BÀI TẬP VỀ NHÀ

CÁC MÔ HÌNH LẬP TRÌNH

Phạm Văn Thông

Ngày 15 tháng 1 năm 2015

1 Các mô hình lập trình

Mô hình lập trình là một khuôn mẫu cho một lớp ngôn ngữ lập trình có chung những đặc điểm cơ bản. Do môi ngôn ngữ lập trình có những phương thức riêng của chúng để thực hiện những quy chuẩn của toán học và logic đặc biệt là *hàm* (function), *biến* (varible), *phương thức* (method), *đối tượng* (object) nên bên cạnh sự đa dạng của ngôn ngữ lập trình cũng dần có sự ra đời của nhiều mô hình lập trình trong đó phổ biến nhất là *lập trình hướng đối tượng* (oriented programing), *lập trình thủ tục* procedural programming) và lập trình cấu trúc (structured programming). Mỗi mô hình lập trình, lại có những điểm đặc trưng riêng về cách sử dụng và tính trừu tượng hóa đối với các quá trình xác định. Các ngon ngử lập trình khác nhau thì hỗ trợ mô hình đặc thù khác nhau (như Java hỗ trợ cho lập trình hướng đối tượng), tuy nhiên, cũng có ngôn ngữ lập trình hỗ trợ nhiều mô hình lập trình (Python và Common Lisp) nên quan hệ giữa mô hình lập trình với ngôn ngữ lập trình có thể phức tạp.

Sau đây giới thiệu một số mô hình lập trình thông dụng.

1.1 Mô hình lập trình mệnh lệnh (Imperative paradigm)

Với mô hình này, ý tưởng cơ bản là cách lệnh gây ảnh hưởng đáng kể đến trạng thái của chương trình. Mỗi imperative program bao gồm:

Declarative statements là các lệnh khai báo, chúng cung cấp các tên cho biến. Các biến này có thể thay đổi giá trị trong quá trình thực hiện chương trình.

Assigment statements là các lệnh gán, thực hiện gán giá trị mới cho biến.

Program flow control statements là các lệnh điều khiển cấu trúc chương trình, chúng xác định trình tự thực hiện các lệnh trong chương trình.

Module Chia chương trình thành các chương trình con bao gồm Functions (hàm) và Procedures (thủ tục).

Các đặc trưng chính của mô hình này là

- *Về mặt nguyên lý và ý tưởng*: Công nghệ phần cứng digital và ý tưởng của Von Neumann.
- *Các bước tính toán được thực hiện với mục đích kiểm soát cấu trúc điều khiển*: chúng ta gọi là các *command* (mệnh lệnh).
- *Các ngôn ngữ đại diện*: FORTRAN, ALGOL, Pascal, Basic, C...
- *Các thủ tục và hàm chính là hình ảnh về sự trừu tượng*: che dấu các lệnh trong chương trình con, có thể coi chương trình con đó là một lệnh.
- *Imperative programming* còn được gọi là lập trình thủ tục.

1.2 Mô hình lập trình hàm (Functional pdaradigm)

Functional Programming (lập trình hàm) là thể loại lâu đời nhất trong ba paradigm lập trình chính. Ngôn ngữ Functional Programming đầu tiên là IFP, được phát minh vào năm 1955, một năm trước khi có FORTRAN, sau đó, LISP ra đời năm 1958, một năm trước khi có COBOL. cả FORTRAN và CObol đều là những ngôn ngữ imperative (hay còn gọi là procedural). Chúng đặc biệt thành công khi ứng dụng vào tính toán khoa học và nghiệp vụ, và trở thành paradigm thống trị trong suốt 30 năm đầu của kỉ nguyên thông tin. Vào những năm 1970, *Object-Oriented paradigm* bắt đầu phát triển, cùng với sự trưởng thành của các ngôn ngữ lập trình hướng đối tượng, Object-Oriented (hướng đối tượng) trở thành paradigm được sử dụng phổ biến nhất cho đến ngày hôm nay.

Từ những năm 1950, mặc dù vẫn phát triển mạnh mẽ và liên tục với các đai diện như SML, OCaml, APL và Clean thì FP vẫn chỉ được sử dụng cho mục đích học thuật và nghiên cứu là chủ yếu. Imperative đạt được những thành công nhờ vào khả năng mô hình hóa các bài toán phức tạp. Còn Functional Paradigm có khả năng gì, tại sao dù chúng ra đời sớm nhưng lại không mang lại thành công như mong đợi? Functional Paradigm xem chương trình là một tập hợp các hàm nhận vào đối số và trả về giá trị. Không giống như Imperative, Functional Paradigm không tạo ra hiệu ứng phụ và sử dụng đệ quy thay cho vòng lặp. Hàm trong Functional Paradigm rất giống với hàm trong toán học vì nó không thay đổi trạng thái của chương trình.

- Một đại lượng khi được gán giá trị thì không bao giờ thay đổi giá trị đó.
- Hàm không sửa đổi giá trị của đối số được truyền vào, và giá trị do hàm trả về hoàn toàn là một giá trị mới.

Về mặt kĩ thuật, cài đặt bên dưới, khi giá trị được gán vào một vùng nhớ thì được đánh dấu là đã sử dụng và không ghi đè lên nữa. Để tạo ra kết quả trả về, các hàm sao chép giá trị rồi chỉnh sửa trên các bản sao đó, không làm ảnh hưởng đến giá trị ban đầu, rồi trả về bản sao đã được chỉnh sửa. Các đại lượng không còn được hàm nào tham chiếu sẽ tự động bị hủy để giải phóng bộ nhớ (đây chính là ý tưởng của bộ thu gom rác trong Java và .NET).

Cơ sở toán học cho Functional Paradigm rất tường minh, cung cấp các giải pháp logic và ngắn gọn cho các vấn đề tính toán. Tuy nhiên do nó không linh hoạt thay đổi trạng thái và chuyên sử dụng đệ quy khiến người ta ưa chuộng các paradigm hác hơn để xử lý các thao tác tính toán thông dụng.

Các đặc trưng của mô hình lập trình hàm

- Nguyên lý và ý tưởng từ toán học và lý thuyết về hàm.
- Các giá trị đã được tạo ra là không thể biến đổi.
- Trừu tượng hóa một biểu thức thành một hàm và ngược lại, một hàm có thể được tính toán như một biểu thức.
- Hàm là lớp giá trị đầu tiên.
- Hàm là những giá trị hoàn chỉnh, tương tự như số, danh sách, ...
- Thích hợp với những tính toán theo yêu cầu.
- Mở ra những khả năng mới.

1.3 Mô hình lập trình logic (Logic paradigm)

Mô hình lập trình logic hoàn toàn khác với các mô hình còn lại. Mô hình này đặc biệt phù hợp với những lĩnh vực liên quan đến việc rút ra những kiến thức từ những sự kiện và quan hệ cơ bản - lĩnh vực trí tuệ nhân tạo. Có vẻ như mô hình này không gắn với những lĩnh vực tính toán nói chung.

Khi mô tả bài toán dưới dạng logic vị từ, ta có thể yêu cầu hệ thống tìm kiếm các lời giải liên quan đến các khai báo đó. Bài toán cần giải được xem là *mục tiêu* mà hệ thống phải chứng mình trên cơ sở các tri thức đã được khai báo. Như thế, toàn bộ các kí hiệu của ngôn ngữ lập trình suy về một công thức đặc biệt:

- Phát sinh từ một yêu cầu.
- Nhằm chứng minh một mục tiêu. Để trả lời cho câu hỏi đó xem nó như là một *đích* và cố chứng minh "đích" đó bằng cách tạo ra những suy diễn trên cơ sở các tri thức đã khai báo.

Các đặc trưng của mô hình lập trình logic

- *Về nguyên tắc và ý tưởng*: Tự động kiểm chứng trong trí tuệ nhân tạo.
- *Dựa trên các chân lý* : Tiên đề *axioms*, các quy luật suy diễn *inference rules*, các truy vấn *queries*.
- Chương trình thực hiện từ việc tìm kiếm có hệ thống trong một tập các sự kiện, sử dụng một tập các luật để đưa ra quyết định.

Một ngôn ngữ logic có thể được dùng trong giai đoạn đặc tả yêu cầu của quy trình xây dựng phần mềm. Hơn thế nữa, logic vị từ cho phép biểu diễn hầu hết các khái niệm và các định lý trong các bộ môn khoa học.

1.4 Mô hình lập trình hướng đối tượng (Object-oriented paradigm)

Mô hình lập trình hướng đối tượng thu hút được sự quan tâm và nổi tiếng từ khoảng 20 năm nay. Lý do là khả năng hỗ trợ mạnh của tính bao gói và gộp nhóm logic của các khía cạnh lập trình. Những thuộc tính này rất quan trọng khi mà kích cỡ các chương trình ngày càng lớn.

Nguyên nhân cơ bản và sâu sắc dẫn đến thành công của mô hình này chính là:

- *Cơ sở lý thuyết đỉnh cao của mô hình*: Một chương trình hướng đối tượng được xây dựng với những quan niệm, tư tưởng làm cơ sở, điều đó rất quan trọng và theo cách đó, tất cả các kĩ thuật cần thiết cho lập trình trở thành thứ yếu.
- Gửi thông điệp giữa các *objects* để mô phỏng sự tiến triển theo thời gian của hàng loạt các hiện tượng trong thế giới thực.

Các đặc trưng của mô hình lập trình hướng đối tượng

- *Nguyên lý và ý tưởng*: Lý thuyết và các mô hình tương tác trong thế giới thực.
- Dữ liệu cũng như các thao tác trên dữ liệu được đóng gói trong *objects*.
- Cơ chế che dấu thông tin được sử dụng để tránh những tác động từ bên ngoài object.
- Các *objects* tương tác với nhau qua việc truyền thông điệp, đó là phép ẩn dụ cho việc thực hiện các thao tác trên một object.
- Trong phần lớn các ngôn ngữ lập trình hướng đối tượng, *objects* được nhóm lại trong các *classes*.
  - *Objects* trong *classes* có chong các thuộc tính, cho phép lập trình trên lớp, thay vì lập trình trên từng đối tượng riêng lẻ.
  - *Classes* đại diện cho *concepts* còn *objects* đại diện cho hiện tượng.
  - *Classes* được tổ chức trong cây phả hệ có kế thừa.
  - Tính kế thừa cho phép mở rộng hay chuyên biệt hóa lớp.

1.5 Mô hình lập trình trực quan (Visual paradigm)

Trong xu hướng phát triển mạnh mẽ hiện nay của tin học, số người sử dụng máy tính cũng tăng lên rất nhanh và máy tính được sử dụng trong hầu hết các lĩnh vực của đời sống nên đòi hỏi các ngôn gnuwx lập trình cũng phải đơn giản, dễ sử dụng và mang tính đại chúng cao. Chính vì vậy, phương pháp lập trình trực quan ra đời.

Đặc điểm của các ngôn ngữ lập trình trực quan là dễ sử dụng, triển khai các ứng dụng một cách nhanh chóng.

Đặc trưng của mô hình lập trình trực quan

- Cho phép xây dựng chương trình theo hướng sự kiện - *Event Driven Programming*, nghĩa là một chương trình ứng dụng được viết theo kiểu này đáp ứng được tình huống xảy ra lúc thực hiện chương trình. Tình huống bao gồm người sử dụng ấn một phím tương ứng, chọn lựa một nút lệnh hoặc gọi một lệnh từ một ứng dụng khác chạy song song cùng lúc.
- Người lập trình trực tiếp tạo ra các khung giao diện (interface), ứng dụng thông qua các thao tác trên màn hình dựa vào các *đối tượng* (object) như *hộp hội thoại* (dialog box) hoặc *nút điều khiển* (control button), những đối tượng này mang các *thuộc tính* (properties) riêng biệt như màu sắc, font chữ ... mà ta chỉ cần chọn lựa trên một danh sách cho sẵn.
- Khi dùng các ngôn ngữ lập trình trực quan, ta rất ít khi phải tự viết các lệnh, tổ chức chương trình ... một các rắc rối mà chỉ cần khai báo việc gì cần làm khi một tình huống xuất hiện.
- Máy tính sẽ dựa vào phần thiết kế và khai báo của lập trình viên để tự động tạo lập chương trình.

1.6 Lập trình song song (Parallel programming)

Tính toán song song (Parallel Computing) là việc chia một công việc ra thành các công việc nhỏ và cho các công việc này thực hiện đồng thời với nhau bởi các hệ thống có nhiều bộ vi xử lý (multiprocessor) hay bộ vi xử lý đa nhân (multicore) nhằm giảm thời gian thực hiện công việc đó xuống. Việc lập trình để tách ra các công việc nhỏ và sắp xếp để xử lý song song thì được gọi là lập trình song song.

Việc lập trình song song không phải lúc nào cũng mang lại hiệu suất coa cho chương trình. Đối với các công việc chỉ tốn ít hiệu suất thì lập trình song song sẽ không mang lại kết qua như mong đợi do khoảng thời gian để khởi tạo và quản lý các tiến trình song song gây nên.

Trong quá trình lập trình song song, lập trình viên khó tránh khỏi việc các tiến trình cùng dùng chung một dữ liệu, điều này sẽ gây ra xung đột dữ liệu hoặc làm dữ liệu bị sai lệch, do đó, lập trình viên cần phải chú ý và áp dụng các kĩ thuật lập trình để giải quyết vấn đề này.

Bộ vi xử lý có nhiều nhân sẽ tăng tốc chương trình song song, tuy nhiên không có nhĩa là nó sẽ tăng lên 100% trên một nhân được thêm vào. Thậm chí chương trình song song không hề tăng hiệu suất lên trong một số trường hợp. Vì vậy, lập trình viên phải biết quyết định khi nào sử dụng lập trình song song bằng cách sử dụng các công cụ đo lường để xác định được tốc độ thực thi của chương trình.

Lập trình song song là một công việc rất phức tạp so với lập trình tuần tự thông thường, người phát triển phải thực hiện một quá trình *song song hóa*, biến đổi các chương trình tuần tự thành chương trình song song có khả năng tận dụng tối đa sức mạnh của hệ thống.

Quá trình song song hóa gồm ba giai đoạn chính:

- Phân chia chương trình thành các công việc con (Sub-task decomposition).
- Phân tích sự phụ thuộc (Dependence analyse).
- Định thời các công việc (Task scheduling).

Nói một cách dễ hiểu thì lập trình song song là lập trình để chạy được đa luồng với các dòng CPU đa nhân hiện nay.

1.7 Lập trình tương tranh (Concurrent programming)

Lập trình tương tranh là một trong những kĩ thuật quan trọng và cũng là một thách thức. Một mặt nó đem lại khả năng sử dụng hiệu quả cho những hệ thống song song và phân tán. Mặt khác những rủi ro trong việc tương tác giữa các tiến trình thực hiện đồng thời có thể gây ra những lỗi phần mềm không dễ dàng tìm ra.

Trong lập trình tương tranh, một số dòng lệnh có thể được thực hiện đồng thời. Mỗi dòng lệnh là một chương trình tuần tự, ngoại trừ một thực tế là các dòng lệnh có thể giao tiếp và gây trở ngại lẫn nhau. Mỗi dòng lệnh là một *luồng* (thread), các chương trình tuần tự còn gọi là các chương trình đơn luồng, còn chương trình viết theo phương pháp tương tranh là chương trình đa luồng.

Đặc điểm của lập trình tương tranh

- Người dùng có thể tương tác với các ứng dụng khi tiến trình đang thực thi.
- Những tiến trình chạy thời gian dài không làm trì hoãn các tiến trình ngắn.
- Các chương trình phức tạp được thực hiện tốt hơn trên các bộ vi xử lý đa luồng.
- Những tiến trình đòi hỏi điều kiện tiên quyết có thể tạm dừng và đợi đến khi đáp ứng được để tiếp tục xử lý.

1.8 Lập trình phân tán (Distributed Programming)

Lập trình phân tán là một dạng của lập trình song song. Lập trình song song tạo ra mối liên hệ giữa máy tính và các đơn vị tính toán, khiến chúng hoạt động đồng thời đối với một vấn đề cụ thể (dự báo thời tiết chẳng hạn). Các đơn vị tính toán có thể được đặt rất gần nhau hoặc tách rời nhau. Khi các đơn vị tính toán được đặt tách rời, người ta gọi đó là lập trình phân tán. Với mô hình lập trình này, các đơn vị tính toán thường rất khác nhau, cũng như sự khác nhau giữa các hệ điều hành và thiết lập mạng máy tính. Những yếu tố đó khiến cho việc lập trình các tính toán của máy tính trở nên tương đối phức tạp và khó khăn.

Lập trình mạng phân tán thường có hai khái niệm chính là peer to peer và client-server. Peer to peer là lập trình ngang hàng giữa hai máy tính, còn lập trình client-server là lập trình cho phép n máy client kết nối tới m máy server - đây cũng là mô hình chúng ta gặp nhiều trong thực tế.

Có thể chia lập trình mạng thành 3 cấp độ, căn cứ theo mức độ thân thiện với developer và khả năng triển khai:

- Mức thấp nhất - lập trình sử dụng socket: Có thể truyền từng byte, từng stream lên trên mạng, ứng dụng này thường gặp trong các ứng dụng mạng cỡ nhỏ.
- Mức cao hơn - Lập trình sử dụng: Triển khai ứng dụng theo mô hình dịch vụ (service) - quản lý mạng theo mô hình kết nối, tạo tham chiếu client trên server và trả về, gọi hàm và truyền dữ liệu thông qua proxy của đối tượng trả về từ server. VD: RMI trên java hay Remoting trên .NET ... Mô hình này thường gặp trong các ứng dụng mạng cỡ trung bình và lớn, đòi hỏi performane cao. VD: Các ứng dụng mạng đòi hỏi kết nối nhiều, liên tục và sử dụng diện rộng như các dịch vụ chat, game online ... Ưu điểm chính của mô hình này là hiệu năng cao, bảo mật nhưng nhược điểm chính là tính đa nền (multiplatform) và tính khả chuyển chưa cao (client và server phải cùng một công nghệ phát triển).

  Mức cao nhất là triển khai ứng dụng theo mô hình triển khai trên web, điển hình nhất ta có thể thấy là các web application và web service, mới nhất giờ có công nghệ WCF của Microsoft. Mô hình này cho phép triển khai trên diện rộng, phục vụ lượng khách hàng lớn, nhưng cũng có nhược điểm là hiệu năng chưa cao, bù lại nó có khả năng mềm dẻo cao. Chúng ta thường thấy những ứng dụng loại này trong các ứng dụng được cung cấp từ một nhà cung cấp nào đó. VD: các số liệu chứng khoán, thời tiết ...

1.9 Lập trình cực hạn (Extreme programming)

Extreme Programming là mô hình nổi tiếng nhất của phương thức Agile (Agile Method). Nó được hình thành bởi một tập các *thực nghiệm* (practice) đơn giản nhưng phụ thuộc lẫn nhau. Những practice này làm việc với nhau để hình thành nên một dạng tổng quát hơn của các thành phần của nó.

Các thực nghiệm

- Khách hàng là thành viên của nhóm phát triển.
- Câu chuyện người dùng.
- Chu kì ngắn.
- Kế hoạch bước lặp.
- Kế hoạch phát hành.
- Các chương trình kiểm tra sự thỏa mãn yêu cầu của sản phẩm.
- Lập trình theo cặp.
- Phát triển theo định hướng test.
- Sở hữu tập thể.
- Tích hợp liên tục.
- Chạy bền.

2 Giới thiệu về mô hình lập trình hướng đối tượng

Lập trình hướng đối tượng (Object Oriented Programming - gọi tắt là OOP) hay chi tiết hơn là *Lập trình định hướng đối tượng* chính là phương pháp lập trình lấy đối tượng làm nền tảng để xây dựng thuật giải, xây dựng chương trình. Thực chất đây không phairl là một phương pháp mới mà là một cách nhin mới (paradigm) trong việc lập trình. Để phân biệt, với phương pháp lập trình theo kiểu cấu trúc mà chúng ta quen thuộc trước đây, hay còn gọi là phương pháp lập trình hướng thủ tục (Produre-Oriented Programming), người lập trình phân tích một nhiệm vụ lớn thành nhiều công việc nhỏ hơn, sau đó dần dần chi tiết, cụ thể hóa để được các vấn đề đơn giản, để tìm ra cách giải quyết vấn đề này dưới dạng những thuật giải cụ thể rõ ràng qua dố dễ dàng minh họa bằng nôn ngữ giải thuật. Cách thức thiết kế như vậy chúng ta gọi là nguyên lý lập trình *top-down*, để thể hiện quá trình suy diễn từ cái chung cho đến cái cụ thể.

Các chương trình con là những chức năng độc lập, sự ghép nối chúng lại với nhau cho chúng ta một hệ thống chương trình để giải quyết vấn đề đặt ra. Chính vì vậy, cách thức phân tích một hệ thống lấy chương trình con làm nền tảng, chương trình con đóng vai trò trung tâm của việc lập trình, được hiểu như phuong pháp lập trình hướng thủ tục. Tuy nhiên khi phân tích để thiết kế một hệ thống không nhất thiết phải luôn suy nghĩ theo hướng "*làm thế nào để giải quyết công việc*", chúng

ta có thể định hướng tư duy theo phong cách *"với một số đối tượng đã có, phải làm gì để giải quyết được công việc được đặt ra"* hoặc *làm cái gì với một số đối tượng đã có đó"*, từ đó cũng có thể giải quyết được những công việc cụ thể. Với phương pháp phân tích trong đó đối tượng đóng vai trò trung tâm của việc lập trình như vậy, người ta gọi là nguyên lý lập trình *bottom-up* (từ dưới lên).

Những đối tượng trong OOP là các kết hợp giữa mã và dữ liệu mà chúng được nhận nhu là một đơn vị duy nhất. Mỗi đối tượng có một tên riêng biệt và tất cả các tham chiếu đến đối tượng đó được tiến hành qua tên của nó, như vậy mỗi đối có khả năng nhận vào các thông báo, xử lý dữ liệu (bên trong của nó), và gửi ra hay trả lời đến các đối tượng khác hay đến môi trường.

2.1 Các tính chất cơ bản

Đối tượng - Object Các dữ liệu và chỉ thị được kết hợp vào một đơn vị đầy đủ tạo nên một đối tượng. Đơn vị này tương đương với một chương trình con và vì thế các đối tượng sẽ được chia thành hai bộ phận chính: phần các **phương thức** (*method*) và phần các **thuộc tính** (*attribute*). Trong thực tế, các **phương thức** của đối tượng là các hàm và các thuộc tính của nó là các biến, các tham số hay hằng nội tại của một đối tượng (hay nói cách khác là tập hợp các dữ liệu tạo thành thuộc tính của đối tượng). Các **phương thức** là phương tiện để sử dụng một đối tượng trong khi các **thuộc tính** sẽ mô tả đối tượng có những tính chất gì.

Các phương thức và các thuột tính thường gắn chặt với thực tế các đặc tính và sử dụng của một đối tượng.

Trong thực tế, các đối tượng thường được trừu tượng hóa qua việc định nghĩa của các lớp (*class*). Tập hợp các giá trị hiện có của các thuộc tính tạo nên **trạng thái** của một đối tượng. mỗi **phương thức** hay mỗi dữ liệu nội tại cùng với các tính chất được định nghĩa (bởi người lập trình) được xem là một **đặc tính** riêng của đối tượng. Nếu không có gì lầm lẫn thì tập hợp các đặc tính này gọi chung là đặc tính của đối tượng.

Lập trình hướng đối tượng là một phương pháp lập trình có các tính chất sau:

Tính trừu tượng - abstraction Đây là khả năng của chương trình bỏ qua hay không chú ý đến một số khía cạnh của thông tin mà nó đang trực tiếp làm việc lên, nghĩa là nó có khả năng tập trung vào những cốt lõi cần thiết. Mỗi đối tượng phục vụ như là một "động từ" có thể hoàn tất các công việc một cách nội bộ, báo cáo, thay đổi trạng thái của nó và liên leajc với các đối tượng khác mà không cần cho biết làm cách nào đối tượng được tiến hành các thao tác. Tính chất này thường được gọi là *sự trừu tượng của dữ liệu*. Tính trừu tượng còn thể hiện qua việc một đối tượng ban đầu có thể không có các biện pháp thi hành. Tính trừu tượng này thường được xác định trong các khái niệm gọi là *lớp trừu tượng* hay *lớp cơ sở trừu tượng*.

Tính đóng gói và che dấu thông tin - encapsulation and infomation hidding Tính chất này không cho phép người sử dụng sử dụng ác đối tượng thay đổi trạng thái nội tại của một đối tượng. Chỉ có các phương thức 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 mà. Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.

Tính đa hình - polymorphism Thể hiện thông qua việc gửi các **thông điệp** (*mesage*). Việc gửi các thông điệp này có thể so sánh như việc gọi các hàm 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. Người lập trình có thể định nghĩa một đặc tính (chẳng hạn thông qua tên của các phương thức) cho một loạt các đối tượng gần nhau nhưng khi thi hành thì cùng một tên gọi mà sự thi hành của mỗi đối tượng sẽ tự động xảy ra tương ứng theo đặc tính của từng đối tượng mà không sợ bị nhầm lẫn.

> Ví dụ khi định nghĩa hai đối tượng "hinh_vuong" và "hinh_tron" thì có một phương thức chung là "chu_vi". Khi gọi phương thức này thì nếu đối tượng là "hinh_vuong" nó sẽ tính theo công thức khác với khi đối tượng là "hinh_tron".

Tính kế thừa - inheritance Đặc tính này cho phép một đối tượng có thể có sẵn các đặc tính mà đối tượng khác đã có thông qua kế thừa. Điều này cho phép các đối tượng chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại. Tuy nhiên không phải ngôn ngử định hướng đối tượng nào cũng có tính chất này.

2.2 Một điểm mới về mẫu hình, về quan điểm

OOP vẫn còn là một đề tài của nhiều tranh cãi về định nghĩa chuẩn xác hay về các nguyên lý của nó.

Trong dạng tổng quát nhất ÔP được hiểu theo cách là một loại thực nghiệm viết chương trình văn bản được phân cấp ra thành nhiều *module*, mà mỗi module đóng vai trò như một lớp vỏ che đại diện cho mỗi kiểu dữ liệu.

Những khái niệm liên hệ đã được góp nhặt lại để tạo thành một khuôn khổ cho việc lập trình. Các triết lý đằng sau việc định hướng đối tượng đã trở nên mạnh mẽ để tạo thành một sự chuyển đổi mẫu hình trong nghành lập trình.

OOP đã phát triển một cách độc lập từ việc nghiên cứu về các ngôn ngữ định hướng của hệ thống mô phỏng đó là SIMULA 67, và từ việc nghiên cứu các kiến trúc của hệ thống có độ tin cậy cao, đó là các kiến trúc CPU và khả năng cơ bản của các hệ điều hành.

Một chức năng đặc sắc của OOP là việc xử lý các kiểu con của các kiểu dữ liệu.

Dữ liệu của các đối tượng một các tổng quát được đòi hỏi trong thiết kế để thỏa màn các yêu cầu của người lạp trình (tức là các lớp).

Các kiểu dữ leeiuj bị giới hạn thêm các điều kiện mặc dù có cùng kiểu dữ liệu với loại hông bị ràng buộc bởi các điều kiện đó, gọi là **kiểu dữ liệu con**. Cả hai loại kiểu dữ liệu này đều dựa vào và đều được điều tiết bởi các hành xử (tức là các phương thức) đã được định nghĩa. Các điều kiện hay yêu cầu này có thể được khi báo rõ ràng hay được giả thiết công nhận ngầm bởi người lập trình. Các ngôn ngữ định hướng đối tượng cung cấp nhiều cơ chế cho việc khẳng định răng các giải thiết đó có tính địa phương cho một phần của chương trình. Các cơ chế này có thể đọc thấy trong các tài liệu về các chương trình định hướng đối tượng.

OOP tự nó đã đang được dùng để khuyến mại cho nhiều sản phẩm và dịch vụ. Các định nghĩa hiện tại và ích lợi của các đặc tính của OOP thường được màu mè hóa bởi các mục đích của thị trường thương mại. Tương tự, nhiều ngôn ngữ lập trình có những quan điểm đặc biệt về OOP, nguyên do là vì chúng tập trung trên thời gian dịch hay tập trung vao thời gian thi hành của chương trình.

Mầu hình của OOP chủ yếu không phải là kiểu lập trình mà là kiểu thiết kế. MỘt hệ thống được thiết kế bởi định nghĩa của các đối tượng mà các đối tượng này sẽ tồn tại trong hệ thống đó, trong mã mà hiện làm việc chưa tương thích với đối tượng, hay là trong người dùng đối tượng do ảnh hưởng của tính chất đóng của đối tượng.

Cũng nên lưu ý rằng có sự khác biệt giữa mâu hình đối tượng và lý thuyết các hệ thống. OOP tập trung trên các đối tượng như là các đơn vị của một hệ thống, trong khi đó, lý thuyết các hệ thống lại tự nó chỉ tập trung vào hệ thoosgn. Như là phần trung gian, người ta có thể tìm thầy các dạng thức thiết kế phần mềm hay các kỹ thuật khác dùng các lớp và các đối tượng như là các viên gạch trong những thành phần lớn hơn. Những thành phần này có thể được xem nhu là bước trung gian từ mẫu hình định hướng đối tượng đến các mô hình "định hướng sống thực" hơn của lý thuyết các hệ thống.

2.3 Một số khái niệm cần biết trong ngôn ngữ OOP hiện đại

Hiện nay các ngôn ngữ OOP phổ biến nhất đều tập trung theo phương pháp phân lớp trong đó có C++, Java, C# và Visual Basic.NET. Ngôn ngữ OOP hay còn gọi là ngôn ngữ lập trình hướng đối tượng, là một phương pháp thiết kế và phát triển phần mềm dựa trên kiến trúc lớp và đối tượng. Sau đây là một số khái niệm mà các ngôn ngữ này thường dùng tới.

Lớp - Class Một lớp có thể được hiểu là khuôn mẫu để tạo ra đối tượng. Trong một lớp, người ta thường dùng các biến để mô tả các thuộc tính và các hàm để mô tả các phương thức của đối tượng. Khi đã định nghĩa được lớp, ta có thể tạo ra các đối tượng từ lớp này. Để việc sử dụng được dễ dàng, thông qua hệ thống hàm tạo (constructor), người ta dùng lớp như một kiểu dữ liệu để tạo ra các đối tượng.

Lớp con - Subclass Lớp con là một lớp thông thường nhưng có thêm tính chất kế thừa một phần hay toàn bộ các đặc tính của một lớp khác. Lớp chia sẽ sự kế thừa gọi là lớp cha (parent class).

Lớp trừu tượng - abstract class Lớp trừu tượng là một lớp mà nó không được thực thể hóa thành một đối tượng thực dụng được. Lớp này được thiết kế nhàm tạo ra một lớp có các đặc tính tổng quá nhưng bản thân lớp đó chưa có ý nghĩa hay không đủ ý nghĩa để có thể tiến hành viết mã cho việc thực thể hóa.

> Ví dụ: Lớp "hinh" được định nghĩa không có dữ liệu nội tại và chỉ có các phương thức (hàm nội tại) "tinh_chu_vi", "tinh_dien_tích". Nhưng vì lớp hình này chưa

xác định được đầy đủ các đặc tính của nó (cụ thể các biến nội tại là toạn độ của các đỉnh nếu là đa giác, là đường bán kính và tọa độ tâm nếu là hình tròn, ...) nên nó chỉ có thể được viết thành một lớp trừu tượng. Sau đó, người lập trình có thể tạo ra các lớp con, chẳng hạn như là lớp "tam_giac", lớp "hinh_tron", lớp "tu_giac",... Và trong các lớp con này người viết mã sẽ cung cấp các dữ liệu nội tại (như là biến nội tại r làm bán kính và hằng nội tại Pi cho lớp "hinh_tron" và sau đó viết mã cụ thể cho các phương thức "tinh_chu_vi" và "tinh_dien_tich").

Phương thức - Method Phương thwusc của một lớp thường được dùng để mô tả các hành vi của đối tượng (hoặc của lớp). Ví dụ như đối tượng thuộc lớp điện thoại có các hành vi sau: Đổ chuông, chuyển tín hiệu từ sóng sang dạng nghe được, chuyển tín hiệu giọng nói sang dạng chuẩn, chuyển tín hiệu lên tổng đài... Khi thiết kế, người ta có thể dùng các phương thức để mô tả và thực hiện hành vi đó được viết tại nội dung của hàm. Khi thực hiện hành vi này, đối tượng có thể phải thực hiện các hành vi khác. Ví dụ như điện thoại phải chuển tín hiệu giónó nói sai dạng chuẩn trước khi chuyển lên tổng đài. Cho nên một phương thức trong một lớp có thể sử dụng phương thức khác trong quá trinhf thự hiện hành vị của mình.

Người ta còn định nghĩa thêm vài loại phương thức đặc biệt:

- Hàm tạo (constructor) là hàm được dùng để tạo ra một đối tượng, cài đặt cái giá trị ban đầu cho các thuộc tính của đối tượng đó.
- Hàm hủy (destructor) là hàm dùng vào việc làm sạch bộ nhớ đã dùng để lưu đối tượng và hủy bỏ tên của một đối tượng sau khi dã dùng xong, trong đó có thể bao gồm cả việc xóa các con trỏ nội tại và trả về các phần bộ nhớ mà đối tượng đã dùng.

Nhiều lớp thư viện có sẵn hàm tạo mặc định (thông thường không có tham số) và hàm hủy.

Thuộc tính - attribute Thuộc tính của một lớp bao gồm các biến, các hằng, hay tham số nội tại của lớp đó. Ở đay, vai trò quan trọng nhất của các thuộc tính là các biến vì chúng sẽ có thể bị thay đổi trong suốt quá trình hoạt động của một đối tượng. Các thuộc tính có thể được xác định kiểu và kiểu của chúng có thể là các kiểu dữ liệu cổ diển hay đó là một lớp đã định nghĩa từ trước. Như đã ghi, khi một lớp đã được thực thể hóa thành đối tượng cụ thể thì tập hợp các giá trị của các biến nội tại làm thành trạng thái của đối tượng. Giống như trường hợp của phương thức, tùy theo người viết mã, biến nội tại có thể chỉ được dùng bên trong các phương thúc của chính lớp đó, có thể cho phép các câu lệnh bên ngoài lớp hay chỉ cho phép các lớp có quan hệ đặc biệt như là quan hệ lớp con, (và quan hệ bạn bè (*friend*) trong C++) được phép dùng tới nó (hay thay đổi giá trị của nó). Mỗi thuộc tính của một lớp còn được gọi là **thành viên dữ liệu** của lớp đó.

Quan hệ giữa lớp và đối tượng Lớp trong quan niệm thông thường là cách phân loại các thực thể dựa theo những đặc điểm chung của các thực thể đó. Do đó, lớp là khái niệm mang tính trừu tượng hóa rất cao. Ví dụ như lớp "người" dùng để chỉ những thực thể sống trên Trái Đất có những thuộc tính: có hai chân, hai bàn tay khéo léo, có tư duy, ngôn ngữ.v.v.. và có phương thức: giao tiếp bằng ngôn ngữ, tư duy, đi, đúng bằng hai chân... Khi đó hai người cụ thể ông A, ông B là các đối tượng thuộc lớp người. Trong ngôn ngữ lập trình, ta cũng hiểu khái niệm lớp tương tự, cho nên ta có quá trình "thực thể hóa" sau, tạo một đối tượng thuộc một lớp đã được ta định nghĩa (phân loại).

Thực thể hóa - Instantiate Là quá trình khai báo để có một tên (có thể xem như là một biến) trở thành một đối tượng từ một lớp nào đó.

Một lớp sau khi được tiến hành thực thể hóa để có một đối tượng cụ thể gọi là một **thực thể**. Hay nói ngược lại, một thực thể là một đối tượng riêng lẻ của một lớp đã định trước. Như các biến thông thường, hai thực thể của cùng một lớp có thể có trạng thái nội tại khác nhau (xác định bởi các giá trị hiện có của các biến nội tại) và do đó hoàn toàn độc lập nhau nếu không có yêu cầu gì đặc biệt từ người lập trình. "Thực thể hóa" gần giống như cá nhân hóa. một lớp khi được "cá nhân hóa" sẽ trở thành một đối tượng cụ thể.

Công cộng - Public Công cộng là một tính chất được dùng để gán cho các phương thức, các biến nội tại, hay các lớp mà khi khai báo thì người lập trình đã cho phép các câu lệnh bên ngoài cũng như các đối tượng khác được phép dùng đến nó.

Ví dụ: Trong C++ khai báo `public: int my_var;` thì biến my_var có hai tính chất là tính công cộng và là một interger. Cả hai tính chất này hợp thành đặc tính của biến my_var khiến nó có thể được sử dụng hay thay đổi giá trị của nó (bởi các câu lệnh) ở mọi nơi bên ngoài lẫn bên trong của lớp.

Riêng tư - Private Riêng tư là sự thể hiện tính chất đóng mạnh nhất (của một đặc tính hay một lớp). Khi dùng tính chất này gán cho một biến, một phương thức thì biến hay phương thức đó chỉ có thể sử dụng bên trong của lớp mà chúng dược định nghĩa. Mọi nỗ lực dùng trực tiếp đến chúng từ bên ngoài qua các câu lệnh hay từ các lớp con sẽ bị phủ nhận hay bị lỗi.

2.4 Bảo tồn (protected)

Tùy theo ngôn ngữ sẽ có vài điểm nhỏ khác nhau về các hiểu tính chất này. Nhìn chung đây là tính chất mà khi dùng để áp dụng cho các phương thức, các biến nội tại, hay các lớp thì chỉ có trong nội bộ của lớp đó hay các lớp con của nó (hay trong nội bộ một gói như trong Java) được phép gọi đến các phương pháp, biến của lớp đó.

So với tính chât riêng tư thì tính bảo tồn rộng rãi hơn về nghĩa chia sẻ dữ liệu hay chức năng. Nó cho phép một số trường hợp được dùng tới các đặc tính của một lớp (từ một lớp con chẳng hạn).

Lưu ý: Các tính chất công cộng, riêng tư và bảo tồn đôi khi còn được dùng để chỉ thị cho một lớp con cách thức kế thừa một lớp cha trong C++.

Đa kế thừa (multiple inheritance) Đây là một tính chất cho phép một lớp con có khả năng kế thừa trực tiếp cùng lúc nhiều lớp khác.

Vài điểm cần lưu ý khi viết mã dùng tính chất đa kế thừa:

- Khi muốn có một sự kế thừa từ nhiều lớp cha thì các lớp này cần phải độc lập và đặc biệt tên của các dữ liệu hay hàm cho phép kế thừa phải có tên khác nhau để tránh lỗi ambiguity. Bởi vì lúc đó, trình dịch sẽ không thể xác định được là lớp con sẽ kế thừa tên nào trong danh sách lớp cha.
- Nhiều ngôn ngữ, ví dụ như Java, không có đa thừa kế, nhưng chúng lại có khái niệm Interface. Với Interface, ta có thể có hầu hết các lới ích mà đa thừa kế mang lại.

Ngoài các khái niệm trên, tùy theo ngôn ngữ, có thể sẽ có các chức năng OOP riêng biệt.