ẩn. Tại nơi sử dụng chỉ cần quan tâm chức năng đó có thể làm gì mà không cần quan tâm tới chi tiết cài đặt của chức năng đó.

### Phân loại lớp

Ngôn ngữ Java hỗ trợ các loại lớp như sau:

**Normal class**: lớp được khai báo với từ khoá **class**. Các thành viên của lớp được khai báo ngay trong thân lớp như các trường (fields), thuộc tính (properties) và các phương thức (methods) hay các sự kiện (events). Khi sử dụng lớp này cần phải tạo ra một đối tượng cụ thể của lớp

**Final class:** lớp không hỗ trợ kế thừa, không thể mở rộng thêm.

|  |
| --- |
| **public** **final** **class** JavaExample {  **public** **static** **void** main(String[] args) {    }    **class** Test **extends** JavaExample {    }  } |

**Sealed** **class (lớp niêm phong)**: có thể coi là dạng linh hoạt hơn của Final class. Trong thực tế có một số trường hợp cần chỉ định lớp nào được thừa kế từ lớp cha, lớp nào không. Sealed class sẽ giúp hạn chế việc kế thừa từ một lớp nào đó. Chúng ta có thể kiểm soát lớp nào được thừa kế từ lớp cha bằng cách sử dụng cặp từ khóa sealed và permits:

|  |
| --- |
| **public** **sealed** **class** JavaExample **permits** JavaExample.Test1, JavaExample.Test2 {  **public** **static** **void** main(String[] args) {    }    **final** **class** Test1 **extends** JavaExample {    }    **final** **class** Test2 **extends** JavaExample {    }  } |

Ở ví dụ trên tại dòng khai báo lớp JavaExample sẽ có từ khóa permits để liệt kê các lớp con sẽ được kế thừa từ lớp này. Lưu ý rằng, cần phải khai báo các lớp con là final để không không class nào được kế thừa từ lớp con này nữa. Nếu có nhu cầu cho phép class Test1 được dùng để cho lớp khác kế thừa thì hãy khai báo class Test1 này với từ khóa non-sealed.

|  |
| --- |
| **public** **sealed** **class** JavaExample **permits** JavaExample.Test1, JavaExample.Test2 {  **public** **static** **void** main(String[] args) {    }    **non-sealed** **class** Test1 **extends** JavaExample {    }    **class** Test3 **extends** Test1 {    }    **final** **class** Test2 **extends** JavaExample {    }  } |

**Static** **class**: lớp tĩnh được khai báo với từ khoá **static** **class.** Người dùng có thể sử dụng lớp bất cứ khi nào trong chương trình mà không cần tạo ra các thể hiện của lớp. Thường được dùng để khai báo các tiện ích hoặc hằng số.

**Abstract** **class**: lớp ảo được khai báo với từ khoá **abstract** **class.** Một lớp trừu tượng có nghĩa là không khởi tạo được đối tượng của lớp này, nhưng cho phép thừa kế để tạo ra lớp con.

### Thuộc tính

Thuộc tính (trường dữ liệu - field) là các biến dùng để lưu trữ dữ liệu của lớp. Kiểu của field có thể là các kiểu dữ liệu có sẵn trong Java hoặc là một kiểu class.

Cú khai báo fields giống như khai báo biến trong Java:

[access modifier] data\_type field\_Name [= initial value];

Ví dụ:

private string customerName = “unknown”;

• **access modifier**: từ khoá khai báo quyền truy cập của thuộc tính bao gồm: public, internal, protected, private. Nếu người dùng không khai báo quyền truy cập thì C# sử dụng quyền truy cập mặc định là private.

• **data\_type**: tên kiểu dữ liệu của thuộc tính. Có thể sử dụng các kiểu dữ liệu cơ sở, enum, array, interface, class, …

• **field\_Name**: tên nhận dạng thuộc tính. Tên thuộc tính được đặt theo quy tắt camelCase hoặc bắt đầu bằng một ký tự\_ và viết hoa đầu từ, tên không chứa khoảng trắng.

Ví dụ: idNumber hoặc \_IdNumber

• initial value: giá trị khởi đầu của thuộc tính nếu có.

Ví dụ: Với lớp Student, ta có khai báo các fields như sau:

|  |
| --- |
| **public** **class** Student  {  //Khai báo fields  **private** string studentName;  **private** **int** idNumber;  **private** DateTime birthday;  **private** string address;  **public** **static** **int** numOfStudent;  } |

Tính đóng gói (Encapsulation) trong OOP thể hiện ở việc ngăn chặn không cho người dùng tác động trực tiếp lên các dữ liệu của lớp. Để thực hiện việc này, ta khai báo quyền truy cập cho các trường (fields) là **private**. Để cung cấp cho người dùng có thể truy xuất, cập nhật thông tin dữ liệu thì lập trình viên sẽ thiết kế các phương thức để lấy dữ liệu (getter) hoặc gán dữ liệu (setter). Việc cung cấp chức năng này sẽ giúp lập trình viên kiểm soát được dữ liệu.

### Cài đặt các phương thức thành viên (Methods)

Một phương thức thành viên của lớp sẽ được định nghĩa bên trong lớp. Phương thức sẽ hoạt động trên tất cả các đối tượng của lớp, và có quyền truy cập vào tất cả các thành viên của lớp.

Tính đóng gói (encapsulation) thể hiện ở chỗ phía client sẽ không nhìn thấy các chi tiết định nghĩa phức tạp bên trong phương thức mà chỉ cần quan tâm đến việc sử dụng phương thức đó như thế nào và kết quả trả về của nó. Đa phần các phương thức của lớp nên để phạm vi truy cập là public, trừ một số phương thức tạo ra với mục đích phụ vụ nội tại trong lớp thì mới để private hoặc một số phương thức muốn tạo ra phục vụ thêm cho các lớp con thì sẽ để phạm vi là protected.

Một số loại phương thức trong lớp như sau:

* Các phương thức khởi tạo (Constructors): gồm các loại sau
  + Phương thức khởi tạo mặc định (default constructor).
  + Phương thức khởi tạo không tham số (no-argument).
  + Phương thức khởi tạo có tham số (parameterized constructor).
  + Phương thức khởi tạo sao chép (copy constructor).
* Phương thức hủy (destroy constructor): dùng để thu hồi bộ nhớ đã cấp phát cho các đối tượng.
* Các phương thức hỗ trợ người dùng truy xuất dữ liệu: phương thức get ().
* Các phương thức hỗ trợ người dùng cập nhật dữ liệu: phương thức set ().
* Phương thức tĩnh

### Phương thức khởi tạo (constructors)

Khi một lớp được tạo ra, phương thức khởi tạo của nó sẽ được gọi. Để đảm bảo các thuộc tính của một đối tượng có giá trị ban đầu hợp lý trước khi người dùng bắt đầu sử dụng đối tượng, người ta sử dụng phương thức khởi tạo (constructors). Trong một lớp có thể có nhiều phương thức khởi tạo (overload).

Đặc điểm của phương thức khởi tạo:

* Tên phương thức khởi tạo phải trùng với tên lớp và không có kiểu trả về kể cả kiểu void
* Là một phương thức đặc biệt được gọi tự động tại thời điểm đối tượng được tạo ra.

Trong Java có các loại phương thức khởi tạo sau:

* ***Phương thức khởi tạo mặc định (default constructor)***: là phương thức khởi tạo do trình biên dịch tự động phát sinh khi không có khai báo phương thức khởi tạo nào. Giá trị mặc định tùy thuộc kiểu dữ liệu của các thuộc tính (số nguyên là 0, số thực là 0.0, boolean là false, đối tượng là null).
* ***Phương thức khởi tạo không tham số (No-args):*** No-args constructor là constructor không có tham số (no arguments), cũng giống như default constructor. Tuy nhiên, nếu định nghĩa một no-args constructor, Java sẽ không tự động tạo ra một default constructor. No-args constructor cho phép kiểm soát cách các trường dữ liệu được khởi tạo. Ví dụ như thiết lập kết nối với cơ sở dữ liệu hoặc thiết lập một số giá trị mặc định cho thuộc tính của đối tượng.

Ví dụ:

|  |
| --- |
| **public** **class** Person {  **private** String name;  **private** **int** age;  // No-args constructor  **public** Person() {  name = “Unknown”;  age = 0;  }  } |

* ***Phương thức khởi tạo có tham số (parametterized constructor)***: là phương thức khởi tạo đối tượng với giá trị khởi đầu của các trường do người dùng truyền vào. Có thể khởi tạo đầy đủ tham số hoặc chỉ một phần tham số cho đối tượng.

Ví dụ 1:

|  |
| --- |
| **public** **class** Person {  **private** String name;  **private** **int** age;  **public** Person(String name, **int** age) {  **this**.name = name;  **this**.age = age;  }  } |

Ví dụ 2:

|  |
| --- |
| **public** **class** Person {  **private** String name;  **private** **int** age;  **public** Person(String name) {  **this**.name = name;  **this**.age = 0;  }  } |

* **Phương thức khởi tạo sao chép**:

Copy constructor trong Java là một constructor đặc biệt được sử dụng để tạo ra một đối tượng mới là bản sao của một đối tượng đã tồn tại. Copy constructor thường được định nghĩa bằng cách sử dụng từ khóa “this” để trỏ đến đối tượng mới được tạo ra.

Để sao chép các giá trị của một đối tượng sang một đối tượng khác trong Java, ngoài copy constructor, ta có thể sử dụng hai phương pháp sau: gán giá trị của một đối tượng cho đối tượng khác và sử dụng phương thức clone() của lớp Object. Tuy nhiên, copy constructor thường được ưu tiên sử dụng trong các trường hợp sau:

* Khi ta muốn tạo ra một đối tượng mới với các giá trị thuộc tính giống với một đối tượng đã tồn tại.
* Khi ta muốn thực hiện các thao tác trên một đối tượng nhưng không muốn ảnh hưởng đến đối tượng ban đầu.
* Khi muốn thực hiện phép toán trên các đối tượng.

Ví dụ:

File Car.java:

|  |
| --- |
| **public** **class** Car {  **private** String model;  **private** **int** year;  // Constructor  **public** Car(String model, **int** year) {  **this**.model = model;  **this**.year = year;  }  // Copy constructor  **public** Car(Car otherCar) {  **this**.model = otherCar.model;  **this**.year = otherCar.year;  }  // Getters and setters  **public** String getModel() {  **return** model;  }  **public** **void** setModel(String model) {  **this**.model = model;  }  **public** **int** getYear() {  **return** year;  }  **public** **void** setYear(**int** year) {  **this**.year = year;  }  } |

File Myclass.java:

|  |
| --- |
| **public** **class** MyClass {  **public** **static** **void** main(String[] args) {  // Tạo một đối tượng Car với model là “BMW” và năm sản xuất là 2021  Car car1 = **new** Car("BMW", 2021);  // Sử dụng copy constructor để tạo một đối tượng Car mới từ car1  Car car2 = **new** Car(car1);  // Thay đổi model và năm sản xuất của car2  car2.setModel("Mercedes");  car2.setYear(2022);  // In ra model và năm sản xuất của cả hai xe  System.***out***.println("Car 1:" + car1.getModel() + "," + car1.getYear());  System.***out***.println("Car 2: " + car2.getModel() + "," + car2.getYear());  }  } |

Kết quả:

*Car 1:BMW,2021*

*Car 2: Mercedes,2022*

Trong Java, một lớp có thể có nhiều hơn một constructor. Khi một đối tượng được khởi tạo, một trong các constructor sẽ được gọi để khởi tạo đối tượng đó. Constructor chaining là cơ chế cho phép gọi một constructor khác trong cùng lớp hoặc lớp cha của nó, thay vì viết lại các đoạn mã khởi tạo lại thuộc tính của lớp. Điều này cho phép tránh việc lặp lại mã trong các constructor khác nhau và tăng tính tái sử dụng của mã.

Khi gọi constructor trong cùng lớp: từ khóa “this” được sử dụng để gọi constructor khác trong lớp đó. Constructor gọi đến sẽ được thực thi trước constructor gọi nó. Lưu ý: Từ khóa “this” phải là câu lệnh đầu tiên trong constructor.

Ví dụ:

|  |
| --- |
| **public** **class** Car {  **private** String model;  **private** **int** year;  // Constructor  **public** Car ()  {  **this**("Unknow", 0);  }  **public** Car(String model, **int** year) {  **this**.model = model;  **this**.year = year;  } |

Trong trường hợp gọi constructor của lớp cha: từ khóa “super” được sử dụng để gọi constructor của lớp cha. Constructor của lớp cha sẽ được thực thi trước constructor của lớp con.  Lưu ý: từ khoá super phải là câu lệnh đầu tiên trong constructor. Nếu không có lệnh “super()” trong hàm khởi tạo của lớp con, Java sẽ tự động gọi hàm khởi tạo mặc định (no-args constructor) của lớp cha.

Ví dụ:

|  |
| --- |
| **public** **class** Animal {  **private** String name;  **private** **int** age;  **public** Animal(String name) {  **this**.name = name;  }  }  **public** **class** Dog **extends** Animal {  **private** String breed;  **public** Dog(String name, String breed) {  **super**(name);  **this**.breed = breed;  } |

### Phương thức hủy đối tượng (destructors / Finalizers)

Trong Java, các đối tượng cấp phát động bằng toán tử new, khi không tồn tại biến nào trỏ đến đối tượng, đối tượng đó xem như không còn cần đến nữa và bộ nhớ dành cho nó có thể được tự động giải phóng bởi bộ thu gom rác (Garbage Collector). Trình thu gom rác hoạt động trong một tuyến đoạn (Thread) độc lập với chương trình. Vì thế chúng ta không phải bận tâm gì với công việc này.

Java cũng cho phép viết hàm hủy, có thể cũng cần thiết cho những trường hợp nào đó. Hàm hủy trong Java chỉ được gọi bởi trình thu gom rác, rất khó đoán trước vào lúc nào hàm hủy sẽ được gọi, do đó hàm huỷ ít được sử dụng. Dạng hàm hủy như sau :

class A {

// Khai báo các thuộc tính

public void method\_1() {

// …

}

protected void finalize() {

// Có thể dùng để đóng tất cả các kết nối

// vào cơ sở dữ liệu trước khi hủy đối tượng.

// …

}

}

### Con trỏ this

Trong định nghĩa phương thức, lập trình viên có thể truy cập tới đối số ngầm định thông qua từ khóa this. Trong thân phương thức thì this dùng tham chiếu tới đối tượng đang thực hiện lời gọi đến phương thức đó.

Một vài lập trình viên luôn sử dụng this để dễ đọc hơn và để chỉ đây là thuộc tính của lớp chứ không phải là một biến toàn cục hay cục bộ, giúp tránh nhầm lẫn.

Ví dụ:

|  |
| --- |
| **public** **class** Car {  **private** String model;  **private** **int** year;  // Constructor  **public** Car(String model, **int** year) {  **this**.model = model;  **this**.year = year;  }    // Getters and setters  **public** String getModel() {  **return** model;  }  **public** **void** setModel(String model) {  **this**.model = model;  }  **public** **int** getYear() {  **return** year;  }  **public** **void** setYear(**int** year) {  **this**.year = year;  }  } |

### Tạo và sử dụng đối tượng

Sau khi khai báo lớp thì ta có thể sử dụng lớp để tạo đối tượng. Lớp tạo ra chính là một kiểu dữ liệu trừu tượng, từ đó có thể tạo ra biến để chứa đối tượng.

Cú pháp :

**TenLop tenBien = new TenLop(<tham số khởi tạo đối tượng>) ;**

Ví dụ :

|  |
| --- |
| **public** **class** MyClass {  **public** **static** **void** main(String[] args) {  // Tạo một đối tượng Car với model là “BMW” và năm sản xuất là 2021  Car car1 = **new** Car("BMW", 2021);  // In ra model và năm sản xuất của cả hai xe  System.***out***.println("Car 1:" + car1.getModel() + "," + car1.getYear());  }  } |

Sau khi tạo và khởi tạo đối tượng (tức tạo thực thể - instance) thì có thể gọi thực thi các chức năng (public) của đối tượng để thực thi (trong lớp khai báo chức năng gì thì đó chính là chức năng của đối tượng).

### Từ khóa Static

Trong Java, từ khóa **static** được sử dụng để quản lý bộ nhớ tốt hơn và nó có thể được truy cập trực tiếp thông qua lớp mà không cần khởi tạo.

Từ khóa static thuộc về lớp chứ không thuộc về instance (thể hiện) của lớp.

Chúng ta có thể áp dụng từ khóa static với các biến, các phương thức, các khối, các lớp lồng nhau (nested class).

Các trường hợp sử dụng static:

1. **Biến static (static variables):** khi khai báo một biến là static, thì biến đó được gọi là biến tĩnh, hay biến static.
2. **Phương thức static (static methods):** khi khai báo một phương thức là static, thì phương thức đó gọi là phương thức static.
3. **Khối static (static blocks):** được sử dụng để khởi tạo thành viên dữ liệu static.
4. **Lớp static (static class)**: một class được có thể được đặt là static chỉ khi nó là một nested class. Một nested static class có thể được truy cập mà không cần một object của outer class (lớp bên ngoài).
5. **Import static**: từ phiên bản Java 5, cho phép import các thành viên tĩnh (static member) của một class hoặc package vào một class khác bằng cách sử dụng từ khóa import và sau đó sử dụng chúng như là thành viên của lớp đó.

### Biến static (static variables)

Trong Java, biến có thể được khai báo cùng với từ khóa static, và lúc đó nó có thể được gọi là **class variable.**

Việc cấp phát bộ nhớ cho biến static chỉ xảy ra một lần khi class được nạp vào bộ nhớ. Giá trị mặc định khi khai báo và khởi tạo biến static và non-static là giống nhau, cụ thể:

primitive integers (long, int, short, byte): **0**

primitive floating points (double, float): **0.0**

boolean: **false**

object references: **null**

Biến static có thể được sử dụng làm thuộc tính chung, để dùng chung dữ liệu cho tất cả objects (hoặc instances) của lớp đó và điều đó giúp cho chương trình tiết kiệm bộ nhớ hơn. Nếu một biến được khai báo với từ khóa static thì có thể truy cập trực tiếp thông qua lớp.

Nếu một biến vừa khai báo từ khóa final vừa khai báo từ khóa static thì nó được xem như là một hằng số. Một hằng số nên được viết hoa và nếu có nhiều từ thì phân cách bằng dấu gạch dưới (**\_)**

Ví dụ 1: public static final PI = 3.14;

Ví dụ 2: Website gpcoder.com có rất nhiều bài viết, mỗi bài viết cần hiển thị địa chỉ của website dưới mỗi bài viết, địa chỉ này giống nhau và có nhiều lớp cần sử dụng. Để tiết kiệm bộ nhớ, dễ dàng chia sẻ và sử dụng ở các lớp khác. Chúng ta có thể sử dụng từ khóa static như sau:

MyWebsite.java

|  |
| --- |
| **public** **class** MyWebsite {  **public** **static** String *WEBSITE* = "upcoder.com.vn";  } |

UsingStaticExample.java

|  |
| --- |
| **public** **class** UsingStaticExample {  **private** String subject;    UsingStaticExample (String subject) {  **this**.subject = subject;  }    **public** **void** print() {  System.***out***.println("Subject = " + subject);  System.***out***.println("Website = " + MyWebsite.WEBSITE);  }    **public** **static** **void** main(String[] args) {  UsingStaticExample ex1 = **new** UsingStaticExample("Core Java");  ex1.print();  System.***out***.println("----");  UsingStaticExample ex2 = **new** UsingStaticExample("Object Oriented Programing");  ex2.print();  }  } |

Kết quả:

|  |
| --- |
| Subject = Core Java  Website = upcoder.com.vn  ----  Subject = Object Oriented Programing  Website = upcoder.com.vn |

Một ví dụ khác thường hay được sử dụng để minh họa cho việc sử dụng từ khóa static là bộ đếm Counter. Nếu có một website và chúng ta cần đếm số lượt truy cập vào trang, chương trình như sau:

|  |
| --- |
| **public** **class** Counter {  **static** **int** *count* = 0;    **public** Counter() {    }    **public** **void** visit() {  *count*++;  **this**.print();  }    **public** **void** print() {  System.***out***.println("count = " + *count*);  }    **public** **static** **void** main(String[] args) {  Counter c1 = **new** Counter();  c1.visit();  Counter c2 = **new** Counter();  c2.visit();  Counter c3 = **new** Counter();  c3.visit();  }  } |

Kết quả:

|  |
| --- |
| count = 1  count = 2  count = 3 |

### Phương thức static (Static Methods)

Nếu một phương thức được khai báo với từ khóa **static** thì phương thức đó được gọi là phương thức static.

Một số đặc điểm:

* Một phương thức static thuộc lớp chứ không phải đối tượng của lớp.
* Một phương thức static có thể được gọi mà không cần tạo khởi tạo (instance) của một lớp.
* Phương thức static có thể truy cập biến static và có thể thay đổi giá trị của nó.
* Một phương thức static chỉ có thể gọi một phương thức static khác, **không** thể gọi được một phương thức **non-static**.
* Một phương thức static **không** thể được sử dụng từ khóa **this** và **super**.
* Người dùng không thể override (đè) phương thức static trong Java, bởi vì kỹ thuật đè (overriding) phương thức được dựa trên quá trình gán (binding) động khi chương trình đang chạy (runtime) và những phương thức static được gán tĩnh trong thời gian biên dịch. Phương thức tĩnh không ràng buộc với thực thể của lớp nên phương thức tĩnh sẽ không thể override (ghi đè).

Khi nào sử dụng từ khóa static cho một phương thức?

* Khi phương thức không phụ thuộc vào trạng thái của đối tượng, nghĩa là không cần sử dụng bất kỳ dữ liệu thành viên nào của đối tượng, mọi thứ được truyền như các tham số (parameter).
* Các phương thức tiện ích là một trường hợp thường được sử dụng nhất trong Java vì chúng có thể được truy cập trực tiếp bằng tên lớp mà không cần tạo bất thể hiện nào. Lớp java.lang.Math là một ví dụ trường hợp sử dụng static method.

Ví dụ:

|  |
| --- |
| **public** **class** UsingStaticExample {  **private** String subject;    UsingStaticExample (String subject) {  **this**.subject = subject;  }    **public** **void** print() {  System.***out***.println("Subject = " + subject);  System.***out***.println("Website = " + MyWebsite.WEBSITE);  }    **public** **static** **void** changeWebsite(String website) {  MyWebsite.WEBSITE = website;  }    //………………………………………………………  **public** **static** **void** main(String[] args) {  UsingStaticExample ex1 = **new** UsingStaticExample("Static in Java");  ex1.print();  System.***out***.println("----");  UsingStaticExample.*changeWebsite*("http://tdc.edu.vn");  ex1.print();  }  } |

Kết quả:

|  |
| --- |
| Subject = Static in Java  Website = upcoder.com.vn  ----  Subject = Static in Java  Website = http://tdc.edu.vn |

### Khối static (stactic blocks)

* Khối static được dùng để khởi tạo hoặc thay đổi giá trị của các biến static.
* Nó được thực thi trước phương thức main tại thời gian tải lớp.
* Một class có thể có nhiều static blocks.

Ví dụ :

|  |
| --- |
| **public** **class** UsingStaticExample {    **private** **static** String *subject*;    **static** {  System.***out***.println("Khối static được gọi");  }    **static** {  *subject* = "Khối static (static blocks)";  }    UsingStaticExample () {  System.***out***.println("hàm main() được gọi");  System.***out***.println("Subject = " + *subject*);  }    **public** **static** **void** main(String[] args) {  UsingStaticExample ex1 = **new** UsingStaticExample();  }  } |

Kết quả :

|  |
| --- |
| Khối static được gọi  hàm main() được gọi  Subject = Khối static (static blocks) |

### Import static trong Java

Java cho phép **import** các thành viên tĩnh (static member) của một class hoặc package vào một class khác bằng cách sử dụng từ khóa **import static** và sau đó sử dụng chúng như là thành viên của lớp đó.

Ví dụ:

File SystemConfig.java

|  |
| --- |
| **package** fit.coder;  **public** **final** **class** SystemConfig {    **public** **static** **final** String ***USER\_NAME*** = "fit-tdc";  **public** **static** **final** String ***PASSWORD*** = "123";  **public** **static** **final** String ***EMAIL*** = "fit@tdc.edu.vn";    **private** SystemConfig() {  }  } |

File StaticImportDemo.java

|  |
| --- |
| **package** fit.coder;  **import** **static** fit.coder.SystemConfig.\*;    **public** **class** StaticImportDemo {    **public** **static** **void** main(String[] args) {  System.***out***.println("Username: " + USER\_NAME);  System.***out***.println("Password: " + PASSWORD);  System.***out***.println("Email: " + EMAIL);  }  } |

Trong ví dụ trên, chúng ta có thể sử dụng trực tiếp thành phần static mà không cần thông qua tên lớp như SystemConfig.USE\_NAME.

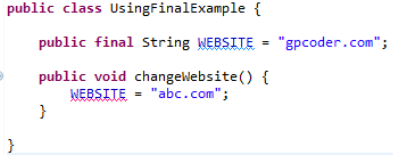
### Từ khóa final

Từ khóa final trong Java được sử dụng để hạn chế thao tác của người dùng.

Các trường hợp sử dụng:

**Biến final**: khi một biến được khai báo với từ khoá final, nó chỉ chứa một giá trị duy nhất trong toàn bộ chương trình (hay dễ hiểu hơn gọi là **biến hằng**).

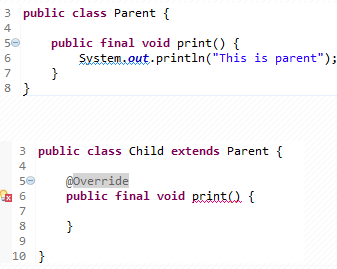
Ví dụ:



Ví dụ trên sẽ ra thông báo lỗi khi cố tình thay đổi giá trị của biến final: *The final field UsingFinalExample.WEBSITE cannot be assigned.*

**Phương thức final**: khi một phương thức được khai báo với từ khoá final, các class con kế thừa sẽ không thể ghi đè (override) phương thức này.

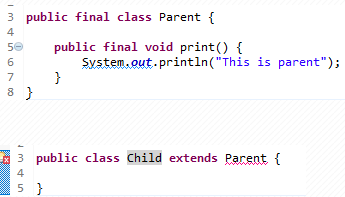
Ví dụ:



Ví dụ trên sẽ được thông báo lỗi khi cố tình ghi đè phương thức final: *Cannot override the final method from Parent.*

**Lớp final**: khi từ khoá final sử dụng cho một lớp, lớp này sẽ không thể được kế thừa.

Ví dụ:



Ví dụ trên sẽ thông báo lỗi khi cố tình kế thừa từ lớp final: *The type Child cannot subclass the final class Parent.*

**Biến static final trống**: Một biến final mà không được khởi tạo tại thời điểm khai báo được gọi là biến final trống. Biến này sẽ được khởi tạo tại thời điểm tạo đối tượng (trong constructor), và một khi đã khởi tạo thì sẽ không thể thay đổi giá trị. Biến final trống cũng có thể là static và sẽ chỉ được khởi tạo trong khối static.

### Sử dụng đối tượng làm tham số cho phương thức

**Tham số** của phương thức là những biến được khai báo trong khai báo phương thức. Tham số đóng vai trò là giá trị đầu vào cho phương thức, hay nói cách khác tham số sẽ là biến mà phương thức dùng để nhận giá trị truyền vào từ người dùng.

Trong Java, có 2 cách để truyền tham số cho phương thức:

* Tham trị (pass by value)
* Tham chiếu (pass by reference)

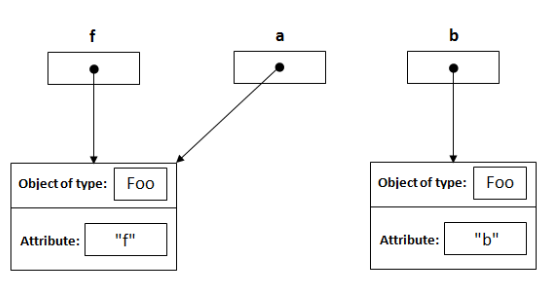
Trong Java, truyền tham trị dành cho các tham số có kiểu dữ liệu nguyên thủy là **byte, short, int, long, float, double, boolean, char**.

Và tất các phương thức có tham số là biến có **kiểu là các lớp (class)** đều là kiểu tham chiếu. Việc thay đổi giá trị của biến tham chiếu bên trong phương thức làm thay đổi giá trị của nó.

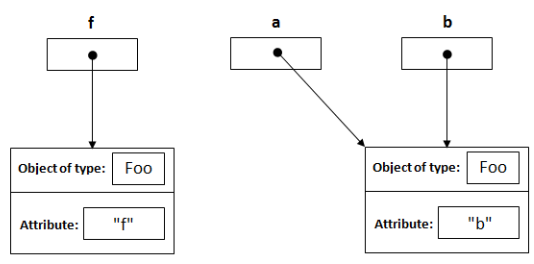
Ví dụ:

|  |
| --- |
| **public** **class** Main {    **public** **static** **void** main(String[] args) {  Foo f = **new** Foo("f");  *changeReference*(f); // It won't change the reference!  *modifyReference*(f); // It will modify the object that the reference variable "f" refers to!  }  **public** **static** **void** changeReference(Foo a) {  Foo b = **new** Foo("b");  a = b;  }  **public** **static** **void** modifyReference(Foo c) {  c.setAttribute("c");  }  } |

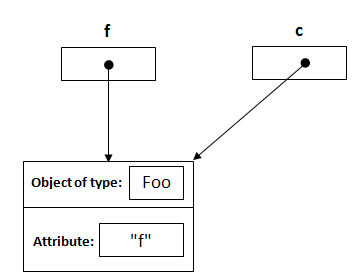
Trong ví dụ trên hàm changeReference(f) sẽ không làm thay đổi f. Giải thích được mình họa như hình dưới:



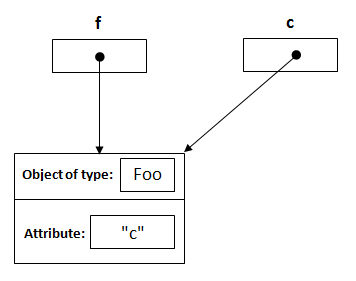
Khi thực hiện a = b:



Còn hàm modifyReference(f) sẽ thay đổi được f. Giải thích được mình họa như hình dưới:



Khi thực hiện c.setAttribute("c"), thành quả như sau:



Đây là lý do tại sau f được thanh đổi.

### Mảng đối tượng

* **Sử dụng mảng đối tượng:**

Ví dụ: Đoạn chương trình sau đây minh họa tạo danh sách (mảng) gồm ba đối tượng lớp Student:

|  |
| --- |
| **File: Program.cs** |
| **//Ví dụ sử dụng static fields**  **using System;**  **namespace VD\_Chuong2**  **{**  **class Program**  **{**  **static void Main(string[] args)**  **{**    **// Tạo danh sách đối tượng Student**  **Student[] arrSV = new Student[3];**  **arrSV[0] = new Student("An",   new DateTime(2020, 1, 1));**  **arrSV[1] = new Student("Binh",   new DateTime(2020, 1, 2));**  **arrSV[2] = new Student("Chau",   new DateTime(2020, 1, 3));**  **//Duyệt từng đối tượng**  **foreach (Student e in arrSV)**  **{**  **Console.WriteLine(e.StudentName);**  **}**  **}  } }** |

Kết quả:

|  |
| --- |
| **An Binh Chau** |

* **Truyền mảng đối tượng cho phương thức:**

Mảng đối tượng này có thể được truyền làm đối số cho phương thức, trong thân phương thức chỉ có thể truy cập tới thành phần public của đối tượng.

**Ví dụ 1**: Quan sát đoạn code sau

|  |
| --- |
| File: Student.cs |
| **// . . .**  **//Phương thức trả về chuỗi thông tin của lớp Student**  **public string ToString()**  **{**  **return($"{studentName} - Birthday: {birthDate :  dd/MM/yyyy} - Sex : {sex}");**  **}** |
| File: Program.cs |
| **//Ví dụ sử dụng static fields**  **using System;**  **namespace VD\_Chuong2**  **{**  **class Program**  **{**  **static void Main(string[] args)**  **{**    **// Tạo danh sách đối tượng Student**  **Student[] arrSV = new Student[3];**  **arrSV[0] = new Student("An",  new DateTime(2020, 1, 1), 1);**  **arrSV[1] = new Student("Binh",**  **new DateTime(2020, 1, 2),0);**  **arrSV[2] = new Student("Chau",**  **new DateTime(2020, 1, 3));**  **//Gọi ham in**  **PrintArrayOfStudent(arrSV);**  **}**  **//Phương thức in mảng đối tượng**  **public static void PrintArrayOfStudent(Student[] arrSV)**  **{**  **//Duyệt từng đối tượng trong mảg**  **foreach (Student e in arrSV)**  **{**  **Console.WriteLine(e.ToString());**  **}**  **}**  **} }** |

**Ví dụ 2**: Phương thức AddNewStudent() được sử dụng để thêm một đối tượng vào cuối mảng bằng cách gọi phương thức Resize của lớp Array để tăng kích thước mảng lên một phần tử, và giá trị của phần tử mới được gán cho phân tử cuối của mảng

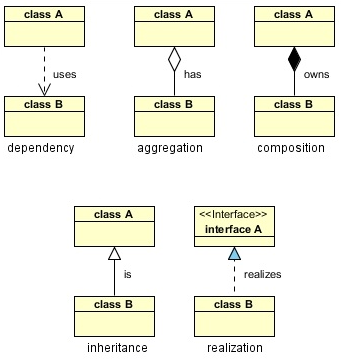
|  |
| --- |
| **File: Program.cs** |
| **//Ví dụ sử dụng mảng đối tượng**  **using System;**  **namespace VD\_Chuong2**  **{**  **class Program**  **{**  **static void Main(string[] args)**  **{**    **// Tạo danh sách đối tượng Student**  **Student[] arrSV = new Student[3];**  **arrSV[0] = new Student("An",  new DateTime(2020, 1, 1), 1);**  **arrSV[1] = new Student("Binh",  new DateTime(2020, 1, 2), 0);**  **arrSV[2] = new Student("Chau",  new DateTime(2020, 1, 3));**  **// Gọi phương thức thêm đối tượng mới vào mảng có sẵn**  **AddNewStudent (ref arrSV, new Student("Duc",   new DateTime(2020, 1, 4), 0, 3););**  **//Gọi phương thức in**  **PrintArray(arrSV);**  **}**  **//Ví dụ Truyền mảng đối tượng dạng tham chiếu**  **//Phương thức thêm đối tượng Student mới vào mảng có sẵn**  **public static void AddNewStudent(  ref Student[] arrSV, Student newStudent)**  **{**  **//Tăng kích thước mảng lên 1 phần tử**  **Array.Resize(ref arrSV, arrSV.Length + 1);**  **arrSV[arrSV.Length - 1] = newStudent;**  **}**  **//Phương thức in mảng đối tượng**  **public static void PrinArray(Student[] arrSV)**  **{**  **//Duyệt từng đối tượng trong mảg**  **foreach (Student e in arrSV)**  **{**  **Console.WriteLine(e.ToString());**  **}**  **}**  **}**  **}** |

Kết quả:

|  |
| --- |
| **An - Birthday: 01/01/2020 - Sex : 1**  **Binh - Birthday: 02/01/2020 - Sex : 1**  **Chau - Birthday: 03/01/2020 - Sex : 1**  **Duc - Birthday: 04/01/2020 - Sex : 1** |

### Biểu diễn mối quan hệ giữa các lớp

Trong lập trình hướng đối tượng, mối quan hệ giữa lớp với các lớp khác có thể biểu diến qua các sơ đồ UML sau:

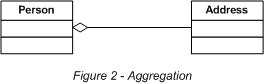


* Quan hệ Dependency (quan hệ - uses): Khi classA *có sử dụng một* thể hiện của classB *như một tham số đầu vào* của method trong class A.

|  |
| --- |
| **public class Address**  **{**  **//members**  **}**  **public class Location**  **{**  **//members**  **public void DoSomething(Address add) { }**  } |

Lớp **Location** sử dụng **object** kiểu **Address** làm tham số cho phương thức thành viên

* **Quan hệ Aggregation (quan hệ - has):** Khi classA *có một thành viên kiểu classB*. Ví dụ:



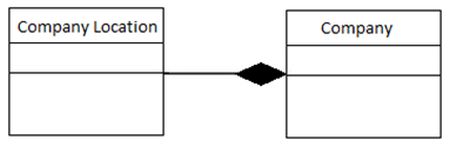
|  |
| --- |
| **public class ClassA**  **{**  **//members**  **}**  **public class ClassB**  **{**  **//members**  **//field1 kiểu ClassA**  **private ClassA field1;**  **//properties**  **public ClassB Field1{get; set;}**  } |

Ví dụ:

|  |
| --- |
| **//field1 kiểu ClassA**  **public class Address**  **{**  **private string numbering;**  **private string streetName;**  **private string city;**  **}**  **public class Location**  **{**  **//members**  **private string code;**  **private string blockName;**  **//field kiểu Address**  **private Address addressOfLocation;**  **//properties**  **public Address AddressOfLocation { get; set; }**  **//Phương thức tạo**  **public Location(string code, string blockName,   Address addressOfLocation = null)**  **{**  **this.code = code;**  **this.blockName = blockName;**  **this.addressOfLocation = addressOfLocation;**  **}**  } |

Ta thấy việc huỷ một đối tượng CompanyLocation không gọi thể tự động gọi phương thức huỷ đối tượng Address. Ta gọi mối quan hệ giữa hai lớp là “CompanyLocation has Address”.

* **Quan hệ Composition (quan hệ - owns):** Khi classA có chứa một thể hiện của classB (phương thức tạo của classB được gọi trong phương thức tạo của classA).



Ví dụ:

|  |
| --- |
| **public class Address**  **{**  **private string numbering;**  **private string streetName;**  **private string city;**  **}**  **public class Location**  **{**  **//members**  **private string code;**  **private string blockName;**  **//field kiểu Address**  **private Address addressOfLocation;**  **//properties**  **public Address AddressOfLocation { get; set; }**  **//Phương thức tạo**  **public Location(string code, string blockName,   Address addressOfLocation = null)**  **{**  **this.code = code;**  **this.blockName = blockName;**  **this.addressOfLocation = addressOfLocation;**  **}**  **}**  **public class Company**  **{**  **//fields**  **private string companyName;**  **private Location companyLocation;**  **//properties**  **public string CompanyName { get; set; }**  **public Location CompanyLocation { get; set; }**  **//constructors**  **public Company (string name, string code,   string blockName )**  **{**  **companyName = name;**  **companyLocation =   new Location(code, blockName);**  **}**  } |

Trong ví dụ trên ta thấy phương thức tạo đối tượng Company gọi phương thức tạo đối tượng Location. Như vậy khi đối tượng kiểu Company bị hủy thì phương thức hủy đối tượng Location cũng tự động được gọi. Quan hệ “owns” thể hiện ở sự tồn tại của đối tượng Location phụ thuộc vào sự tồng tại của đối tượng Company.

Các quan hệ Inheritance và Realization liên quan đến các lớp kế thừa chứng ta sẽ tìm hiểu trong chương 4.

### Bài tập áp dụng

**BÀI TẬP THỰC HÀNH SỐ 2**

1. **Thông tin chung:**

* Mã số bài tập : HW1-LTHĐT
* Hình thức nộp bài : Nộp qua hệ thống EL môn học
* Thời hạn nộp bài : … / … / ……
* Nội dung : Chương 2: Xây dựng lớp

**Chuẩn đầu ra cần đạt:**

|  |  |
| --- | --- |
| L.O.1 | Phân tích, thiết kế chương trình theo hướng đối tượng. Cài đặt được sơ đồ lớp cho các chương trình vừa và nhỏ; |
| L.O.8 | Làm bài tập và nộp bài đúng quy định. |

**BÀI TẬP CƠ BẢN**

1. Sinh viên trả lời các câu hỏi sau
2. Giải thích sự khác nhau giữa bổ từ truy xuất (Access modifier)
3. Có thể có bao nhiêu constructor và destructor trong một lớp? Giải thích sự khác nhau giữa chúng?
4. Nêu mục đích của các phương thức set, get.
5. Nêu tác dụng của từ khoá static trên các thành viên của lớp
6. Nêu các mối quan hệ không có tính kế thừa giữa hai class. Cho ví dụ minh hoạ.

Với mỗi bài toán sau hãy vẽ sơ đồ lớp sau đó cài đặt sơ đồ lớp đó.

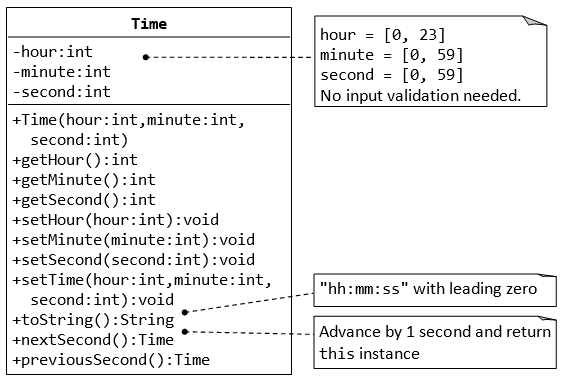
Viết chương trình tạo lớp “Rectangle” với thuộc tính chiều rộng và chiều cao. Tính diện tích và chu vi của hình chữ nhật. Tại chương trình chính hãy tạo ra 2 đối tượng và khởi tạo chúng, sau đó in ra chu vi, diện tích của chúng.

**Bài 3.**

Viết chương trình tạo lớp “Circle” với thuộc tính bán kính sao cho có thể truy cập và chỉnh sửa thuộc tính này. Tính diện tích và chu vi của hình tròn.

**Bài 4.**

Viết chương trình cho phép tạo ra các đối tượng Time (thời gian) được mô tả theo sơ đồ lớp sau:



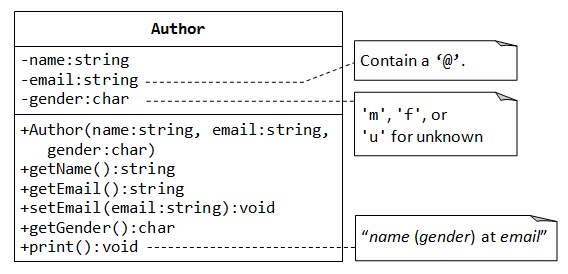
**Mô tả lớp Time:**

* Lớp Time bao gồm dữ liệu hour, minute, second với các giá trị hour hợp lệ trong khoảng từ 0 đến 23, minute và second trong khoảng từ 0 đến 59 và quyền truy cập và private.
* Lớp Time chỉ hỗ trợ một phương thức tạo có đầy đủ tham số và các hỗ trợ đầy đủ các thuộc tính get, set cho các trường.
* Phương thức toString() trả về chuỗi theo định dạng hh:mm:ss.
* Phương thức NextSecond(): trả về đối tượng Time là một giây sau.
* Phương thức PreviousSecond(): trả về đối tượng Time là một giây trước.

Viết đoạn chương trình theo yêu cầu sau:

* cho phép người dùng tạo đối tượng Time với các dữ liệu do người dùng nhập vào từ bàn phím. Nếu dữ liệu sai báo lỗi và không cho phép tạo đối tượng. Gọi phương thức toString() để in chuỗi theo định dạng ra màn hình.
* Thực hiện gán lại giá trị cho các trị cho tất cả các trường của đối tượng vừa tạo, trường hợp giá trị mới không hợp lệ thì thông báo lỗi. In giá trị mới của đối tượng ra màn hình.
* Gọi phương thức PreviousSecond() và in kết quả ra màn hình.
* Gọi phương thức NextSecond() và in kết quả ra màn hình.

Thực hiện khai báo class Author (tác giả) được mô tả theo sơ đồ lớp sau:



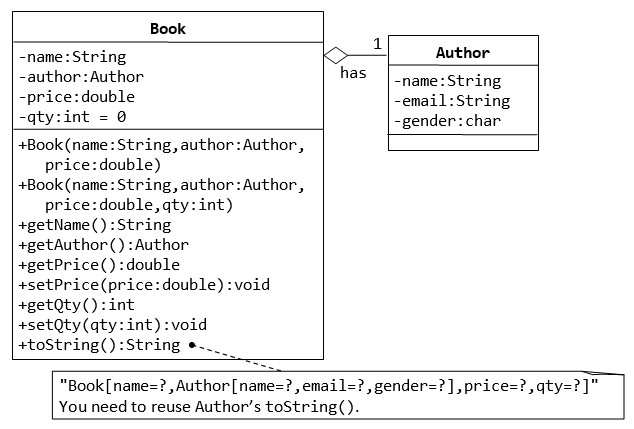
Mô tả lớp Author:

* Lớp Author gồm các trường name, email, gender. Trường email bắt buộc phải có ký tự '‘@'’ trường gender (phái) chỉ được phép chứa giá trị là một trong các ký tự m (male - nam), f (femail - nữ) hoặc u (unknown ).
* Phương thức tạo cho phép truyền đầy đủ tham số hoặc chỉ có thông tin về name, email.
* Các trường name, gender không hỗ trợ các hành động gán dữ liệu từ bên ngoài lớp. Người dùng có quyền xem và gán lại giá trị cho gender.
* Phương thức Print() in chuỗi thông tin theo định dạng ra màn hình.

Viết đoạn chương trình cho phép tạo ra các đối tượng Author với các đầy đủ tham số và đối tượng chỉ có hai tham số là name và gender. Thử nghiệm tất cả thuộc tính get, set các trường trên các đối tượng vừa tạo. Gọi phương thức Print() để in thông tin của các đối tượng ra màn hình.

Viết đoạn chương trình cho phép nhập một mảng các đối tượng kiểu Author. Thực hiện tìm kiếm một đối tượng bằng cách nhập vào name từ bàn phím, in ra thông tin của đối tượng tìm thấy.

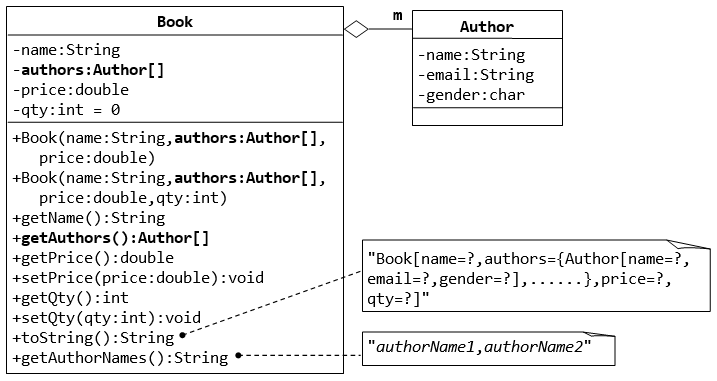
1. Hãy cài đặt viết chương trình sử dụng thử nghiệm lớp sau:



Viết đoạn chương trình cho phép tạo ra các đối tượng Book với các đầy đủ tham số. Thử nghiệm tất cả thuộc tính get, set các trường trên các đối tượng vừa tạo. Gọi phương thức toString() để in thông tin của các đối tượng ra màn hình.

Viết đoạn chương trình cho phép nhập một mảng các đối tượng kiểu Book. Thực hiện tìm kiếm một đối tượng bằng cách nhập vào name từ bàn phím, in ra thông tin của đối tượng tìm thấy.

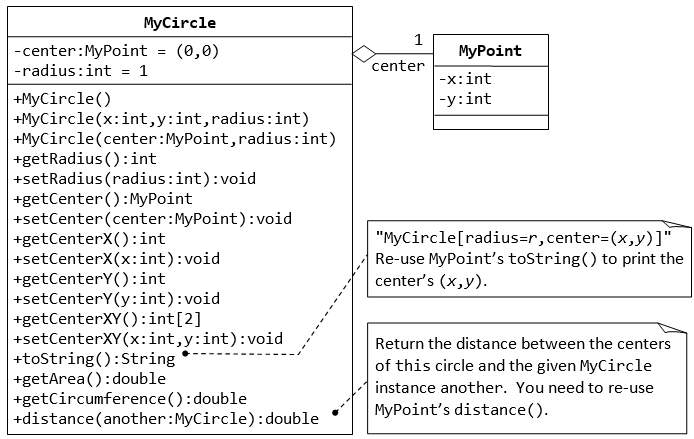
1. Hãy cài đặt viết chương trình sử dụng thử nghiệm lớp sau



Viết đoạn chương trình cho phép tạo ra các đối tượng Book với các đầy đủ tham số. Thử nghiệm tất cả thuộc tính get, set các trường trên các đối tượng vừa tạo. Gọi phương thức toString() để in thông tin của các đối tượng ra màn hình.

Viết đoạn chương trình cho phép nhập một mảng các đối tượng kiểu Book. Thực hiện tìm kiếm một đối tượng bằng cách nhập vào name từ bàn phím, in ra thông tin của đối tượng tìm thấy.

1. Hãy cài đặt viết chương trình sử dụng thử nghiệm lớp sau



Viết đoạn chương trình cho phép nhập một mảng các đối tượng kiểu MyCircle Thực hiện in ra thông tin của đối tượng thỏa điều kiện diện tích >=100.

**BÀI TẬP NÂNG CAO**

1. Viết chương trình quản lý sinh viên. Biết rằng mỗi sinh viên cần lưu trữ các thuộc tính sau : họ tên , lớp, điểm trung bình tích lũy.

Hãy thiết kế lớp SinhVien với những thuộc tính đã mô tả và các phương thức sau:

1. Khởi tạo và huỷ đối tượng
2. Xếp loại cho sinh viên biết :
   * Nếu đtb>8.5🡪Giỏi
   * Nếu đtb>=7🡪 Khá
   * Nếu đtb>=5🡪 Trung Bình
   * Còn lại Yếu
3. Viết chương trình chính cho phép tạo ra các đối tượng của lớp SinhVien và gọi các chức năng của đối tượng.
4. Viết chương trình để quản lý danh sách các máy tính của một cửa hàng. Biết rằng thông tin của một máy tính bao gồm: Loại máy**,** Nơi sản xuất **,** Thời gian bảo hành
5. Xây dựng lớp MayTinh với các thuộc tính đã cho và một số phương thức sau: Khởi tạo, hủy, xuất thông tin, phương thức đếm số lượng đối tượng máy tính có trong hệ thống
6. Viết chương trình chính cho phép tạo ra danh sách gồm n máy tính của cửa hàng. Sau đó thực hiện các chức năng sau:
7. Viết phương thức thống kê xem có bao nhiêu máy có thời gian bảo hành là 1 năm.
8. In ra danh sách các máy tính có xuất xứ từ Mỹ
9. In ra số lượng đố tượng máy tính có trong hệ thống

|  |  |
| --- | --- |
|  | TÍNH KẾ THỪA |

Chương này nhằm trang bị cho sinh viên kiến thức về kế thừa, các cơ chế và cách sử dụng kế thừa trong lập trình. Qua đó sinh viên có thể lập trình chuyên sâu hơn theo hướng đối tượng.

### Quan hệ giữa các lớp đối tượng

Giữa các lớp đối tượng có những loại quan hệ sau:

* Quan hệ một một (1-1): hai lớp đối tượng được gọi là có quan hệ 1-1 với nhau khi một đối tượng của lớp này quan hệ với một đối tượng của lớp kia và một đối tượng lớp kia cũng quan hệ với đối tượng của lớp này. Ví dụ:
* Một lớp học có một giáo viên chủ nhiệm
* Một quốc gia có một thủ đô
* Quan hệ một nhiều (1-n): hai lớp được gọi là có quan hệ một – nhiều với nhau khi mà một đối tượng thuộc lớp này có quan hệ với nhiều đối tượng của lớp kia và một đối tượng của lớp kia có quan hệ duy nhất với một đối tượng thuộc lớp này. Ví dụ:
* Một lớp học có nhiều học sinh
* Một công ty có nhiều nhân viên
* Một nhạc sĩ có nhiều tác phẩm
* Quan hệ nhiều nhiều (n-n): hai lớp được gọi là có quan hệ nhiều nhiều với nhau khi một đối tượng thuộc lớp này có quan hiệu với nhiều đối tượng của lớp kia, và một đối tượng của lớp kia có quan hệ với nhiều đối tượng của lớp này. Ví dụ:
* Một bác sĩ khám nhiều bệnh nhân và một bênh nhân có nhiều bác sĩ
* Quan hệ đặc biệt hóa, tổng quát hóa: hai lớp đối tượng mà đối tượng này là trường hợp đặc biệt của đối tượng kia và đối tượng kia là trường hợp tổng quát của đối tượng này thì được gọi là quan hệ đặc biệt hóa, tổng quát hóa. Ví dụ:
* Tam giác cân là một trường hợp đặc biệt của tam giác
* Sinh viên là một con người

### Tổng quát hóa, đặc biệt hóa

Một chương trình lớn chứa rất nhiều chức năng và phải xử lý rất nhiều dữ liệu nên rất khó để quản lý chúng trong một class, do đó cần có một cấu trúc phù hợp để quản lý chương trình lớn.

Tổng quát hóa (Generalization): Là việc trích ra những đặt tính chung nhất từ hai hoặc nhiều lớp để tạo nên lớp cha (lớp tổng quát)

Đặc biệt hóa (Specialization): Là tạo một lớp con từ một lớp đã tồn tại (mối quan hệ is\_a)

Ví dụ:

|  |  |
| --- | --- |
| Generalization Specialization |  |
| Kết quả hình ảnh cho bank account inheritance | Một Bank Account có hai loại: Checking Account and Saving Account. Checking Account và Saving Account được kế thừa các đặc tính như Account Number, Total Balance, … từ Bank Account và các lớp con cũng có thêm đặc tính riêng của nó như fee, intertRate. |

### Khái niệm kế thừa

Kế thừa trong lập trình hướng đối tượng là một đặc điểm dùng để biểu diễn mối quan hệ đặc biệt hóa – tổng quát hóa giữa các class. Các lớp được trừu tượng hóa và tổ chức thành một sơ đồ phân cấp lớp.

Kế thừa là công cụ thiết kế cho phép lập trình viên sử dụng các lớp có sẵn và sản sinh ra các lớp mới với các hành vi cao hơn hoặc khác biệt hơn. Các hành vi này không từ code có sẵn. Nghĩa là khi sử dụng cơ chế kế thừa, lập trình viên không được tác động đến code trong base class. Ngoài ra, lập trình viên có thể tận dụng code có sẵn trong base class mà không nhân bản nó trong derived class.

Có hai mức độ kế thừa khác nhau là: đơn kế thừa và đa kế thừa.

* Đơn kế thừa khi chỉ có một lớp cha để kế thừa
* Đa kế thừa khi có đồng thời nhiều lớp cha

Đối tượng class con hoàn toàn có thể đóng vai trò cả đối tượng class cha hoặc có thể thay thế đối tượng class cha khi cần thiết, nhưng ngược lại thì không cho phép.

### Lợi ích của kế thừa

Kế thừa trong OOP cho phép chúng ta định nghĩa một class mới dựa trên thuộc tính và phương thức của một class đã có trước. Giúp chúng ta tối ưu được thời gian công sức phát triển ứng dụng, đồng thời cũng dễ dàng nâng cấp và chỉnh sửa mã nguồn. Điều đó có nghĩa là, khi chúng ta định nghĩa một class mới, không nhất thiết phải viết lại tất cả mã nguồn cho nó, mà có thể kế thừa dựa trên một class đã tồn tại.

Tính kế thừa là một đặc tính của Lập trình hướng đối tượng. Chương trình cho phép sử dụng lại (reuses/ inherit), mở rộng (extend), chỉnh sửa (modify). Kế thừa là một đặc tính quan trọng trong OOP cho phép tái sử dụng mã nguồn và tránh trùng lắp dữ liệu.

### Cú pháp khai báo kế thừa

Cú pháp khai báo kế thừa như sau:

|  |
| --- |
| **class BaseClass**  **{**  **//Thành phần của lớp cơ sở**  **};**    **class DerivedClass : BaseClass**  **{**  **//Thành phần bổ sung của lớp dẫn xuất**  **};** |

BaseClass: gọi là lớp cha hay lớp cơ sở (supper class)

DerivedClass: gọi là lớp con hay lớp kế thừa (sub class)

Gọi constructor của base class bằng từ khoá base ngay phía sau khai báo constructors của derived class.

|  |
| --- |
| **class BaseClass**  **{**  **Public BaseClass(parameters)**  **};**    **class DerivedClass : BaseClass**  **{**  **Public DerivedClass (parameters): base (parameters)**  **};** |

### Tầm vực trong kế thừa

Khả năng truy cập các thành phần trong lớp cha của lớp con như sau:

|  |  |  |
| --- | --- | --- |
| BaseClass | Khả năng truy cập trong Assembly | Khả năng truy cập ngoài Assembly |
| Private | No | No |
| Public | Yes | Yes |
| Protected | Yes\* | Yes\* |
| Internal | Yes | No |
| Protected internal | Yes\* | Yes\* |

Yes\*: chỉ có các lớp con mới có khả năng truy cập các thuộc tính Protected của lớp cha.

Các thành viên Private không được kế thừa. Các thành viên public, protected, internal và internal protected được kế thừa trong cùng môi trường assembly. Trừ thành viên internal không được kế thừa ngoài môi trường assembly, các thành viên kiểu public, protected, và internal protected đều được kế thừa.

Các thành viên sau không được kế thừa:

* Phương thức tạo tĩnh (Static constructors)
* Phương thức khởi tạo (Instance constructors)
* Phương thức huỷ (Finalizers)

Quan sát ví dụ sau để hiểu hơn về cơ chế kế thừa:

|  |
| --- |
| **class BaseClass**  **{**  **// Other details**  **public void Function\_B()**  **{**  **Console.WriteLine("In function 'B’");**  **}**  **};**  **class DerrivedClass : BaseClass**  **{**  **// Other details**  **public void Function\_D()**  **{**  **Console.WriteLine("In function 'D’");**  **}**  **};** |

Code phía client:

|  |
| --- |
| **class DerrivedClass : BaseClass**  **{**  **// Other details**  **public void Function\_D()**  **{**  **Console.WriteLine("In function 'D’");**  **}**  **};** |

Kết quả:

|  |
| --- |
| **In function 'B'**  **In function 'B'**  **In function 'D'** |

Mặc dù mã nguồn của lớp DerrivedClass không hiển thị định nghĩa của một phương thức có tên Function\_B nhưng bản thân nó có một phương thức mà nó kế thừa từ lớp BaseClass.

Lưu ý rằng kế thừa chỉ hoạt động theo một hướng. Đoạn mã sau đây là không đúng: myB. Function\_D(); vì một object của BaseClass không phải là một object của Derivedclass.

Không có giới hạn số lượng lớp kế thừa từ lớp cơ sở:

|  |
| --- |
| **class DerrivedClass1: BaseClass {/\*Details \*/};**  **class DerrivedClass2: BaseClass {/\* Details\*/};**  **class DerrivedClass3: BaseClass {/\* Details\*/};** |

Lớp DerrivedClass1, DerrivedClass2 và DerrivedClass3 được kế thừa từ lớp BaseClass.

C# Ver 6.0 không hỗ trợ một lớp đa kế thừa trực tiếp từ các lớp cha mà kế thừa thông qua định nghĩa interface. Vì chi phí thực hiện đa kế thừa trong định nghĩa class rất cao nhưng tần suất dùng đa kế thừa trong chương trình thực tế thì rất ít.

### Phân loại kế thừa

* Mỗi lớp trong C# chỉ có duy nhất một base class. Trong C# có các loại kế thừa sau:
* Đơn kế thừa (Single Inheritance): một lớp chỉ kế thừa từ một lớp khác
* Kế thừa nhiều cấp (Multilevel Inheritance): Một lớp vừa là lớp cơ sở của lớp này vừa là lớp dẫn xuất của lớp khác
* Kế thừa Hierarchical: Nhiều lớp kế thừa từ một lớp. Một lớp cha có nhiều lớp con.
* Multiple inheritance: Một lớp được kế thừa từ nhiều lớp (chỉ thực hiện được thông qua interface)
* Hybrid Inheritance: là sự kết hợp của Hierachical inheritance và Multiple inheritance. Trong C# kế thừa chỉ thực hiện được thông qua interface.

### Một số mô hình kế thừa

|  |  |
| --- | --- |
| Single inheritance (Kế thừa dạng đơn) | |
| 1. //Base Class 2. class A 3. { 4. }; 5. //Derived Class 6. class B : A 7. { 8. }; | Class A  Class B |

|  |  |
| --- | --- |
| Multi-level inheritance (Kế thừa dạng nhiều cấp) | |
| //Base Class  class A  {  };  //Derived Class  class B : A  {    };  //Derived Class  class C : B  {  } |  |

|  |  |
| --- | --- |
| Hierarchical inheritance (Kế thừa dạng cây phả hệ) | |
| class A  {  };  class B : A  {  } ;  class C: A  {  }; |  |

|  |  |
| --- | --- |
| Multiple inheritance (Kế thừa dạng đa) | |
| //Base Class  interface A  {    }  interface B  {     }  //Derived Class  class C : A, B  {  }; | Interface A  Interface B  Class C |

|  |  |
| --- | --- |
| Hybric inheritance(Kế thừa dạng phức hợp: nhiều dạng kết hợp) | |
| //Base Class  interface A{  };  //Derived Class  interface B : A{  } ;  // Derived Class  interface C: A{  };  //Derived Class  class D : B, C{  }; | **Class A**  **Class B**  **Class C**  **Class D** |

### Bài tập áp dụng

**BÀI TẬP THỰC HÀNH SỐ 3**

1. **Thông tin chung:**

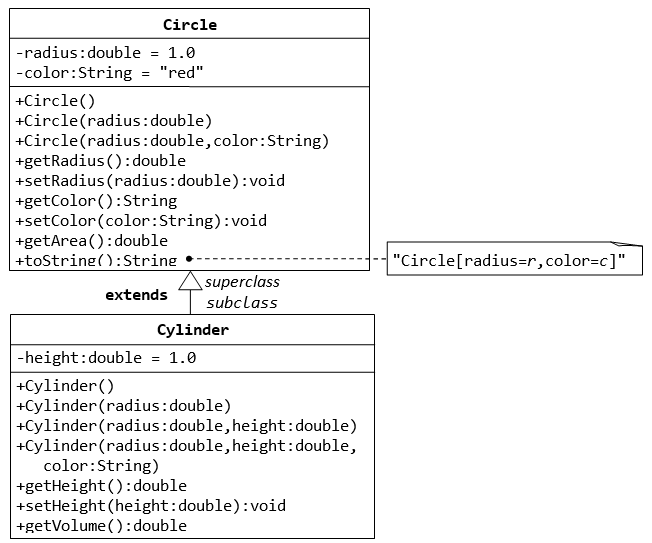
* Mã số bài tập : HW2-LTHĐT
* Hình thức nộp bài : Nộp qua hệ thống EL của môn học
* Thời hạn nộp bài : … / … / ……
* Nội dung : Chương 3. Tính Kế thừa

**Chuẩn đầu ra cần đạt:**

|  |  |
| --- | --- |
| L.O.2 | Sử dụng cơ chế kế thừa và đa hình trong thiết kế hướng đối tượng |
| L.O.8 | Làm bài tập và nộp bài đúng quy định; |

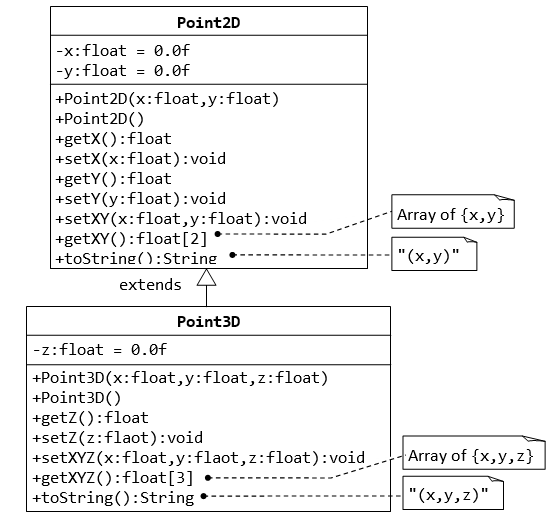
**BÀI TẬP CƠ BẢN**

1. **Hãy cài đặt các lớp sau**



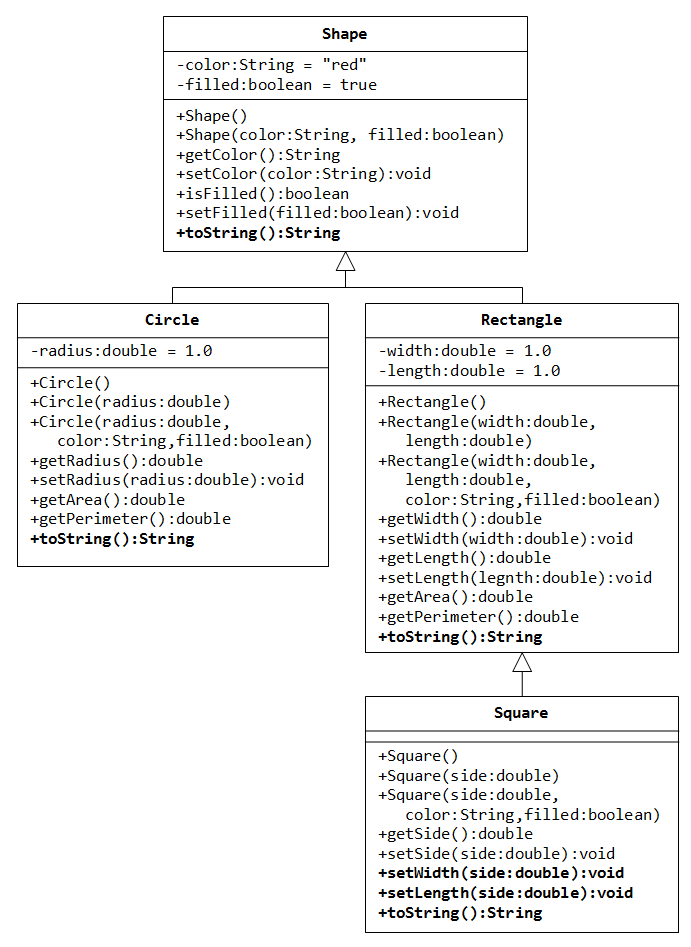
Thực hiện viết chương trình chạy thử nghiệm khởi tạo các đối tượng/ danh sách đối tượng và gọi sử dụng tất cả các phương thức của các lớp trên

1. **Hãy cài đặt các lớp sau :**



Thực hiện viết chương trình chạy thử nghiệm khởi tạo các đối tượng/ danh sách đối tượng và gọi sử dụng tất cả các phương thức của các lớp trên

1. **Hãy cài đặt các lớp sau:**



Thực hiện viết chương trình chạy thử nghiệm khởi tạo các đối tượng/ danh sách đối tượng và gọi sử dụng tất cả các phương thức của các lớp trên

1. Để quản lý các biên lai thu tiền điện, người ta cần các thông tin như sau:

* Thông tin về hộ sử dụng điện, chỉ số cũ, chỉ số mới
* Số tiền phải trả của hộ = (Số mới - số cũ) \* 750).

Xây dựng ứng dụng quản lý biên lai tiền điện theo mô tả sau:

1. Hãy xây dựng lớp KhachHang để lưu trữ các thông tin riêng của mỗi hộ sử dụng điện và các phương thức hỗ trợ cần thiết. Biết các thông tin riêng của mỗi hộ sử dụng điện gồm: Họ tên chủ hộ, số nhà, mã số công tơ của mỗi hộ sử dụng điện.
2. Xây dựng lớp BienLai để quản lý việc sử dụng và thanh toán tiền điện.
3. Xây dựng chương trình chính có các chức năng sau:

* Nhập vào các thông tin cho n biên lai và hiển thị các thông tin vừa nhập
* Tính tổng tiền mà người quản lý thu được từ các biên lai.

1. Viết chương trình quản lý điểm của sinh viên với mỗi sinh viên có các thông tin: Mã sinh viên, tên sinh viên, lớp học và môn học, biết rằng một sinh viên chỉ thuộc một lớp học và có nhiều môn học.

Thông tin lớp học bao gồm: Tên lớp, khóa. Thông tin về môn học bao gồm: Tên môn, số tín chỉ, điểm. Yêu cầu chương trình có các chức năng sau:

* Nhập thông tin cho n sinh viên sao cho mỗi sinh viên có đủ thông tin
* In ra danh sách các sinh viên vừa nhập gồm các thông tin: mã sinh viên, tên sinh viên, tên lớp và khóa.
* In ra phiếu báo điểm cho từng sinh viên theo mẫu:

|  |  |  |
| --- | --- | --- |
| * **Phiếu Báo Điểm** * Mã sinh viên: *SV001* Tên sinh viên: *Nguyễn Hải Hà* * Lớp: *Tin2* Khóa: *52* | | |
| BẢNG ĐIỂM | | |
| * **Tên môn** | * **Số tín chỉ** | * **Điểm** |
| * Cơ sở dữ liệu | * 4 | * 8 |
| * Lập trình HĐT | * 3 | * 7 |
| * Hệ điều hành | * 5 | * 9 |

**BÀI TẬP NÂNG CAO**

**Bài 6**. Phòng đào tạo của Trường đại học ABC cần xây dựng ứng dụng quản lý sinh viên để dễ dàng theo dõi tình hình học tập và rèn luyện của sinh viên trong trường. Biết rằng mỗi sinh viên được mô tả bởi m*ã sinh viên, họ tên, năm sinh và danh sách các môn học mà sinh viên đã tích lũy đến thời điểm hiện tại*. Biết rằng với mỗi môn học cần được lưu trữ thông tin: *mã môn học, tên môn học, số tín chỉ và điểm môn học.*

Hãy xây dựng các lớp cần thiết cho ứng dụng. Với lớp sinh viên cần xây dựng các chức năng sau:

1. Nhập dữ liệu cho sinh viên
2. Xem thông tin sinh viên (ngoài những thông tin được mô tả, cần xuất thêm thông tin về điểm trung bình tích lũy, xếp loại học tập)

Mô phỏng mô hình khi xuất dữ liệu:

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**MaSV: sv01**

**Ho ten: Nguyen Van An**

**Nam sinh: 1996**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Danh sach gom 2 mon hoc:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * **STT** | * **Ma Mon** | * **Ten Mon** | * **So tin chi** | * **Diem** |
| * **1** | * **Csc1060001** | * **Nhap mon** | * **3** | * **5** |
| * **2** | * **Csc1070001** | * **KTLT** | * **3** | * **5** |

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Diem trung binh tich luy: 5.0**

**Xep loai: Trung binh**

1. Tính điểm trung bình tích lũy (TBTL) theo công thức:

TBTL = (điểm môn 1. \* tín chỉ môn 1 + điểm môn 2 \* tín chỉ môn 2 + …)/(tín chỉ 1 + tín chỉ 2 +…)

1. Xếp loại học tập sinh viên. Biết rằng
   * + - 10 > TBTL >= 9 : Xuất sắc
       - 9> TBTL >= 8: Giỏi
       - 8 > TBTL >= 7: Khá
       - 7 > TBTL >=5: Trung bình
       - TBTL < 5: Yếu
2. Xây dựng thêm lớp DanhSachSinhVien. Biết rằng mỗi danh sách cần lưu trữ mảng sinh viên và số lượng sinh viên thực tế. Các phương thức cần xây dựng cho lớp này bao gồm: Nhập danh sách, xuất danh sách, tìm sinh viên có điểm trung bình cao nhất. Sau đó tại chương trình chính cho phép tạo ra 1 danh sách sinh viên (L) của trường (đối tượng của lớp DanhSachSinhVien) và gọi các chức năng như sau:

L.NhapDanhSach();

L.XuatDanhSach();

L.TinhDiemTBCaoNhat();Pub

**Bài 7**. Xây dựng class TAILIEU gồm các thuộc tính sau:

* Mã tài liệu
* Loại tài liệu
* Tên tài liệu
* Tác giả
* Nhà xuất bản
* Ghi chú

Dựa trên class TAILIEU, xây dựng chương trình quản lý thự viên trong môi trường console. Yêu cầu chương trình:

* Chương trình cho phép nhập mới tài liệu, thông tin tài liệu được lưu giữ trong file TAILIEU.DAT.
* Chương trình cho phép duyệt và xem thông tin tài liệu lần lượt từ tài liệu đầu tiên cho đến tài liệu cuối cùng và ngược lại
* Chương trình cho phép xóa tài liệu đang được hiển thị trên màn hình.

|  |  |
| --- | --- |
|  | TÍNH ĐA HÌNH |

Chương này nhằm trang bị cho sinh viên kiến thức về sử dụng tính đa hình trong lập trình. Qua đó sinh viên có thể lập trình chuyên sâu hơn theo hướng đối tượng.

### Khái niệm Đa hình

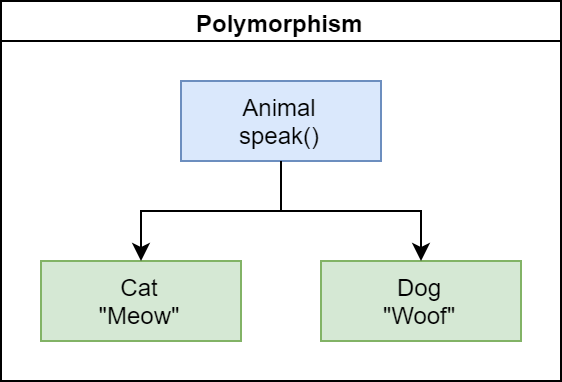
Lập trình hướng đối tượng (OOP) cung cấp bốn khả năng làm tăng khả năng thiết kế, cấu trúc và tái sử dụng code là: Abstraction (Tính trừu tượng), Encapsulation (Tính đóng gói), Inheritance (Tính kế thừa), Polymorphism (Tính đa hình).

**Tính đa hình (polymorphism)** trong lập trình có nghĩa là các thực thể (biến, phương thức, đối tượng) có thể có nhiều hình thái khác nhau tùy theo ngữ cảnh của chương trình.

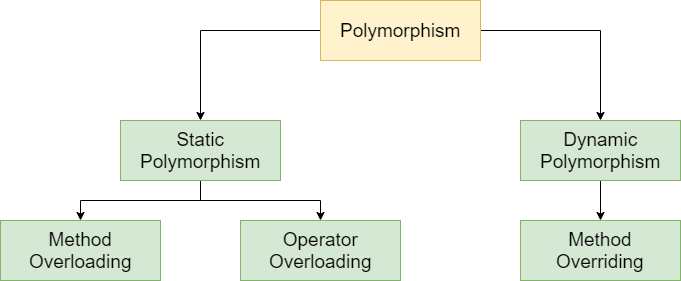
Ví dụ 1: Một người có những đặc tính chung như họ tên, ngày sinh, chiều cao, …. Nhưng có thể đóng nhiều vai trò khác nhau như cha mẹ, học sinh, khách hàng, ...



Ví dụ 2: Lớp Animal đại diện cho các con vật có tiếng kêu. Tuy nhiên mỗi đối tượng khác nhau lại có tiếng kêu khác nhau. Speak () là phương thức chung dùng để phát tiếng kêu của các con vật. Tuy nhiên, tuỳ theo từng loài vật phương thức này hoạt động khác nhau như lớp Mèo (Cat) thì có tiếng kêu là “Meow”, lớp Chó (Dog) thì phát ra tiếng kêu là Woof.



### Phân loại Đa hình



Có hai loại đa hình:

* Đa hình tĩnh (Compile Time, Static Polymorphism)
* Đa hình động (RunTime, Dynamic Polymorphism)

### Đa hình tĩnh - Static Polymorphism

Là dạng đa hình thể hiện qua phương pháp nạp chồng các phương thức (functions overloading), quá tải toán tử (operators overloading). Các khai báo về các tham số, các kiểu tham số khác nhau được xác định trước cho phương thức tại thời điểm biên dịch chương trình.

* **Functions** **overloading**: là các phương thức có cùng tên nhưng khác nhau về số lượng tham số hoặc kiểu dữ liệu của tham số.
* **Operator** **overloading**: thông thường các toán tử sẽ được áp dụng trên các biến, hằng số. Nhưng khi toán tử đã được overloaded thì có thể xem như tên một phương thức và áp dụng trên các đối tượng.

### Nạp chồng phương thức

Functions overloading là một cách thể hiện sự đa hình tĩnh. Nó là khả năng định nghĩa các phương thức có cùng tên nhưng có nhiều dạng đối số. Thông thường nạp chồng hay quá tải phương thức thể hiện qua sự khác nhau về số lượng tham số, kiểu tham số, thư tự các tham số truyền vào cho phương thức.

**Ví dụ 1**: Nạp chồng phương thức bằng cách thay đổi số lượng đối số truyền cho phương thức

Quan sát đoạn code sau:

|  |
| --- |
| **/\*Ví dụ sử dụng đa hình tĩnh – function overoading\*/**  **using System;**  namespace MethodOverloading  **{**  **class Program**  **{**  **// Method Overloading**  **public int Add(int x, int y)**  **{**  **return x + y;**  **}**  **public int Add(int x, int y, int z)**  **{**  **return x + y + z;**  **}**  **static void Main(string[] args)**  **{**  **Program prog = new Program();**  **Console.WriteLine("Addition of Number is   {0}", prog.Add(10, 20));**  **Console.WriteLine("Addition of Number is   {0}", prog.Add(10, 20, 30));**  **Console.ReadLine();**  **}**  **}**  } |

Kết quả:

|  |
| --- |
| Addition of Number is 30  Addition of Number is 60 |

Đoạn chương trình có hai phương thức cùng tên Add nhưng khác nhau về danh sách đối số. Phương thức Add phù hợp sẽ được lựa chọn ngay tại thời điểm biên dịch chương trình, tuỳ thuộc vào số lượng đối số truyền vào.

* **Ví dụ 2**: Quá tải phương thức bằng cách thay đổi kiểu tham số truyền cho phương thức

Quan sát ví dụ sau

|  |
| --- |
| File: program.cs |
| **//Ví dụ sử dụng static fields**  **using System;**  **namespace VD\_Chuong2**  **{**  **class Program**  **{**  **static void Main(string[] args)**  **{**    **//sV1**  **Student sV1 = new Student("An",   new DateTime(2000, 8, 20), 1, 2);   sV1.Result[0] = 8;**  **sV1.Result[1] = 9;**  **//sV2**  **Student sV2 = new Student("An",   new DateTime(2000, 1, 20), 1, 3);**  **sV2.Result[0] = 2;**  **sV2.Result[1] = 3;**  **sV2.Result[0] = 6;**    **//sV3**  **Student sV3 = new Student("Chau",   new DateTime(2000, 11, 1),0, 3);**  **sV3.Result[0] = 8;**  **sV3.Result[1] = 9;**  **sV3.Result[0] = 10;**    **// Tạo danh sách đối tượng Student**  **Student[] arrSV1 = new Student[0];**  **AddStudentToArray(ref arrSV1, sV1);**  **AddStudentToArray(ref arrSV1, sV2);**  **AddStudentToArray(ref arrSV1, sV3);**      **//Sử dụng overloading function**  **// Gọi kiểu GetMax(Student[] arrSV)**  **Console.WriteLine("Max diem trung binh:{0}", GetMax(arrSV1));**  **// Gọi kiểu GetMax(float[] arrSV)**  **float[] arrA = GetArrayOfAverage(arrSV1);**  **Console.WriteLine("Max diem trung binh:{0}", GetMax(arrA));**  **}**    **//Ví dụ Truyền mảng đối tượng dạng tham chiếu**  **//Phương thức thêm đối tượng Student mới vào mảng có sẵn**  **public static void AddStudentToArray(ref Student[] arrSV, Student newStudent)**  **{**  **//Tăng kích thước mảng lên 1 phần tử**  **Array.Resize(ref arrSV, arrSV.Length + 1);**  **arrSV[arrSV.Length - 1] = newStudent;**  **}**  **//Tính điểm trung bình của tất cả SV, trả về mảng số thực**  **public static float[]GetArrayOfAverage(  Student[] arrSV)**  **{**  **float[] arrA = new float[arrSV.Length];**  **for(int i = 0; i<arrSV.Length; i++)**  **{**  **arrA[i] = Student.GetAverage(arrSV[i]);**  **}**  **return arrA;**  **}**  **//Quá tải phương thức (Functions overloading)**  **//Tìm SV có điểm trung bình lớn nhất**  **public static float GetMax(params   float[] result )**  **{**  **float max = result[0];**  **for (int i = 1; i < result.Length; i++)**  **{**  **if (max < result[i])**  **{**  **max = result[i];**  **}**  **}**  **return max;**  **}**  **public static float GetMax(Student[] arrSV)**  **{**  **float max = Student.GetAverage(arrSV[0]);**  **int imax = 0;**  **for (int i = 1; i < arrSV.Length; i++)**  **{**  **float temp=Student.GetAverage(arrSV[i]);**  **if (max < temp)**  **{**  **max = temp;**  **imax = i;**  **}**  **}**  **return max;**  **}**  **}**  **}** |

* Kết quả:

|  |
| --- |
| Max diem trung binh: 8.5  Max diem trung binh: 8.5 |

* Đoạn chương trình trên thực hiện overloading phương thức GetMax() để tìm điểm trung bình lớn nhất của danh sách sinh viên. Có hai cách thực hiện để tìm được giá trị điểm trung bình lớn nhất:
  + - Cách 1: gọi phương thức GetMax() với tham số truyền vào là mảng các đối tượng kiểu Student. Khi đó phương thức GetAverage() của lớp Student được gọi để tính được điểm trung bình cho từng sinh viên.
    - Cách 2: Ta tính điểm trung bình của tất cả các sinh viên bằng cách gọi phương thức GetAverage () và lưu kết quả vào một mảng kiểu float. Sau đó ta truyền mảng kiểu float này cho phương thức GetMax () để lấy về giá trị lớn nhất.
* Như vậy các phương thức GetMax () có cùng tên nhưng khác nhau danh sách tham số đầu, kiểu tham số đầu vào và cách thức thực hiện. Tại thời điểm biên dịch, trình biên dịch sẽ quyết định phương thức phù hợp để thực hiện.

### Nạp chồng toán tử

Nạp chồng toán tử (Overloading Operator) là khả năng làm cho các toán tử có thể sử dụng trên nhiều kiểu dữ liệu khác nhau. Nó cung cấp thêm khả năng của các toán tử, trong C# có thể sử dụng trên kiểu dữ liệu do người dùng tự định nghĩa. Các toán tử được nạp chồng trong C# là các phương thức với các tên đặc biệt: từ khóa operator được theo sau bởi biểu tượng cho toán tử đang được định nghĩa. Tương tự như bất kỳ phương thức nào khác, một toán tử được nạp chồng có một kiểu trả về và một danh sách tham số.

Cú pháp phương thức nạp chồng toán tử:

|  |
| --- |
| public static return\_type operator operator\_symbol (parameters)  {   // Code  } |

Ví dụ: public static Fraction operator \* (Fraction a, Fraction b);

Tạo một lớp Phân số gồm có giá trị tử số và mẫu số. Thực hiện nạp chồng toán tử + để thực hiện cộng hai phân số. Kết quả trả về là một phân số.

Quan sát đoạn code sau:

|  |
| --- |
| **Fraction.cs** |
| **//Lớp Fractio sử dụng quá tải toán tử \***  **using System;**  **class Fraction**  **{**  **//fields**  **private int numerator; //tử số**  **private int deminorator; //mẫu số**  **//constructors**  **public Fraction()**  **{**  **numerator = 0;**  **deminorator = 1;**  **}**  **public Fraction(int numerator, int deminorator)**  **{**  **this.numerator = numerator;**  **this.deminorator = deminorator;**  **}**  **//Ham in**  **public void Print()**  **{**  **Console.Write(numerator + "/" + deminorator);**  **}**  **//Overloading operator\***  **public static Fraction operator \*  (Fraction a, Fraction b)**  **{**  **Fraction result = new Fraction();**  **result.numerator = a.numerator \* b.numerator;**  **result.deminorator=a.deminorator\*b.deminorator;**  **return result;**  **}**  **}** |

Chương trình thử nghiệm:

|  |
| --- |
| Program.cs |
| **using System;**  **namespace Test**  **{**  **class Program**  **{**  **static void Main(string[] args)**  **{**  **Fraction a = new Fraction(3, 4);**  **Fraction b = new Fraction(1, 2);**  **Fraction c = a \* b;**  **c.Print();**  **Console.ReadKey();**  **}**  **}**  **}**  **}** |

Kết quả:

|  |
| --- |
| 3/8 |

C# hỗ trợ các toán tử có thể thực hiện nạp chồng (overloadable operator) như sau:

* Unary operators: + - ! ~ ++ -- true false
* Binary operators: + - \* / % & | ^ << >> == != > < >= <=

Các toán tử không thể nạp chồng, hoặc không thể nạp chồng trực tiếp trong C#:

Toán tử logic: &&, ||

Toán tử gán: = += -= \*= /= %=

Toán tử khác : . ?: ?? . =>

### Đa hình động - Dynamic Polymorphism

Là dạng đa hình thể hiện qua phương pháp ghi đè phương thức (function overriding). Phương thức có nhiều dạng khác nhau và các cách gọi phương thức được quyết định ở thời điểm chương trình thực thi. Trong C#, đa hình động có thể thực hiện thông qua từ khoá ***override*** hoặc ***virtual.***

**Virtual function**: là phương thức được định nghĩa trong base class với từ khóa virtual. Các derived class định nghĩa lại phương thức này với cùng tên, cùng kiểu trả về, có thể cùng danh sách tham số nhưng lại khác nhau các thực hiện. Virtual function đảm bảo rằng phương thức được gọi sẽ thực thi theo cách tương thích với đối tượng.

Ví dụ 1: Quan sát đoạn chương trình sau

|  |
| --- |
| **/\*Ví dụ sử dụng đa hình động\*/**  **using System;**  **namespace MethodOverriding**  **{**  **public class Employee**  **{**  **public virtual void PrintName(string fname,   string lname)**  **{**  **Console.WriteLine("Employee Full Name {0}",  fname + " " + lname);**  **}**  **}**  **public class Child : Employee**  **{**  **public override void PrintName(string fname,**  **string lname)**  **{**  **Console.WriteLine("Child Full Name {0}",**  **fname + " " + lname);**  **}**  **}**    **}** |

Đoạn chương trình chạy thử nghiệm:

|  |
| --- |
| **/\*Chương trình thử nghiệm thể hiện tính đa hình động\*/**  **public class MethodOverridingDemo**  **{**  **static void Main(string[] args)**  **{**  **Employee emp;**  **emp = new Employee();**  **emp.PrintName("Shri", "VT");**    **emp = new Child();**  **emp.PrintName("Shrimant", "VT");**  **Console.ReadLine();          }** |

Kết quả:

|  |
| --- |
| **Employee Full Name Shri VT**  **Child Full Name Shrimant VT** |

Class Child được kế thừa từ Employee. Ta thấy đối tượng emp được khai báo kiểu Employee nhưng có lúc có thể đóng vai trò như đối tượng Child tại thời điểm chương trình chạy bằng câu lệnh:

emp = new Child();

Như vậy tính đa hình động liên quan đến các lớp kế thừa.

Ví dụ 2: Quan sát đoạn code sau:

|  |
| --- |
| **/\*Ví dụ sử dụng đa hình động\*/**  **using System;**  **namespace RealTimeMethodOverriding**  **{**  **/\*Base class: User\*/**  **public class User**  **{**  **public virtual void UserLogin(string uname,   string password, string role)**  **{**  **if (role == "user"&&(uname == "user" &&  password == "user@123"))**  **{**  **Console.WriteLine("{0} is valid and  Loged sucessfully.........", role);**  **}**  **else**  **{**  **Console.WriteLine("Invalid User Name   or Password!");**  **}**  **}**  **}**  **/\*Derrived class: Admin\*/**  **public class Admin : User**  **{**  **public override void UserLogin(string uname,  string password, string role)**  **{**  **if (role=="Admin"&&(uname == "Admin" && password == "admin@123"))**  **{**  **Console.WriteLine("{0} is valid and Loged successfully", role);**  **}**  **else**  **{**  **Console.WriteLine("Invalid User Name   or Password!");**  **}**  **}**  **}**  **/\*Derrived class: SuperAdmin\*/**  **public class SuperAdmin : User**  **{**  **public override void UserLogin(string uname,   string password, string role)**  **{**  **if (role=="SuperAdmin"&&(uname == "SuperAdmin" && password=="admin@123"))**  **{**  **Console.WriteLine("{0} is valid and Loged sucessfully", role);**  **}**  **else**  **{**  **Console.WriteLine("Invalid User Name   or Password!");**  **}**  **}**  **}** |

Đoạn chương trình thử nghiệm:

|  |
| --- |
| **class Program**  **{**  **static void Main(string[] args)**  **{**  **User user;**  **user = new Admin();**  **user.UserLogin("Admin", "admin@123", "Admin");**  **user = new SuperAdmin();**  **user.UserLogin("SuperAdmin", "admin@123",   "SuperAdmin");**  **Console.ReadLine();**  **}**  **}** |

Đối tượng user thể hiện tính đa hình động, tại mỗi thời điểm khác nhau user đóng vai trò là đối tượng kiểu Admin hoặc SuperAdmin.

Kết quả:

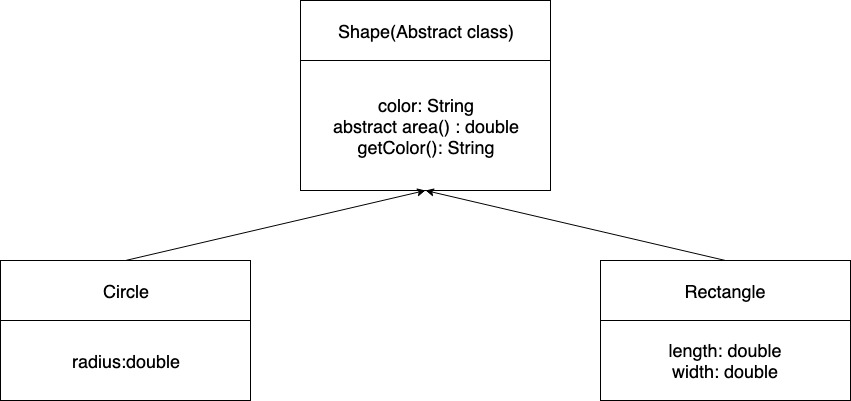
|  |
| --- |
| **Admin is valid and Loged sucessfully.........**  **SuperAdmin is valid and Loged sucessfully.........** |

**So sánh Nạp chồng phương thức và Ghi đè phương thức:**

|  |  |  |
| --- | --- | --- |
|  | **Function Overloading (Nạp chồng phương thức)** | **Function overriding**  **(Ghi đè phương thức)** |
|  |  |  |
| Định nghĩa | Các phương thức trong cùng class có cùng tên nhưng khác nhau về danh sách tham số hoặc kiểu tham số | Các phương thức trong các lớp con có cùng tên, cùng kiểu trả về và cùng danh sách tham số với lớp cha, chỉ khác nhau về cách thực thi |
| Ý nghĩa | Các phương thức khác nhau trong cùng một class có cùng tên nhưng khác nhau về hình thức | Các phương thức trong lớp con định nghĩa lại các thực hiện phương thức đã khai báo trong lớp cha |
| Hành vi | Mở rộng hành vi của các phương thức (extend) | Thay đổi hành vi của phương thức (redefine) |
| Loại đa hình | Compile Time polymorphis | Run Time Polymorphism |
| Tính kế thừa | Không cần kế thừa | Phải có lớp kế thừa |
| Phạm vi phương thức | public, protected, private | public |

### Abstract class và Abstract methods

Khi tạo ra một lớp con kế thừa từ một lớp cha, chúng ta chỉ định nghĩa các thuộc tính mới và thuộc tính mở rộng. Trong một số trường hợp, chúng ta tạo ra một lớp nhưng không bao giờ tạo ra các thể hiện (instance) của lớp đó. Quan sát sơ đồ lớp sau:



Lớp Shape chứa các dữ liệu và hành động chung của các hình nhưng trong thực tế chúng ta không có đối tượng kiểu lớp Shape mà chỉ có thể tạo ra các thể hiện của các lớp con Circle hoặc lớp Rectangle.

Lớp trừu tượng (Abstract Class) là lớp dùng để định nghĩa những thuộc tính và hành vi chung của những lớp khác… Hay nói cách khác: Lớp trừu tượng là lớp dùng để khai báo thuộc tính và phương thức cho các lớp khác sử dụng.

Một số điểm cần lưu ý:

* Trong derived class, phải sử dụng từ khoá override khi định nghĩa phương thức mà đã được khai báo với từ khoá abstract trong base class.
* Trong derived class, không bắt buộc phải định nghĩa phượng thức trừu tượng đã khai báo trong base class.
* Có thể chứa phương thức tạo và phương thức huỷ
* Có thể thực hiện các chức năng với các phương thức không trừu tượng
* Không hỗ trợ đa kế thừa (multiple inheritance)
* Không thể là một static class

Ví dụ:

|  |
| --- |
| **// Ví dụ sử dụng Abstract class**  **using System;**  **namespace AbstractClassExample**  **{**  **abstract class Shape**  **{**  **private string color;**    **public string Color**  **{**  **get**  **{**  **return color;**  **}**  **set**  **{**  **color = value;**  **}**  **}**  **public void toString()**  **{**  **Console.WriteLine("My color is " + this.color);**  **}**  **public abstract int GetArea();**  **}**  **class Square : Shape**  **{**  **int side;**  **public Square(int side)**  **{**    **this. side = side;**  **}**  **public override int GetArea()**  **{**  **return this.side\* this.side;**  **}**  **}**    **public class Test**  **{**  **static void Main()**  **{**  **var sq = new Square(12);**  **sq.Color = "Red";**    **Console.WriteLine($"Area of the square = {sq.GetArea()}");**  **sq.toString();**  **}**  **}**      **}** |

Kết quả:

|  |
| --- |
| Area of the square = 144  My color is Red  Press any key to continue . . . |

Phương thức toString() là phương thức chung và giống nhau cho tất cả các lớp hình nên được định nghĩa trong lớp Shape. Các lớp con kế thừa không cần định nghĩa lại.

Phương thức GetArea() là phương thức chung nhưng công thức tính khác nhau cho từng lớp hình nên chúng ta không thể định nghĩa được phương thức này trong lớp shape mà bắt buộc các lớp con kế thừa phải định nghĩa lại. Vì vậy phương thức GetArea() là một phương thức ảo được khai báo với từ khoá “abstract”. Lớp chứa ít nhất một phương thức ảo thì cũng là lớp ảo, không thể khai báo các phương thức tạo cho lớp này.

Các lớp con khi định nghĩa lại phương thức GetArea() phải sử dụng từ khoá override

### Từ khóa virtual

Phương thức ảo là một phương thức thành viên được khai báo và định nghĩa trong base class và nó có thể được định nghĩa lại (Overriden) bởi các con lớp kế thừa.

Phương thức ảo (Virtual function) đảm bảo các phương thức được gọi đúng theo đối tượng mà không cần tham chiếu đến phương thức nào sẽ được gọi. Chúng là thể hiện của tính đa hình theo thời gian thực. Phương thức được khai báo với từ khoá virtual ở base class và từ khoá override ở derived class.

Một số nguyên tắc khi khai báo Virtual function:

* Virtual function không thể dùng chung với các từ khoá static, abstract, private hay override.
* Virtual function nên được truy cập tham chiếu của một kiểu base class bằng từ khoá new để đạt được tính đa hình động.
* Nguyên mẫu phương thức (prototype) phải giống nhau ở base class và derived class.
* Một lớp có thể có virtual destructor nhưng không có virtual constructors

Ví dụ: Quan sát đoạn code sau:

|  |
| --- |
| **// Ví dụ sử dụng Virtual fuction**  **using System;**  **namespace VirtualFunction**  **{**  **class Shape**  **{**  **private string color;**  **public string Color**  **{**  **get**  **{**  **return color;**  **}**  **set**  **{**  **color = value;**  **}**  **}**  **public virtual void Print()**  **{**  **Console.WriteLine("I'm a shape");**  **}**  **}**  **class Square : Shape**  **{**  **int side;**  **public Square(int side)**  **{**  **this.side = side;**  **}**    **public override void Print()**  **{**  **Console.WriteLine("I'm a Square");**  **}**  **}**  **// Sử dụng phương thức Print() đa hình theo run-time**  **public class Test**  **{**  **static void Main()**  **{**  **Shape sq;**  **sq= new Square(12);**  **sq.Color = "Red";**  **sq.Print();**  **sq = new Shape();**  **sq.Print();**    **}**  **}**  **}** |

Kết quả:

|  |
| --- |
| I'm a Square  I'm a shape |

Ta thấy lớp shape đã định nghĩa phương thức Print(), nếu không có từ khoá virtual thì lớp kế thừa Square không thể định nghĩa lại phương thức Print() bằng từ khoá override.

Một abstract class vẫn có thể chứa virtual functions

### Từ khóa sealed

Từ khoá Sealed (Niêm phong) được dùng để khai báo một lớp hay một phương thức không được kế thừa hay override.

Trong sealed class các phương thức không được dùng phạm vi là protected

Vì sealed method không cho phép override nên trong định nghĩa phải có từ khoá override.

Sealed class không nhất thiết phải có sealed methods.

Ví dụ:

|  |
| --- |
| **// Ví dụ sử dụng Abstract class**  **using System;**  **namespace VirtualFunction**  **{**  **class Shape**  **{**  **private string color;**  **public string Color**  **{**  **get**  **{**  **return color;**  **}**  **set**  **{**  **color = value;**  **}**  **}**  **public virtual void Print()**  **{**  **Console.WriteLine("I'm a shape");**  **}**  **}**  **class Square : Shape**  **{**  **int side;**  **public Square(int side)**  **{**  **this.side = side;**  **}**  **public sealed override void Print()**  **{**  **Console.WriteLine("I'm a shape");**  **}**  **}**  **class Box : Square**  **{**  **int height;**  **public Box(int height, int side) : base(side)**  **{**  **this.height = height;**  **}**  **public override void Print()   //ERROR: cannot inherit member Square.Print() because it is sealed**  **{**  **Console.WriteLine("I'm a shape");**  **}**  **}**  **}** |

Đoạn chương trình báo lỗi “cannot inherit member Square.Print() because it is sealed” vì không thể định nghĩa lại phương thức Print với khai báo sealed.

### Kiểu dữ liệu Object

Là một kiểu dữ liệu cơ bản của tất cả các kiểu dữ liệu trong .NET.

Mọi kiểu dữ liệu đều được kế thừa từ System.Object. Thuộc kiểu dữ liệu tham chiếu.

Kiểu dữ liệu object cung cấp một số phương thức ảo cho phép người dùng overload để sử dụng.

Một số phương thức hỗ trợ cho kiểu Object:

|  |  |
| --- | --- |
| **Phương thức/ thuộc tính** | **Ý nghĩa** |
| ToString() | Trả về kiểu chuỗi của đối tượng (chuyển từ kiểu dữ liệu nào đó về kiểu chuỗi) |
| GetHashCode() | Trả về mã băm của đối tượng. |
| Equals() | So sánh 2 đối tượng và trả về true khi 2 đối tượng có giá trị bằng nhau, ngược lại trả về false. |
| GetType | Trả về kiểu dữ liệu của đối tượng. |
| BaseType | Trả về kiểu của dữ liệu gốc của base class trên cùng |
| Namespace | Trả về chuỗi tên namespace chứa object hiện tại |

Ví dụ 1: Sử dụng kiểu dữ liệu Object

|  |
| --- |
| **// C# Program to demonstrate the Object class**  **using System;**  **class UsingObjectClass**  **{**  **// Main Method**  **static void Main(string[] args)**  **{**  **Object obj1 = new Object();**  **object obj2 = obj1; ;**  **int i = 10;**  **// Các phương thức của lớp object**  **Console.WriteLine(obj1.ToString()); //Object**  **Console.WriteLine(obj1.GetHashCode());//46104728**  **Console.WriteLine(obj1.Equals(i)); //false**  **Console.WriteLine(obj1.Equals(obj1)); //true**  **Type t1 = obj1.GetType();**  **Type t2 = i.GetType();**  **Console.WriteLine("obj1=new Object();");**  **Console.WriteLine(t1.BaseType);**  **Console.WriteLine(t1.Name);**  **Console.WriteLine(t1.FullName);**  **Console.WriteLine(t1.Namespace);**  **Console.WriteLine();**  **Console.WriteLine("For int i = 10;:");**  **Console.WriteLine(t2.BaseType);**  **Console.WriteLine(t2.Name);**  **Console.WriteLine(t2.FullName);**  **Console.WriteLine(t2.Namespace);**  **}**  **}** |

Kết quả:

|  |
| --- |
| **System.Object 46104728 False True**  **obj1 = new Object(); Object System.Object System**  **For int i = 10;: System.ValueType Int32 System.Int32 System** |

**Ví dụ 2: Sử dụng đa hình về kiểu dữ liệu sử dụng object**

Quan sát đoạn code sau:

|  |
| --- |
| **//Ví dụ sử dụng kiểu dữ liệu object**  **using System;**  **namespace Test**  **{**  **class Program**  **{**  **static public bool Compare(object a, object b)**  **{**  **bool sum = (a== b);**  **return sum;**  **}**  **static void Main(string[] args)**  **{**  **float a = 5.5f;**  **float b = 2.9f;**  **int num1 = 1;**  **float num2 = 1.0f;**  **string s1 = "Thu Duc";**  **string s2 = "Thu Duc";**  **Console.WriteLine("{0}", Compare(a,b));**  **Console.WriteLine("{0}", Compare(num1, num2));**  **Console.WriteLine("{0}", Compare(s1, s2));**  **Console.ReadKey();**  **}**  **} }** |

Kết quả:

|  |
| --- |
| **False False True** |

Ta thấy phương thức Compare không chỉ ra kiểu dữ liệu cụ thể mà khai báo với kiểu object. Khi sử dụng phương thức này, người dùng có thể truyền biến với các kiểu dữ liệu tuỳ ý như: bool, char, int, float, double, string, struct, class,…

### Bài tập áp dụng

**BÀI TẬP THỰC HÀNH SỐ 4**

1. **Thông tin chung:**

* Mã số bài tập : HW2-LTHĐT
* Hình thức nộp bài : Nộp qua hệ thống EL của môn học
* Thời hạn nộp bài : … / … / ……
* Nội dung : Chương 4. Kế thừa – Đa hình

**Chuẩn đầu ra cần đạt:**

|  |  |
| --- | --- |
| L.O.2 | Sử dụng cơ chế kế thừa và đa hình trong thiết kế hướng đối tượng |
| L.O.8 | Làm bài tập và nộp bài đúng quy định; |

**BÀI TẬP CƠ BẢN:**

**Hãy cài đặt các lớp sau:**

1. Trả lời các câu hỏi sau:
2. Điểm khác nhau giữa Composition và lớp kế thừa Inheritance là gì?
3. Giải thích sự khác nhau giữa các thành phần protected và thành phần private?
4. Các Constructor và Destructor hoạt động như thế nào trong Derived class?
5. So sánh Overloading methods và Overriding methods.
6. Nêu các nguyên tắc khi sử dụng virtual methods.
7. So sánh static binding và dynamic binding. Cho vi dụ minh hoạ.

**Bài 2.** Tạo một lớp Phân số gồm có giá trị tử số và mẫu số và các phương thức tiện ích như nhập, xuất, in ra màn hình, rút gọn phân số. Thực hiện phương thức nạp chồng toán tử sau :

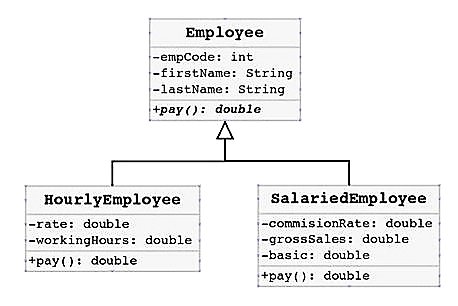
* 1. Các toán tử một ngôi : +, -
  2. Các toán tử hai ngôi : **+ , - , \*, /**
  3. Các toán tử so sánh == , > , >=, <, <=, !=
  4. Toán tử gán: =

**Bài 3**. **Cài đặt sơ đồ lớp sau :**



Thực hiện viết chương trình chạy thử nghiệm khởi tạo các đối tượng/ danh sách đối tượng và gọi sử dụng tất cả các phương thức của các lớp trên

**Bài 4.** **Cài đặt sơ đồ lớp sau :**



Employee::pay(): là một abstract method

HourlyEmployee::pay(): *getRate() \* getWorkingHours()*

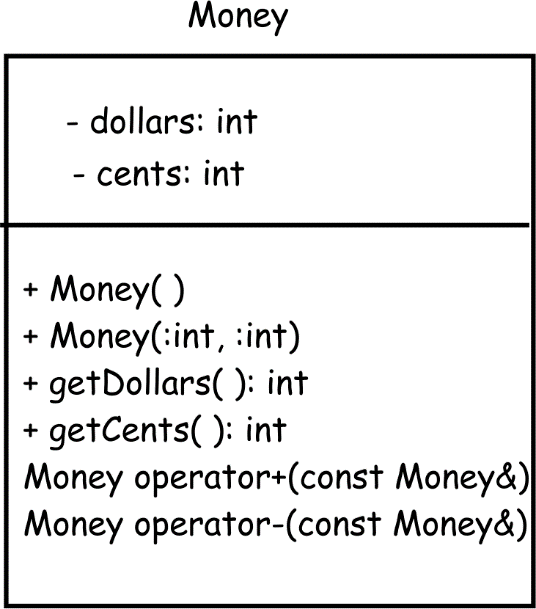
SalerieEmployee::pay() : *getBasic() + getCommissionRate() \* getGrossSale()*

Thực hiện viết chương trình chạy thử nghiệm:

-Khởi tạo danh sách bao gồm một số phần tử HourlyEmployee và SalariedEmployee

-Gọi sử dụng tất cả các phương thức của các lớp trên

**Bài 5.** **Cài đặt sơ đồ lớp sau và viết chương trình sử dụng thử nghiệm tất cả các tính năng của các lớp**



Thực hiện viết chương trình chạy thử nghiệm khởi tạo danh sách đối tượng Money và gọi sử dụng tất cả các phương thức của các lớp trên

**Bài 6.** Viết chương trình cho phép tạo ra các đơn thức và thực hiện các chức năng sau cho mỗi đơn thức đó:

1. Khởi tạo đơn thức
2. Phương thức hủy
3. Xuất đơn thức
4. Lấy giá trị hệ số, số mũ
5. Thay đổi lại giá trị hệ số, số mũ
6. Tính giá trị đơn thức
7. Cộng đơn thức

**BÀI TẬP NÂNG CAO:**

**Bài 7.** Xây dựng lớp MyArray chứa tối đa 10 số nguyên và các phương thức tiện ích như tạo mảng, xuất mảng ra màn hình. Xây dựng các toán tử sau trên mảng:

1. ++: tăng các phần tử trên mảng 1 đơn vị.
2. -- : giảm các phần tử trên mảng 1 đơn vị.
3. + (num) : cộng thêm cho các phần tử trên mảng một số nguyên num. Trả về mảng mới.
4. – (num): trừ thêm cho các phần tử trên mảng một số nguyên num. Trả về mảng mới.
5. + : cộng hai mảng. Trả về mảng mới với giá trị các phần tử là tổng của từng phần tử tương ứng trên hai mảng. Số phần tử mảng trả về là số phần tử lớn nhất giữa hai mảng.

Ví dụ: arr1 = [1, 5, 6, 7, 8]

arr2 = [3, 1, 2, 8, 0, 9, 5]

arr3 = arr1 + arr2 = [4, 6, 8, 15, 8, 9, 5]

1. \* : nhân hai mảng. Trả về mảng mới là mảng hai chiều với giá trị trên dòng thứ i là tích của các phần tử của mảng 1 với phần tử thứ i của mảng 2.

Ví dụ: = [1, 5, 6, 7, 8]

= [3, 1]

|  |  |
| --- | --- |
|  | INTERFACE |

Chương này nhằm giới thiệu cho sinh viên các khái niệm về Interface và cách sử dụng Interface trong kế thừa

### Khái niệm Interface

Interface là bản thiết kế của một lớp. Nó giống như một abstract class vì các phương thức được khai báo bên trong interface chỉ là khai báo prototype, không chứa định nghĩa phương thức. Interface có thể được một class hoặc struct khác kế thừa.

Interface chứa định nghĩa các thuộc tính giao tiếp, phương thức và sự kiện, mà các thành viên của Interface đó. Các Interface chỉ chứa khai báo của các thành viên này. Việc định nghĩa các thành viên là trách nhiệm của lớp kế thừa. Nó thường giúp ích trong việc cung cấp một Cấu trúc chuẩn mà các lớp kế thừa nên theo.

* **Các đặc điểm interface:**
* Interface có các thành viên là methods, properties, indexers, events và không chứa field.
* Một interface có thể implement nhiều interface cơ sở (base interface), một class hoặc struct có thể implement nhiều interface.
* Bất kỳ class hay struct nào implement một interface thì implement tất cả các thành viên và định nghĩa đầy đủ các thành viên của interface đó.
* Các thành viên của interface không được phép định nghĩa (definition) mà chỉ được khai báo (declaration).
* Interface không có constructors.
* Nếu một lớp implement từ nhiều interface có cùng tên thành viên thì trong lớp phải chỉ rõ thành viên đó thuộc interface nào (explicit interface).
* **Mục đích sử dụng interface:**
* Vì C# không hỗ trợ đa kế thừa nên interface ra đời như là 1 giải pháp cho việc đa kế thừa này.
* Trong một hệ thống việc trao đổi thông tin giữa các thành phần cần được đồng bộ và có những thống nhất chung. Vì thế dùng interface sẽ giúp đưa ra những quy tắc chung mà bắt buộc các thành phần trong hệ thống này phải làm theo mới có thể trao đổi với nhau được.

### Khai báo interface

**Cú pháp:**

**[modifier] interface <interface-name> [: interface-base]**

**{**

**// Khai báo các thành phần bên trong interface**

**}**

Interface có **modifier** là public hoặc internal, nếu không ghi rõ mặc định là internal.

**interface**là từ khoá dùng để khai báo 1 interface.

**<interface-name>**là tên do người dùng đặt và tuân theo các quy tắc đặt tên

**Lưu ý** là để tránh nhầm lẫn với lớp kế thừa thì khi đặt tên interface người ta thường thêm tiền tố “**I**” để nhận dạng.

Cú pháp khai báo interfacekế thừa từ interface khác

|  |
| --- |
| **// inheriting the interface**  **access\_modifier interface Interface\_name: BaseInterface\_name**  **{**  **// Your code**  **}** |

Ví dụ:

|  |
| --- |
| **public interface IScanner**  **{**  **void Scan();**  **}**  **public interface IPrinter**  **{**  **void Print();**  **}** |

|  |
| --- |
| **public interface IPhotocopier : IScanner, IPrinter**  **{**  **void Copy();**  **}** |

### Cài đặt Interface

Cài đặt class implement một interface

-Tạo interface

|  |
| --- |
| **interface IShape**  **{**  **// methods**  **double ChuVi();**  **double DienTich();**  **string toString();**  **}** |

-Tạo class:

|  |
| --- |
| **class Square : IShape**  **{**  **private double size;**  **public Square(double x)**  **{**  **size = x;**  **}**    **public double Size**  **{**  **get { return size; }**  **set { size = value; }**  **}**  **public double ChuVi()**  **{**  **return 4 \* size;**  **}**  **public double DienTich()**  **{**  **return size \* size;**  **}**  **public string toString()**  **{**  **string str = "hinh vuong canh = " + size**  **+ "\n dien tich = " + DienTich()**  **+ "\n chu vi = " + ChuVi();**  **return str;**  **}**  **}** |

#### Cài đặt nhiều giao diện

Trong C#, không hỗ trợ đa kế thừa. Vì vậy, sử dụng interface để hiện thực đa kế thừa (multiple-implementations).

* Định nghĩa interface:

|  |
| --- |
| **interface IShape**  **{**  **// methods**  **double getChuVi();**  **double getDienTich();**  **}**  **interface IShapeDisplay**  **{**  **void Display();**  **}** |

* Định nghĩa class:

|  |
| --- |
| **class Rectangle : IShapeDisplay, IShape**  **{**  **private double dai;**  **private double rong;**  **public Rectangle(double xDai, double xRong)**  **{**  **dai = xDai;**  **rong = xRong;**  **}**  **public double Dai**  **{**  **get {return dai; }**  **set {dai = value; }**  **}**  **public double Rong**  **{**  **get {return rong; }**  **set {rong = value; }**  **}**  **public double getChuVi()**  **{**  **return 2 \* (dai + rong);**  **}**  **public double getDienTich()**  **{**  **return dai \* rong;**  **}**  **public void Display()**  **{**  **Console.WriteLine("hinh chu nhat: \n dai = {0} \n rong = {1} \n chu vi = {2} \n dien tich = {3} ", \_dai, \_rong, getChuVi(), getDienTich());**  **}**  **}** |

-Ứng dụng:

|  |
| --- |
| **static void Main(string[] args)**  **{**  **//khoi tao**  **Rectangle re = new Rectangle(5, 7);**  **Circle ci = new Circle(6);**    **//xuat thong tin**  **re.Display();**  **ci.Display();**  **Console.ReadKey();**  **}** |

Kết quả:

|  |
| --- |
| hinh chu nhat:  dai = 5  rong = 7  chu vi = 24  dien tich = 35 |

### Truy xuất phương thức của giao diện

Một lớp implement từ nhiều interface có cùng tên thành viên thì trong lớp phải chỉ rõ thành viên đó thuộc interface nào (explicit interface).

-Tạo interface

|  |
| --- |
| **interface IShape**  **{**  **// methods**  **double getChuVi();**  **double getDienTich();**  **void Display();**  }  **interface IShapeDisplay**  **{**  **void Display();**  } |

-Tạo class

|  |
| --- |
| **class Rectangle : IShapeDisplay, IShape**  **{**  **private double dai;**  **private double rong;**  **public Rectangle(double xDai, double xRong)**  **{**  **dai = xDai;**  **rong = xRong;**  **}**  **public double Dai**  **{**  **get { return dai; }**  **set { dai = value; }**  **}**  **public double Rong**  **{**  **get { return rong; }**  **set { rong = value; }**  **}**  **public double getChuVi()**  **{**  **return 2 \* (dai + rong);**  **}**  **public double getDienTich()**  **{**  **return dai \* rong;**  **}**  **void IShapeDisplay.Display()**  **{**  **Console.WriteLine($"hinh chu nhat: \n dai = {dai} \n  rong = {rong} \n chu vi = { getChuVi()}”)**  **}**  **void IShape.Display()**  **{**  **Console.WriteLine("day la minh hoa hinh chu nhat:");**  **}**  **}**  **class Circle: IShape, IShapeDisplay**  **{**  **private double banKinh;**    **public Circle(double xBanKinh)**  **{**  **banKinh = xBanKinh;**    **}**  **public double BanKinh**  **{**  **get { return banKinh; }**  **set { banKinh = value; }**  **}**    **public double getChuVi()**  **{**  **return 2 \* 3.14159 \* banKinh;**  **}**  **public double getDienTich()**  **{**  **return 3.14159 \* banKinh \* banKinh;**  **}**  **void IShapeDisplay.Display()**  **{**  **Console.WriteLine($"hinh tron: \n ban kinh =   {banKinh} \n chu vi = { getChuVi()}\n  dien tich = { getDienTich()}");**  **}**  **void IShape.Display()**  **{**  **Console.WriteLine("day la minh hoa hinh tron:");**  **}**  **}** |

* Ứng dụng:

|  |
| --- |
| **static void Main(string[] args)**  **{**  **//khoi tao**  **Rectangle re = new Rectangle(5, 7);**  **IShape Shape = (IShape)re;**  **IShapeDisplay shapeDisplay1 = (IShapeDisplay)re;**  **//explicit**  **Console.WriteLine("---explicit interface---");**  **Shape.Display();**  **shapeDisplay1.Display();**    **//polymorphism**  **Console.WriteLine("----polymorphism----");**  **IShapeDisplay[] shapeArr = { new Rectangle(4,6),  new Circle(7) };**  **for (int i = 0; i < 2; i++)**  **{**  **shapeArr[i].Display();**  **}**  **Console.ReadKey();**  **}** |

### Một số trường hợp sử dụng interface

Các trường hợp sau nên sử dụng interface:

* Khi cần tạo ra một nhóm các chức năng có liên quan đến nhau mà một non-abstract class hay một struct phải định nghĩa.

Khi một lớp implement một interface thì phải định nghĩa đầy đủ các phương thức của interface đó, vì thế khi muốn tạo ra các lớp thực hiện một số nhiệm vụ như nhau thì interface sẽ là lựa chọn tốt nhất trong trường hợp này. Interface được dùng để **qui định kiến trúc** cho lớp nào kế thừa từ nó. Việc xây dựng kiến trúc giữa các lớp rất quan trọng điều đó giúp cho chương trình thêm tính chặt chẽ, có logic và giúp cho lập trình viên dễ kiểm soát khi viết code.

* Khi một lớp cần kế thừa các hành vi từ nhiều lớp thì phải sử dụng thông qua interface vì C# không hổ trợ đa kế thừa.

### Bài tập áp dụng

**BÀI THỰC HÀNH SỐ 5**

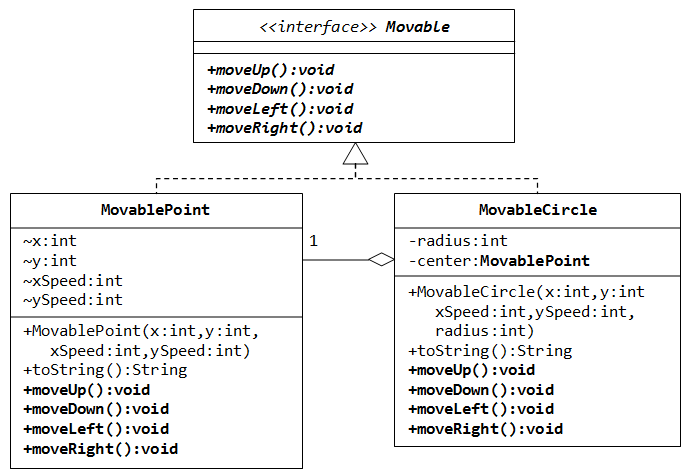
1. **Thông tin chung:**

* Mã số bài tập : HW3 - LTHĐT
* Hình thức nộp bài : Nộp qua Moodle môn học
* Thời hạn nộp bài : … / … / ……
* Nội dung : Chương 5. Lớp giao diện
* **Chuẩn đầu ra cần đạt:**

|  |  |
| --- | --- |
| L.O.3 | Sử dụng giao diện trong thiết kế hướng đối tượng |
| L.O.8 | Làm bài tập và nộp bài đúng quy định; |

**BÀI TẬP CƠ BẢN:**

1. Hãy cài đặt sơ đồ lớp sau:

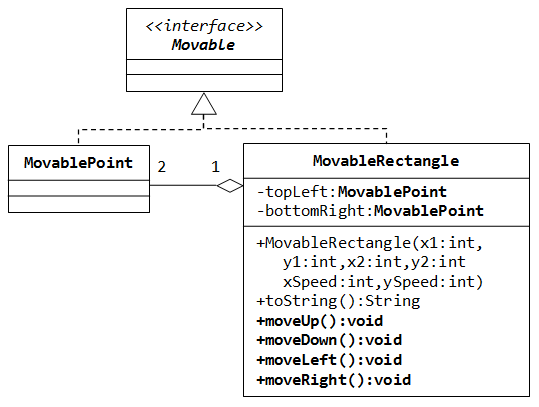


Thực hiện viết chương trình chạy thử nghiệm:

-Khởi tạo danh sách các đối tượng MovableCircle

-Gọi sử dụng tất cả các phương thức của MovableCircle

1. Cài đặt sơ đồ lớp sau

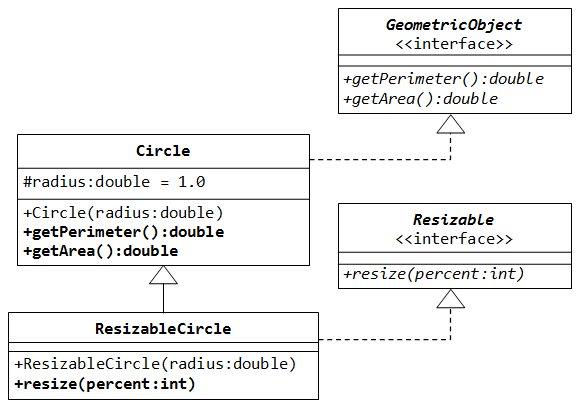


Thực hiện viết chương trình chạy thử nghiệm:

-Khởi tạo danh sách các đối tượng MovableRectangle

-Gọi sử dụng tất cả các phương thức của MovableRectangle

1. Cài đặt sơ đồ sau:

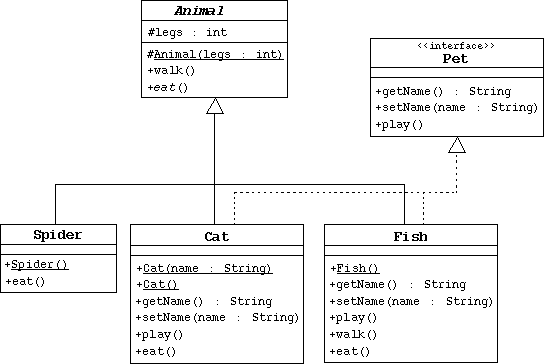


Thực hiện viết chương trình chạy thử nghiệm:

-Khởi tạo danh sách các đối tượng Circle, ResizableCircle

-Gọi sử dụng tất cả các phương thức của Circle, ResizableCircle

1. Cài đặt sơ đồ sau:



1. Tạo lớp Animal là một Abstract class chứa abstract method eat(). Phương thức walk() in ra trạng thái về di chuyển có bao gồm số chân (legs). Ví dụ : “Đi bẳng 4 chân”.
2. Tạo lóp Spider class kế thừa từ Animal. Định nghĩa phương thức tạo với tất cả Spiders đi bằng 8 chân.
3. Tạo Pet class theo sơ đồ.
4. Tạo lớp Cat kết thừa từ Animal và Pet. Phương thức tạo bao gồm cat có 4 chân.. Cài đặt các phương thức Pet() và Eat()
5. Cài đặt Fish class. Override phương thức Animal với thông báo “ Không thể đi và không có chân”
6. Tạo chương trình thử nghiệm bao gồm các lệnh sau:
7. Fish d = new Fish();
8. Cat c = new Cat("Fluffy");
9. Animal a = new Fish();
10. Animal e = new Spider();  
     Pet p = new Cat();

**BÀI TẬP NÂNG CAO**

1. Một công ty quản lý tiền lương nhân viên, mỗi nhân viên đều có mã số, họ tên, năm sinh. Lương của mỗi loại nhân viên được tính khác nhau.

* Dạng người lao động lãnh lương từ ngân sách Nhà nước được gọi là cán bộ, công chức (dạng biên chế).
* Dạng người lao động nhận lương từ ngân sách cơ quan được gọi là nhân viên hợp đồng.

Như vậy, hệ thống quản lý có hai loại nhân viên là nhân viên biên chế (NVBC) và nhân viên hợp đồng (NVHD)

* + NVBC: gồm các đối tượng là cán bộ, công chức có các thuộc tính: lương căn bản, hệ số lương, tiền phụ cấp chức vụ.

Lương = lương căn \* hệ số lương \* tiền phụ cấp

* + NVHD: gồm các đối tượng làm hợp đồng gồm các thuộc tính: tiền công lao động một giờ, số giờ làm việc trong tháng.

Lương =Tiền công lao động một giờ \* số giờ làm việc trong tháng

1. Viết và cài đặt sơ đồ lớp cho bài toán với các thuộc tính và một số phương thức: Khởi tạo, nhập, xuất thông tin và phương thức tính tiền lương của mỗi loại nhân viên.
2. Viết chương trình để quản lý tiền lương trong công ty:
   1. Cho phép người dùng nhập vào số lượng (n) nhân viên trong công ty.
   2. Xây dựng phương thức nhập thông tin cho n nhân viên đó
   3. Xây dựng phương thức xuất thông tin cho n nhân viên đó
   4. Xây dựng phương thức tính tổng tiền lương phải trả trong tháng
   5. Xây dựng để tìm và xuất thông tin của nhân viên của nhân viên có lương cao nhất, thấp nhất trong công ty.
   6. Void minMax(NhanVien\* nv[], int n)
3. Một công ty được giao nhiệm vụ quản lý các phương tiện giao thông gồm các loại: ô tô, xe máy, xe tải.

* Mỗi loại phương tiện giao thông cần quản lý: Hãng sản xuất, năm sản xuất, giá bán và màu.
* Các ô tô cần quản lý: số chỗ ngồi, kiểu động cơ
* Xe máy cần quản lý: công suất
* Xe tải cần quản lý: trọng tải.

1. Xây dựng các lớp cần thiết cho bài toán với các phương thức sau:
   1. Khởi tạo
   2. Xuất thông tin đối tượng
2. Xây dựng chương trình chính thực hiện các chức năng sau:
   1. Nhập đăng ký phương tiện (Mỗi đối tượng tạo ra sẽ được đưa thông tin vào thông qua phương thức khởi tạo)
   2. Tìm phương tiện theo màu
   3. Tìm phương tiện sản xuất trước năm x (x được cung cấp từ người dùng).
3. Công ty XYZ cần xây dựng một ứng dụng quản lý nhân sự và tính lương cho nhân viên trong công ty như sau:

Quản lý thông tin về nhân viên: Họ tên, ngày sinh, địa chỉ. Biết ngày sinh là một đối tượng của lớp **NgaySinh** gồm 3 thông tin (Ngày, Tháng, Năm)

Tính lương cho nhân viên:

Hiện công ty có 3 loại nhân viên và cách tính lương tương ứng cho từng loại nhân viên như sau:

* + - Nhân viên sản xuất: Lương = **số sản phẩm** \* 3000
    - Nhân viên công nhật: Lương = **số ngàylàm** \* 40 000

1. Xây dựng lớp **NgaySinh** với các thành phần thuộc tính kiểu int (**Ngày, Tháng, Năm** ) và phương thức **NhapNS** (nhập ngày sinh), **XuatNS (**xuất dưới dạng: ví dụ **25-2**-**1987)**
2. Xây dựng các lớp (NhanVien, NVSX (nhân viên sản xuất), NVCN (nhân viên công nhật)) với các thành phần thuộc tính và phương thức **Nhập, Xuất và Tính lương**.
3. Viết chương trình cho phép quản lý nhân viên trong công ty như sau:
   1. Phương thức nhập danh sách gồm nhiều nhân viên (nhập số -1 để kết thúc việc nhập)
   2. Phương thức xuất thông tin các nhân viên trong công ty
   3. Phương thức tìm nhân viên có lương cao nhất với cấu trúc sau:

**NhanVien**\* tim(**NhanVien**\* nv[ ] , int n) // n là số lượng nhân viên

|  |  |
| --- | --- |
|  | LẬP TRÌNH KHÁI QUÁT |

Chương này nhằm giới thiệu cho sinh viên các khái niệm lập trình khái quát, Xây dựng phương thức và lớp khái quát. Qua đó sinh viên có thể sử dụng các class template template để việc bảo trì, thay đổi hay phát triển chương trình dể dàng hơn

### Khái niệm Generic

**Generic Programming (lập trình khái quát)** là một cách tiếp cận lập trình mà trước hết lập trình viên định nghĩa một code chung. Nghĩa là phương pháp lập trình mà kiểu dữ liệu không được xác định tại giai đoạn xây dựng đơn vị code như định nghĩa class, phương thức, kiểu dữ liệu cụ thể của các thành phần được xác định cụ thể tại thời điểm khởi tạo và sử dụng.

Một số ngôn ngữ cho phép lập trinhg tổng quát như Ada, C#, Delphi, F#, Java, Nim, Python, Swift, TypeScript, Visual Basic .NET, …

Generics trong C# cho phép **tham số hóa kiểu dữ liệu**. Khi kiểu dữ liệu là một tham số, chúng ta có thể tạo ra cấu trúc, class, phương thức, interface, … để sử dụng với nhiều kiểu dữ liệu khác nhau mà không cần phải viết lại code cho từng kiểu dữ liệu riêng rẽ. Nhờ vậy generics giúp tái sử dụng code hiệu quả.

### Một số đặc điểm của generic trong C#

Generic trong C# có các đặc điểm sau:

* Generic classes, methods, interface, … cho phép lựa chọn kiểu dữ liệu ở giai đoạn sử dụng, không cần phải chỉ ra cụ thể ở giai đoạn định nghĩa.
* Lập trình generic yêu cầu phải cung cấp một tham số kiểu type\_parameter thay thế đặt trong cặp dấu <>; tên tham số kiểu type\_parameter thường là một chữ cái in hoa nằm cuối bảng chữ cái (thông dụng nhất là T, U, V);
* Trong code có thể sử dụng tham số kiểu type\_parameter này tương tự như bất kỳ kiểu dữ liệu “thật” nào;
* Số lượng tham số kiểu type\_parameter không giới hạn; nếu có nhiều tham số kiểu type\_parameter thì các tham số này được phân tách bởi dấu phẩy.

Lập trình generics nên được sử dụng trong các trường hợp sau:

* Nếu có sự trùng lặp code về mặt logic và cách xử lý dữ liệu, chỉ khác biệt về kiểu dữ liệu.
* Nếu lúc xây dựng class chưa xác định được kiểu dữ liệu của các thành viên

So với C++ template, generic trong C# có một số điểm khác biệt như sau:

### Generic được áp dụng cho các đối tượng nào trong C#?

Generics trong C# có thể áp dụng cho: class, method, interface, delegate.

Tùy vào đối tượng áp dụng, **tham số kiểu** của generic chỉ khác biệt về phạm vi tác dụng.

Đối với class và interface, **tham số kiểu** tác dụng trong toàn bộ code của class. **tham số kiểu** này có thể sử dụng làm kiểu cho biến thành viên, thuộc tính, kiểu trả về của phương thức thành viên, kiểu tham số của phương thức thành viên.

Đối với method, **tham số kiểu** chỉ có tác dụng trong phạm vi code của method đó. Nghĩa là **tham số kiểu** có thể được sử dụng làm kiểu của biến cục bộ, kiểu trả về, kiểu tham số của method.

Collection<T> là tập hợp các dữ liệu không cùng kiểu. Các class cà interface được tập hợp vào namespace System.Collections.Generic. Một số các lớp collection kiểu generic thông dụng như: List<T>, Dictionary<TKey, TValue>, HashSet<T>, SortList<TKey, TValue>, Stack<T>, Queue<T> ... những lớp generic này trong namespace System.Collections. Generic.

Một số generic class thường sử dụng:

|  |  |
| --- | --- |
| **Generic Collection** | **Mô tả** |
| List<T> | Chứa danh sách các đối tượng có thể truy cập bằng chỉ số (index). Cung cấp các phương thức tìm kiếm, sắp xếp và thao tác với danh sách |
| Dictionary<TKey, TValue> | Đại diện cho tập hợp của các cặp (key,value) |
| Hashset<T> | Tập hợp các giá trị không trùng lặp |
| SortList<TKey, TValue> | Tập hợp các cặp key, value. Tự động thêm các phần tử theo thứ tự tăng dần, mặc định sắp xếp theo key |
| Stack<T> | Lưu trữ các giá trị dưới dạng LIFO (Last In First Out). Cung cấp các phương thức thêm giá trị vào cuối danh sách Push(), lấy giá trị ở cuối danh sách Pop() và truy xuất giá trị hiện tại Peek() |
| Queue<T> | Lưu trữ các giá trị dưới dạng FIFO (First In First Out). Cung cấp các phương thức thêm giá trị vào cuối danh sách EnQueue(), lấy giá trị ở đầu danh sách DeQueue(). |

Một số generic interface thường sử dụng:

|  |  |
| --- | --- |
| Generic interface | Mô tả |
| IEnumerable<T> | Sử dụng kế thừa từ interface này khi muốn đối tượng có thể thực hiện duyệt các phần tử bằng foreach. Nó định nghĩa phương thức GetEnumerator trả về một enumerator. |
| ICollection<T> | Định nghĩa các phương thức thao tác với generic collection. Giao diện này được triển khai bở các generic collection như: lấy tổng phần tử bằng thuộc tính Count, sao chép các phần tử vào mảng CopyTo(), thêm phần Add(), xóa phần tử Remove(), Clear() |
| IList<T> | Tập hợp một danh sách các phần tử truy cập được theo chỉ số (index) của nó. Nó có indexer, phương thức để chèn phần tử Insert(), xóa phần tử RemoveAt(). |
| IDictionary<TKey, TValue> | Tập hợp các phần tử mà mỗi phần tử là một cặp ( key, value). Interface cung cấp các phương thức Add(), Clear(), Contains(), GetEnumerator(), Remove() |
| IComparer<Tkey, TValue> | Định nghĩa phương thức thực hiện so sánh hai đối tượng |
| IEqualityComparer<<Tkey, TValue> | Định nghĩa phương thức thực hiện so sánh bằng giữa hai đối tượng |

### Generic fields

Một generic class có thể chứa một hoặc nhiều generic fields tuy nhiên nó không thể khởi tạo. Khi đó một kiểu dữ liệu T được dùng như kiểu tổng quát.

Ví dụ:

|  |
| --- |
| class DataStore<T>  {  public T data;  } |

|  |
| --- |
| class DataStore<T>  {  public T[] data = new T[10];  } |

### Generic methods

Để khai báo tổng quát mẫu phương thức chúng ta khai báo cú pháp như sau:

|  |
| --- |
| **<Modifier> <return\_type> function\_name   (T arg1, T arg2, …)**  **{**  **// body of function**  **}** |

* T: naming convention (any data types)

**Ví dụ 1:**

|  |
| --- |
| **File: DataStore.cs** |
| **//Ví dụ Định nghĩa generic class**  **using System;**  **using System.Collections;**  **using System.Collections.Generic;**  **namespace DataStoreApp**  **{**  **class DataStore<T>: IEnumerable, IComparer**  **{**  **private T[] data = new T[10];**    **public void AddOrUpdate(int index, T item)**  **{**  **if (index >= 0 && index < 10)**  **data[index] = item;**  **}**  **public T GetData(int index)**  **{**  **if (index >= 0 && index < 10)**  **return data[index];**  **else**  **return default(T);**  **}**  **//Định nghĩa phương thức của IEnumerator   //để sử dụng foreach iterator**  **public IEnumerator GetEnumerator()**  **{**  **return data.GetEnumerator();**  **}**  **public void Print()**  **{**  **foreach (var e in data)**  **{**  **if(e != null)**  **Console.WriteLine(e);**  **}**  **}**  **//Định nghĩa phương thức của IComparer   //để so sánh hai đối tượng**  **public int Compare(object x, object y)**  **{**  **return ((new   CaseInsensitiveComparer()).Compare(y, x));**  **}**    **//Quá tải phương thức**  **public void AddOrUpdate(T data1, T data2)**  **{**  **for(int i = 0; i < data.Length; i++)**  **{**  **if(Compare(data[i], data1)== 0)**  **{**  **data[i] = data2;**  **}**  **}**  **}**  **//public void AddOrUpdate<U>(T data1, U data2) { }**  **//public void AddOrUpdate(T data) { }**  **}**  **}** |

|  |
| --- |
| **File: Program.cs** |
| **//Ví dụ sử dụng Generic Class DataStore**  **using System;**  **namespace DataStoreApp**  **{**    **class Program**  **{**  **static void Main(string[] args)**  **{**  **//Update**  **DataStore<string> cities=new DataStore<string>();**  **cities.AddOrUpdate(0, "Ha Noi");**  **cities.AddOrUpdate(1, "Hai Phong");**  **cities.AddOrUpdate(2, "Da Nang");**  **cities.Print();**    **DataStore<int> empIds=new DataStore<int>();**  **empIds.AddOrUpdate(0, 50);**  **empIds.AddOrUpdate(1, 100);**  **empIds.AddOrUpdate(2, 150);**  **empIds.Print();**  **//Update**  **cities.AddOrUpdate("Hai Phong", "Ho Chi Minh");**  **cities.Print();**  **}**  **}**  **}** |

**Ví dụ: Xây dựng template phương thức hoán vị**

|  |
| --- |
| File: Program.cs |
| **//Ví dụ sử dụng Generic method**  **using System;**  **namespace GenericProgramming**  **{**  **class Program**  **{**  **static void Main(string[] args)**  **{**  **//khai bao bien**  **//Kiểu số nguyên**  **int soNguyen1 = 5;**  **int soNguyen2 = 7;**  **//Kiểu số thực**  **double soThuc1 = 6.5;**  **double soThuc2 = 7.8;**  **//Kiểu Student**  **Student sV1 = new Student("An",   new DateTime(2020, 1, 1), 1);**  **Student sV2 = new Student("Binh",   new DateTime(2020, 1, 2), 0);**  **//hoan vi so nguyen**  **Console.WriteLine("Hoan vi so nguyen");**  **Console.WriteLine($"Truoc hoan vi:\n  Num1 = {soNguyen1} \n Num 2 = {soNguyen2}");**  **Swap(ref soNguyen1, ref soNguyen2);**  **Console.WriteLine($"Sau hoan vi:\n  Num1 = {soNguyen1} \n Num 2 = {soNguyen2}");**  **//hoan vi so thuc**  **Console.WriteLine("Hoan vi so thuc");**  **Console.WriteLine($"Truoc hoan vi:\n  Num1 = {soThuc1} \n Num 2 = {soThuc2}");**  **Swap(ref soThuc1, ref soThuc2);**  **Console.WriteLine($"Sau hoan vi:\n  Num1 = {soThuc1} \n Num 2 = {soThuc2}");**  **//hoan vi object**  **Console.WriteLine("Hoan vi objects");**  **Console.WriteLine($"Truoc hoan vi:\n  Ten SV1 = {sV1.StudentName}\n  Ten SV2 = {sV2.StudentName}");**  **Swap(ref sV1, ref sV2);**  **Console.WriteLine($"Sau hoan vi:\n  Ten SV1 = {sV1.StudentName} \n  Ten SV2 = {sV2.StudentName}");**  **Console.ReadKey();**  **}**  **//Generic method**  **private static void Swap<T>(ref T a, ref T b)**  **{**  **T temp = b;**  **b = a;**  **a = temp;**  **}**  **}**  **}** |

Kết quả:

|  |
| --- |
| \*\*\*Hoan vi so nguyen\*\*\*  Truoc hoan vi:  Num1 = 5  Num 2 = 7  Sau hoan vi:  Num1 = 7  Num 2 = 5  \*\*\*Hoan vi so thuc\*\*\*  Truoc hoan vi:  Num1 = 6.5  Num 2 = 7.8  Sau hoan vi:  Num1 = 7.8  Num 2 = 6.5  \*\*\*Hoan vi objects \*\*\*  Truoc hoan vi:  Ten SV1 = An  Ten SV2 = Binh  Sau hoan vi:  Ten SV1 = Binh  Ten SV2 = An |

Trong ví dụ trên chỉ xây dựng một phương thức khái quát **Swap<T>(ref T a, ref T b)** nhưng có thể gọi sử dụng truyền tham số đầu vào cho biến kiểu built-in và reference.

### Generic class trong C#

So với generic methods ở phần trên, cách sử dụng **tham số kiểu** \_type parameter đối với generic class là không khác biệt. Sự khác biệt lớn nhất là: phạm vi có ý nghĩa của **tham số kiểu** \_type parameter bây giờ là toàn bộ class, thay vì chỉ trong một phương thức.

Khi có nhiều class cùng chung logic, chỉ khác biệt về kiểu dữ liệu cần xử lý thì có thể viết một generic class thay cho viết nhiều class riêng rẽ. Nó sẽ giúp chúng ta tránh phải viết code lặp nhiều lần.

Cú pháp khai báo lớp template:

|  |
| --- |
| * **template <class T>** * **class Tenlop** * **{** * **// thành phần trong lớp** * **}** * **// định nghĩa các phương thức trong lớp** * **template <class T>**   **<Kiểu trả về> Tenlop <T>:: Tênphương thức (các tham số)**  **{**  **//Nội dung phương thức**  **}** |

Ví dụ:

Trước tiên ta định nghĩa khuôn mẫu lớp KeyValue gồm hai field có kiểu đại diện là K và V như sau:

|  |
| --- |
| **File: KeyValue.cs** |
| **//Ví dụ định nghĩa class template**  **using System;**  **namespace UsingGenericClass**  **{**  **public class KeyValue<K, V>**  **{**  **//fields**  **private K key;**  **private V value;**  **//properties**  **public K Key**  **{**  **get**  **{**  **return key;**  **}**  **set**  **{**  **key = value;**  **}**  **}**  **public V Value**  **{**  **get**  **{**  **return value;**  **}**  **set**  **{**  **this.value = value;**  **}**  **}**  **//constructor**  **public KeyValue(K key, V value)**  **{**  **this.key = key;**  **this.value = value;**  **}**  **//Methods**  **public string ToString()**  **{**  **string s = $"Phone = {key}, Name={value}";**  **return s;**  **}**  **}**  **}** |

* Ứng dụng PhoneName sử dụng class template KeyValue tương ứng với hai trường K kiểu **int** và V kiểu **string** như sau:

|  |
| --- |
| **File: PhoneNameApp.cs** |
| **using System;**  **namespace UsingGenericClass**  **{**  **class PhoneNameInt : KeyValue<int, string>**  **{**  **public PhoneNameInt(int key, string value) : base(key, value)**  **{**  **}**  **}**  **class PhoneNameApp**  **{**  **static void Main(string[] args)**  **{**  **// Khai báo đối tượng**  **PhoneNameInt entry1 =   new PhoneNameInt(0903123456, "Thu An");**  **// Kiểu trả về là int.**  **int phoneInt = entry1.Key;**  **// Kiểu trả về này là string.**  **string phoneString = entry2.Key;**  **Console.WriteLine(entry1.ToString());**  **Console.ReadKey();**  **}   }**  **}** |

* Mặt khác ứng dụng PhoneName sử dụng class template KeyValue tương ứng với hai trường K kiểu **string** và V kiểu **string** như sau**:**

|  |
| --- |
| **File: PhoneNameApp.cs** |
| **using System;**  **namespace UsingGenericClass**  **{**  **class PhoneNameInt : KeyValue<string, string>**  **{**  **public PhoneNameInt(int key, string value) : base(key, value)**  **{**  **}**  **}**  **class PhoneNameApp**  **{**  **static void Main(string[] args)**  **{**  **// Khai báo đối tượng**  **PhoneNameInt entry1 =   new PhoneNameInt("0903123456", "Thu An");**  **// Kiểu trả về là int.**  **string phoneStr = entry1.Key;**  **// Kiểu trả về này là string.**  **string phoneStr = entry1.Key;**  **Console.WriteLine(entry1.ToString());**  **Console.ReadKey();**  **}   }**  **}** |

### Bài tập áp dụng

**BÀI THỰC HÀNH SỐ 6**

1. **Thông tin chung:**

* Mã số bài tập : HW3 - LTHĐT
* Hình thức nộp bài : Nộp qua Moodle môn học
* Thời hạn nộp bài : … / … / ……
* Nội dung : Chương 6. Lập trình khái quát

**Chuẩn đầu ra cần đạt:**

|  |  |
| --- | --- |
| L.O.3 | Lập trình khái quát bằng phương pháp dùng template |
| L.O.8 | Làm bài tập và nộp bài đúng quy định. |

**BÀI TẬP CƠ BẢN**

1. Xây dựng generic classes chứa các xử lý trên mảng một chiều sau:
   1. Nhập các phần tử cho mảng một chiều
   2. Duyệt tuần tự các phần tử mảng
   3. In các phần tử mảng ra màn hình
   4. Thêm một phần tử vạo cuối mảng
   5. Xóa một phần tử tại vị trí index
   6. Xóa toàn bộ các phần tử của mảng
   7. Cập nhật giá trị của phần tử tại vị trí index thành giá trị mới
   8. Cập nhật tất cả các phần tử tro.ng mảng có giá trị value1 thành value2
   9. Sắp xếp mảng theo thứ tự tăng dần hoặc giảm dần
   10. Viết ứng sử dụng generic class vừa tạo ở trên với các kiểu số thực, nguyên, phân số, chuỗi, object.
2. Xây dựng generic class chứa mảng dữ liệu gồm hai trường key và value có kiểu dữ liệu tùy ý, các methods sau:
   1. toán tử = = thực hiện so sánh hai đối tượng biết chúng bằng nhau khi hai trường value bằng nhau
   2. Phương thức Equals() trả về true khi hai đối tượng giống cả key và valuephương thức truy cập mảng dữ liệu bằng index và
   3. Phương thức sao chép mảng dữ liệu cho một mảng khác tương ứng.

**BÀI TẬP NÂNG CAO**

1. Xây dựng khuôn hình lớp mô tả lớp danh sách các phần tử (List). Danh sách này dùng để tạo ra các đối tượng như: Danh sách số nguyên, số thực, chuỗi,...

Thuộc tính:

* Mảng phần tử (arr)
* Số phần tử tối đa (MAXSIZE)
* Số lượng phần tử thực tế (n)

Phương thức:

* addList(): Thêm một phần tử vào danh sách.
* display(): Xuất danh sách
* getSize (): trả về kích thước của danh sách
* delete(): Xóa một phần tử trong danh sách

1. Xây dựng khuôn hình lớp Stack

Thuộc tính:

* Mảng phần tử (arr)
* Số phần tử tối đa (MAXSIZE)
* Số lượng phần tử thực tế (n)

Phương thức:

* push(): Thêm một phần tử vào danh sách.
* display(): Xuất danh sách
* getSize (): trả về kích thước của danh sách
* pop(): Xóa một phần tử trong danh sách
* isEmpty(); // kiểm tra stack rỗng hay không
* isFull(); // kiểm tra stack đầy hay không
* top(); // xem phần tử trên đỉnh stack

-----Hết-----

PHỤ LỤC

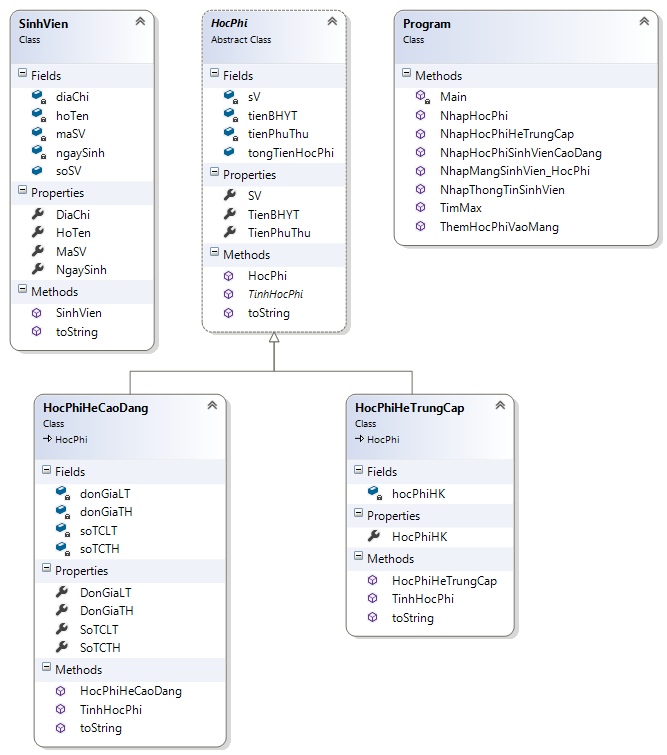
* XâydựngỨng dụng Quản lý học phí của sinh viên hệ niên chế và hệ tín chỉ trong một trường Cao Đẳng.

***Gợi ý****: Hệ thống cho phép ghi lại các thông tin cá nhân của sinh viên. Sinh viên hệ trung cấp học theo hệ niên chế và Sinh viên Cao đẳng học theo hệ tính chỉ.Học phí từng học kỳ được tính như sau:*

*Hoc phí theo hệ niên chế = học phí học kỳ + bảo hiểm y tế + phụ thu*

*Hoc phí theo hệ tính chỉ = số tín chỉ lý thuyết \* đơn giá môn lý thuyết + số tính chỉ thực hành \* đơn giá môn thực hành + bảo hiểm y tế + phụ thu*

**Mô hình lớp tham khảo:**

****

**Code cài đặt tham khảo:**

* **Cài đặt lớp SinhVien:**

|  |
| --- |
| **File: SinhVien.cs** |
| **/\* File: SinhVien.cs**  **\*/**  **using System;**  **namespace QLSinhVien2**  **{**  **public class SinhVien**  **{**  **//fields**  **private string maSV;**  **private string hoTen;**  **private string diaChi;**  **private DateTime ngaySinh;**  **public static int soSV = 0;**  **//Properites**  **public string MaSV**  **{**  **get**  **{**  **return maSV;**  **}**  **private set**  **{**  **maSV = value;**  **}**  **}**  **public string HoTen**  **{**  **get**  **{**  **return hoTen;**  **}**  **private set**  **{**  **hoTen = value;**  **}**  **}**  **public string DiaChi**  **{**  **get**  **{**  **return diaChi;**  **}**  **set**  **{**  **diaChi = value;**  **}**  **}**  **public DateTime NgaySinh**  **{**  **get**  **{**  **return ngaySinh;**  **}**  **set**  **{**  **ngaySinh = value;**  **}**  **}**  **//constructors**  **public SinhVien(string maSV, string hoTen, string diaChi, DateTime ngaySinh)**  **{**  **this.MaSV = maSV;**  **this.HoTen = hoTen;**  **this.DiaChi = diaChi;**  **this.NgaySinh = ngaySinh;**  **soSV++;**  **}**  **//method**  **public string toString()**  **{**  **string s = $"{maSV} - {hoTen} - {ngaySinh}";**  **return s;**  **}**  **}**  **}** |

* **Cài đặt lớp HocPhi:**

|  |
| --- |
| **File: HocPhi.cs** |
| **/\* File: HocPhi.cs**  **\*/**  **using System;**  **namespace QLSinhVien2**  **{**  **public abstract class HocPhi**  **{**  **//Fields**  **private SinhVien sV;**  **private long tienBHYT; //Tiền thu bảo hiểm y tế**  **private long tienPhuThu; // Tiền phụ thu**  **//Properties**  **public SinhVien SV**  **{**  **get**  **{**  **return sV;**  **}**  **set**  **{**  **sV = value;**  **}**  **}**  **public long TienBHYT**  **{**  **get**  **{**  **return tienBHYT;**  **}**  **set**  **{**  **tienBHYT = value;**  **}**  **}**  **public long TienPhuThu**  **{**  **get**  **{**  **return tienPhuThu;**  **}**  **set**  **{**  **tienPhuThu = value;**  **}**  **}**  **//Construcotrs**  **public HocPhi(SinhVien sV, long tienBHYT, long tienPhuThu)**  **{**  **this.sV = sV;**  **this.tienBHYT = tienBHYT;**  **this.tienPhuThu = tienPhuThu;**  **}**  **//Methods**  **abstract public long TinhHocPhi();**  **virtual public string toString()**  **{**  **string s = $"{sV.toString()} - {tienBHYT} - {tienPhuThu}";**  **return s;**  **}**  **}**  **}** |

* **Cài đặt các lớp kế thừa:**

|  |
| --- |
| **File:HocPhiHeCaoDang.cs** |
| **/\* File: HocPhiHeCaoDang.cs**  **\* Date:**  **\* Name**  **\*/**  **using System;**  **namespace QLSinhVien2**  **{**  **public class HocPhiHeCaoDang: HocPhi**  **{**  **//fields**  **int soTCLT; //số tín chỉ lý thuyết**  **int soTCTH; //Số tín chỉ thực hành**  **long donGiaLT;//Đơn giá môn lý thuyết**  **long donGiaTH; //Đơn giá môn thực h**  **public HocPhiHeCaoDang(SinhVien sV, long tienBHYT, long tienPhuThu, int soTCLT, int soTCTH, long donGiaLT, long donGiaTH) : base(sV, tienBHYT, tienPhuThu)**  **{**  **this.soTCLT = soTCLT;**  **this.soTCTH = soTCTH;**  **this.donGiaTH = donGiaTH;**  **this.donGiaLT = donGiaLT;**  **}**  **public int SoTCLT**  **{**  **get**  **{**  **return soTCLT;**  **}**  **set**  **{**  **soTCLT = value;**  **}**  **}**  **public int SoTCTH**  **{**  **get**  **{**  **return soTCTH;**  **}**  **set**  **{**  **soTCTH = value;**  **}**  **}**  **public long DonGiaLT**  **{**  **get**  **{**  **return donGiaLT;**  **}**  **set**  **{**  **donGiaLT = value;**  **}**  **}**  **public long DonGiaTH**  **{**  **get**  **{**  **return donGiaTH;**  **}**  **set**  **{**  **donGiaTH = value;**  **}**  **}**  **public override long TinhHocPhi()**  **{**  **long hp = 0;**  **hp = soTCLT \* DonGiaLT + SoTCTH \* donGiaTH + TienBHYT + TienPhuThu;**  **return hp;**  **}**  **public override string toString()**  **{**  **string s = $"{SV.toString()}\t {TinhHocPhi()}";**  **return s;**  **}**  **}**  **}** |

|  |
| --- |
| **File: HocPhiHeTrungCap.cs** |
| /\* File: HocPhiHeTrungCap.cs  \* Date:  \* Name:  \*/  using System;  namespace QLSinhVien2  {  public class HocPhiHeTrungCap : HocPhi  {  //Fileds  long hocPhiHK;  //Properties  public long HocPhiHK  {  get  {  return hocPhiHK;  }  set  {  hocPhiHK = value;  }  }  //Constructors  public HocPhiHeTrungCap(SinhVien sv, long tienBHYT, long tienPhuThu, long hocPhiHK): base(sv, tienBHYT, tienBHYT)  {  this.hocPhiHK = hocPhiHK;  }  //Methods  public override long TinhHocPhi()  {  long hp = hocPhiHK + TienBHYT + TienPhuThu;  return hp;  }  public override string toString()  {  string s = $"{SV.toString()} - {TinhHocPhi()}";  return s;  }  }  } |

* **Chương trình thử nghiệm:**

|  |
| --- |
| **File: Program.cs** |
| /\* File: Program.cs: Chương trình quản lý Sinh viên - Học phí  \* Date:  \* Name:  \*/  using System;  namespace QLSinhVien2  {  class Program  {  static void Main(string[] args)  {  //static fields  SinhVien.soSV = 0;  //Tạo mảng học phí có thể chưa cả hai loại Cao Đẳng và Trung cấp  HocPhi[] arrHp = new HocPhi[0];  NhapMangSinhVien\_HocPhi(ref arrHp);  //In mảng học phí  foreach (var item in arrHp)  {  Console.WriteLine(item.toString());  }  //Thống kê  Console.Write("Hoc phi lon nhat la:");  Console.WriteLine(TimMax(arrHp));  //sử dụng biến static  Console.WriteLine("So SV da tao: " + SinhVien.soSV);    }  //Nhập Mảng SV và Học phí  public static void NhapMangSinhVien\_HocPhi(ref HocPhi[] arrHp)  {  int soSV = 0;  Console.WriteLine("Nhap thong tin Sinh Vien va hoc phi: ");  Console.Write("Nhap so luong sinh vien: ");  int.TryParse(Console.ReadLine(), out soSV);  for (int i = 0; i < soSV; i++)  {  //\*\*\*Nhập thông tin Sinh viên  SinhVien sv = NhapThongTinSinhVien();  char chon = '\0';  do  {  Console.Write("Chon 1-SV Cao Dang ; 2 - SV Trung Cap. Ban chon: ");  char.TryParse(Console.ReadLine(), out chon);  } while (chon != '1' && chon != '2');    HocPhi hp;  switch(chon)  {  case '1':  //Nhập thông tin học phí cho sv1  hp = NhapHocPhiSinhVienCaoDang(sv);  //Thêm vào mảng học phí  ThemHocPhiVaoMang(ref arrHp, hp);  break;  case '2':  //Nhập thông tin học phí cho sv1  hp = NhapHocPhiHeTrungCap(sv);  //Thêm vào mảng học phí  ThemHocPhiVaoMang(ref arrHp, hp);  break;  }  }  }  //Tim học phí lớn nhất trong mang HocPhiHeCaoDang  public static long TimMax(HocPhi[] arrHp)  {  long max = arrHp[0].TinhHocPhi();  foreach (var item in arrHp)  {  long t = item.TinhHocPhi();  if (max < t)  {  max = t;  }  }  return max;  }  //Thêm 1 phần tử vào mảng học phí  public static HocPhi[] ThemHocPhiVaoMang(ref HocPhi[] arrHp, HocPhi hp )  {  //tăng kích thước mảng  Array.Resize(ref arrHp, arrHp.Length + 1);  //gán vào cuối mảng  arrHp[arrHp.Length -1] = hp;  return arrHp;  }  //Nhập hoc phí hệ trung cấp  public static HocPhiHeTrungCap NhapHocPhiHeTrungCap(SinhVien sv)  {    long hocPhiHK = 0;  long tienBHYT = 0;  long tienPhuThu = 0;  Console.WriteLine();  Console.WriteLine("\*\*\*\*Nhap hoc phi Trung Cap\*\*\*\*\*");  NhapHocPhi(out tienBHYT, out tienPhuThu);  Console.Write("Nhap hoc phi hoc ky: ");  long.TryParse(Console.ReadLine(), out hocPhiHK);  HocPhiHeTrungCap obj = new HocPhiHeTrungCap(sv, tienBHYT, tienPhuThu, hocPhiHK);  return obj;  }  //Nhập học phí hệ cao đẳng  public static HocPhiHeCaoDang NhapHocPhiSinhVienCaoDang(SinhVien sv)  {  int soTCLT = 0;  int soTCTH = 0;  long donGiaLT = 0;  long donGiaTH = 0;  long tienBHYT = 0;  long tienPhuThu = 0;  Console.WriteLine();  Console.WriteLine("\*\*\*\* Nhap hoc phi Cao Dang \*\*\*\*\*");  NhapHocPhi(out tienBHYT, out tienPhuThu);  Console.Write("Nhap so tin chi lý thuyet: ");  int.TryParse(Console.ReadLine(), out soTCLT);  Console.Write("Nhap so tin chi thuc hanh : ");  int.TryParse(Console.ReadLine(), out soTCTH);  Console.Write("Nhap don gia lý thuyet: ");  long.TryParse(Console.ReadLine(), out donGiaLT);  Console.Write("Nhap don gia thuc hanh: ");  long.TryParse(Console.ReadLine(), out donGiaTH);  HocPhiHeCaoDang obj = new HocPhiHeCaoDang(sv, tienBHYT, tienPhuThu, soTCLT, soTCTH, donGiaLT, donGiaTH);  return obj;  }  //Nhap hoc phi  public static void NhapHocPhi(out long tienBHYT, out long tienPhuThu)  {  Console.WriteLine();  Console.Write("Nhap tien BHYT:");  long.TryParse(Console.ReadLine(), out tienBHYT);  Console.Write("Nhap tien phu thu:");  long.TryParse(Console.ReadLine(), out tienPhuThu);  }  //Nhập thông tin Sinh Viên  public static SinhVien NhapThongTinSinhVien()  {  string maSV = "";  string hoTen = "";  string diaChi = "";  DateTime ngaySinh;  Console.WriteLine("\*\*\*\*Nhap Thong tin Sinh vien\*\*\*\*\*");  Console.Write("Nhap ma SV: ");  maSV = Console.ReadLine();  Console.Write("Nhap ho ten SV: ");  hoTen = Console.ReadLine();  Console.Write("Nhap dia chi SV: ");  diaChi = Console.ReadLine();  Console.Write("Nhap ngay sinh SV: ");  DateTime.TryParse(Console.ReadLine(), out ngaySinh);  SinhVien sv = new SinhVien(maSV, hoTen, diaChi, ngaySinh);  return sv;  }  }  } |

**Kết quả:**

|  |
| --- |
| **Nhap thong tin Sinh Vien va hoc phi:**  **Nhap so luong sinh vien: 2**  **\*\*\*\*Nhap Thong tin Sinh vien\*\*\*\*\***  **Nhap ma SV: SV001**  **Nhap ho ten SV: Tran Nham**  **Nhap dia chi SV: Tien Giang**  **Nhap ngay sinh SV: 1/1/2003**  **Chon 1-SV Cao Dang ; 2 - SV Trung Cap. Ban chon: 1**  **\*\*\*\* Nhap hoc phi Cao Dang \*\*\*\*\***  **Nhap tien BHYT:560000**  **Nhap tien phu thu:50000**  **Nhap so tin chi ly thuyet: 10**  **Nhap so tin chi thuc hanh : 15**  **Nhap don gia ly thuyet: 30000**  **Nhap don gia thuc hanh: 35000**  **\*\*\*\*Nhap Thong tin Sinh vien\*\*\*\*\***  **Nhap ma SV: SV002**  **Nhap ho ten SV: Xuan Tai**  **Nhap dia chi SV: Tp.HCM**  **Nhap ngay sinh SV: 25/12/2000**  **Chon 1-SV Cao Dang ; 2 - SV Trung Cap. Ban chon: 2**  **\*\*\*\*Nhap hoc phi Trung Cap\*\*\*\*\***  **Nhap tien BHYT:580000**  **Nhap tien phu thu:20000**  **Nhap hoc phi hoc ky: 2500000**  **SV001 - Tran Nham - 01/01/2003 12:00:00 AM 1435000**  **SV002 - Xuan Tai - 25/12/2000 12:00:00 AM - 3660000**  **Hoc phi lon nhat la:3660000**  **So SV da tao: 2**  **Press any key to continue . . .** |

**TÀI LIỆU THAM KHẢO**

[1] Keogh and Mario Giannini, OOP Demystified: A Self-Teaching Guide, McGraw-Hill/Osborne, 2004

[2] Richard L.Halterman , Fundamentals of C++ program, 2017

[3] Trần Đan Thư, Lập trình hướng đối tượng, Nhà xuất bản khoa học và kỹ thuật, 2010, 370.

1. Pool là một vùng nhớ đặc biệt nằm trong vùng nhớ Heap (Heap memory), dùng để lưu trữ các biến được tạo ra từ string. Pool giúp tối ưu lưu trữ và dùng vùng nhớ khi khái báo String từ đó giúp hạn chế tình trạng tràn bộ nhớ Heap. [↑](#footnote-ref-1)