Chapter 20 : CÁC LỚP DẪN XUẤT

Không nhân lên các đối tượng mà không cần thiết.

• Giới thiệu

• Các lớp có nguồn gốc

Chức năng Thành viên; Người xây dựng và Người phá hủy

• Phân cấp lớp

Nhập trường; Chức năng ảo; Trình độ chuyên môn rõ ràng; Kiểm soát Ghi đè; sử dụng Base Mem-

các loại bia; Loại trả lại Thư giãn

• Các lớp trừu tượng

• Kiểm soát truy cập

thành viên được bảo vệ; Quyền truy cập vào các lớp cơ sở; using-Khai báo và Kiểm soát Truy cập

• Con trỏ đến thành viên

Con trỏ đến các thành viên chức năng; Con trỏ tới Thành viên Dữ liệu; Thành viên gốc và thành viên có nguồn gốc

• Lời khuyên

**20.1 Giới thiệu**

Từ Simula, C ++ đã vay mượn ý tưởng của các lớp và cấu trúc phân cấp lớp. Ngoài ra, nó vay mượn ý tưởng thiết kế rằng các lớp nên được sử dụng để mô hình hóa các khái niệm trong thế giới của lập trình viên và ứng dụng. C ++ cung cấp các cấu trúc ngôn ngữ hỗ trợ trực tiếp các khái niệm thiết kế này. Ngược lại, việc sử dụng các tính năng ngôn ngữ hỗ trợ cho các ý tưởng thiết kế sẽ phân biệt việc sử dụng C ++ hiệu quả. Việc sử dụng các cấu trúc ngôn ngữ chỉ như các đạo cụ ký hiệu cho các kiểu lập trình truyền thống là bỏ sót những điểm mạnh chính của C ++.  
Một khái niệm không tồn tại biệt lập. Nó cùng tồn tại với các khái niệm liên quan và tạo ra phần lớn sức mạnh của nó từ các mối quan hệ với các khái niệm khác. Ví dụ, cố gắng giải thích ô tô là gì. Bạn sẽ sớm giới thiệu các khái niệm về bánh xe, động cơ, người lái xe, người đi bộ, xe tải, xe cứu thương, đường, dầu, vé chạy quá tốc độ, nhà nghỉ, v.v. Vì chúng tôi sử dụng các lớp để biểu thị các khái niệm, vấn đề trở thành cách thể hiện mối quan hệ giữa các khái niệm. Tuy nhiên, chúng ta không thể diễn đạt mối quan hệ arbi-trary trực tiếp bằng ngôn ngữ lập trình. Ngay cả khi chúng tôi có thể, chúng tôi sẽ không muốn. Để trở nên hữu ích, các lớp của chúng ta nên được định nghĩa hẹp hơn so với các khái niệm hàng ngày của chúng ta - và chính xác hơn.  
Khái niệm về một lớp dẫn xuất và các cơ chế ngôn ngữ liên quan của nó được cung cấp để thể hiện các mối quan hệ thứ bậc, nghĩa là, để thể hiện tính chung giữa các lớp. Ví dụ, con-cepts của một hình tròn và một hình tam giác có liên quan với nhau ở chỗ cả hai đều là hình dạng; nghĩa là họ có chung khái niệm về một hình dạng.  
  
Kế thừa giao diện thường được gọi là đa hình thời gian chạy. Ngược lại, việc sử dụng thống nhất các lớp không liên quan đến tính kế thừa được cung cấp bởi các khuôn mẫu thường được gọi là tính đa hình thời gian biên dịch.  
  
Phần thảo luận về phân cấp lớp được tổ chức thành ba chương  
  
• Các lớp có nguồn gốc: Chương này giới thiệu các tính năng cơ bản của ngôn ngữ hỗ trợ lập trình hướng đối tượng. Các lớp cơ sở và dẫn xuất, các hàm ảo và kiểm soát truy cập được bao phủ.  
• Phân cấp lớp: Chương này tập trung vào việc sử dụng các lớp cơ sở và lớp dẫn xuất để tổ chức mã một cách hiệu quả xung quanh khái niệm phân cấp lớp. Phần lớn chương này được dành để thảo luận về các kỹ thuật lập trình, nhưng các khía cạnh kỹ thuật của đa kế thừa cũng được đề cập.  
• Nhận dạng loại thời gian chạy: Chương này mô tả các kỹ thuật để điều hướng phân cấp lớp một cách rõ ràng.  
  
Có thể tìm thấy phần giới thiệu ngắn gọn về ý tưởng cơ bản về tổ chức phân cấp của các kiểu trong Chương  
  
3: các lớp cơ sở và dẫn xuất và các hàm ảo. Các chương này sẽ xem xét các tính năng cơ bản này và các kỹ thuật thiết kế và lập trình liên quan của chúng một cách chi tiết hơn.

**20.2: Các lớp có nguồn gốc**

Cân nhắc xây dựng một chương trình giao dịch với những người được một công ty tuyển dụng:

**struct Nhân viên {**

**string first\_name, family\_name;**

**char middle\_initial;**

**Ngày\_công\_dụng;**

**ngắn depar tment;**

**// ...};**

Tiếp theo, chúng ta có thể cố gắng xác định một người quản lý:

**struct Manager {**

**Employee emp; // hồ sơ nhân viên của người quản lý**

**list<Employee∗> group; // người quản lý**

**short level//;...};**

Một nhà quản lý cũng là một nhân viên; dữ liệu Nhân viên được lưu trữ trong thành viên emp của một đối tượng Người quản lý. Điều này có thể hiển nhiên đối với người đọc là con người - đặc biệt là người đọc cẩn thận - nhưng không có gì nói với trình biên dịch và các công cụ khác rằng Người quản lý cũng là Nhân viên. Nhân viên ∗, vì vậy người ta không thể đơn giản sử dụng cái này khi cái kia được yêu cầu. Đặc biệt, người ta không thể đưa Người quản lý vào danh sách Nhân viên mà không viết mã đặc biệt. Tuy nhiên, cả hai giải pháp đều không phù hợp và có thể khá mù mờ. Cách tiếp cận đúng là tuyên bố rõ ràng rằng Người quản lý là Nhân viên, với một vài thông tin được thêm vào:

**struct Manager : public Employee {**

**list<Employee∗> group;**

**short level;**

**// ...};**

Trình quản lý có nguồn gốc từ Nhân viên và ngược lại, Nhân viên là một lớp cơ sở cho Trình quản lý.( Các quản lý lớp có các thành viên của lớp Nhân viên (tên\_người đầu tiên, phòng ban, v.v.) ngoài các thành viên của lớp các thành viên riêng (nhóm, cấp độ, v.v.).Người quản lý lớp có các thành viên của Nhân viên lớp ngoài các thành viên của chính nó.