MỤC LỤC

[MỤC LỤC I](#_Toc512700295)

[DANH MỤC BẢNG VII](#_Toc512700296)

[DANH MỤC HÌNH VIII](#_Toc512700297)

[LỜI NÓI ĐẦU 1](#_Toc512700298)

[CHƯƠNG 1. GIỚI THIỆU VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VÀ NGÔN NGỮ LẬP TRÌNH JAVA 3](#_Toc512700299)

[1.1. Giới thiệu về lập trình hướng đối tượng 3](#_Toc512700300)

[1.1.2. Đối tượng 4](#_Toc512700301)

[1.1.3. Lớp 5](#_Toc512700302)

[1.1.4. Trừu tượng hóa và đóng gói dữ liệu 6](#_Toc512700303)

[1.1.5. Tính kế thừa 6](#_Toc512700304)

[1.1.6. Tính đa hình 7](#_Toc512700305)

[1.2. Giới thiệu về ngôn ngữ lập trình Java 8](#_Toc512700306)

[1.2.1. Giới thiệu ngôn ngữ lập trình Java 8](#_Toc512700307)

[1.2.2. Các đặc trưng của Java 9](#_Toc512700308)

[1.2.2.1. Đơn giản 9](#_Toc512700309)

[1.2.2.2. Hướng đối tượng 10](#_Toc512700310)

[1.2.2.3. Độc lập phần cứng và hệ điều hành 10](#_Toc512700311)

[1.2.2.4. Mạnh mẽ 11](#_Toc512700312)

[1.2.2.5. Bảo mật 11](#_Toc512700313)

[1.2.2.6. Phân tán 11](#_Toc512700314)

[1.2.2.7. Đa luồng 11](#_Toc512700315)

[1.2.2.8. Động 11](#_Toc512700316)

[1.2.3. Giới thiệu và cài đặt Bộ công cụ JDK (Java Development Kit) 11](#_Toc512700317)

[1.2.3.1. Trình biên dịch, 'javac' 12](#_Toc512700318)

[1.2.3.2. Trình thông dịch, 'java' 12](#_Toc512700319)

[1.2.3.3. Trình dịch ngược, 'javap' 12](#_Toc512700320)

[1.2.3.4. Công cụ sinh tài liệu, 'javadoc' 12](#_Toc512700321)

[1.2.3.5. Chương trình tìm lỗi - Debug, 'jdb‘ 12](#_Toc512700322)

[1.2.3.6. Chương trình xem Applet , 'appletviewer‘ 12](#_Toc512700323)

[CHƯƠNG 2. CÁC THÀNH PHẦN CƠ BẢN TRONG JAVA 13](#_Toc512700324)

[2.1. Giới thiệu chương trình Java đơn giản 13](#_Toc512700325)

[2.2. Các kiểu dữ liệu nguyên thủy (primitive) của Java 14](#_Toc512700326)

[2.3. Biến và hằng số trong Java 15](#_Toc512700327)

[2.3.1. Biến 15](#_Toc512700328)

[2.3.1.1. Cú pháp khai báo biến trong Java như sau: 16](#_Toc512700329)

[2.3.2. Hằng số 16](#_Toc512700330)

[2.4. Các toán tử trong Java 17](#_Toc512700331)

[2.4.1. Toán tử số học 18](#_Toc512700332)

[2.4.2. Toán tử gán gộp (compound assignment operators) 19](#_Toc512700333)

[2.4.3. Các toán tử tăng, giảm 19](#_Toc512700334)

[2.4.4. Biểu thức (expression) 21](#_Toc512700335)

[2.4.5. Chuyển kiểu tự động 22](#_Toc512700336)

[2.4.6. Toán tử ép kiểu (type cast operator) 23](#_Toc512700337)

[2.5. Biểu thức logic 24](#_Toc512700338)

[2.5.1. Toán tử quan hệ (relational operators) 24](#_Toc512700339)

[2.5.2. Toán tử logic 25](#_Toc512700340)

[2.6. Các toán tử thao tác bit (bitwise operators) 26](#_Toc512700341)

[2.6.2. Phép phủ định bit 26](#_Toc512700342)

[2.6.3. Phép toán AND bit 26](#_Toc512700343)

[2.6.4. Phép toán OR bit 27](#_Toc512700344)

[2.6.5. Phép toán XOR bit 27](#_Toc512700345)

[2.6.6. Phép toán dịch phải số số học 27](#_Toc512700346)

[2.6.7. Phép toán dịch trái số học 28](#_Toc512700347)

[2.6.8. Phép toán dịch phải số không dấu 28](#_Toc512700348)

[2.7. Nhập, xuất đơn giản của Java 29](#_Toc512700349)

[2.7.1. Xuất đơn giản 29](#_Toc512700350)

[2.7.2. Nhập đơn giản 29](#_Toc512700351)

[2.8. Cấu trúc điều khiển của Java 31](#_Toc512700352)

[2.8.1. Cấu trúc lựa chọn 31](#_Toc512700353)

[2.8.1.1. Cấu trúc lựa chọn if 31](#_Toc512700354)

[2.8.1.2. Cấu trúc lựa chọn if…else 33](#_Toc512700355)

[2.8.1.3. Toán tử điều kiện ? 35](#_Toc512700356)

[2.8.1.4. Cấu trúc switch 36](#_Toc512700357)

[2.8.2. Cấu trúc lặp 38](#_Toc512700358)

[2.8.2.1. Cấu trúc lặp while 39](#_Toc512700359)

[2.8.2.2. Cấu trúc lặp for 40](#_Toc512700360)

[2.8.2.3. Cấu trúc lặp do … while 42](#_Toc512700361)

[2.8.2.4. Câu lệnh break và continue 43](#_Toc512700362)

[2.9. Mảng trong java 44](#_Toc512700363)

[2.9.1. Giới thiệu mảng 44](#_Toc512700364)

[2.9.2. Các thao tác cơ bản với mảng 45](#_Toc512700365)

[2.9.2.2 Nhập điểm cho các sinh viên từ bàn phím 45](#_Toc512700366)

[2.9.2.3 Tính tổng điểm của tất cả các sinh viên 46](#_Toc512700367)

[2.9.2.4 Hiển thị điểm của các sinh viên ra màn hình 46](#_Toc512700368)

[2.9.2.5 Tìm sinh viên có điểm bằng với điểm nhập từ bàn phím 46](#_Toc512700369)

[2.9.2.6 Tìm điểm cao nhất và xác định sinh viên có điểm cao nhất 46](#_Toc512700370)

[2.9.3. Mảng trong Java 47](#_Toc512700371)

[2.9.3.1. Sử dụng các phần tử của mảng 47](#_Toc512700372)

[2.9.3.2. Mảng đối tượng 48](#_Toc512700373)

[2.9.3.3. Khởi tạo cho mảng 49](#_Toc512700374)

[2.9.3.4. Truyền đối số mảng cho phương thức 49](#_Toc512700375)

[2.9.4. Mảng nhiều chiều 50](#_Toc512700376)

[BÀI TẬP CHƯƠNG 2 52](#_Toc512700377)

[CHƯƠNG 3. LỚP VÀ ĐỐI TƯỢNG TRONG JAVA 56](#_Toc512700378)

[3.1. Đối tượng và lớp trong Java 56](#_Toc512700379)

[3.1.1. Đối tượng 56](#_Toc512700380)

[3.1.2. Lớp (class) 56](#_Toc512700381)

[3.2. Thuộc tính và phương thức của lớp 58](#_Toc512700382)

[3.2.1. Thuộc tính 58](#_Toc512700383)

[3.2.2. Phương thức 59](#_Toc512700384)

[3.2.2.1. Cú pháp định nghĩa phương thức như sau: 59](#_Toc512700385)

[3.2.2.2. Danh sách đối số 60](#_Toc512700386)

[3.2.2.3. Kết quả trả về từ phương thức 61](#_Toc512700387)

[3.2.2.4. Nạp chồng phương thức (method overloading) 61](#_Toc512700388)

[3.3. Phương thức tạo (constructor) 62](#_Toc512700389)

[3.4. Phương thức lấy và thiết lập giá trị các thuộc tính 65](#_Toc512700390)

[3.4.1. Phương thức thiết lập giá trị cho thuộc tính (setter) 65](#_Toc512700391)

[3.4.2. Phương thức lấy giá trị của thuộc tính (getter) 65](#_Toc512700392)

[3.5. Hiển thị trạng thái của đối tượng 65](#_Toc512700393)

[3.6. Phương thức tĩnh (static method) và thuộc tính tĩnh (static field) 67](#_Toc512700394)

[3.6.1. Thuộc tính tĩnh 67](#_Toc512700395)

[3.6.2. Phương thức tĩnh (static methods) 68](#_Toc512700396)

[3.6.3. Khai báo hằng số (final) 68](#_Toc512700397)

[3.7. Gói (package) 68](#_Toc512700398)

[3.7.1. Khái niệm 68](#_Toc512700399)

[3.7.2. Sử dụng gói 69](#_Toc512700400)

[3.8. Kết tập (composition), kế thừa (inheritance) và đa hình (polymorphism) 69](#_Toc512700401)

[3.8.1. Kết tập 69](#_Toc512700402)

[3.8.2. Kế thừa 75](#_Toc512700403)

[3.8.2.1. Định nghĩa 75](#_Toc512700404)

[3.8.2.2. Truy xuất vào các thành phần của lớp cha 80](#_Toc512700405)

[3.8.2.3. Ghi đè phương thức (method overriding) và che dấu thuộc tính (variable hiding) 82](#_Toc512700406)

[3.8.2.4. Quyền truy cập 84](#_Toc512700407)

[3.8.3. Đa hình (polymorphism) 85](#_Toc512700408)

[3.8.3.1. Mảng đối tượng 85](#_Toc512700409)

[3.8.3.2. Đa hình 87](#_Toc512700410)

[3.9. Lớp trừu tượng (abstract class) và giao diện (interface) 88](#_Toc512700411)

[3.9.1. Lớp trừu tượng 88](#_Toc512700412)

[3.9.2. Interface 93](#_Toc512700413)

[BAI TẬP CHƯƠNG 3 97](#_Toc512700414)

[CHƯƠNG 4. XỬ LÝ NGOẠI LỆ (EXCEPTION HANDLING) 107](#_Toc512700415)

[4.1. Giới thiệu 107](#_Toc512700416)

[4.2. Xử lý ngoại lệ với java 107](#_Toc512700417)

[4.3. Phân loại ngoại lệ 110](#_Toc512700418)

[4.3.2. Lỗi (Error) 110](#_Toc512700419)

[4.3.3. Ngoại lệ (Exception) 111](#_Toc512700420)

[4.3.4. Thao tác với ngoại lệ 111](#_Toc512700421)

[4.4. Tự tạo lớp xử lý ngoại lệ 112](#_Toc512700422)

[4.5. Cơ chế gom và xử lý rác trong Java 114](#_Toc512700423)

[BÀI TẬP CHƯƠNG 4 115](#_Toc512700424)

[CHƯƠNG 5. CÁC LỚP CƠ SỞ VÀ CẤU TRÚC DỮ LIỆU 116](#_Toc512700425)

[5.1. Các lớp cơ bản của Java 116](#_Toc512700426)

[5.1.1. Lớp Math 116](#_Toc512700427)

[5.1.2. Lớp Calendar 119](#_Toc512700428)

[5.1.3. Lớp Random 122](#_Toc512700429)

[5.1.4. Lớp String 123](#_Toc512700430)

[5.1.5. Lớp StringBuffer 126](#_Toc512700431)

[5.1.6. Lớp StringBuilder 128](#_Toc512700432)

[5.2. Cấu trúc tuyển tập đối tượng (collection framework) 130](#_Toc512700433)

[5.2.2. List interface 130](#_Toc512700434)

[5.2.2.1. Lớp ArrayList 130](#_Toc512700435)

[5.2.2.2. Lớp LinkedList 132](#_Toc512700436)

[5.2.3. Set interface 134](#_Toc512700437)

[5.2.3.1. Lớp HashSet 134](#_Toc512700438)

[5.2.3.2. Lớp LinkedHashSet 137](#_Toc512700439)

[5.2.4. Map interface 140](#_Toc512700440)

[5.2.4.1. Lớp HashMap 140](#_Toc512700441)

[5.2.4.2. Lớp LinkedHashMap 142](#_Toc512700442)

[5.2.4.3. Lớp TreeMap 143](#_Toc512700443)

[5.2.4.4. Lớp HashTable 143](#_Toc512700444)

[5.3. Phương thức tìm kiếm và sắp xếp của Collections 144](#_Toc512700445)

[5.3.1. Sắp xếp 144](#_Toc512700446)

[5.3.2. Tìm kiếm 152](#_Toc512700447)

[BÀI TẬP CHƯƠNG 5: 156](#_Toc512700448)

[CHƯƠNG 6. CÁC LUỒNG NHẬP XUẤT FILE 160](#_Toc512700449)

[6.1. Giới thiệu 160](#_Toc512700450)

[6.2. Luồng nhập xuất văn bản 161](#_Toc512700451)

[6.2.1. Ghi vào file văn bản 161](#_Toc512700452)

[6.2.2. Đọc file văn bản 163](#_Toc512700453)

[6.3. Luồng nhập xuất nhị phân 164](#_Toc512700454)

[6.3.1. Ghi vào file nhị phân 165](#_Toc512700455)

[6.3.2. Đọc file nhị phân 166](#_Toc512700456)

[6.4. Luồng nhập xuất đối tượng 167](#_Toc512700457)

[6.4.1. Ghi đối tượng vào file 167](#_Toc512700458)

[6.4.2. Đọc đối tượng từ file 169](#_Toc512700459)

[BÀI TẬP CHƯƠNG 6 171](#_Toc512700460)

[TÀI LIỆU THAM KHẢO 172](#_Toc512700461)

DANH MỤC BẢNG

[Bảng 2.1. Các kiểu dữ liệu nguyên thủy của Java 14](#_Toc512699821)

[Bảng 2.2. Các lớp bao bọc tương ứng với các kiểu dữ liệu nguyên thủy 15](#_Toc512699822)

[Bảng 2.3. Bảng hằng số nguyên ở một số hệ đếm 17](#_Toc512699823)

[Bảng 2.4. Các toán tử số học chính của Java 18](#_Toc512699824)

[Bảng 2.5. các toán tử gán gộp trong Java 19](#_Toc512699825)

[Bảng 2.6. Toán tử tăng và giảm trong Java 19](#_Toc512699826)

[Bảng 2.7. Minh họa một số biểu thức đơn giản trong Java 22](#_Toc512699827)

[Bảng 2.8. Các toán tử quản hệ của Java 24](#_Toc512699828)

[Bảng 2.9. Các toán tử logic của Java 25](#_Toc512699829)

[Bảng 2.10. Bảng chân lý của toán tử && (AND) 25](#_Toc512699830)

[Bảng 2.11. Bảng chân lý của toán tử || (OR) 25](#_Toc512699831)

[Bảng 2.12. Bảng chân lý của toán tử ! (NOT) 25](#_Toc512699832)

[Bảng 2.13. Các phép toán thao tác bit 26](#_Toc512699833)

[Bảng 2.14. Bảng chân lý của phép phủ định bit 26](#_Toc512699834)

[Bảng 2.15. Bảng chân lý của phép AND bit 26](#_Toc512699835)

[Bảng 2.16. Bảng chân lý của phép OR bit 27](#_Toc512699836)

[Bảng 2.17. Bảng chân lý của phép XOR bit 27](#_Toc512699837)

[Bảng 2.18. Một số phương thức cơ bản của lớp Scanner 30](#_Toc512699838)

[Bảng 2.19. Xếp loại sinh viên dựa vào điểm 37](#_Toc512699839)

[Bảng 2.20. Quan hệ giữa điểm và xếp loại 54](#_Toc512699840)

[Bảng 2.21. Quan hệ giữa số điện và đơn giá 54](#_Toc512699841)

[Bảng 2.22. Mối quan hệ giữa số ngày nghỉ và đơn giá 54](#_Toc512699842)

[Bảng 3.1. Tổng hợp các quyền truy cập 84](#_Toc512699843)

[Bảng 5.1. Một số hàm toán học thông dụng của lớp Math 116](#_Toc512699844)

[Bảng 5.2. Một số phương thức chính của lớp Random 122](#_Toc512699845)

[Bảng 5.3. Một số phương thức thông dụng của lớp String 125](#_Toc512699846)

[Bảng 5.4. Một số phương thức tạo quan trọng của lớp StringBuffer 126](#_Toc512699847)

[Bảng 5.5. Một số phương thức quan trọng của lớp StringBuffer 127](#_Toc512699848)

[Bảng 5.6. Một số phương thức tạo chính của lớp StringBuilder 128](#_Toc512699849)

[Bảng 5.7. Một số phương thức quan trọng của lớp StringBuilder 129](#_Toc512699850)

[Bảng 5.8. Các phương thức tạo của lớp HashMap 141](#_Toc512699851)

DANH MỤC HÌNH

[Hình 1.1. Tổ chức của dữ và hàm trong lập trình hướng đối tượng 4](#_Toc512699911)

[Hình 1.2. Biểu diễn một đối tượng 5](#_Toc512699912)

[Hình 1.3. Minh họa tính chất kế thừa của OOP 7](#_Toc512699913)

[Hình 1.4. Minh họa tính đa hình của OOP 8](#_Toc512699914)

[Hình 1.5. Minh họa đặc trưng độc lập phần cứng của Java 10](#_Toc512699915)

[Hình 2.1. Chương trình Java đơn giản 13](#_Toc512699916)

[Hình 2.2. Minh họa autoboxing và unboxing 15](#_Toc512699917)

[Hình 2.3. Lưu đồ thuật toán của cấu trúc if 32](#_Toc512699918)

[Hình 2.4. Lưu đồ thuật toán của cấu trúc if…else 34](#_Toc512699919)

[Hình 2.5. Lưu đồ thuật toán của switch 37](#_Toc512699920)

[Hình 2.6. Lưu đồ thuật toán của cấu trúc while 39](#_Toc512699921)

[Hình 2.7. Lưu đồ thuật toán của cấu trúc for 41](#_Toc512699922)

[Hình 2.8. Lưu đồ thuật toán cấu trúc lặp do … while 42](#_Toc512699923)

[Hình 3.1. Minh họa đối tượng thực và đối tượng trong ngôn ngữ lập trình 56](#_Toc512699924)

[Hình 3.2. Biểu đồ lớp Author 70](#_Toc512699925)

[Hình 3.3. Biểu đồ lớp Book 70](#_Toc512699926)

[Hình 3.4. Biểu đồ lớp Person 75](#_Toc512699927)

[Hình 3.5. Minh họa lớp Employee kế thừa từ lớp Person 76](#_Toc512699928)

[Hình 3.6. Biểu đồ lớp kế thừa nhiều cấp 80](#_Toc512699929)

[Hình 3.7. Biểu đồ lớp minh họa lớp trừu tượng 89](#_Toc512699930)

[Hình 3.8. Minh họa đa thừa kế với Interface 94](#_Toc512699931)

[Hình 3.9. Biểu đồ lớp Person 97](#_Toc512699932)

[Hình 3.10. Biểu đồ lớp DiaChi 98](#_Toc512699933)

[Hình 3.11. Biểu đồ lớp NhanVien 98](#_Toc512699934)

[Hình 3.12. Biều đồ lớp BenhNhan 99](#_Toc512699935)

[Hình 3.13. Biểu đồ lớp SinhVien 99](#_Toc512699936)

[Hình 3.14. Biểu đồ lớp PartTimeEmployee 100](#_Toc512699937)

[Hình 3.15. Biểu đồ lớp của ResizableCircle 101](#_Toc512699938)

[Hình 3.16. Biểu đồ lớp Rectangle và Triangle 101](#_Toc512699939)

[Hình 4.1. Biểu đồ lớp xử lý ngoại lệ của Java 110](#_Toc512699940)

[Hình 4.2. Minh họa kết quả của ngoại lệ tự định nghĩa 114](#_Toc512699941)

[Hình 5.1. Kết quả của chương trình mô phỏng các hàm toán học thông dụng 119](#_Toc512699942)

[Hình 5.2. Kết quả chương trình minh họa sử dụng Calendar 122](#_Toc512699943)

[Hình 5.3. Minh họa kết quả chạy chương trình demo lớp Random 123](#_Toc512699944)

[Hình 5.4. Minh họa hoạt động của hằng xâu kí tự. 124](#_Toc512699945)

[Hình 5.5. Biểu đồ lớp của Collection và Map 130](#_Toc512699946)

[Hình 5.6. Minh họa danh sách liên kết 132](#_Toc512699947)

[Hình 5.7. Minh họa hoạt động của iterator 136](#_Toc512699948)

[Hình 6.1. Nội dung file văn bản vidu.txt 163](#_Toc512699949)

[Hình 6.2. Nội dung của file vidu.dat mở bằng Notepad 166](#_Toc512699950)

LỜI NÓI ĐẦU

Ngôn ngữ lập trình Java là một ngôn ngữ lập trình hướng đối tượng hoàn toàn vì nó mang đầy đủ các tính chất của lập trình hướng đối tượng như trừu tượng dữ liệu; tính đóng gói; tính kế thừa và đa hình. Ngoài ra, một chương trình Java dù nhỏ đến đâu cũng phải gắn với một lớp. Ngôn ngữ lập trình Java còn là một ngôn ngữ mạnh vì nó độc lập với kiến trúc phần cứng của máy tính và độc lập với hệ điều hành đúng như câu khẩu hiệu của Java “write one runs every where (viết một lần chạy ở mọi nơi)”. Thêm vào đó, Java còn cung cấp cơ chế xử lý ngoại lệ, tự động dọn rác để tối ưu bộ nhớ, hỗ trợ đa luồng và bảo mật tốt. Do đó, Java là ngôn ngữ lập trình thường được chọn để xây dựng các ứng dụng lớn, yêu cầu tính bảo mật cao như trong ngân hàng, hàng không, quốc phòng, … Nắm được ngôn ngữ lập trình Java căn bản giúp sinh viên dễ dàng học tiếp các môn học về Java nâng cao (servlet và JSP), cũng như học lập trình Android. Nhằm tạo điều kiện thuận lợi cho sinh viên khoa Công nghệ thông tin có một cuốn tài liệu đơn giản, dễ hiểu và đầy đủ hỗ trợ cho việc học và tự học kiến thức về Java căn bản nên nhóm tác giả quyết định viết cuốn giáo trình “Lập trình Java căn bản”. Nội dung của giáo trình gồm 6 chương:

Chương 1: Giới thiệu về lập trình hướng đối tượng và ngôn ngữ lập trình Java, nội dung của chương chủ yếu tập trung trình bày các khái niệm liên quan đến lập trình hướng đối tượng và các đặc trưng của ngôn ngữ lập trình Java.

Chương 2: Các thành phần cơ bản trong Java, nội dung của chương này trình bày các kiến thức liên quan đến các kiểu dữ liệu, toán tử, biểu thức và các cấu trúc lập trình của Java.

Chương 3: Lớp và đối tượng trong Java, chương này chủ yếu trình bày các kiến thức liên quan đến lớp như thuộc tính, phương thức của lớp, phương thức tạo, tính đóng gói, kế thừa, đa hình. Chương này cũng trình bày cách tạo và giải phóng đối tượng.

Chương 4: Xử lý ngoại lệ, nội dung của chương chủ yêu tập trung trình bày cơ chế xử lý lỗi của ngôn ngữ lập trình Java.

Chương 5: Các lớp cơ sở và cấu trúc dữ liệu, nội dung của chương trình bày các lớp cơ bản hay sử dụng của Java như lớp thao tác với các hàm toán học Math, xử lý dữ liệu thời gian Calendar, tạo số ngẫu nhiên Random. Nội dung chính của chương trình bày tuyển tập (collections) của Java như danh sách (list), tập hợp (set) và ánh xạ (map).

Chương 6: Các luồng nhập xuất file, nội dung cơ bản của chương là trình bày ba luồng nhập xuất cơ bản của Java là nhập xuất file văn bản, nhập xuất file nhị phân và nhập xuất file đối tượng.

Mặc dù nhóm tác giả đã cố gắng, tuy nhiên cuốn giáo trình không thể tránh khỏi thiếu sót. Nhóm tác giả rất mong nhận được các ý kiến đóng góp từ các đồng nghiệp và bạn đọc để cuốn giáo trình ngày càng hoàn thiện hơn. Một lần nữa nhóm tác giả xin chân thành cảm ơn đến gia đình, bạn bè và đồng nghiệp đã động viên, đóng góp các ý kiến quý báu để nhóm tác giả hoàn thành cuốn giáo trình này.

1. GIỚI THIỆU VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VÀ NGÔN NGỮ LẬP TRÌNH JAVA

Sau khi đọc song chương này sinh viên nắm được các tính chất của lập trình hương đối tượng nói chung và ngôn ngữ lập trình Java nói riêng như tính trừu tượng dữ liệu; đóng gói; kế thừa; đa hình. Các khái niệm của lập trình hướng đối tượng như lớp (class); đối tượng (object). Ngoài ra, chương này cũng giới thiệu một số đặc trưng và lịch sử phát triển và các thành phần của ngôn ngữ lập trình Java.

* 1. Giới thiệu về lập trình hướng đối tượng

Lập trình hướng đối tượng (OOP: Object Oriented Programming) được phát triển để khắc phục những nhược điểm của ngôn ngữ lập trình trước đó là:

* Không quản lý được sự thay đổi dữ liệu khi có nhiều chương trình cùng thay đổi một biến dùng chung. Vấn đề này đặc biệt nghiêm trọng khi các ứng dụng ngày càng lớn, người ta không thể kiểm soát được việc truy nhập dữ liệu đến các biến dùng chung.
* Không tiết kiệm tài nguyên con người: Giải thuật gắn liền với cấu trúc dữ liệu, do đó nếu thay đổi cấu trúc dữ liệu sẽ thay đổi giải thuật, điều này có nghĩa là chúng ta phải viết chương trình lại từ đầu.

Lập trình hướng đối tượng coi dữ liệu là thành phần quan trọng trong phát triển chương trình và không cho phép nó di chuyển một cách tự do trong hệ thống. Lập trình hướng đối tượng gắn chặt dữ liệu với các hàm thao tác lên chúng và bảo vệ để không cho phép sự thay đổi tùy tiện từ bên ngoài hàm. Lập trình hướng đối tượng cho phép chia một bài toán thành các thực thể (Entities), được gọi là các đối tượng (Objects) và xây dựng dữ liệu và các hàm xung quanh các đối tượng này. Hình 1.1 sau mô tả việc tổ chức của dữ liệu và các hàm trong lập trình hướng đối tượng. Dữ liệu của một đối tượng chỉ có thể được truy cập bới các hàm của chính đối tượng đó. Tuy nhiên, một hàm của đối tượng này có thể truy xuất đến một hàm của đối tượng khác.

Lập trình hướng đối tượng có một số đặc trưng cơ bản sau:

* Nhấn mạnh vào dữ liệu hơn là các thủ tục (procedure).
* Chương trình được chia thành các đối tượng.
* Các cấu trúc dữ liệu được thiết kế sao cho chúng mô tả các đối tượng.
* Các hàm thao tác trên dữ liệu của đối tượng được ràng buộc vào nhau trong một cấu trúc dữ liệu.
* Dữ liệu được che dấu sao cho các hàm bên ngoài không thể truy cập vào được.
* Các đối tượng có thể trao đổi thông tin với nhau thông qua các hàm.
* Dữ liệu và các hàm mới có thể dễ dàng thêm vào đối tượng khi cần thiết.
* Chương trình được thiết kế theo kỹ thuật từ dưới lên (bottom up)

Dữ liệu

Hàm

Đối tượng A

Dữ liệu

Hàm

Đối tượng B

Dữ liệu

Hàm

Đối tượng C

Tổ chức của dữ và hàm trong lập trình hướng đối tượng

Chúng ta cần nắm được một số khái niệm được sử dụng rộng rãi trong lập trình hướng đối tượng. Chúng bao gồm:

* Đối tượng (objects)
* Lớp (Classes)
* Trừu tượng và đóng gói dữ liệu (Data abstraction and encapsulation)
* Kế thừa (Inheritance)
* Đa hình (Polymorphism)
* Liên kết động (Dynamic binding)
* Truyền thông báo (Message passing)
  + 1. Đối tượng

Đối tượng là thực thể cơ bản được tạo ra trong lúc thi hành chương trình của một chương trình hướng đối tượng. Đối tượng có thể tượng trưng cho một người, một địa danh, một tài khoản ngân hàng, một bảng dữ liệu hoặc một mục bất kỳ mà chương trình phải xử lý. Đối tượng cũng có thể biểu diễn dữ liệu do người sử dụng định nghĩa ví dụ như véc tơ, thời gian và danh sách. Như vậy, công việc lập trình chính là phân tích tính chất của các đối tượng và quá trình truyền thông tin giữa chúng. Các đối tượng của chương trình nên được chọn sao cho càng sát với các đối tượng của thế giới thực càng tốt. Các đối tượng của chương trình cũng sử dụng không gian của bộ nhớ và được cấp phát địa chỉ giống như kiểu cấu trúc của ngôn ngữ lập trình C.

Khi một chương trình được thực thi thì các đối tượng tương tác với nhau thông qua việc gửi các thông báo từ đối tượng này đến đối tượng khác. Ví dụ trong một chương trình chúng ta có 2 đối tượng là customer (khách hàng) và đối tượng account (tài khoản) thì đối tượng khách hàng có thể gửi một thông báo đến đối tượng tài khoản để yêu cầu vấn tin tài khoản của khách hàng. Mỗi một đối tượng sẽ chứa dữ liệu và các hàm thao tác trên dữ liệu này. Trong lập trình hướng đối tượng các dữ liệu được gọi là các thuộc tính (attributes), các hàm được gọi là các phương thức (methods). Các đối tượng có thể tương tác với nhau mà không cần biết thông tin chi tiết về thuộc tính và phương thức của nhau. Thông tin mà các đối tượng cần là kiểu thông báo được chấp nhận và dữ liệu trả về của đối tượng. Hình vẽ 1.2 bên dưới minh họa biểu diễn đối tượng.

**Đối tượng**: SinhVien

**Dữ liệu**:

Tên

Ngày sinh

Điểm

**Hàm**:

Tính tổng điểm

Tính điểm trung bình

Hiển thị thông tin

Biểu diễn một đối tượng

* + 1. Lớp

Như chúng ta đã biết, các đối tượng chứa dữ liệu và các mã lệnh xử lý dữ liệu đó. Toàn bộ dữ liệu và mã lệnh của một đối tượng được tạo ra bởi kiểu dữ liệu do người lập trình định nghĩa và được gọi là lớp (class). Trên thực tế thì đối tượng chính là biến có kiểu dữ liệu là lớp. Mỗi khi một lớp đã được định nghĩa thì chúng ta có thể tạo ra vô số đối tượng thuộc về lớp đó. Mỗi đối tượng liên kết với dữ liệu kiểu lớp tạo ra nó. Do đó, một lớp là tập hợp của các đối tượng có kiểu tương tự nhau. Ví dụ xoài, táo và cam được tạo ra từ một lớp hoa quả. Như vậy, lớp là kiểu dữ liệu do người lập trình định nghĩa ra và có tính chất giống như kiểu dữ liệu có sẵn của ngôn ngữ lập trình.

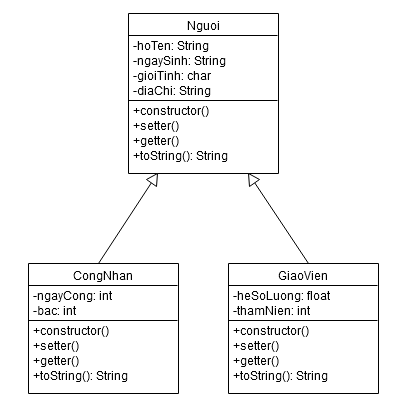
* + 1. Trừu tượng hóa và đóng gói dữ liệu

Việc gói dữ liệu cùng với các hàm vào một lớp được hiểu là đóng gói dữ liệu. Đóng gói dữ liệu là một đặc trưng nổi bật của lập trình hướng đối tượng. Việc đóng gói giúp cho dữ liệu không bị truy cập trực tiếp từ bên ngoài và chỉ những hàm được gói trong lớp mới có thể truy cập được. Các hàm này cung cấp một giao diện giữa dữ liệu của đối tượng với chương trình. Việc cách ly dữ liệu để tránh truy cập trực tiếp từ bên ngoài còn được gọi là che dấu thông tin (information hiding) hoặc che dấu dữ liệu (data hiding). Do có tính chất đóng gói nên 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. Đây là tính chất đảm bảo sự toàn vẹn của đối tượng. Trừu tượng dữ liệu là thu thập các đặc trưng cần thiết của một đối tượng mà không bao gồm các thông tin chi tiết hoặc sự giải thích về thuộc tính này. Lớp sử dụng khái niệm trừu tượng hóa và định nghĩa một danh sách các thuộc tính trừu tượng. Ví dụ như kích thước (size), màu sắc (color), giá cả (cost) và các hàm thao tác trên các thuộc tính này. Các thuộc tính đôi khi còn được gọi là dữ liệu thành viên (data members) bởi vì chúng chứa thông tin. Các hàm thao tác trên các dữ liệu này đôi khi còn được gọi là hàm thành viên (member function).

* + 1. Tính kế thừa

Tính kế thừa là cho phép một lớp chia sẻ các thuộc tính và phương thức đã được định nghĩa trong một hoặc nhiều lớp khác. Lớp được kế thừa gọi là lớp cha (supper class) còn lớp kế thừa được gọi là lớp con (sub class) hoặc lớp dẫn xuất (drived class). Hình vẽ 1.3 bên dưới minh họa tính kế thừa. Trong đó, lớp Nguoi là lớp cha, hai lớp con kế thừa từ lớp cha này là CongNhan và GiaoVien.

Trong lập trình hướng đối tượng khái niệm kế thừa cung cấp cho chúng ta ý tưởng sử dụng lại. Điều này có nghĩa là bạn có thể thêm các các thuộc tính và phương thức vào một lớp đã có mà không làm thay đổi chúng bằng cách tạo ra một lớp con kế thừa lớp cha này và bổ sung thêm các thuộc tính và phương thức mới. Lớp con mới tạo sẽ có các tính chất kết hợp của cả lớp cha và con. Như vậy, sức hấp dẫn và thế mạnh của tính kế thừa là cho phép người lập trình sử dụng lại các lớp.

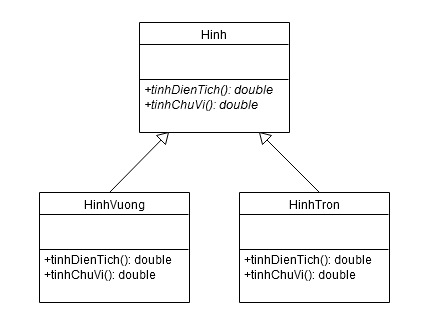


Minh họa tính chất kế thừa của OOP

* + 1. Tính đa hình

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 hàm bên trong 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 phương thức cho một loạt các đối tượng gần nhau. Nhưng khi thực thi thì dùng cùng một tên gọi mà sự thực thi của mỗi đối tượng sẽ tự động xảy ra tương ứng theo đặc tính của từng đối tượng mà không bị nhầm lẫn.

Ví dụ khi định nghĩa hai đối tượng HinhVuong và HinhTron có một phương thức chung tinhChuVi. Khi gọi phương thức này thì nếu đối tượng thuộc lớp HinhVuong sẽ tính theo công thức khác so với đối tượng thuộc lớp HinhTron. Hình vẽ 1.4 bên dưới minh họa tính đa hình của OOP. Trong hình vẽ này hai lớp HinhVuong và HinhTron kế thừa từ lớp cha Hinh. Một đặc trưng của lập trình hướng đối tượng là tổng quát hóa nghĩa là những thuộc tính và phương thức nào mà tất cả các lớp con đều có sẽ được biểu diễn ở trong lớp cha. Như trong ví dụ này, cả hai lớp con đều có phương thức chung là tính chu vi nên phương thức này sẽ được khai báo ở trong lớp cha Hinh. Do tính đa hình nên bạn có thể khai báo một đối tượng có kiểu dữ liệu là lớp cha thì nó có khả năng tham chiếu đến các đối tượng tạo ra từ các lớp con. Khi tham chiếu đến đối tượng tạo ra từ lớp con nào thì sẽ gọi đúng phương thức được định nghĩa trong lớp con đó.



Minh họa tính đa hình của OOP

Như vậy, nếu chúng ta khai báo một biến có kiểu dữ liệu là lớp cha Hinh. Khi biến này tham chiếu đến đối tượng của lớp con HinhVuong và gọi phương thức tinhChuVi() sẽ tiến hành tính chu vi vủa hình vuông còn nếu nó tham chiếu đến đối tượng thuộc lớp HinhTron thì gọi phương thức tinhChuVi() sẽ tính chu vi của hình tròn.

* 1. Giới thiệu về ngôn ngữ lập trình Java
     1. Giới thiệu ngôn ngữ lập trình Java

Java là một ngôn ngữ lập trình được Sun Microsystems giới thiệu vào tháng 6 năm 1995. Từ đó, nó đã trở thành một công cụ lập trình của các lập trình viên chuyên nghiệp. Java được xây dựng trên nền tảng của C và C++. Do vậy nó sử dụng các cú pháp của C và các đặc trưng hướng đối tượng của C++.

Vào năm 1991, một nhóm các kỹ sư của Sun Microsystems có ý định thiết kế một ngôn ngữ lập trình để điều khiển các thiết bị điện tử như Tivi, máy giặt, lò nướng, … Mặc dù C và C++ có khả năng làm việc này nhưng trình biên dịch lại phụ thuộc vào từng loại CPU.

Trình biên dịch thường phải tốn nhiều thời gian để xây dựng nên rất đắt. Vì vậy để mỗi loại CPU có một trình biên dịch riêng là rất tốn kém. Do đó nhu cầu thực tế đòi hỏi một ngôn ngữ chạy nhanh, gọn, hiệu quả và độc lập thiết bị tức là có thể chạy trên nhiều loại CPU khác nhau, dưới các môi trường khác nhau. “Oak” đã ra đời và vào năm 1995 được đổi tên thành Java. Mặc dù mục tiêu ban đầu không phải cho Internet nhưng do đặc trưng không phụ thuộc thiết bị nên Java đã trở thành ngôn ngữ lập trình cho Internet. Hiện tại, Java đã thuộc về Oracle sau thương vụ mua lại Sun vào 20.04.2009 với trị giá 7.4 tỷ USD.

Java là ngôn ngữ lập trình hướng đối tượng, do vậy không thể dùng Java để viết một chương trình hướng chức năng. Java có thể giải quyết hầu hết các công việc mà các ngôn ngữ khác có thể làm được.

Java là ngôn ngữ vừa biên dịch vừa thông dịch. Đầu tiên mã nguồn được biên dịch bằng công cụ JAVAC để chuyển thành dạng ByteCode. Sau đó được thực thi trên từng loại máy cụ thể nhờ chương trình thông dịch. Mục tiêu của các nhà thiết kế Java là cho phép người lập trình viết chương trình một lần nhưng có thể chạy trên bất cứ phần cứng cụ thể.

Ngày nay, Java được sử dụng rộng rãi để viết chương trình chạy trên Internet. Java là ngôn ngữ lập trình hướng đối tượng độc lập thiết bị, không phụ thuộc vào hệ điều hành. Java không chỉ dùng để viết các ứng dụng chạy đơn lẻ hay trong mạng mà còn là nền tảng để xây dựng các trình điều khiển thiết bị cho điện thoại di động…

* + 1. Các đặc trưng của Java
* Đơn giản
* Hướng đối tượng
* Độc lập phần cứng và hệ điều hành
* Mạnh
* Bảo mật
* Phân tán
* Đa luồng
* Động
  + - 1. Đơn giản

Những người thiết kế mong muốn phát triển một ngôn ngữ dễ học và quen thuộc với đa số người lập trình. Do vậy Java được loại bỏ các đặc trưng phức tạp của C và C++ như thao tác con trỏ, thao tác nạp đè (overload),… Java không sử dụng lệnh “goto” cũng như file header (.h). Cấu trúc “struct” và “union” cũng được loại bỏ khỏi Java.

* + - 1. Hướng đối tượng

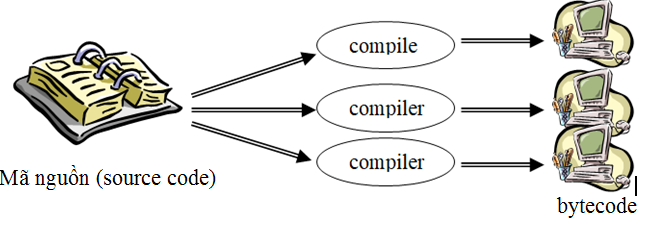
Java được thiết kế xoay quanh mô hình hướng đối tượng. Vì vậy trong Java, tiêu điểm là dữ liệu và các phương thức thao tác lên dữ liệu đó. Dữ liệu và các phương thức mô tả trạng thái và cách ứng xử của một đối tượng trong Java.

* + - 1. Độc lập phần cứng và hệ điều hành

Đây là khả năng một chương trình được viết tại một máy nhưng có thể chạy được bất kỳ đâu. Chúng được thể hiện ở mức mã nguồn và mức nhị phân.

Ở mức mã nguồn, người lập trình cần mô tả kiểu cho mỗi biến. Kiểu dữ liệu trong Java nhất quán cho tất cả các hệ điều hành và phần cứng khác nhau. Java có riêng một thư viện các lớp cơ sở. Vì vậy chương trình Java được viết trên một máy có thể dịch và chạy trơn tru trên các loại máy khác mà không cần viết lại.

Ở mức nhị phân, một chương trình đã biên dịch có thể chạy trên nền khác mà không cần dịch lại mã nguồn. Tuy vậy cần có phần mềm máy ảo Java (sẽ đề cập đến ở phần sau) hoạt động như một trình thông dịch tại máy thực thi.



Minh họa đặc trưng độc lập phần cứng của Java

Trình biên dịch sẽ chuyển các chương trình viết bằng C, C++ hay ngôn ngữ khác thành mã máy nhưng phụ thuộc vào CPU. Nên khi muốn chạy trên loại CPU khác, chúng ta phải biên dịch lại chương trình.

Môi trường phát triển của Java được chia làm hai phần: Trình biên dịch và trình thông dịch. Không như C hay C++, trình biên dịch của Java chuyển mã nguồn thành dạng bytecode độc lập với phần cứng mà có thể chạy trên bất kỳ CPU nào. Nhưng để thực thi chương trình dưới dạng bytecode, tại mỗi máy cần phải có trình thông dịch của Java hay còn gọi là máy ảo Java. Máy ảo Java chuyển bytecode thành mã lệnh mà CPU thực thi được.

* + - 1. Mạnh mẽ

Java yêu cầu chặt chẽ về kiểu dữ liệu và phải mô tả rõ ràng khi viết chương trình. Chúng sẽ kiểm tra lúc biên dịch và cả trong thời gian thông dịch. Vì vậy, Java loại bỏ các kiểu dữ liệu dễ gây ra lỗi.

* + - 1. Bảo mật

Java cung cấp một số lớp để kiểm tra bảo mật. Ở lớp đầu tiên, dữ liệu và các phương thức được đóng gói bên trong lớp. Chúng chỉ được truy xuất thông qua các giao diện mà lớp cung cấp. Java không hỗ trợ con trỏ vì vậy không cho phép truy xuất bộ nhớ trực tiếp. Nó cũng ngăn chặn không cho truy xuất thông tin bên ngoài của mảng bằng kỹ thuật tràn và cũng cung cấp kỹ thuật dọn rác trong bộ nhớ. Các đặc trưng này tạo cho Java an toàn và có khả năng cơ động cao.

Trong lớp thứ hai, trình biên dịch kiểm soát để đảm bảo mã an toàn. Lớp thứ ba được đảm bảo bởi trình thông dịch. Chúng kiểm tra xem bytecode có đảm bảo các qui tắc an toàn trước khi thực thi. Lớp thứ tư kiểm soát việc nạp các lớp lên bộ nhớ để giám sát việc vi phạm giới hạn truy xuất trước khi nạp vào hệ thống.

* + - 1. Phân tán

Java dùng để xây dựng các ứng dụng có thể làm việc trên nhiều nền tảng phần cứng, hệ điều hành và giao diện đồ họa. Java được thiết kế cho các ứng dụng chạy trên mạng. Vì vậy chúng được sử dụng rộng rãi trên Internet, nơi sử dụng nhiều nền tảng khác nhau.

* + - 1. Đa luồng

Chương trình Java sử dụng kỹ thuật đa tiến trình (Multithread) để thực thi các công việc đồng thời. Java cũng cung cấp giải pháp đồng bộ giữa các tiến trình. Đặc tính hỗ trợ đa tiến trình này cho phép xây dựng các ứng dụng trên mạng chạy uyển chuyển.

* + - 1. Động

Java được thiết kế như một ngôn ngữ động để đáp ứng cho những môi trường mở. Các chương trình Java bổ sung các thông tin cho các đối tượng tại thời gian thực thi. Điều này cho phép khả năng liên kết động các mã.

* + 1. Giới thiệu và cài đặt Bộ công cụ JDK (Java Development Kit)

Sun Microsystem đưa ra ngôn ngữ lập trình Java qua sản phẩm có tên là Java Development Kit (JDK) (bộ công cụ phát triển ứng dụng Java). Ba phiên bản chính là:

* Java 1.0 - Sử dụng lần đầu vào năm 1995
* Java 1.1 – Ðưa ra năm 1997 với nhiều ưu điểm hơn phiên bản trước.
* Java 2 – Phiên bản mới nhất
* JDK bao gồm Java Plug-In, chúng cho phép chạy trực tiếp Java Applet hay JavaBean bằng cách dùng JRE thay cho sử dụng môi trường thực thi mặc định của trình duyệt.
* JDK chứa các công cụ sau:
  + - 1. Trình biên dịch, 'javac'

Cú pháp:

javac [options] sourcecodename.java

* + - 1. Trình thông dịch, 'java'

Cú pháp:

java [options] classname

* + - 1. Trình dịch ngược, 'javap'

Cú pháp:

javap [options] classname

* + - 1. Công cụ sinh tài liệu, 'javadoc'

Cú pháp:

javadoc [options] sourcecodename.java

* + - 1. Chương trình tìm lỗi - Debug, 'jdb‘

Cú pháp:

jdb [options] sourcecodename.java

hoặc

jdb -host -password [options] sourcecodename.java

* + - 1. Chương trình xem Applet , 'appletviewer‘

Cú pháp:

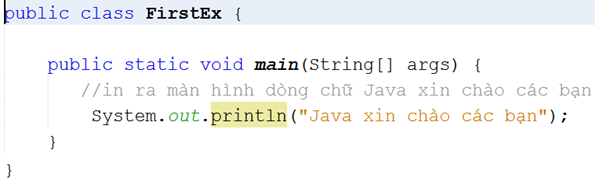
appletviewer [options] sourcecodename.java / url

1. CÁC THÀNH PHẦN CƠ BẢN TRONG JAVA

Sau khi đọc xong chương này sinh viên sẽ nắm được các kiến thức về các thành phần cơ bản trong java như cấu trúc của một chương trình, các kiểu dữ liệu, cú pháp khai báo biến số và hằng số. Chương này cũng giới thiệu các cấu trúc lập trình của java như cấu trúc tuần tự, cấu trúc rẽ nhánh và cấu trúc vòng lặp.

* 1. Giới thiệu chương trình Java đơn giản

Chúng ta xem xét một chương trình Java đơn giản hiện thị dòng thông báo ra màn hình “Java xin chào các bạn”. Nội dung của chương trình như hình vẽ 2.1 bên dưới.



Chương trình Java đơn giản

Chúng ta phân tích chi tiết dòng lệnh đầu tiên của chương trình.

* Từ khóa **public** được gọi là quyền truy cập. Quyền truy cập dùng để điều khiển các phần khác của chương trình có thể sử dụng mã này. Quyền truy cập sẽ được đề cập chi tiết trong chương 3.
* Từ khóa **class** thể hiện rằng mọi thứ của chương trình Java đều tồn tại bên trong một lớp. Theo sau từ khóa **class** là tên của lớp. Cú pháp để đặt tên lớp trong Java cũng giống như các ngôn ngữ lập trình khác là bắt đầu là chữ, các ký tự tiếp theo có thể là chữ hoặc số, không được sử dụng từ khóa để đặt cho tên lớp và các chữ cái đầu của các từ trong tên lớp nên dùng chữ hoa.
* Tiếp theo là phương thức **main,** phương thức này sẽ được chạy đầu tiên khi thực thi một chương trình Java. Như vậy một chương trình Java muốn được thi hành thì phải có một lớp chứa phương thức này.
* Dòng lệnh **System.out.println(“Java xin chào các bạn”);** dùng để in dòng thông báo ra màn hình.

Để chú thích trong Java bạn có thể sử dụng một trong 2 cách sau:

* Chú thích trong một dòng //dòng chú thích ở đây
* Chú thích nhiều dòng /\* các dòng chú thích đặt ở đây \*/

Chú thích sẽ được chương trình biên dịch của Java bỏ qua khi chuyển từ mã nguồn sang dạng bytecode.

* 1. Các kiểu dữ liệu nguyên thủy (primitive) của Java

Java cung cấp 8 kiểu dữ liêu nguyên thủy được mô tả trong bảng 2.1 sau:

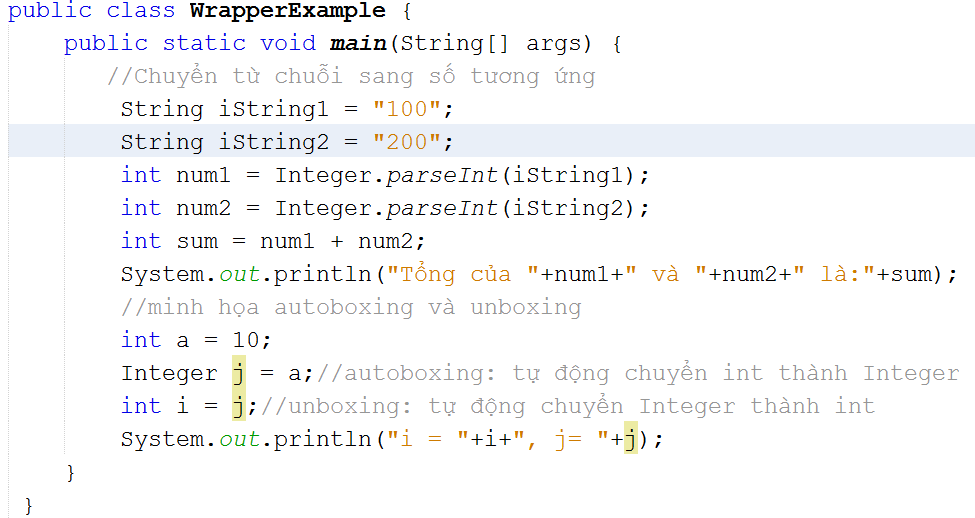
* + - * 1. Các kiểu dữ liệu nguyên thủy của Java

|  |  |  |
| --- | --- | --- |
| **Kiểu dữ liệu** | **Kích thước tính theo bit** | **Phạm vi biểu diễn** |
| byte | 8 | -128 đến 127 |
| char | 16 bit mã Unicode | Các ký tự trọng bảng mã Unicode. |
| short | 16 bit | -32768 đến 32767 |
| int | 32 bit | -2,147,483,648 đến 2,147,483,647 |
| long | 64 bit | -9,223,372,036’854,775,808 đến +9,223,372,036’854,775,807 |
| float | 32 bit biểu diễn dấu phẩy động | -3.40292347E+38 đến +3.40292347E+38 |
| double | 64 bit biểu diễn dấu phẩy động | -1,79769313486231570E+308 đến +1,79769313486231570E+308 |
| boolean | 1 bit | true hoặc false |

Tương ứng với 8 kiểu dữ liệu nguyên thủy của Java sẽ có 8 lớp Wrapper (bao bọc) tương ứng dùng để làm nhiệm vụ chuyển từ kiểu dữ liệu nguyên thủy thành đối tượng tương ứng và ngược lại; thực hiện một số thao tác với kiểu dữ liệu nguyên thủy tương ứng; ... Bắt đầu từ J2SE 5.0 bổ sung thêm chức năng autoboxing và unboxing sẽ tự động chuyển kiểu nguyên thủy thành kiểu đối tượng và ngược lại. Bảng 2.2 sau liệt kê 8 lớp bao bọc (wrapper class) tương ứng với 8 kiểu dữ liệu nguyên thủy của Java.

* + - * 1. Các lớp bao bọc tương ứng với các kiểu dữ liệu nguyên thủy

|  |  |
| --- | --- |
| **Kiểu dữ liệu nguyên thủy** | **Wrapper class** |
| byte | Byte |
| char | Character |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| boolean | Boolean |

Hình 2.2 sau minh họa chương trình Java sử dụng một số phương thức hữu ích của lớp bao bọc và đặc điểm autoboxing và unboxing.

Minh họa autoboxing và unboxing

Kết quả thu được khi chạy chương trình trên như sau:



* 1. Biến và hằng số trong Java
     1. Biến

Biến số trong Java được khai báo trong một lớp hoặc trong một phương thức, thông thường biến số được khai báo ở phần bắt đầu của lớp và phương thức, mỗi biến cần có **kiểu dữ liệu** và **tên.** Quy tắc đặt tên biến trong Java cũng giống như các ngôn ngữ lập trình khác có nghĩa là tên biến là sự kết hợp các các kí tự chữ, số, dấu gạch nối \_, ký tự đầu tiên của tên biến nên đặt là chữ và được viết thường, ký tự đầu tiên của các từ tiếp theo nên để chữ hoa. Tên biến nên đặt có ý nghĩa liên quan đến dữ liệu mà nó chứa. Tên biến không được đặt trùng với từ khóa của Java. Ví dụ chúng ta đặt tên biến là sum (tong) để phản ánh biến này chứa tổng của một dãy số.

Kiểu dữ liệu của biến có thể là một trong các kiểu dữ liệu nguyên thủy hoặc kiểu tham chiếu (mảng, lớp và giao diện (Interface)).

* + - 1. Cú pháp khai báo biến trong Java như sau:

**Kiểu dữ liệu Tên biến** [ = giá trị khởi tạo];

* Trong đó kiểu dữ liệu và tên biến là bắt buộc còn giá trị khởi tạo cho biến là tùy chọn. Bạn có thể khai báo nhiều biến có cùng kiểu dữ liệu trên một dòng, các biến được ngăn cách nhau bởi dấu phẩy “,”. Bạn nên khai báo mỗi biến trên một dòng để cho chương trình sáng sủa.
* Ví dụ sau đây minh họa khai báo 3 biến: một biến kiểu nguyên; một biến kiểu số thực 32 bit; một biến kiểu kí tự. Hai biến đầu không được khởi tạo giá trị, biến thứ 3 được khởi tạo giá trị ban đầu là kí tự ‘A’.

int i;

float f;

char c = ‘A’;

Chú ý, trong Java nếu biến khai báo mà không được khởi tạo thì các biến kiểu số sẽ mang giá trị 0, kiểu tham chiếu sẽ mang giá trị null, kiểu kí tự sẽ mang giá trị rỗng.

* + 1. Hằng số

Ngôn ngữ Java cung cấp một số loại hằng số như sau:

* Hằng số nguyên là các số nguyên không được bắt đầu bằng số 0, Java xem số bắt đầu là số 0 là số trọng hệ 8. Bảng 2.3 minh họa một số hằng số nguyên ở một số hệ đếm khác nhau:
  + - * 1. Bảng hằng số nguyên ở một số hệ đếm

|  |  |  |
| --- | --- | --- |
| **Hệ thập phân** | **Hệ mười sáu** | **Hệ tám** |
| 16 | 0x10 | 020 |
| 25 | 0X19 | 031 |

* Hằng số thực có thể bắt đầu là 0. Ví dụ một số hằng số thực: 0.5, 3.14, 2.2E10. Các hằng số thực này ngầm định là kiểu double, để xác định hằng số thực kiểu float thì kết thúc phải là f hoặc F. Ví dụ hằng số thực kiểu float 0.5f hoặc 0.5F. Chú ý, Java không cho phép thực hiện phép gán mà làm mất độ chính xác. Ví dụ sau minh họa cho hằng số thực kiểu double và float.

float fNum;

//Phép gán sau là không cho phép vì làm mất độ chính xác

fNum = 0.5;//vì 0.5 Java ngầm định đó là kiểu double

//Để phép gán hợp lệ thì phải xác định hằng số thực là float

fNum = 0.5f;

* Hằng xâu kí tự được bao trong dấu nháy kép “Hello every body”.
* Hằng kí tự là các kí tự được bao trong dấu nháy đơn. Ví dụ ‘a’ hoặc ‘A’.
* Hằng kí hiệu: Bạn có thể sử dụng một tên tương ứng với hằng số. Cú pháp để khai báo hằng số giống như khai báo biến nhưng thêm từ khóa **final**. Quy tắc đặt tên hằng là dùng chữ hoa, các từ được ngăn cách với nhau bởi dấu gạch nối \_. Ví dụ định nghĩa hằng tên DISC\_RATE và có gián trị là 10 (tỷ lệ giảm giá 10%).

final float DISC\_RATE = 10;

* 1. Các toán tử trong Java

Ngôn ngữ lập trình Java cung cấp một số loại toán tử sau:

* Toán tử số học
* Toán tử gán
* Toán tử quan hệ
* Toán tử logic
* Toán tử thao tác bit
  + 1. Toán tử số học

Bảng 2.4 sau chỉ ra các toán tử số học chính và thứ tự ưu tiên của chúng theo thứ tự giảm dần.

* + - * 1. Các toán tử số học chính của Java

|  |  |
| --- | --- |
| **Ký hiệu toán tử** | **Ý nghĩa** |
| \* | Phép nhân số học |
| / | Phép chia số học |
| % | Phép chia lấy phần dư |
| + | Phép cộng số học |
| - | Phép trừ số học |

Bạn cần chú ý rằng phép chia / và phép chia lấy phần dư %. Khi thực hiện chia hai số nguyên cho nhau kết quả thu được là phần nguyên của phép chia. Ví dụ 5/2 sẽ thu được kết quả là 2. Muốn kết quả trả về là đúng thì bạn phải ép số chia hoặc số bị chia sang số thực kiểu float hoặc double. Phép chia lấy phần dư có thể thực hiện được với cả số thực. Ví dụ sau minh họa sử dụng các phép toán số học cơ bản.

public class MathOperatorDemo {

public static void main(String[] args) {

int a = 100;

int b = 21;

int c;

float f;

f = a/b;

c = a%b;

System.out.println(a+"/"+b+" = "+f);

System.out.println(a+"%"+b+" = "+c);

}

}

Kết quả hiển thị ra màn hình khi chạy chương trình trên

*a/b = 4.0*

*a%b = 16*

* **Chú ý**: Khi chia 2 số nguyên kết quả thu được là phần nguyên. Do đó khi chia 100 cho 21 thu được kết quả là 4, mặc dù chúng ta đã lưu kết quả vào biến kiểu số thực float. Như vậy, muốn chia 2 số nguyên được kết quả đúng thì chúng ta cần ép kiểu số bị chia hoặc số chia sang số thực. Cú pháp ép kiểu chúng ta sẽ xem xét chi tiết ở phần tiếp theo.
  + 1. Toán tử gán gộp (compound assignment operators)

Bảng 2.5 bên dưới liệt kê các toán tử gán gộp trong Java. Các toán tử này thường được sử dụng khi chúng ta muốn cập nhật giá trị của một biến.

* + - * 1. các toán tử gán gộp trong Java

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Ví dụ** | **Câu lệnh tương đương** |
| += | thuong+=500 | thuong = thuong + 500 |
| –= | quy–=500 | quy = quy – 500 |
| \*= | luong\*=1.2 | luong = luong \* 1.2 |
| /= | heso/1.5 | heso = heso / 1.5 |
| %= | ngay%=7 | ngay = ngay % 7 |

* + 1. Các toán tử tăng, giảm

Java cung cấp 2 toán tử cho phép tăng hoặc giảm giá trị của một biến trong khi tính toán giá trị của một biểu thức. Bảng 2.6 bên dưới liệt kê các toán tử tăng và giảm của Java.

* + - * 1. Toán tử tăng và giảm trong Java

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Toán tử** | **Ví dụ** | **Mô tả** | **Câu lệnh tương đương** | |
| ++ | ++i; | Tăng trước | i = i + 1; | i+=1; |
| i++; | Tăng sau | i = i + 1; | i+=1; |
| -- | --i; | Giảm trước | i = i – 1; | i-=1; |
| i--; | Giảm sau | i = i – 1; | i-=1; |

* Toán tử tăng trước: Cú pháp ++<tên biến>;, với toán tử này biến sẽ được tăng lên 1 trước khi được sử dụng trong tính toán.
* Toán tử tăng sau: Cú pháp <tên biến>++; với toán tử này biến sẽ được tăng lên 1 sau khi được sử dụng trong tinh toán.
* Toán tử giảm trước: Cú pháp --<tên biến>;, với toán tử này biến sẽ được giảm đi 1 trước khi được sử dụng trong tính toán.
* Toán tử giảm sau: Cú pháp <tên biến>--;, với toán tử này biến sẽ được giảm đi 1 sau khi được sử dụng trong tính toán.

Ví dụ minh họa các phép toán tăng và giảm như đoạn chương trình dưới đây:

class InDeArithOper {

public static void main(String args[]) {

int i = 5;

int j = 10;

int k;

/\* Java không cho phép một biến được sử dụng mà chưa được khởi tạo giá trị do đó câu lệnh println bên dưới sẽ tạo ra lỗi\*/

//System.out.println("i=" + i + ",j=" + j+ ",k="

// + k);

k = 0;

System.out.println("i=" + i + ",j=" + j + ",k="   
 + k);

i++; /\* cộng 1 vào i, giá trị của i là 6 \*/

System.out.println("i=" + i + ",j=" + j + ",k="   
 + k);

++i; /\* cộng 1 vào i, giá trị của i là 7 \*/

System.out.println("i=" + i + ",j=" + j + ",k="   
 + k);

i--; /\* i trừ đi 1, giá trị của i là 6 \*/

System.out.println("i=" + i + ",j=" + j + ",k="   
 + k);

--i; /\* i trừ đi 1, giá trị của i là 5 \*/

System.out.println("i=" + i + ",j=" + j + ",k="   
 + k);

k = i++ + j; /\* k là 15, i là 6 \*/

System.out.println("i=" + i + ",j=" + j + ",k="  
 + k);

k = ++i + j; /\* k là 17, i là 7 \*/

System.out.println("i=" + i + ",j=" + j + ",k="   
 + k);

k = i-- + j; /\* k là 17, i là 6 \*/

System.out.println("i=" + i + ",j=" + j + ",k="   
 + k);

k = --i + j; /\* k là 15, i là 5 \*/

System.out.println("i=" + i + ",j=" + j + ",k="   
 + k);

}

}

Kết quả khi chạy chương trình trên như sau:

i = 5, j = 10, k = 0

i = 5, j = 10, k = 0

i = 6, j = 10, k = 15

i = 7, j = 10, k = 17

i = 6, j = 10, k = 17

i = 5, j = 10, k = 15

* + 1. Biểu thức (expression)

Biểu thức được sử dụng để tính toán ra một giá trị. Một biểu thức bao gồm một hoặc số toán hạng (giá trị) kết hợp với các toán tử (phép toán).

Tất cả các tên biến, tên hàm, tên mảng, hằng số, lời gọi hàm và tham chiếu đến mảng đều được coi là biểu thức. Áp dụng toán tử một ngôi, 2 ngôi một cách thích hợp đối với các biểu thức trên cũng thu được một biểu thức. Một biểu thức đặt trong dấu ngoặc tròn cũng là một biểu thức.

Ví dụ một số biểu thức hợp lệ trong java:

3 ; ‑2 ; num ; a = b \* c + d ; a += (b/c++)

Một biểu thức có thể được thành lập từ một số biểu thức nhỏ hơn, các biểu thức nhỏ này được gọi là biểu thức con (sub pression). Ví dụ trong biểu thức   
a += (b/c++) thì b/c++ là một biểu thức con.

Một đặt trưng quan trọng của ngôn ngữ lập trình Java là tất cả các biểu thức đều cho một giá trị. Để tính toán giá trị của một biểu thức cần áp dụng thứ tự ưu tiên của các phép toán trong biểu thức.

* + - * 1. Minh họa một số biểu thức đơn giản trong Java

|  |  |
| --- | --- |
| **Biểu thức** | **Giá trị** |
| -4+6 | 2 |
| c = 3 + 8 | 11 |
| x = 6 + (y = 5) | 11 (chú ý y được gán giá trị là 5 và biểu thức (y=5) là một biểu thức con) |

* + 1. Chuyển kiểu tự động

Khi cần tính toán biểu thức mà các thành phần có kiểu dữ liệu khác nhau (ví dụ một số nguyên và một số thực). Trong trường hợp này Java tự động chuyển các thành phần thành một kiểu chuẩn sau đó mới tiến hành tính toán giá trị của biểu thức. Nguyên tắc chuyển là tránh để mất mát thông tin. Như vậy, Java sẽ tự động chuyển từ kiểu dữ liệu có kích thước nhỏ lên kiểu có kích thước lớn hơn. Ví dụ tiến hành cộng một số nguyên với một số thực kiểu float thì số nguyên sẽ được tự động chuyển lên kiểu float sau đó mới thực hiện tính toán.

Chương trình Java bên dưới minh họa việc chuyển kiểu tự động:

public class TypeConv {

public static void main(String[] args) {

int i = 10;

float t = 5.0f;

float f;

/\*phép gán thực hiện được vì chuyển số nguyên  
thành số thực không mất độ chính xác\*/

f = i;

/\*phep gán số nguyên bằng số thực sẽ tạo ra   
lỗi biên dịch vì làm mất độ chính xác\*/

// i = f;

System.out.println("i = "+i+", f = "+f);

f = i + t; /\*i được tự động chuyển thành   
 kiểu float rồi thực hiện phép cộng\*/

System.out.println("i = "+i+", f = "+f);

}

}

Kết quả hiển thị ra màn hình khi thực hiện chương trình Java trên

i = 10, f = 10.0

i = 10, f = 15.0

* + 1. Toán tử ép kiểu (type cast operator)

Toán tự ép kiểu sẽ chuyển kiểu dữ liệu này sang một kiểu dữ liệu xác định. Phép ép kiểu cho phép chuyển từ kiểu dữ liệu có kích thước lớn xuống kiểu dữ liệu có kích thước nhỏ. Ví dụ chuyển từ số thực float thành số nguyên. Cú pháp để ép kiểu như sau: **(kiểu dữ liệu)mục dữ liệu;**

Ví dụ ép kiểu biến thực *f* thành kiểu số nguyên.

int i;

float f = 3.14f;

i = (int)f;/\*không báo lỗi vì chúng ta chủ động ép  
 kiểu\*/

i = f;/\*báo lỗi vì chuyển kiểu sẽ bị mất độ chính xác   
 nên không được phép\*/

Chương trình Java sau minh họa việc ép kiểu.

public class TypeCastDemo {

public static void main(String[] args) {

int i = 10;

int j = 3;

float f;

//i = f; lỗi vì mất độ chính xác

f = 7.5F;

i = (int)f;/\*không lỗi vì chúng ta chủ động   
 chuyển kiểu\*/

System.out.println("i = "+i+", f = "+f);

f = i/j;/\*chia lấy phần nguyên vì cả số bị   
 chia và số chia đều là nguyên\*/

System.out.println("i = "+i+", f = "+f);

f = (float)i/j;/\*ép số bị chia thành số thực   
 float\*/

System.out.println("i = "+i+", f = "+f);

f = i/(float)j;/\*ép số chia thành số thực   
 float\*/

System.out.println("i = "+i+", f = "+f);

f = (float)i/(float)j;/\*ép cả số bị chia và   
 số chia thành float\*/

System.out.println("i = "+i+", f = "+f);

}

}

Kết quả hiển thị lên màn hình khi thực hiện chương trình Java trên:

i = 7, f = 7.5

i = 7, f = 2.0

i = 7, f = 2.3333333

i = 7, f = 2.3333333

i = 7, f = 2.3333333

* 1. Biểu thức logic

Biểu thức logic có giá trị là **true** hoặc **false.** Biểu thức logic thường được thành lập từ các toán tử quan hệ và logic.

* + 1. Toán tử quan hệ (relational operators)

Toán tử quan hệ được sử dụng để thành lập lên các biểu thức logic. Bảng 2.8 sau liệt kê các toán tử quan hệ của ngôn ngữ lập trình Java.

* + - * 1. Các toán tử quản hệ của Java

|  |  |
| --- | --- |
| **Toán tử quan hệ** | **Mô tả** |
| != | Khác |
| == | Bằng |
| > | Lớn hơn |
| < | Nhỏ hơn |
| >= | Lớn hơn hoặc bằng |
| <= | Nhỏ hơn hoặc bằng |

Ví dụ minh họa:

int a = 25, b = 30, c = 25, d = 0;

a < b cho kết quả là true

a == c cho kết quả là true

a > c cho kết quả là false

a != d cho kết quả là true

* + 1. Toán tử logic

Toán tử logic được dùng kết hợp với các toán tử quan hệ để thành lập nên các biểu thức logic. Bảng 2.9 sau liệt kê các toán tử logic của ngôn ngữ lập trình Java.

* + - * 1. Các toán tử logic của Java

|  |  |
| --- | --- |
| **Toán tử** | **Ý nghĩa** |
| && | Thực hiện phép AND logic |
| || | Thực hiện phép OR logic |
| ! | Thực hiện phép phủ định NOT |

Để thực hiện các toán tử logic bên trên cần dựa vào bảng chân lý (true table) của các toán tử này như trong các bảng từ 2.10 đến 2.12 sau:

* + - * 1. Bảng chân lý của toán tử && (AND)

|  |  |  |
| --- | --- | --- |
| **x** | **y** | **x && y** |
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

* + - * 1. Bảng chân lý của toán tử || (OR)

|  |  |  |
| --- | --- | --- |
| **x** | **y** | **x || y** |
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

* + - * 1. Bảng chân lý của toán tử ! (NOT)

|  |  |
| --- | --- |
| **x** | **!x** |
| false | true |
| true | false |

* 1. Các toán tử thao tác bit (bitwise operators)

Ngôn ngữ lập trình Java cung cấp các phép toán thao tác bit được liệt kê trong bảng 2.13 sau:

* + - * 1. Các phép toán thao tác bit

|  |  |
| --- | --- |
| **Toán tử** | **Ý nghĩa** |
| ~ | Phép phủ định bit |
| & | Phép AND bit |
| | | Phép OR bit |
| ^ | Phép XOR bit |
| >> | Phép dịch phải bit số có dấu |
| << | Phép dịch trái bít số có dấu |
| >>> | Phép dịch phải số không dấu |

* + 1. Phép phủ định bit

Đấy là phép toán một ngôi, thực hiện phủ định từng bit của toán hạng theo bảng 2.14 sau:

* + - * 1. Bảng chân lý của phép phủ định bit

|  |  |
| --- | --- |
| **x** | **~x** |
| 0 | 1 |
| 1 | 0 |

Ví dụ minh họa:

Giả sử a = 10101010b thì ~a = 01010101b (b là ký hiệu cho số ở hệ nhị phân)

* + 1. Phép toán AND bit

Phép toán AND bit là phép toán 2 ngôi, thực hiện phép AND từng bit của toán hạng thứ nhất với bit tương ứng của toán hạng thứ 2 theo bảng chân lý 2.15 sau:

* + - * 1. Bảng chân lý của phép AND bit

|  |  |  |
| --- | --- | --- |
| **x** | **y** | **x&y** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Ví dụ: Giả sử *x* = 10101010b và *y* = 11001100b thì *x&y* = 10001000b.

* + 1. Phép toán OR bit

Phép toán OR bit là phép toán 2 ngôi, thực hiện phép OR từng bit của toán hạng thứ nhất với bit tương ứng của toán hạng thứ 2 theo bảng chân lý 2.16 sau:

* + - * 1. Bảng chân lý của phép OR bit

|  |  |  |
| --- | --- | --- |
| **x** | **y** | **x|y** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Ví dụ minh họa: Giả sử *x* = 10101010b và *y* = 11001100b thì *x|y* = 11101110b.

* + 1. Phép toán XOR bit

Phép toán XOR bit là phép toán 2 ngôi, thực hiện phép XOR từng bit của toán hạng thứ nhất với bit tương ứng của toán hạng thứ 2 theo bảng chân lý 2.17 sau:

* + - * 1. Bảng chân lý của phép XOR bit

|  |  |  |
| --- | --- | --- |
| **x** | y | **x^y** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Ví dụ: Giả sử *x* = 10101010b và *y* = 11001100b thì *x^y* = 01100110b.

* + 1. Phép toán dịch phải số số học

Toán tử dịch phải số có dấu (dịch phải số học) là phép toán dịch các bit của một toán hạng sang phải với số bít yêu cầu và bảo toàn bit dấu (bit có trọng số cao nhất). Toán hạng ở bên trái phép toán là toán hạng cần dịch, còn toán hạng đứng bên phải là số bit cần dịch. Ví dụ muốn dịch toán hạng *x* sang phải 2 vị trí thì chúng ta viết *x*>>*2.* Phép toán dịch phải số học dùng để thực hiện phép chia cho 2n với *n* là số bit cần dịch.

Ví dụ minh họa: Giả sử *x* = 10101010b thì *x*>>1 = 11010101b

* + 1. Phép toán dịch trái số học

Toán tử dịch trái số có dấu (dịch trái số học) là phép toán dịch toàn bộ các bit của một toán hạng sang trái với số bít yêu cầu và bảo toàn bit dấu (bit có trọng số cao nhất). Toán hạng ở bên trái phép toán là toán hạng cần dịch, còn toán hạng đứng bên phải là số bit cần dịch. Ví dụ muốn dịch toán hạng *x* sang trái 2 vị trí thì chúng ta viết *x*<<*2.* Phép toán dịch trái số học dùng để thực hiện phép nhân với 2n với *n* là số bit cần dịch.

Ví dụ minh họa: Giả sử *x* = 10101010b thì *x*<<1 = 11010100b

* + 1. Phép toán dịch phải số không dấu

Toán tử dịch phải số không dấu là phép toán dịch toàn bộ các bit của một toán hạng sang phải với số bít yêu cầu. Toán hạng ở bên trái phép toán là toán hạng cần dịch, còn toán hạng đứng bên phải là số bit cần dịch. Ví dụ muốn dịch toán hạng *x* không dấu sang phải 2 vị trí thì chúng ta viết *x*>>>*2*.

Ví dụ minh họa: *x* = 10101010b thì *x*>>>1 = 01010101b

Chương trình Java sau minh họa cho các phép toán thao tác bit

public class BitWiseDemo {

public static void main(String[] args) {

// khai báo biến khởi tạo giá trị ban đầu hệ

// 16

int a = 0x000F;//a = 0000000000001111b = 15

int b = 0x2222;//b = 0010001000100010b = 8738

int c;

System.out.println("Giá trị của a ở hệ thập  
 phân:"+a);

System.out.println("Giá trị của b ở hệ thập   
 phân:"+b);

c = ~a;//c = 1111111111110000b = -16

System.out.println("~a = "+c);

c = a&b;//c = 0000000000000010b = 2

System.out.println("a & b = "+c);

c = a|b;//c = 0010001000101111b = 8751

System.out.println("a | b = "+c);

c = b>>2;//c =00001000100010000b = 2184

System.out.println("b>>2 = "+c);

c = a<<2;//c = 0000000000111100b = 60

System.out.println("a<<2 = "+c);

}

}

Kết quả hiển thị ra màn hình khi thực hiện chương trình Java trên như sau:

Giá trị của a ở hệ thập phân:15

Giá trị của b ở hệ thập phân:8738

~a = -16

a & b = 2

a | b = 8751

b>>2 = 2184

a<<2 = 60

* 1. Nhập, xuất đơn giản của Java
     1. Xuất đơn giản

Để hiện thị lên màn hình đơn giản sử dụng câu lệnh:

System.out.println(thông tin cần hiển thị);

Trong đó thông tin cần hiển thị nếu là xâu kí tự thì được bao trong cặp dấu nháy kép “”, in thêm giá trị của biến thì dùng thêm dấu cộng (+) trước tên biến.

Ví dụ minh họa:

int tuoi = 21;

System.out.println(“Tuổi của bạn là: ”+tuoi);

Câu lệnh bên trên hiển thị thông tin ra màn hình song sẽ đưa con trỏ xuống đầu dòng mới. Để hiển thị thông tin mà không đưa con trỏ xuống dòng chúng ta sử dụng câu lệnh:

System.out.print(thông tin cần hiển thị);

* + 1. Nhập đơn giản

Để nhập dữ liệu từ bàn phím bạn có thể sử dụng các phương thức của lớp **Scanner** trong trong gói **util** của Java. Các phương thức của lớp Scanner cho phép chúng ta nhập số và xâu kí tự từ bàn phím. Một số phương thức thông dụng của đối tượng lớp Scanner để nhập dữ liệu được liệt kê trong bảng 2.18 sau:

* + - * 1. Một số phương thức cơ bản của lớp Scanner

|  |  |
| --- | --- |
| **Phương thức** | **Ý nghĩa** |
| nextInt() | Nhập số nguyên từ bàn phím |
| nextLong() | Nhập số nguyên kiểu long từ bàn phím |
| nextFloat() | Nhập số thực kiểu float từ bàn phím |
| nextDouble() | Nhập số thực kiểu double từ bàn phím |
| next() | Nhập xâu kí tự từ bàn phím không có dấu cách (space) |
| nextLine() | Nhập xâu kí tự từ bàn phím chứa cả dấu cách |

Để sử dụng được các phương thức của lớp Scannner chúng ta cần thực hiện như sau:

* import gọi chưa lớp Scanner vào chương trình bằng cách sử dụng câu lệnh với cú pháp sau ở sau phần khai báo gói:

import java.util.Scanner;

* Khai báo và tạo đối tượng thuộc lớp Scanner với cú pháp sau:

Scanner sn = new Scanner(System.in);

* Ví dụ chương trình Java sau dùng để nhập 2 số nguyên từ bàn phím tính tổng và hiển thị kết quả ra màn hình.

package nhapxuatdemo;

import java.util.Scanner;

public class NhapXuatDemo {

public static void main(String[] args) {

Scanner sn = new Scanner(System.in);

int num1;

int num2;

int sum;

System.out.print("Mời nhập số thứ nhất:");

num1 = sn.nextInt();

System.out.print("Mời nhập số thứ hai:");

num2 = sn.nextInt();

sum = num1 + num2;

System.out.println("Tổng của "+num1+" và  
 "+num2+" la: "+sum);

}

}

Kết quả hiển thị lên màn hình khi chạy chương trình Java trên là:

Mời nhập số thứ nhất:10

Mời nhập số thứ hai:5

Tổng của 10 và 5 la: 15

**Chú ý**: Khi câu lệnh nhập một xâu ngay sau câu lệnh nhập một số thì trong bộ đệm đã có sẵn enter do câu lệnh nhập số để lại nên xâu nhận ngay enter mà bạn không có cơ hội nhập giá trị cho nó. Do đó, để tránh trường hợp này thì bạn cần sử dụng phương thức nextLine() để loại bỏ enter sau đó dùng phương thức này một lần nữa để nhập xâu.

Ví dụ minh họa:

Scanner sn = new Scanner(System.in);

int num;

String s;

System.out.print(“Mời nhập một số nguyên: ”);

num = sn.nextInt();

sn.nextLine();//đọc dấu enter trong bộ đệm

System.out.print(“Mời nhập một chuỗi kí tự: ”);

s = sn.nextLine();

* 1. Cấu trúc điều khiển của Java
     1. Cấu trúc lựa chọn
        1. Cấu trúc lựa chọn if

Cú pháp của cấu trúc lựa chọn if trong Java như sau:

**if**(biểu thức điều kiện){

//câu lệnh hoặc một nhóm các câu lệnh;

}

* Nếu kết quả của biểu thức điều kiện có giá trị là **true** thì khối lệnh (các câu lệnh nằm trong dấu ngoặc nhọn mở { và đóng }) sẽ được thi hành.
* Nếu kết quả là **false** thì các câu lệnh nằm sau vị trí kết thúc của **if** (câu lệnh nằm sau dấu đóng ngoặc nhọn }) sẽ được thi hành.
* Nếu chỉ có một câu lệnh ở bên trong thân của **if** thì chúng ta không cần mở và đóng ngoặc nhọn, nhưng trong thực tế lập trình thì bạn nên bổ sung cặp mở đóng ngoặc này để chương trình nhìn sáng sủa, rõ ràng và sau này dễ dàng bổ sung hoặc lược bỏ các câu lệnh trong thân của **if**.

Lưu đồ thuật toán của cấu trúc if

Biểu thức điều kiện

Một câu lệnh hoặc một số câu lệnh

true

false

Lưu đồ thuật toán của cấu trúc if

Ví dụ sau minh họa chương trình java nhập 2 số nguyên từ bàn phím sau đó in thông báo ra màn hình về sự tương quan giữa 2 số vừa nhập.

package ifdemo;

import java.util.Scanner;

public class IFDemo {

public static void main(String[] args) {

Scanner sn = new Scanner(System.in);

int num1;

int num2;

System.out.print("Mời nhập số thứ nhất:");

num1 = sn.nextInt();

System.out.print("Mời nhập số thứ hai:");

num2 = sn.nextInt();

if(num1>num2){

System.out.println("Số thứ nhất lớn hơn   
 số thứ hai");

}

if(num1<num2){

System.out.print("Số thứ nhất nhỏ hơn số  
 thứ hai");

}

if(num1==num2){

System.out.print("Số thứ nhất bằng số thứ   
 hai");

}

}

}

* + - 1. Cấu trúc lựa chọn if…else

Cú pháp

**if**(biểu thức điều kiện){

// các câu lệnh ứng với biểu thức điều kiện có

// kết quả là true;

}

**else**{

// các câu lệnh ứng với biểu thức điều kiện có

// kết quả là false;

}

* Nếu biểu thức điều kiện có kết quả là **true** thì khối lệnh nằm trong **if** sẽ được thi hành. Trái lại, thì khối các câu lệnh ở **else** sẽ được thi hành.

Lưu đồ thuật toán được trình bày như hình 2.4 sau:

Ví dụ minh họa cấu trúc if …else

public class Test {

public static void main(String args[]) {

int x = 30;

if( x < 20 ) {

System.out.print("Đây là câu lệnh if");

}else {

System.out.print("Đây là câu lệnh else");

}

}

}

Vì biểu thức điều kiện x<20 có kết quả là false nên các câu lệnh trong else sẽ được thi hành. Do đó, kết quả hiển thị lên màn hình như sau:

Đây là câu lệnh else

Biểu thức điều kiện

câu lệnh của if

câu lệnh của else

true

false

Lưu đồ thuật toán của cấu trúc if…else

Cấu trúc if…else có thể lồng nhau với cú pháp như sau:

**if**(biểu thức điều kiện 1) {

// Thực hiện khi biểu thức điều kiện 1 là true

}**else if**(biểu thức điều kiện 2) {

// Thực hiện khi biểu thức điều kiện 2 là true

}**else if**(biểu thức điều kiện 3) {

// Thực hiện khi biểu thức điều kiện 3 là true

}**else** {

// Thực hiện khi tất cả các biểu thức điều

// kiện đều false.

}

Chương trình sau minh họa cấu trúc if…else lồng nhau:

public class Test {

public static void main(String args[]) {

int x = 30;

if( x == 10 ) {

System.out.print("Giá trị của x là 10");

}else if( x == 20 ) {

System.out.print("Giá trị của x là 20");

}else if( x == 30 ) {

System.out.print("Giá trị của x là 30");

}else {

System.out.print("Đây là câu lệnh else");

}

}

}

Kết quả in ra màn hình khi thực hiện chương trình trên là:

Giá trị của x là 30

* + - 1. Toán tử điều kiện ?

Cú pháp của toán tử điều kiện trong java như sau:

(Biểu thức điều kiện) ? biểu thức 1 : biểu thức 2;

* Ví dụ minh họa chương trình java sau kiểm tra một kí tự có phải là chữ số không.

public class ToanTuQuanHe {

public static void main(String[] args) {

char ch = '0';

String s = (ch>='0'&&ch<='9')? "Kí tự số" :  
 "Kí tự chữ";

System.out.println(s);

}

}

Kết quả hiển thị ra màn hình khi thực thi chương trình trên là:

Kí tự số

* + - 1. Cấu trúc switch

Java cung cấp cấu trúc switch để cho phép chọn một trong một số các lựa chọn. Bạn có thể sử dụng cấu trúc switch để tránh phải sử dụng nhiều lệnh if ... else lồng nhau.

Cú pháp của cấu trúc switch như sau:

**switch**(biểu thức) {

**case** giá trị 1:

// các câu lệnh ứng với giá trị 1;

**break**;

**case** giá trị 2:

// các câu lệnh ứng với giá trị 2;

**break;**

// Số lượng của case không hạn chế.

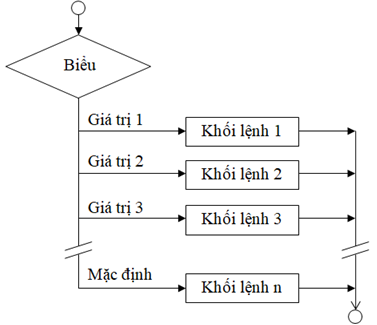
**default** : // tùy chọn mặc định

// các câu lệnh;

}

Một số quy tắc sau áp dụng cho câu lệnh switch:

* Biểu thức phải có kết quả là số nguyên, hoặc biến thể của số nguyên (short, byte, char), xâu kí tự hoặc kiểu liệt kê.
* Số lượng **case** là không hạn chế trong thân của câu lệnh **switch**, theo sau mỗi **case** là một giá trị và dấu 2 chấm :
* Kiểu dữ liệu của các giá trị sau **case** phải có cùng kiểu dữ liệu với biết thức của **switch**.
* Khi giá trị của biểu thức của **switch** trùng với giá trị nào của **case** thì các câu lệnh tương ứng với **case** đó được thực hiện cho đến khi gặp câu lệnh **break**.
* Khi gặp câu lệnh **break** thì cấu trúc **switch** kết thúc và câu lệnh đặt ngay sau **switch** sẽ được thi hành.
* Không nhất thiết **case** nào cũng phải có lệnh **break**. Khi **case** không có **break** được thi hành thì sẽ thực hiện sang các **case** kế tiếp cho đến khi gặp **break**.
* Câu lệnh **switch** có thể có trường hợp tùy chọn **default** (mặc định) và phải đặt ở cuối của câu lệnh này. Trường hợp mặc định được sử dụng để thực hiện nhiệm vụ khi giá trị biểu thức của **switch** không trùng với bất kỳ giá trị nào của **case**. Trường hợp mặc định không cần có **break**.



Lưu đồ thuật toán của switch

* **Ví dụ**: Viết chương trình Java nhập vào một điểm dạng chữ (A, B, C, D, F) từ bàn phím. Hiển thị thông báo loại ra màn hình với bảng ánh xạ sau:
  + - * 1. Xếp loại sinh viên dựa vào điểm

|  |  |
| --- | --- |
| **Điểm** | **Loại** |
| A | Xuất sắc |
| B | Giỏi |
| C | Khá |
| D, E | Trung bình |
| F | Trượt |

import java.util.Scanner;

public class SwitchDemo {

public static void main(String args[]) {

char grade = '';

Scanner sn = new Scanner(System.in);

System.out.println("Mời nhập điểm (A, B, C,  
 D, E, F):");

grade = Character.toUpperCase(sn.next().charAt(0)); switch(grade) {

case 'A':

System.out.println("Xuất sắc");

break;

case 'B':

System.out.println("Giỏi”);

break;

case 'C':

System.out.println("Khá");

break;

case 'D':

case ‘E’:

System.out.println("Trung bình");

case 'F' :

System.out.println("Trượt. Bạn phải học  
 lại");

break;

default :

System.out.println("Bạn nhập điểm không   
 hợp lệ");

}

System.out.println("Điểm của bạn là: " +   
 grade);

}

}

**Chú ý**: Trong ví dụ trên có sử dụng phương thức **charAt()** của lớp String để lấy ra kí tự tại vị trí xác định của xâu.

* + 1. Cấu trúc lặp

Cấu trúc lặp trong lập trình cho phép chúng ta thực hiện một câu lệnh hoặc một nhóm các câu lệnh nhiều lần. Ngôn ngữ lập trình Java cung cấp các cấu trúc lặp sau:

* Cấu trúc lặp kiểm tra điều kiện trước **while**
* Cấu trúc lặp với số lần lặp biết trước **for**
* Cấu trúc lặp kiểm tra điều kiện sau **do …while**
  + - 1. Cấu trúc lặp while

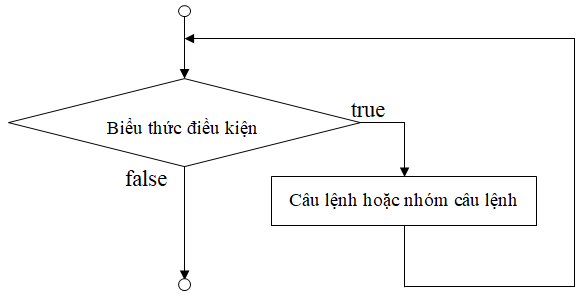
**Cú pháp**

**while** (biểu thức điều kiện){

//một câu lệnh hoặc nhóm các câu lệnh;

}

* **Hoạt động**: Khi thực hiện câu lệnh ***while*** nếu biểu thức điều kiện ở sau ***while*** có giá trị ***true*** thì các các lệnh trong thân vòng ***while*** sẽ được thi hành. Quá trình này sẽ được lặp đi lặp lại cho đến khi biểu thức điều kiện sau ***while*** có giá trị ***false***. Khi biểu thức điều kiện có giá trị ***false*** thì câu lệnh ngay sau vòng lặp ***while*** sẽ được thi hành.
* Lưu đồ thuật toán



Lưu đồ thuật toán của cấu trúc while

Ví dụ: Viết chương trình in ra màn hình các số nguyên từ 1 đến 10.

public class WhileDemo {

public static void main(String args[]) {

int i = 1;

while( i <= 10 ) {

System.out.println("Giá trị của i: " + i );

i++;

}

}

}

Kết quả khi chạy ra màn hình

Giá trị của i: 1

Giá trị của i: 2

Giá trị của i: 3

Giá trị của i: 4

Giá trị của i: 5

Giá trị của i: 6

Giá trị của i: 7

Giá trị của i: 8

Giá trị của i: 9

Giá trị của i: 10

* + - 1. Cấu trúc lặp for

**Cú pháp**

**for** (biểu thức khởi tạo; biểu thức điều kiện; biểu  
 thức cập nhật giá trị){

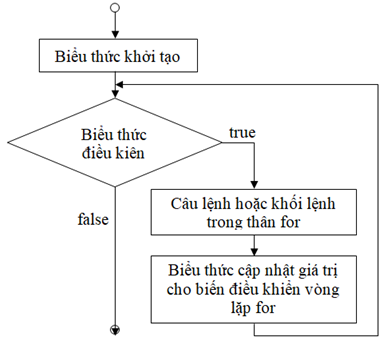
//một câu lệnh hoặc một nhóm các câu lệnh;

}

**Hoạt động**

* Khi thực hiện vòng lặp ***for*** thì biểu thức khởi tạo được thực hiện đầu tiên và chỉ thực hiện duy nhất một lần. Biểu thức này thường được sử dụng để khai báo và khởi tạo giá trị cho biến điều khiển vòng lặp.
* Tiếp theo biểu thức điều kiện được tính toán, nếu biểu thức này có giá trị là ***true*** thì các câu lệnh trong thân vòng lặp được thi hành.
* Sau đó biểu thức cập nhật giá trị cho biến điều khiển vòng lặp được thực thi, biểu thức này có thể bỏ trống.
* Tiếp theo biểu thức điều kiện lại được tính toán lại, nếu biểu thức này có giá trị ***true*** thì các công việc (thực hiện các câu lệnh ở thân vòng ***for***, cập nhật giá trị cho biến điều khiển vòng lặp, tính toán lại giá trị của biểu thức điều kiện) tiếp tục được lặp lại cho đến khi biểu thức điều kiện có giá trị là ***false*** thì câu lệnh ngay sau lệnh ***for*** sẽ được thi hành.
* Biểu thức khởi tạo, biểu thức điều kiện, biểu thức cập nhật giá trị có thể là một hoặc một số câu lệnh. Nếu là nhiều câu lệnh thì các câu lệnh được ngăn cách với nhau bởi dấu phẩy.

**Lưu đồ thuật toán**



Lưu đồ thuật toán của cấu trúc for

* **Ví dụ minh họa**: Viết chương trình tính tổng các số nguyên từ 1 đến 10. Hiển thị kết quả tính được ra màn hình.

public class Test {

public static void main(String args[]) {

int sum = 0;

for(int i = 1; i <= 10; i = i + 1) {

sum = sum + i;

}

System.out.println("Tổng các số tự nhiên từ 1   
 đến 10: " + sum);

}

}

Kết quả thực hiện chương trình trên là:

Tổng các số tự nhiên từ 1 đến 10: 55

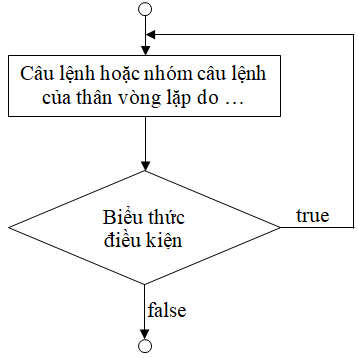
* + - 1. Cấu trúc lặp do … while
* Cú pháp

**do**{

//một câu lệnh hoặc nhóm các câu lệnh;

}**while**(biểu thức điều kiện);

* **Hoạt động**: Khi thực hiện câu lệnh ***do … while*** thì câu lệnh trong thân vòng lặp được thi hành sau đó biểu thức điều kiện được tính toán. Nếu biểu thức điều kiện có giá trị ***true*** thì các câu lệnh trong thân vòng lặp lại được thi hành. Quá trình này lặp đi lặp lại cho đến khi biểu thức điều kiện có giá trị là ***false***.
* Lưu đồ thuật toán



Lưu đồ thuật toán cấu trúc lặp do … while

**Ví dụ minh họa**: Xét chương trình Java sau:

public class DoWhileDemo {

public static void main(String[] args) {

int i = 1;

do{

System.out.println("Giá trị của i = "+i);

i++;

}while(i<=10);

}

}

Kết quả hiển thị ra màn hình khi chạy chương trình trên:

Giá trị của i = 1

Giá trị của i = 2

Giá trị của i = 3

Giá trị của i = 4

Giá trị của i = 5

Giá trị của i = 6

Giá trị của i = 7

Giá trị của i = 8

Giá trị của i = 9

Giá trị của i = 10

* **Chú ý:** Các vòng lặp có thể lồng nhau tức là trong thân của vòng lặp này lại chứa một vòng lặp khác và khi đó vòng lặp bên trong sẽ thực hiện hết số lần lặp của mình thì vòng lặp bên ngoài mới tiếp tục được lặp.

Ví dụ chương trình Java sau in ra các bảng cửu chương từ 1 đến 10.

public class MuptiplierTable {

public static void main(String[] args) {

for(int i=1;i<=10;i++){

System.out.println("Bảng cửu chương "+i);

for(int j=1;j<=10;j++){

System.out.println(i+" x "+j+" = "+i\*j);

}

}

}

}

* + - 1. Câu lệnh break và continue

Khi sử dụng các cấu trúc lặp của Java bạn có thể sử dụng câu lệnh continue và break. Lệnh continue dùng để bỏ qua lần lặp hiện tại để bắt đầu vòng lặp mới còn lệnh break dùng để kết thúc vòng lặp.

Ví dụ minh họa lệnh continue:

System.out.println("continue khi i bang 2:");

for (int i = 1; i <= 3; i++) {

if (i == 2) {

System.out.print("[continue]");

continue;

}

System.out.print("[i:" + i + "]");

}

Kết quả hiện thị lên màn hình khi chạy chương trình trên như sau:

continue khi i bang 2:

[i:1][continue][i:3]

Ví dụ minh họa lệnh break:

System.out.println("\nbreak khi i bang 2:");

for (int i = 1; i <= 3; i++) {

if (i == 2) {

System.out.print("[break]");

break;

}

System.out.print("[i:" + i + "]");

}

Kết quả hiển thị lên màn hình khi chạy đoạn chương trình trên như sau:

break khi i bang 2:

[i:1][break]

* 1. Mảng trong java
     1. Giới thiệu mảng

Khi một chương trình cần thao tác trên một tập dữ liệu có cùng đặc điểm, ví dụ như điểm của sinh viên của một lớp, trong trường hợp này thay vì sử dụng rất nhiều biến khác nhau, mỗi biến lưu điểm của một sinh viên, thì bạn có thể sử dụng một cấu trúc được gọi là mảng (array). Như vậy, mảng cho phép sử dụng một tên để tham chiếu đến một tập dữ liệu cùng kiểu. Mỗi phần tử của mảng được tham chiếu thông qua tên của mảng và một chỉ số. Chỉ số của mảng trong Java bắt đầu từ 0 (tương tự như ngôn ngữ lập trình C). Ví dụ, studMarks[5] tham chiếu đến phần tử thứ 6 của mảng StudMarks. Mỗi phần tử của mảng được sử dụng như là một biến.

Mảng có hai loại là một chiều và nhiều chiều. Ví dụ nếu chúng ta muốn lưu điểm 3 môn học của các sinh viên trong một lớp có 80 sinh viên thì chúng ta sử dụng mảng 2 chiều, marks(30,3) có 30 dòng và 3 cột, mỗi dòng tương ứng với một sinh viên trong lớp, mỗi cột tương ứng với điểm của một môn. Như vậy, marks(1,2) tham chiếu đến phần tử ở dòng thứ 2 và cột thứ 3 của mảng marks. Về mặt logic thì phần tử này lưu điểm môn thứ 3 của sinh viên thứ 2 trong lớp.

Không gian nhớ (RAM) dành cho mảng thường được thiết lập tại thời điểm khai báo biến mảng, khi chúng ta xác định số phần tử của mảng. Tuy nhiên trong Java mảng được xem như là đối tượng, do đó bộ nhớ dành cho các phần tử của mảng không được cấp phát cho tới tận khi chúng ta sử dụng lệnh **new** để tạo ra đối tượng mảng. Ví dụ, int a[]; là khai báo một mảng có tên là a và dùng để lưu các số nguyên, khi chúng ta muốn lưu 10 số nguyên vào mảng a chúng ta cần sử dụng câu lệnh a = new int[10]; một mảng 2 chiều có 4 dòng và 8 cột, lưu các số nguyên có thể được khai báo như sau: int b[][] = new int[4][8];kích thước của mảng không có giới hạn mà nó chỉ phụ thuộc vào dung lượng bộ nhớ của máy tính. Như chúng ta đã đề cập ở bên trên, mỗi phần tử của mảng có thể được tham chiếu một cách trực tiếp nhưng thế mạnh thực sự của lập trình là khi chỉ số của mảng được sử dụng kết hợp với vòng lặp.

Ví dụ: đoạn mã giả bên dưới dùng để khởi tạo 10 phần tử của một mảng chứa điểm sinh viên giá trị 0.

FOR subscript = 0 to 9

studMarks[subscript] = 0

END FOR

* + 1. Các thao tác cơ bản với mảng

Giả sử rằng chúng ta sử dụng mảng *studMarks* đã được khai báo có kích thước 80 phần tử, mỗi phần tử chứa điểm của một sinh viên trong một lớp. Sau đây chúng ta minh họa một số thuật toán (viết dưới dạng mã giả) thường gặp khi thao tác với mảng một chiều giống như mảng studMarks.

* + - 1. **Thuật toán tổng quát**

FOR student = 0 to 79

PROCESS studMarks[student]

END FOR

#### Nhập điểm cho các sinh viên từ bàn phím

FOR student = 0 to 79

input studMarks[student]

END FOR

#### Tính tổng điểm của tất cả các sinh viên

totalMark = 0

FOR student = 0 to 79

totalMark = totalMark + studMarks[student]

END FOR

PRINT totalMark

#### Hiển thị điểm của các sinh viên ra màn hình

FOR student = 0 to 79

PRINT stud\_marks[student]

END FOR

#### Tìm sinh viên có điểm bằng với điểm nhập từ bàn phím

INPUT markWanted

student = 0

WHILE (student< 80) and (studMarks[student]<>markWanted)

student = student+1

END WHILE

#### Tìm điểm cao nhất và xác định sinh viên có điểm cao nhất

maximum = studMarks[0]

maxStudent = 0

FOR student = 1 to 79

IF studMarks[student] > maximum THEN

maximum = studMarks[student]

maxStudent = student

END IF

END FOR

Trong thuật toán bên trên chúng ta không những tìm điểm cao nhất mà còn xác định sinh viên nào có điểm cao nhất. Chú ý rằng sinh viên nào có điểm cao nhất được xác định thông qua biến *maxStudent*.

* + 1. Mảng trong Java

Trong Java chúng ta có thể khai báo mảng có kiểu dữ liệu bất kỳ. Một số câu lệnh bên dưới minh họa vừa thực hiện chức năng khai báo và tạo đối tượng mảng:

int a[] = new int[10];//mảng số nguyên

float af[] = new float[200];//mảng số thực

double matrix[][] = new double[10][20]; /\* mảng 2   
 chiều \*/

static final int MAX = 100; // hằng số kích thước của  
 // mảng

Person personDB[] = new Person[MAX]; //mảng chứa các

// đối tượng

* Số nguyên ở bên trong dấu ngoặc vuông xác định số lượng phần tử của mảng. Chỉ số của mảng luôn bắt đầu từ 0. Do đó phần tử cuối cùng của mảng có chỉ số nhỏ hơn 1 so với kích thước của mảng. Chúng ta truy xuất một phần tử của mảng bằng cách sử dụng tên mảng và chỉ số đặt trong dấu ngoặc vuông.
* Chú ý, mảng personDB được khai báo để chưa các đối tượng tạo ra từ lớp Person nhưng chưa có đối tượng thuộc lớp Person được tạo.
* Bởi vì tạo ra đối tượng mảng và khai báo mảng là tách biệt nhau, nên kích thước của mảng không cần thiết phải xác định tại lúc viết chương trình mà có thể được xác định khi chạy chương trình. Ví dụ đoạn chương trình sau minh họa kích thước của mảng được xác định tại lúc chạy chương trình:

int num = 0;

int a[];

Scanner sn = new Scanner(System.in);

System.out.print(“Mời bạn nhập số phần tử của  
 mảng:”);

num = sn.nextInt();

A = new int[num];

* Trong trường hợp này kích thước của mảng a được xác định bằng cách người sử dụng nhập vào từ bàn phím.
  + - 1. Sử dụng các phần tử của mảng

Bạn cần chú ý rằng khác với ngôn ngữ lập trình C không kiểm tra việc truy cập đến các phần tử nằm ngoài phạm vi của mảng tức là trong C nếu bạn khai báo một mảng có kích thước là 100 phần tử thì khi bạn truy cập phần tử có chỉ số là 100 hoặc lớn hơn vẫn không báo lỗi. Trình biên dịch của Java kiểm tra chỉ số của mảng có nằm trong phạm vi cho phép không (từ 0 đến kích thước của mảng trừ đi 1). Nếu bạn cố tình truy cập vào phần từ nằm ngoài phạm vi cho phép của mạng thì sẽ gặp lỗi ngoại lệ khi chạy chương trình. Ví dụ như đối với mảng personDB có kích thước là 100 phần tử thì khi bạn cố truy xuất vào phần từ có chỉ số 100 (personDB[100]) hoặc lớn hơn thì sẽ gặp lỗi ngoại lệ khi thực thi chương trình.

* + - 1. Mảng đối tượng

Xét đoạn chương trình sau dùng để lưu tham chiếu đến đối tượng tạo từ lớp Person vào mảng personDB đã được khai báo ở bên trên. Thông tin chi tiết và lớp và đối tượng sẽ được đề cập trong chương 3 của giáo trình.

Person aPerson = new Person("Nguyen", "Quang");

// Vi dụ lưu đối tượng vào mảng

personDB[0] = aPerson;

Chú ý rằng với đoạn chương trình bên trên thì cả biến aPerson và phần tử đầu tiên của mảng personDB[0] cùng tham chiếu đến một đối tượng. Do đó, bất kỳ thay đổi gì đối với aPerson cũng sẽ ảnh hưởng đến personDB[0] và ngược lại. Nếu bạn muốn lưu một bản sao chi tiết của aPerson thì bạn cần tạo một đối tượng Person khác và thiết lập giá trị của các thuộc tính tương ứng với giá trị của aPerson như đoạn mã sau:

personDB[1] = new Person();

personDB[1].setSurname(aPerson.getSurname());

personDB[1].setFirstname(aPerson.getFirstname());

// các câu lệnh tương tự cho các thuộc tính còn lại

// của lớp Person

Rõ ràng chúng ta thấy việc làm bên trên là một công việc nhàm chán trừ khi lớp Person cung cấp một phương thức tạo có đối số là kiểu Person. Trong trường hợp này một bản sao mới có thể được lưu như câu lệnh sau:

personDB[2] = new Person(aPerson);

Do đó việc tạo phương thức tạo cho phép đối số là đối tượng của chính lớp đó là một ý tưởng hay và bạn nên áp dụng cho các lớp mà bạn xây dựng.

* + - 1. Khởi tạo cho mảng
* Mảng có thể được khởi tạo tại thời điểm khai báo như ví dụ sau:

int a[] = {5, 4, 3, 2, 1};

* Với ví dụ trên kích thước của mảng được xác định là số phần tử nằm trong dấu ngoặc nhọn và đặc biệt là có một toán tử **new** không tường minh xuất hiện ở đây.
  + - 1. Truyền đối số mảng cho phương thức
* Mảng có thể được sử dụng là đối số của phương thức bằng cách khai báo chúng là biến mảng sử dụng cặp đấu ngoặc vuông ([ ]) trong danh sách đối số. Ví dụ sau là phương thức tính tổng phần tử của mảng số thực, mảng cần tính là đối số của phương thức.

/\*\*

\* Tính tổng các phần tử của mảng số thực

\* @param x là mảng cần tính tổng

\* @return tổng các phần tử của mảng

\*/

double sumArray (double x[]) {

double sum = 0.0;

int k;

for (k = 0; k < x.length; k++) {

sum += x[k];

}

return (sum);

}

* Như chúng ta đã đề cập bên trên, mảng trong Java là một đối tượng và nó có thuộc tính ***length*** dùng để phản ánh số phần tử của mảng.
* Đoạn chương trình sau chứa lời gọi phương thức sumArray để tính tổng các phần tử của một mảng kiểu double:

double a[] = {1.2, 1.4, -1.1, 2.5};

double total = sumArray(a);

System.out.println(“Tổng các phần tử của mảng   
 là:”+total);

* Bạn cũng có thể cập nhật các phần tử của mảng được truyền là tham số của phương thức bởi vì mảng là đối tượng do đó chúng ta truyền một tham chiếu đến mảng cho phương thức. Chúng ta xét đoạn chương trình sau thực hiện cập nhật giá trị cho các phần tử của mảng bằng cách sử dụng đối số của phương thức là mảng:

/\*\*

\* Cập nhật giá trị cho các phần tử của mảng bằng một giá trị xác định

\* @param a mảng các số nguyên

\* @param val giá trị cần cập nhật cho các phần tử của mảng

\*/

void setArray (int a[], int val) {

int k;

for (k = 0, k < a.length; k++) {

a[k] = val;

}

}

int x[] = new int[250];

// Ví dụ gọi phương thức và truyền đối số mảng

setArray(x ,-1);

Với lời gọi và truyền tham số cho phương thức bên trên bạn đã khởi tạo giá trị cho mảng x có 250 phần tử mang giá trị -1.

* + 1. Mảng nhiều chiều
* Bạn có thể khai báo mảng với số chiều tùy ý. Tuy nhiên, mảng với số chiều lớn hơn 3 là hầu như không sử dụng. Ví dụ sau khai báo một mảng 2 chiều:

int a[][] = new int[4][3];

* Biến a được khai báo là một mảng có 4 thành phần, mỗi thành phần lại là một mảng 3 phần tử kiểu số nguyên. Như vậy, mảng hai chiều tương đương với một ma trận với số dòng là chiều thứ nhất và số cột là chiều thứ 2 của mảng 2 chiều. Tương tự như mảng một chiều, để truy xuất vào các phần tử dựa vào chỉ số của dòng và cột được đặt trong các cặp dấu ngoặc vuông: a[1][2] là phần tử ở dòng thứ 2 và cột thứ 3. Nguyên tắc này cũng áp dụng cho mảng có số chiều là 3 hoặc lớn hơn. Các câu lệnh bên dưới minh họa khai báo, tạo mảng 3 và 4 chiều:

int b[][][] = new int[2][3][4]; // 3 chiều

int c[][][][] = new int[20][50][15][4]; // 4 chiều

* Đoạn chương trình sau minh họa truy xuất vào các phần tử của mảng 2 chiều:

for (i=0; i<4; i++) {

for (j=0; j<3; j++) {

a[i][j] = i+j;

}

}

* Bạn có thể khởi tạo cho mảng nhiều chiều tương tự như cách khởi tạo cho mảng một chiều bằng cách sử dụng dấu ngoặc nhọn lồng nhau như sau:

int a[][] = { {1,2,5}, {3,2,6}, {3,2,1}, {3,8,7} };

* Bạn có thể truyền tham số mảng nhiều chiều cho phương thức giống như cách truyền mảng một chiều. Đoạn chương trình sau minh họa phương thức tính tổng các phần tử của mảng 2 chiều:

/\*\*

\* Tính tổng các phần tử của mảng 2 chiều kiểu double

\* @param x mảng 2 chiều cần tính tổng

\* @return trả về kết quả sum là tổng các phần tử của   
 mảng

\*/

double sum2d(double x[][]) {

double sum = 0.0;

int r = x.length;//số dòng của mảng

int c = x[0].length;//số cột của mảng

for (int k=0; k < r; k++) {

for ( int j=0; j<c; j++) {

sum += x[k][j];

}

}

return(sum);

}

//Khai báo mảng 2 chiều a

double a[][] = {{1,2,3},{2,3,4},{5,6,7}};

// Gọi phương thức sum2d để tính tổng các phần tử

// của mảng a

double total = sum2d(a);

System.out.println(“Tổng các phần tử của mảng a   
 la:”+total);

BÀI TẬP CHƯƠNG 2

**Bài 1.** Cho biết kết quả của các biến khi thực hiện các câu lệnh java sau:

a) b) c)

int i = 5; int i = 10; int i = 5;

int j = 6; int j = 11; int j = 2;

int x; int x; int x = i%j;

x = i++ + ++j; x = --i + j--; float y = i/j;

d) e) f)

int i = 5; int i = 5; int i = 5;

int j = 2; int j = 2; int j = 2;

float x = (float)i/j; float x = i/(float)j; float x = (float)i/(float)j;

**Bài 2.** Cho biết kết quả của các biểu thức logic sau:

1. 2 < 3 || 4 > 3 b) 2 < 3 && 4 > 3 c) 2 < 3 && 4 < 3

d) !(2 > 4) e) 2 == 3 f) 4 != 3

**Bài 3.** Cho biết kết quả khi thực hiện các phép toán thao tác bít sau với a = 10 và   
b = 100:

1. a | b b) a & b c) a ^ b

d) ~ a e) a >> 2 f) b << 2

**Bài 4.** Cho biết kết quả hiển thị ra màn hình khi thực hiện chương trình java sau:

public class DemoWhile {

public static void main(String[] args) {

int i = 1;

while(i<=20){

if(i%2==0){

System.out.println("i = "+i);

i++;

}

}

}

}

**Bài 5.** Viết chương trình java nhập vào từ bàn phím một số nguyên, một số thực, một kí tự và một chuỗi kí tự. Hiển thị ra màn hình các thông tin nhập được.

**Bài 6.** Viết chương trình java nhập vào 2 số nguyên từ bàn phím. Tính và hiển thị ra màn hình tổng, hiệu, tích và thương của 2 số nhập được.

**Bài 7.** Viết chương trình java nhập vào 2 số thực từ bàn phím và một kí tự (‘A’, ‘S’, ‘M’, ‘D’). Căn cứ vào kí tự nhập được hãy thực hiện các công việc sau:

* Nếu kí tự là ‘A’ thì tính tổng 2 số vừa nhập và hiển thị kết quả ra màn hình.
* Nếu kí tự là ‘S’ thì tính hiệu 2 số vừa nhập và hiển thị kết quả ra màn hình.
* Nếu kí tự là ‘M’ thì tính tích 2 số vừa nhập và hiển thị kết quả ra màn hình.
* Nếu kí tự là ‘D’ thì tính thương 2 số vừa nhập và hiển thị kết quả ra màn hình.
* Nếu không phải các kí tự trên thì hiển thị thông báo “*Kí tự vừa nhập không hợp lệ*”

**Bài 8.** Viết chương trình Java thực hiện tính và hiển thị ra màn hình tổng các số chẵn từ 1 đến 100.

**Bài 9.** Viết chương trình Java thực hiện tính và hiển thị ra màn hình tổng các số lẻ từ 1 đến 100.

**Bài 10.** Viết chương trình Java nhập vào hai cạnh của một hình chữ nhật từ bàn phím. Tính và hiển thị diện tích và chu vi ra màn hình.

**Bài 11.** Viết chương trình Java nhập vào bán kính của một hình tròn. Tính và hiển thị ra màn hình diện tích và chu vi của hình tròn có bán kính vừa nhập.

**Bài 12.** Viết chương trình Java nhập vào điểm thi của sinh viên trong thang điểm 100, sau đó hiển thị loại của sinh viên đó theo bảng sau:

* + - * 1. Quan hệ giữa điểm và xếp loại

|  |  |
| --- | --- |
| **Điểm** | **Xếp loại** |
| Từ 90 đến 100 | A |
| Từ 80 đến cận 90 | B |
| Từ 70 đến cận 80 | C |
| Từ 50 đến cận 70 | D |
| Nhỏ hơn 50 | F |

**Bài 13.** Viết chương trình Java nhập vào giá bán và phần trăm giảm giá của một sản phẩm. Tính và hiển thị ra màn hình giá bán sau khi giảm.

**Bài 14.** Viết chương trình Java nhập vào từ bàn phím số điện tiêu thụ trong tháng của một hộ gia đình. Tính và hiển thị ra màn hình số tiền phải thanh toán trong tháng của hộ gia đình đó. Biết đơn giá được tính theo bảng sau:

* + - * 1. Quan hệ giữa số điện và đơn giá

|  |  |
| --- | --- |
| **Số điện** | **Đơn giá** |
| Từ 0 đến 50 | 1484 |
| Từ 51 đến 100 | 1533 |
| Từ 101 đến 200 | 1786 |
| Từ số 201 trở đi | 2242 |

**Bài 15.** Viết chương trình Java nhập vào từ bàn phím số ngày lưu trú và đơn giá ngày của một nhà nghỉ từ bàn phím. Tính và hiển thị ra màn hình tiền phòng phải thanh toán biết rằng quan hệ giữa số ngày nghỉ và đơn giá như bảng sau:

* + - * 1. Mối quan hệ giữa số ngày nghỉ và đơn giá

|  |  |
| --- | --- |
| **Số ngày** | **Đơn giá** |
| 1 ngày | Bằng đơn giá |
| Từ ngày đến ngày thứ 4 | Giảm 20% |
| Từ ngày thứ 5 đến ngày thứ 10 | Giảm 40% |
| Từ ngày thứ 11 trở đi | Giảm 60% |

**Bài 16.** Viết chương trình nhập vào số nguyên n từ bàn phím. Tính và hiển thị giai thừa n ra màn hình. Biết rằng n giai thừa được tính như sau: n!=1\*2\*3\*…\*n.

**Bài 17.** Viết chương trình Java tìm và hiển thị tất cả các số có 3 chữ số thỏa mãn điều kiện sau:

**Bài 18.** Viết chương trình Java nhập một số nguyên từ bàn phím. Đưa ra màn hình thông báo số vừa nhập có phải là số nguyên tố hay không. Biết rằng số nguyên tố là số chỉ chia hết cho 1 và chính nó.

**Bài 19.** Viết chương trình Java nhập vào số giờ, số phút, số giây sau đó tính và hiển thị ra màn hình tổng số giây được quy đổi từ số giờ, số phút và số giây đã nhập.

**Bài 20.** Viết chương trình Java nhập vào nhiệt độ C. Tính và hiển thị ra màn hình nhiệt độ tương ứng ở độ F. Biết rằng mối quan hệ giữa 2 nhiệt độ như sau:

0C = (0F – 32)/1.8

**Bài 21.** Viết chương trình Java nhập vào số phần tử của mảng n và mảng số nguyên a. Hiển thị các phần tử vừa nhập được ra màn hình.

**Bài 22.** Viết chương trình Java nhập vào số phần tử của mảng n và mảng số thực a. Tính tổng giá trị của các phần tử và hiển thị giá trị tính được ra màn hình.

**Bài 23.** Viết chương trình Java nhập vào số sinh viên n và một mảng chứa điểm của n sinh viên (điểm là số nguyên tính trong thang 100). In ra màn hình điểm cao nhất, sinh viên có điểm cao nhất và điểm của tất cả các sinh viên theo chiều tăng dần.

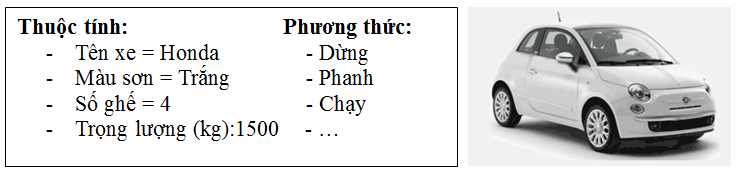
**Bài 24.** Viết chương trình Java nhập vào số lượng sinh viên của một lớp n và điểm thi tiếng anh của các sinh viên này (điểm thi là số nguyên và chấm trong thang 100). Nhập một điểm thi bất kỳ từ bàn phím, tìm sinh viên có điểm bằng điểm vừa nhập.

1. LỚP VÀ ĐỐI TƯỢNG TRONG JAVA

Sau khi học xong chương này sinh viên sẽ nắm được các kiến thức về lập trình hướng đối tượng với java như trừu tượng dữ liệu, che dấu thông tin, kế thừa, đa hình. Biết cách định nghĩa lớp và khai báo đối tượng trong java.

* 1. Đối tượng và lớp trong Java
     1. Đối tượng

Mỗi đối tượng (object) trong thế giới thực đều có các thuộc tính (properties) và các hành động. Ví dụ đối tượng xe hơi có thuộc tính là: tên, màu sơn, trọng lượng, số ghế ngồi, model và các hành động như dừng, chạy, phanh, … Mỗi đối tượng trong thế giới thực đều có thể mô hình hóa thành một đối tượng tương ứng trong ngôn ngữ lập trình hướng đối tượng nói chung và ngôn ngữ lập trình Java nói riêng, với các thuộc tính tương đương với các trường (fields) dữ liệu hoặc thuộc tính và các hành động tương ứng với các hàm hay còn gọi là phương thức (methods). Hình vẽ 3.1 bên dưới minh họa đối tượng xe ô tô trong thế giới thực và các thuộc tính và phương thức tương ứng với đối tượng đó trong ngôn ngữ lập trình Java.

Minh họa đối tượng thực và đối tượng trong ngôn ngữ lập trình

* + 1. Lớp (class)

Lớp là bản thiết kế dùng để tạo ra các đối tượng. Cú pháp để định nghĩa lớp trong ngôn ngữ lập trình Java như sau:

[Quyền truy cập] **class** <Tên lớp> {

//Khai báo các thuộc tính của lớp

[quyền truy cập] <kiểu dữ liệu 1> <tên thuộc tính 1>[=giá trị khởi tạo];

[quyền truy cập] <kiểu dữ liệu 2> <tên thuộc tính 2>[=giá trị khởi tạo];

…

[quyền truy cập] <kiểu dữ liệu n> <tên thuộc tính n>[=giá trị khởi tạo];

//Định nghĩa phương thức khởi tạo

//Định nghĩa các phương thức khác

[quyền truy cập] <kiểu dự liệu trả về> tên  
 phương thức([danh sách đối số hình thức]){

//Các câu lệnh của phương thức

}

}

Trong đó:

* **class** là từ khóa của Java dùng để định nghĩa lớp.
* Những thông tin đặt trong cặp dấu [] là thông tin tùy chọn nên có thể có hoặc không. Những thông tin đặt trong cặp dấu <> là bắt buộc phải có.
* Quyền truy cập có thể là: **public**, **private**, **protected** hoặc để trống.
* Tên lớp nên viết hoa các chữ cái đầu của các từ, tên thuộc tính và phương thức chỉ viết hoa chữ cái đầu của các từ bắt đầu từ thứ 2. Tên thuộc tính, lớp và phương thức không được đặt trùng với từ khóa và không được bắt đầu là số.

Ví dụ định nghĩa lớp XeHoi (xe hơi) như sau:

public class XeHoi{

private String ten;

private String mau;

private int soGhe;

private float trongLuong;

public XeHoi()

{

/\*Phương thức tạo không đối số dùng để khởi tạo giá trị mặc định cho các thuộc tính\*/

}

public XeHoi(String ten, String mau, int soGhe,float  
 trongLuong){

this.ten = ten;

this.mau = mau;

this.soGhe = soGhe;

this.trongLuong = trongLuong;

}

//Các phương thức dùng để thiết lập các thuộc

// tính lớp

public void setTen(String ten){

this.ten = ten;

}

//các phương thức thiết lập cho các thuộc tính còn lại

// Các phương thức dùng để lấy giá trị của các

// thuộc tính

public String getTen(){

return this.ten;

}

//các phương thức lấy giá trị của các thuộc tính còn lại

}

Đối tượng được tạo ra từ lớp với cú pháp như sau:

**Tên­­\_lớp** tên\_đối\_tượng = **new** Tên\_lớp([các giá trị   
 khởi tạo]);

Trong đó **new** là từ khóa của Java dùng để tạo đối tượng mới.

Ví dụ tạo đối tượng **xe** từ lớp XeHoi bên trên như sau:

XeHoi xe = **new** XeHoi();

* 1. Thuộc tính và phương thức của lớp
     1. Thuộc tính

Thuộc tính của một lớp là một biến được khai báo bên trong lớp và bên ngoài tất cả các phương thức (hàm).

Cú pháp để khai báo thuộc tính như sau:

[Quyền truy cập] <kiểu dữ liệu> <tên thuộc tính>[=giá trị khởi tạo];

Trong đó:

* Quyền truy cập có thể bỏ trống hoặc là **public, private** hoặc **protected**. Quyền truy cập sẽ được đề cập chi tiết ở phần sau của sách. Để che dấu thông tin các thuộc tính nên khai có quyền truy cập là **private**.
* Kiểu dữ liệu có thể là các kiểu dữ liệu nguyên thủy của Java hoặc là kiểu lớp.
* Tên thuộc tính được viết hoa chữ cái đầu từ từ thứ 2 và không được đặt trùng với các từ khóa của java và không được bắt đầu bằng số.

Ví dụ khai báo thuộc tính tên của một người như sau:

private String ten;

* + 1. Phương thức

Phương thức của Java là một tập các câu lệnh được nhóm lại với nhau để thực hiện một nhiệm vụ.

* + - 1. Cú pháp định nghĩa phương thức như sau:

[Quyền truy cập][Tiền tố] [Kiểu dữ liệu trả về] <Tên phương thức>([danh  
 sách đối số]){

//các câu lệnh trong thân phương thức

}

Trong đó:

* Quyền truy cập giống như của khai báo thuộc tính
* Tiền tố dùng để xác định phương thức có thể là static, final, abstract, synchronized. Phương thức có tiền tố là static (tĩnh) hay còn gọi là phương thức lớp có nghĩa là gọi phương thức thông qua tên lớp chứ không thông qua đối tượng. Phương thức có tiền tố là final để không cho phép lớp con viết lại (overriding) phương thức này của lớp cha. Phương thức có tiền tố abstract có nghĩa là phương thức trừu tượng, chúng ta chỉ cần khai báo nguyên mẫu của phương thức mà không cần định nghĩa thân. Phương thức có tiền tố synchronized dùng để ngăn chặn một phương thức của một đối tượng bị gọi đồng thời bởi nhiều luồng khác nhau trong lập trình đa luồng.
* Kiểu dữ liệu trả về là tùy chọn, phần này không có đối với phương thức tạo, còn các phương thức khác bắt buộc phải có kiểu trả về và có thể là kiểu void, kiểu dữ liệu nguyên thủy hoặc kiểu đối tượng.
* Tên phương thức bắt buộc phải có, tên phương thức nên đặt để phản ánh nhiệm vụ của nó, chữ cái đầu của từ đầu tiên là chữ thường, còn của các từ tiếp theo là chữ hoa. Tên phương thức phải bắt đầu là ký tự chữ, không chứa dấu cách và không được đặt trùng với từ khóa của Java.
* Danh sách đối số là tùy chọn, có phương thức có đối số, có phương thức không có đối số. Khi muốn thực hiện một nhiệm vụ với các giá trị đầu vào khác nhau thì phương thức phải có đối số.
* Các câu lệnh trong thân của phương thức dùng để thực hiện nhiệm vụ của phương thức.

Ví dụ minh họa phương thức tính bình phương của một số nguyên.

public int binhPhuong(int n){

return n\*n;

}

* + - 1. Danh sách đối số

Khi chúng ta muốn thức hiện một thao tác với các thông tin đầu vào khác nhau thì cần xây dựng một phương thức có danh sách đối số. Ví dụ chúng ta muốn tính bình phương của nhiều số nguyên khác nhau thì chúng ta xây dựng phương thức tính bình phương một số nguyên như bên trên với đối số *n,* đối số này gọi là đối số hình thức (formal parameters). Sau đó muốn tính bình phương của số nguyên nào chỉ cần gọi phương thức ***binhPhuong*** và truyền vào giá trị của số nguyên chúng ta cần tính bình phương, giá trị truyền vào này được gọi là đối số thực (actual parameters). Trong ngôn ngữ lập trình Java chúng ta có thể coi đối số hình thức là các biến cục bộ và nó được khởi tạo giá trị bằng đối số thực khi phương thức được gọi. Ví dụ khi ta gọi phương thức ***binhPhuong(4)*** thì biến ***n*** được khởi tạo giá trị bằng 4 và thực hiện các câu lệnh trong thân phương thức đó là câu lệnh trả về kết quả của ***n*\**n*** tức là 16. Trong Java nếu đối số có kiểu dữ liệu nguyên thủy thì việc truyền đối số là truyền theo kiểu giá trị, có nghĩa là một bản sao của đối số thực được tạo ra và phương thức sẽ sử dụng bản sao này nên đối số gốc không bị thay đổi. Nếu đối số truyền vào là kiểu đối tượng thì được gọi là truyền theo kiểu tham chiếu (reference) và phương thức sẽ thao tác trực tiếp trên đối số thực truyền vào.

* + - 1. Kết quả trả về từ phương thức

Để trả về kết quả cho phương thức gọi thì trong thân của phương thức bị gọi chứa câu lệnh ***return biểu thức;***

* + - 1. Nạp chồng phương thức (method overloading)

Một đặc điểm nổi bật của ngôn ngữ lập trình Java và một số ngôn ngữ lập trình khác ví dụ như C++ là khả năng sử dụng một tên cho nhiều phương thức (nhiều phương thức có cùng tên). Các phương thức có cùng tên nhưng phải có số lượng đối số hoặc kiểu giá trị của các đối số khác nhau vì khi gọi một phương thức Java không chỉ căn cứ vào tên mà còn căn cứ vào kiểu và số lượng đối số thực trong lời gọi phương thức để gọi chính xác phương thức mà chúng ta cần. Điều này giúp chúng ta chỉ cần nhớ một tên của phương thức nhưng thực hiện với các đối số khác nhau.

Ví dụ sau minh họa hai phương thức có cùng tên là findLargest dùng để trả về số lớn nhất trong ba số nguyên và 3 số thực như sau:

public int **findLargest**(int n1,int n2,int n3)

{

int largest = n1;

if(n2>largest){

largest = n2;

}

if(n3>largest){

largest = n3;

}

return largest;

}

public float **findLargest**(float n1,float n2,float n3)

{

float largest = n1;

if(n2>largest){

largest = n2;

}

if(n3>largest){

largest = n3;

}

return largest;

}

* Như vậy, hai phương thức định nghĩa bên trên là khác nhau vì phương thức thứ nhất có kiểu dữ liệu của đối số là số nguyên còn phương thức thứ hai có kiểu dữ liệu của đối số là số thực. Do đó, hai phương thức này là khác nhau về nguyên mẫu (signature). Chú ý kiểu dữ liệu trả về của phương thức không phải là một phần của nguyên mẫu. Ví dụ sau đây minh họa hai phương thức sau không thể đồng thời tồn tại vì chúng không khác nhau về nguyên mẫu.

public int **findLargest**(int n1, int n2, int n3)

public float **findLargest**(int n1, int n2, int n3)

Số lượng đối số là một phần của nguyên mẫu. Do đó 2 phương thức sau là khác nhau vì chúng khác nhau về nguyên mẫu.

public int **findLargest**(int n1, int n2, int n3, int n4)

public int **findLargest**(int n1, int n2, int n3)

* 1. Phương thức tạo (constructor)

Phương thức tạo trong Java dùng để khởi tạo giá trị ban đầu cho các thuộc tính của đối tượng khi nó mới được tạo ra. Phương thức tạo trong Java có quyền truy cập là public, không có kiểu trả về và có tên phải trùng với tên của lớp. Phương thức tạo có thể có đối số và không có đối số. Khi định nghĩa một lớp mà không có phương thức tạo nào cả thì Java tự động cung cấp phương thức tạo mặc định không có đối số làm nhiệm vụ khởi tạo giá trị cho các thuộc tính có kiểu số là 0, kiểu boolean là false và kiểu tham chiếu là null. Để tường minh bạn nên định nghĩa phương thức tạo không đối số làm nhiệm vụ khởi tạo giá trị mặc định cho các thuộc tính trong khi khai báo lớp thay vì sử dụng phương thức tạo mặc định mà Java cung cấp. Ví dụ sau mình họa các phương thức tạo cho lớp người gồm các thuộc tính tên, địa chỉ, ngày sinh, giới tính.

public class Nguoi{

private String ten;

private String diaChi;

private String ngaySinh;

private char gioiTinh;//nam là ‘M’, nữ là ‘F’

//Phương thức tạo không đối số

public Nguoi(){

ten = “”;

diaChi = “”;

ngaySinh = “”;

gioiTinh = ‘M’;

}

//Phương thức tạo có một đối số

public Nguoi(String ten){

this.ten = ten;

}

//Phương thức tạo có 2 đối số

public Nguoi(String ten,String diaChi){

this.ten = ten;

this.diaChi = diaChi;

}

//Phương thức tạo có 3 đối số

public Nguoi(String ten,String diaChi,String ngaySinh){

this.ten = ten;

this.diaChi = diaChi;

this.ngaySinh = ngaySinh;

}

//Phương thức tạo có đầy đủ đối số

public Nguoi(String ten,String diaChi,String ngaySinh,char gioiTinh){

this.ten = ten;

this.diaChi = diaChi;

this.ngaySinh = ngaySinh

this.gioiTinh = gioiTinh;

}

}

Trong ví dụ trên có sử dụng từ khóa **this** dùng để tham chiếu đến đối tượng hiện tại (đối tượng đang sử dụng). Như vậy, để phân biệt giữa thuộc tính và đối số trùng tên thì bạn sử dụng tham chiếu này. Do đó ***this.ten*** là thuộc tính còn ***ten*** là đối số. Bạn cũng có thể sử dụng từ khóa này để gọi các phương thức tạo khác trong cùng lớp. Do đó chúng ta có thể viết lại các phương thức tạo bên trên như sau:

//Phương thức tạo có 2 đối số

public Nguoi(String ten,String diaChi){

this(ten);

this.diaChi = diaChi;

}

//Phương thức tạo có 3 đối số

public Nguoi(String ten,String diaChi,String ngaySinh){

this(ten,diaChi);

this.ngaySinh = ngaySinh;

}

//Phương thức tạo có đầy đủ đối số

public Nguoi(String ten,String diaChi,String ngaySinh,char gioiTinh){

this(ten,diaChi,ngaySinh);

this.gioiTinh = gioiTinh;

}

Bạn cũng có thể sử dụng đối số của phương thức tạo chính là đối tượng được tạo ra từ chính lớp đó. Mục đích của phương thức tạo này là tạo ra một đối tượng mới copy toàn bộ giá trị các thuộc tính của một đối tượng đã có. Ví dụ minh họa phương thức tạo của lớp bên trên sử dụng đối số chính là đối tượng được tạo ra từ lớp này.

public Nguoi(Nguoi p){

this.ten = p.getTen();

this.diaChi = p.getDiaChi();

this.ngaySinh = p.getNgaySinh();

this.gioiTinh = p.getGioiTinh();

}

* 1. Phương thức lấy và thiết lập giá trị các thuộc tính

Một trong những đặc trưng của lập trình hướng đối tượng là che dấu thông tin có nghĩa là các phương thức của lớp này không thể truy cập trực tiếp vào các thuộc tính riêng tư (private) của lớp khác. Do đó, khi xây dựng lớp cần cung cấp một tập các phương thức để giao tiếp với các lớp khác tức là cho phép các phương thức của lớp kia lấy giá trị hoặc thay đổi giá trị thuộc tính của lớp này. Các phương thức này được gọi là setter và getter.

* + 1. Phương thức thiết lập giá trị cho thuộc tính (setter)

Nguyên mẫu của phương thức thiết lập giá trị cho thuộc tính như sau:

public void setXyz(kiểu dữ liệu của thuộc tính xyz){

this.xyz = xyz;

}

Trong đó: *xyz* là tên của thuộc tính.

Ví dụ thiết lập thuộc tính ten có kiểu String như sau:

public void setTen(String ten){

this.ten = ten;

}

* + 1. Phương thức lấy giá trị của thuộc tính (getter)

Để lấy giá trị của thuộc tính bạn sử dụng phương thức getter có nguyên mẫu như sau:

public kiểu dữ liệu của thuộc tính getXyz(){

return this.xyz;

}

Trong đó: *xyz* là tên của thuộc tính

Ví dụ phương thức lấy thuộc tính *ten* kiểu String sẽ được viết như sau:

public String getTen(){

return this.ten;

}

* 1. Hiển thị trạng thái của đối tượng

Để hiển thị trạng thái (giá trị các thuộc tính) của một đối tượng thay vì bạn lặp đi lặp lại câu lệnh hiển thị giá trị của từng thuộc tính của đối tượng. Ví dụ để hiển thị giá trị của thuộc tính ten, diaChi, ngaySinh và gioiTinh của một đối tương người p nào đó thì trong phương thức *main()* bạn sử dụng các câu lệnh sau:

System.out.println(“Tên là: ”+p.getTen());

System.out.println(“Địa chỉ là: ”+p.getDiaChi());

System.out.println(“Ngày sinh là: ”+p.getNgaySinh());

System.out.println(“Giới tính là: ”+p.getGioiTinh());

Nếu bạn có thể hiển thị tất cả các thuộc tính của đối tượng *p* bằng một câu lệnh sau thì rất tiện lợi:

System.out.println(“Giá trị các thuộc tính của p là:”+p);

Câu lệnh bên trên là đúng cú pháp trong Java vì các lớp trong Java đều mặc định kế thừa lớp khởi thủy Object, lớp này đã định nghĩa sẵn phương thức **toString()** để trả về một xâu chứa thông tin về lớp tạo ra đối tượng và địa chỉ bộ nhớ để lưu đối tượng. Do đó, khi bạn ghép một xâu với một đối tượng thì Java tự động gọi phương thức **toString()** để lấy thông tin của đối tượng. Thông thường, thì thông tin này là không có ý nghĩa với người sử dụng. Như vậy, để phương thức **toString()** trả về xâu chứa giá trị thuộc tính của đối tượng thì chúng ta cần viết đè phương thức ***toString()*** có nguyên mẫu như sau:

public String toString(){

return xâu cần hiển thị;

}

Ví dụ để hiển thị các thuộc tính của đối tượng được tạo ra từ lớp Nguoi thì trong lớp này bạn cần viết đè phương thức **toString()** như sau:

public String toString(){

return “Tên = \t” + this.ten + ”\n”+

“Địa chỉ = \t” + this.diaChi + ”\n”+

“Ngày sinh = \t” + this.ngaySinh + ”\n”+

“Giới tính = \t” + this.gioiTinh;

}

Phương thức **toString()** được Java gọi một cách tự động mỗi khi đối tượng được sử dụng tại những vị trí mà chúng ta cần trả về một xâu kí tự. Ví dụ bạn dùng dấu cộng để nối xâu: “Thông tin của người là:” + p; khi đó phương thức ***toString()*** của đối tượng *p* sẽ được gọi một cách tự động.

* 1. Phương thức tĩnh (static method) và thuộc tính tĩnh (static field)
     1. Thuộc tính tĩnh

Khi bạn khai báo thuộc tính của lớp là ***static*** thì khi đó chỉ có duy nhất một thuộc tính này của lớp, khác với các thuộc tính bình thường mỗi đối tượng sẽ có bản sao của tất cả các thuộc tính. Nói cách khác tất cả các đối tượng sẽ dùng chung thuộc tính ***static.*** Giả sử bạn muốn tạo số định danh (id) duy nhất cho mỗi một nhân viên được tạo ra và định danh này tăng dần thì ngoài các thuộc tính chung như lớp Nguoi chúng ta cần bổ sung thêm thuộc tính id và nextId như sau:

public class NhanVien{

private String ten;

private String diaChi;

private String ngaySinh;

private String gioiTinh;

private int id;

private static int nextId = 1;

//Phương thức tạo không đối số

public NhanVien(){

id = nextId;

nextId++;

}

//Phương thức tạo có đủ đối số

public NhanVien(String ten,String diaChi,String ngaySinh,String gioiTinh){

this.ten = ten;

this.diaChi = diaChi;

this.ngaySinh = ngaySinh;

this.gioiTinh = gioiTinh;

id = nextId;

nextId++;

}

}

Như vậy, tất cả các đối tượng được tạo ra từ lớp NhanVien sẽ sử dụng chung thuộc tính ***nextId,*** ban đầu giá trị của *nextId* = 1. Mỗi nhân viên được tạo ra từ lớp NhanVien sẽ có thuộc tính *id = nextId*, sau đó tăng *nextId* lên 1. Do đó khi nhân viên thứ nhất được tạo ra từ lớp NhanVien có *id = 1* và thuộc tính *nextId = 2*, khi nhân viên thứ 2 được tạo ra sẽ có thuộc tính *id = 2* và thuộc tính *nextId = 3*, …

* + 1. Phương thức tĩnh (static methods)

Phương thức tĩnh là phương thức gắn với lớp, hoạt động không dựa vào đối tượng. Như vậy để gọi phương thức tĩnh chúng ta không cần tạo thiết tạo ra đối tượng mà gọi thông qua tên lớp. Ví dụ để tính *xy* chúng ta sử dụng phương thức tĩnh *pow* của lớp Math như sau: Math.pow(x,y). Bởi vì phương thức tĩnh không hoạt động dựa vào đối tượng nên chúng không thể truy xuất vào các thuộc tính, và phương thức khác của lớp trừ thuộc tính và phương thức tĩnh. Thông thường phương thức tĩnh được sử dụng để truy cập các thuộc tính tĩnh của lớp. Ví dụ phương thức getNextId() sau dùng để trả về giá trị của thuộc tĩnh tĩnh nextId.

public static int getNextId(){

return nextId;

}

* + 1. Khai báo hằng số (final)

Bạn sử dụng kết hợp từ khóa static với final dùng để khai báo hằng số trong java với cú pháp như sau:

[quyền\_truy\_cập] static final <kiểu dữ liệu> tên\_hằng\_số = giá trị;

**Trong đó**: tên hằng số nên viết hoa và sử dụng dấu gạch nối “\_” để ngăn cách giữa các từ.

Ví dụ khai báo hằng số chứa lương cơ bản như sau:

static final double LUONG\_CO\_BAN = 1200000;

* 1. Gói (package)
     1. Khái niệm

Với ngôn ngữ lập trình java mỗi lớp (class) được lưu trong một tệp riêng biệt. Tên của tệp phải trùng với tên của lớp. Một tệp mã nguồn của Java có phần mở rộng là “.java”. Các lớp liên quan có thể nhóm lại với nhau thành một gói (package). Gói rất tiện lợi cho việc tổ chức chương trình và để phân tách chương trình của bạn với thư viện mã được cung cấp bởi người khác. Mục đích sử dụng gói để đảm bảo tên lớp là duy nhất. Do đó, nếu có hai lớp trùng tên thì nó phải nằm ở hai gói khác nhau. Ngoài ra, gói còn có chức năng hạn chế quyền truy cập đối với các thành phần bên ngoài gói. Để khai báo một lớp thuộc gói nào đó thì bạn phải dùng từ khóa package để khai báo gói và dòng này phải là dòng đầu tiên trong câu lệnh khai báo lớp. Cú pháp để khai báo gói như sau:

package <tên gói>;

* + 1. Sử dụng gói

Một lớp có thể sử dụng tất cả các lớp khác trong cùng gói và các lớp public của gói khác. Để sử dụng lớp public của gói khác bạn có hai cách:

* Sử dụng tên gói đầy đủ ở trước tên các lớp cần sử dụng:

**Ví dụ**:

java.util.Date today = new java.util.Date();

* Sử dụng từ khóa import để dùng các lớp public của một gói nào đó. Bạn có thể import một lớp hoặc toàn bộ các lớp của một gói.

**Ví dụ:**

import java.util.Date;

Câu lệnh trên chỉ sử dụng lớp Date của gói util. Để sử dụng tất cả các lớp của gói này bạn sử dụng câu lệnh sau:

import java.util.\*;

Khi đã import lớp hoặc các lớp của một gói rồi thì việc sử dụng các lớp này giống như việc sử dụng các lớp trong chính gói đó.

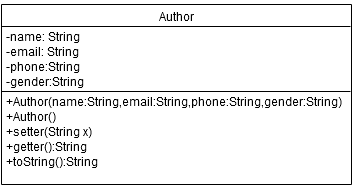
**Ví dụ**: Khi đã import lớp Date của gói util vào lớp cần sử dụng thì tại lớp này, để tạo ra đối tượng của lớp Date chúng ta chỉ cần sử dụng câu lệnh sau:

Date today = new Date();

* 1. Kết tập (composition), kế thừa (inheritance) và đa hình (polymorphism)
     1. Kết tập

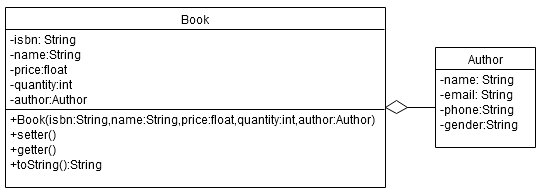
Trong java có 2 cách để sử dụng lại mã (code) là kết tập và kế thừa. Kết tập có nghĩa là bạn định nghĩa một lớp mới chứa một lớp đã tồn tại. Nói cách khác là kiểu dữ liệu của thuộc tính lớp mới là kiểu đối tượng chứ không phải là kiểu nguyên thủy.

Chúng ta minh họa cách kết khối thông qua ví dụ sau: Xét lớp tác giả (Author) có biểu đồ lớp như hình 3.2 bên dưới. Trong đó setter() và getter() là đại diện cho các phương thức thiết lập và lấy thuộc tính của lớp.



Biểu đồ lớp Author

Xét lớp sách (Book) có biểu đồ lớp như hình 3.3 sau. Trong lớp này kiểu dữ liệu của thuộc tính author là Author.



Biểu đồ lớp Book

Ví dụ sau cài đặt cho các lớp trong biểu đồ lớp mô tả trong hình 3.3

Cài đặt lớp Author

package book;

public class **Author** {

// Khai báo các thuộc tính private

private String name;

private String email;

private String phone;

private String gender;

// Phương thưc tạo đầy đủ đối số

public Author(String name, String email, String phone,   
 String gender) {

this.name = name;

this.email = email;

this.phone = phone;

this.gender = gender;

}

// Phương thức thiết lập các thuôc tính

public void setName(String name) {

this.name = name;

}

public void setEmail(String email) {

this.email = email;

}

public void setPhone(String phone) {

this.phone = phone;

}

public void setGender(String gender) {

this.gender = gender;

}

//Phương thức lấy giá trị các thuộc tính

public String getName() {

return name;

}

public String getEmail() {

return email;

}

public String getPhone() {

return phone;

}

public String getGender() {

return gender;

}

// Phương thức toString() trả về thông tin tác giả

public String toString() {

return “Tên:”+name + ", Địa chỉ email:" + email+”,   
 Số điện thoại”+phone+”, Giới tính:” +gender;

}

}

Cài đặt cho lớp Book

package book;

public class **Book** {

// Khai báo các thuộc tính private

private String isbn

private String name;

private Author author;

private float price;

private int quantity;

// Phương thức tạo đầy đủ đối số

public Book(String isbn, String name, Author   
 author, float price, int quantity) {

this.isbn = isbn;

this.name = name;

this.author = author;

this.price = price;

this.quantity = quantity;

}

// Getters và Setters

public String getIsbn(){

return isbn;

}

public String getName() {

return name;

}

public Author getAuthor() {

return author; // trả về thành viên author tham

// chiếu đến đối tượng tạo bởi lớp Author

}

public float getPrice() {

return price;

}

public int getQuantity() {

return quantity;

}

public void setIsbn(String isbn) {

this.isbn = isbn;

}

public void setName(String name) {

this.name = name;

}

public void setAuthor(Author author) {

this.author = author;

}

public void setPrice(float price) {

this.price = price;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

// Phương thức toString()trả về xâu chứa thông tin chi

//tiết của sách public String toString() {

return "Sách:” + name + " của tác giả " +   
 author;//tự động gọi author.toString()

}

}

Cài đặt lớp TestBook để kiểm tra lớp Book

package book;

public class **TestBook** {

public static void main(String[] args) {

// Tạo đối tượng aut để khởi tạo tác giả cho sách

Author aut = new Author("Quách Tuấn Ngọc",   
 "tuanngocquach@gmail.com",”098676537”,”Nam”);

System.out.println(aut); // Hiển thị thông tin

//tác giả

// Kiểm tra phương thức tạo và toString

// của lớp Book

Book expBook = new Book("12937473",”Lập trình  
 Java” aut, 90000f, 100);

System.out.println(expBook); // Hiển thị thông

//tin của sách

// Kiểm tra các phương thức setter và getter

expBook.setPrice(85000f);

expBook.setQuantity(88);

System.out.println(expBook); // Hiển thị thông

//tin mới của sách

System.out.println("Tên sách là: " +   
 expBook.getName());

System.out.println("Giá sách là: " +   
 expBook.getPrice());

System.out.println("Số lượng sách là: " +   
 expBook.getQuantity());

System.out.println("Tác giả của sách là: " +  
 expBook.getAuthor()); // tự động gọi phương

// thức toString() của lớp Author

System.out.println("Tên tác giả là: " +  
 expBook.getAuthor().getName());

System.out.println("Email của tác giả là: " +   
 expBook.getAuthor().getEmail());

System.out.println("Giới tính của tác giả là: "  
 + expBook.getAuthor().getGender());

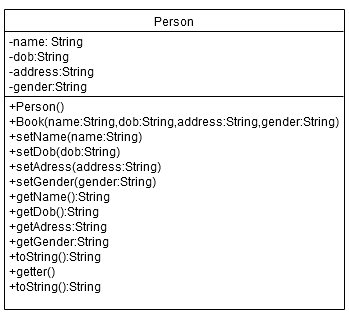
}

}

* + 1. Kế thừa
       1. Định nghĩa

Kế thừa là kỹ thuật sử dụng các lớp chuyên biệt hóa hoặc tổng quát hóa.

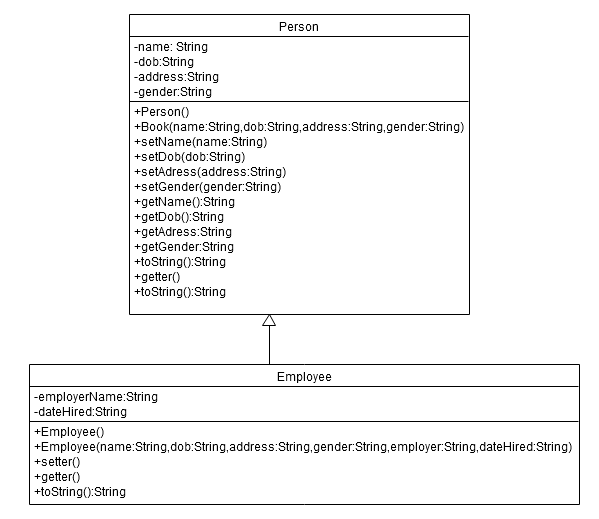
* Chuyên biệt hóa (specialise) có nghĩa là xem xét một số tính chất của một nhóm các đối tượng giống nhau và xác định một vài tính chất chỉ có trên một số ít đối tượng. Ví dụ nếu xem xét lớp Person (cá nhân) có biểu đồ lớp như hình vẽ 3.4 bên dưới sẽ thuộc về nhóm người (people), chia sẻ một số tính chất chung như: tên (name), ngày sinh (date of birth:dob), địa chỉ (address) và giới tính (gender). Một số người làm việc cho một ai đó (nhưng không phải tất cả). Do đó sẽ có một tập con của tập người và được gọi là người lao động (Employee). Tập con này chứa thêm các đặc trưng chung mà tất cả người lao động đều có ví dụ như tên của ông chủ (hoặc công ty), ngày bắt đầu làm việc. Như vậy, chúng ta có thể nói người lao động là người nhưng điều ngược lại thì không đúng.



Biểu đồ lớp Person

* Tổng quát hóa (generalisation) có nghĩa là xem xét một tập các đối tượng và tìm ra các thuộc tính chung giữa chúng. Ví dụ, xem xét một số đối tượng người lao động (Employee), sinh viên (Student), giáo viên (Teacher) thì cả ba nhóm người này đều có thuộc tính chung như tên, ngày sinh, địa chỉ, giới tính. Những tính chất chung này có thể tách ra và thành lập một nhóm mới gọi là người (Person).

Chuyên biệt hóa và tổng quát hóa dẫn đến kết quả là các thuộc tính và phương thức của lớp không phải lặp đi lặp lại mà chia sẽ giữa các lớp liên quan với nhau. Hình vẽ 3.5 minh họa lớp người lao động (Employee) kế thừa từ lớp người (Person). Như vậy, tất cả các thuộc tính và phương thức (viết tắt là setter cho các phương thức set và getter cho các phương thức get) được áp dụng cho đối tượng tạo ra từ lớp Employee giống như đối tượng được tạo ra từ lớp Person. Lớp Person được gọi là lớp cha hay lớp cơ sở còn lớp Employee được gọi là lớp con hay lớp dẫn xuất.



Minh họa lớp Employee kế thừa từ lớp Person

Trong Java để kế thừa một lớp chúng ta sử dụng từ khóa extends với cú pháp như sau:

class Base {

//Định nghĩa các thuộc tính

//Định nghĩa các phương thức

}

class Driver extends Base {

//Các thuộc tính bổ sung cho lớp con

//Các phương thức bổ sung cho lớp con hoặc viết lại của

//lớp cha

}

Sau đây là chương trình java cài đặt minh họa cho sơ đồ lớp hình 3.5. Chúng ta sử dụng gói kethua để chứa 2 lớp trên.

package kethua;

public class Person {

private String name;

private String dob;

private String address;

private String gender;

public Person() {

}

public Person(String name, String dob, String   
 address, String gender) {

this.name = name;

this.dob = dob;

this.address = address;

this.gender = gender;

}

public void setName(String name) {

this.name = name;

}

public void setDob(String dob) {

this.dob = dob;

}

public void setAddress(String address) {

this.address = address;

}

public void setGender(String gender) {

this.gender = gender;

}

public String getName() {

return name;

}

public String getDob() {

return dob;

}

public String getAddress() {

return address;

}

public String getGender() {

return gender;

}

public String toString() {

return "tên = " + name + ", ngày sinh = " +   
 dob + ", địa chỉ = " + address + ",   
 giới tính = " + gender;

}

}

package kethua;

public class Employee extends Person {

private String employerName;

private String hiredDate;

public Employee() {

super();

}

public Employee(String name,String dob,String   
 address,String gender,String   
 employerName, String hiredDate)   
 {

super(name,dob,address,gender);

this.employerName = employerName;

this.hiredDate = hiredDate;

}

public void setEmployerName(String employerName)  
 {

this.employerName = employerName;

}

public void setHiredDate(String hiredDate) {

this.hiredDate = hiredDate;

}

public String getEmployerName() {

return employerName;

}

public String getHiredDate() {

return hiredDate;

}

public String toString() {

return super.toString()+ ", Tên ông chủ=" +   
 employerName + ", Ngày làm việc=" +   
 hiredDate;

}

public static void main(String[] args){

Employee emp = new Employee("Nguyễn Văn   
 A","20/10/1996","Thái Bình","Nam","Hoàng   
 Anh Hùng","01/09/2018");

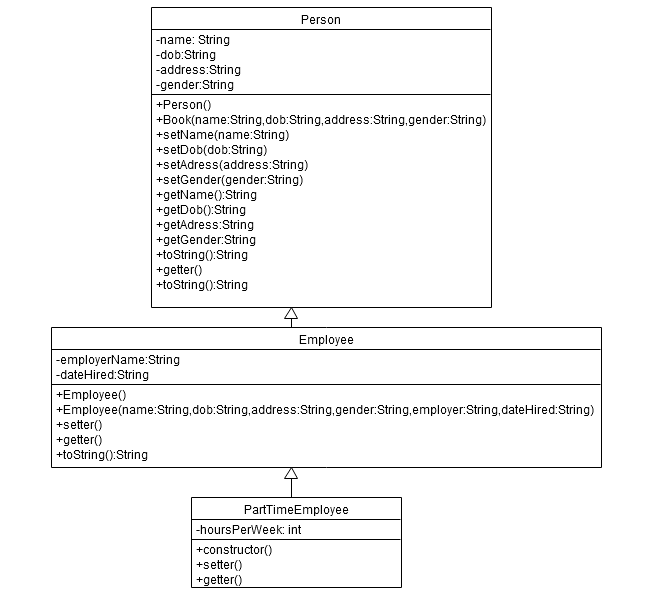
System.out.println("Thông tin nhân   
 viên:"+em);

}

}

Trong ví dụ trên khi đối tượng emp được tạo ra thì nó copy toàn bộ thuộc tính và tính chất của lớp Person. Do đó, bạn sử dụng đối tượng của lớp con có thể truy xuất vào các thành phần public hoặc protected của lớp cha.

Ngôn ngữ lập trình Java cho phép kế thừa nhiều mức. Ví dụ bạn có lớp người lao động theo giờ (PartTimeEmployee) kế thừa các thuộc tính và phương thức của lớp người lao động và bổ sung thêm thuộc tính số giờ làm trong tuần (hoursPerWeek). Biểu đồ lớp kế thừa hai cấp cho trường hợp này được minh họa trên hình 3.6 sau:



Biểu đồ lớp kế thừa nhiều cấp

* + - 1. Truy xuất vào các thành phần của lớp cha

Để truy xuất vào các thành phần của lớp cha Java sử dụng từ khóa **super** trong các tình huống sau:

* Phân biệt thành phần của lớp cha và lớp con nếu chúng trùng tên.
* Gọi phương thức tạo của lớp cha trong phương thức tạo của lớp con.

Nếu lớp con kế thừa từ lớp cha và hai lớp này có một số thành phần trùng tên thì bạn sử dụng từ khóa **super** để phân biệt thành phần của lớp cha với cú pháp như sau:

super.tên thuộc tính;

super.phương thức();

Ví dụ bên trên trong phương thức toString() của lớp Employee muốn gọi tới phương thức toString() của lớp Person chúng ta sử dụng cú pháp super.toString();

Ví dụ dưới đây minh họa việc sử dụng từ khóa super để gọi các phương thức và thuộc tính của lớp cha.

package kethua;

public class SuperClass {

int num = 20;

// phương thức dislay của lớp cha

public void display() {

System.out.println("Đây là phương thức hiển thị của  
 lớp cha");

}

}

package kethua;

public class SubClass extends SuperClass {

int num = 10;

// Phương thức hiển thị của lớp con

public void display() {

System.out.println("Đây là phương thức hiển thị của   
 lớp con");

}

public void myMethod() {

// Tạo đối tượng của lớp con

SubClass sub = new SubClass();

// Gọi phương thức hiển thị của lớp con thông qua đối

//tượng sub

sub.display();

// Gọi phương thức hiển thị của lớp cha sử dụng từ

// khóa super

super.display();

// Hiển thị ra màn hình giá trị của biến num của lớp

// con

System.out.println("Giá trị của biến num của lớp   
 con:"+ sub.num);

// printing the value of variable num of superclass

System.out.println("Giá trị của biến num của lớp   
 cha:"+ super.num);

}

//Phương thức main để kiểm tra

public static void main(String args[]) {

SubClass obj = new SubClass();

obj.myMethod();

}

}

Kết quả hiển thị ra màn hình khi chạy chương trình trên như sau:

*Đây là phương thức hiển thị của lớp con*

*Đây là phương thức hiển thị của lớp cha*

*Giá trị của biến num của lớp con:10*

*Giá trị của biến num của lớp cha:20*

Để gọi phương thức tạo của lớp cha chúng ta sử dụng từ khóa **super** với cú pháp sau:

super(); gọi phương thức tạo không đối số của lớp cha và

super(giá trị); gọi phương thức tạo có đối số của lớp cha.

* + - 1. Ghi đè phương thức (method overriding) và che dấu thuộc tính (variable hiding)

Lớp con kế thừa các thuộc tính và phương thức của lớp cha. Lớp con không những sử dụng phương thức và thuộc tính này như nó vốn có trong lớp cha mà còn có thể ghi đè phương thức của lớp cha bằng cách định nghĩa lại phương thức này ở lớp con và che dấu thuộc tính của lớp cha bằng cách khai báo thuộc tính cùng tên. Như chúng ta đã biết tất cả các lớp trong Java đều được kế thừa từ lớp **Object**, trong lớp này đã có sẵn phương thức **toString()** trả về địa chỉ của đối tượng trong bộ nhớ. Thông thường thì thông tin này là không hữu ích đối với người sử dụng mà chúng ta cần phương thức **toString()** trả về thông tin thuộc tính của đối tượng. Do đó trong các lớp tự tạo nên ghi đè (viết lại) phương thức **toString()** để trả về thuộc tính của lớp. Ví dụ sau minh họa ghi đè phương thức **toString()** của lớp cha **Object**.

public class OveridingDemo {

private String name;

private String address;

public OveridingDemo() {

}

public OveridingDemo(String name, String address)   
 {

this.name = name;

this.address = address;

}

@Override

public String toString() {

return "name: " + name + "\naddress: " +   
 address ;

}

public static void main(String[] args) {

OveridingDemo ord = new OveridingDemo("Nguyễn   
 Văn An","Thái Bình");

System.out.println(ord);

}

}

Kết quả khi chạy chương trình trên như sau:

*name: Nguyễn Văn An*

*address: Thái Bình*

Nếu bạn không ghi đè phương thức **toString()** của lớp cha **Object** thì khi thực thi chương trình sẽ trả về kết quả như sau:

overidingdemo.OveridingDemo@118f375

* + - 1. Quyền truy cập

Quyền truy cập dùng để xác định xem một thuộc tính hoặc phương thức của lớp được truy cập như thế nào ở trong chính lớp đó và các lớp khác. Java cung cấp 4 quyền truy cập là: public, private, protected và mức gói (package).

* Quyền truy cập là public (công cộng) có nghĩa là thuộc tính hoặc phương thức này có khả năng truy cập trực tiếp từ bên trong và bên ngoài lớp chưa nó.
* Quyền truy cập là private (riêng tư) có nghĩa là thuộc tính hoặc phương thức chỉ có khả năng truy cập trực tiếp ở bên trong lớp, các lớp khác kể cả lớp kế thừa cũng không truy cập trực tiếp được.
* Quyền truy cập là protected (bảo vệ) có nghĩa là thuộc tính hoặc phương thức có khả năng truy cập trực tiếp từ bên trong lớp và các lớp kế thừa còn các lớp khác không truy cập trực tiếp được.
* Quyền truy cập mặc định (package) tức là khi khai báo thuộc tính và phương thức của một lớp phần quyền truy cập được bỏ trống thì có nghĩa quyền truy cập mức gói và khi đó thì nó có khả năng truy cập trực tiếp từ các lớp nằm trong cùng gói với nó.
  + - * 1. Tổng hợp các quyền truy cập

|  |  |  |
| --- | --- | --- |
| **Quyền truy cập** | **Từ khóa** | **Ý nghĩa** |
| Mặc định (default) | Phần quyền truy cập bỏ trống | Các thành viên trong cùng gói có khả năng truy cập được |
| Riêng tư | private | Các thành viên trong cùng lớp có khả năng truy cập được |
| Bảo vệ | protected | Các thành viên trong cùng lớp, trong cùng gói và lớp con có khả năng truy cập được |
| Công cộng | public | Có thể truy cập từ bất cứ đâu trong chương trình |

* + 1. Đa hình (polymorphism)
       1. Mảng đối tượng

Nếu bạn có một mảng các đối tượng của một lớp cụ thể nào đó thì mảng này có thể được sử dụng để lưu các đối tượng được tạo ra từ lớp con bất kỳ của nó.

Giả sử chúng ta có khai báo:

Object myObjects[] = new Object[10];

Khi đó mảng myObjects có thể chứa đối tượng được tạo ra từ bất kỳ lớp nào bởi vì tất cả các đối tượng đều được tạo ra từ lớp là con của lớp **Object**. Như vậy, các phần tử của mảng myObject có thể tham chiếu đến các đối tượng khác nhau như sau:

myObjects[0] = new Person("Hoang Manh Hung","Thai  
 Binh");

myObjects[1] = new String("This is a string");

myObjects[2] = new Button("Click me");

Như vậy, phần tử thứ nhất của mảng tham chiếu đến đối tượng Person, phần tử thứ 2 tham chiếu đến một đối tượng chuỗi kí tự, phần tử thứ ba tham chiếu đến đối tượng là nút lệnh. Như chúng ta đã đề cập, tất cả các đối tượng tạo ra từ các lớp đều có phương thức toString() được viết đè hoặc kế thừa từ lớp khởi thủy Object. Do đó, bạn có thể sử dụng đoạn chương trình bên dưới để in ra thông tin của các đối tượng:

for (int i = 0; i < 10; i++) {

println(“myObjects[“ + i + “] = “ +   
 myObjects[i].toString());

}

Đối với mỗi phần tử của mảng cơ chế đa hình sẽ gọi phương thức toString() tương ứng với các lớp Person, String và Button. Việc làm này không thể thực hiện tại thời điểm biên dịch chương trình bởi vì tất cả thông tin mà trình biên dịch biết chỉ là myObjects là mảng chứa các đối tượng. Nó không biết là loại đối tượng nào sẽ được lưu vào mảng tại thời điểm chạy chương trình. Do đó, tại thời điểm chạy chương trình hệ thống java mới xác định phiên bản nào của toString() sẽ được sử dụng.

Điều gì sẽ xảy ra nếu chúng ta không định nghĩa phương thức toString() cho lớp Person. Cơ chế đa hình sẽ tìm kiếm phương thức toString() trong lớp cha của Person. Bởi vì Object là lớp cha duy nhất của Person nên nó sẽ sử dụng phương thức toString() được định nghĩa ở trong lớp này. Phương thức toString() của lớp Object đơn giản chỉ trả về một xâu chứa thông tin là tên của đối tượng và địa chỉ bắt đầu của vùng nhớ cấp phát cho đối tượng. Thông tin này là không hữu ích đối với người sử dụng do đó chúng ta nên viết đè phương thức toString() ở trong lớp Person để in ra các thông tin có ý nghĩa hơn đối với người sử dụng đó là giá trị của các thuộc tính của lớp này.

Bạn chú ý rằng, Object không phải là một lớp trừu tượng do đó phương thức toString() của nó đã được định nghĩa một cách hoàn chỉnh. Tuy nhiên, chúng ta nên viết đè phương thức này ở các lớp con để trả về thông tin ý nghĩa hơn.

Như vậy, kế thừa cùng với cơ chế gắn kết sau (**late binding)** tạo ra đa hình**.** Cơ chế này chỉ đơn giản là phương thức thực sự được gọi sẽ xác định vào thời điểm chạy chương trình chứ không phải tại thời điểm biên dịch chương trình. Để minh họa cho vấn đề này chúng ta xét ví dụ sau:

Giả sử bạn đã định nghĩa lớp Person và mảng chưa đối tượng Person có tên là personList.

Person personList[] = new Person[100];

và các lớp Employee, PersonalContact và PartTimeEmployee với các mối quan hệ như sau:

class Employee extends Person

class PartTimeEmployee extends Employee

class PersonalContact extends Person

Như vậy, bạn sử dụng tính kế thừa để định nghĩa 3 lớp. Lớp Employee và PersonalContact kế thừa lớp Person và lớp PartTimeEmployee kế thừa lớp Employee.

Mỗi lớp đều chứa phương thức toString() như chúng ta đã đề cập ở bên trên. Phương thức này sẽ in ra màn hình giá trị các thuộc tính của đối tượng.

Bởi vì mảng personList được định nghĩa để lưu đối tượng Person nên thực tế nó có thể lưu được đối tượng tạo ra từ lớp Person và bất kỳ đối tượng nào tạo ra từ lớp con của Person. Điều này có nghĩa là personList có thể đồng thời lưu đối tượng kiểu Person, Employee, PersonalContact và PartTimeEmployee.

Bạn có thể nói mảng personList được điền với các loại đối tượng khác nhau (Person, Employee, PersonalContact và PartTimeEmployee). Tính đa hình xuất hiện khi chúng ta viết mã lệnh (code) không biết chính xác kiểu dữ liệu của đối tượng được lưu trong mảng nhưng nó vẫn hoạt động bình thường, ví dụ như đoạn mã bên dưới dùng để hiển thị ra màn hình một số thông tin chi tiết của mỗi phần tử trong mảng personList:

for (i = 0; i < numPersons; i++) {

System.out.println(“Item i is ->” +  
 personList[i].toString());

}

Một câu hỏi đặt ra cho đoạn mã trên là phương thức toString() nào sẽ thực sự được gọi khi thực thi đoạn chương trình trên? Phương thức toString() được gọi sẽ tùy thuộc vào đối tượng lưu trong mảng. Nếu nó là một đối tượng tạo từ lớp Employee thì phương thức toString() của lớp Employee sẽ được gọi, nếu nó là đối tượng tạo ra từ lớp PersonalContact thì phương thức toString() của lớp PersonContact sẽ được gọi, nói cách khác thì tính đa hình của phương thức toString() sẽ tùy thuộc vào kiểu dữ liệu của đối tượng. Thực tế thì việc gắn lời gọi phương thức với phương thức thực tế được thực thi bị trì hoãn từ thời điểm biên dịch đến thời điểm chạy chương trình được gọi là gắn kết động (dynamic binding). Tuy nhiên, tại thời điểm biên dịch chương trình vẫn có sự kiểm tra xem kiểu dữ liệu mà mảng personList lưu trữ, trong trường hợp này là Person có phương thức toString() được định nghĩa hay không. Câu trả lời là chắc chắn có, kể cả trong trường hợp lớp Person không chứa phương thức toString() một cách tường minh thì nó cũng có phương thức này nhờ kế thừa từ lớp Object. Do đó trình biên dịch sẽ tiến hành dịch chương trình một cách bình thường (không có lỗi).

Ưu điểm nổi bật của tính đa hình là bạn có thể bổ sung thêm lớp con cho lớp Person ví dụ lớp Student mà đoạn chương trình bên trên vẫn chạy bình thường, không cần chỉnh sửa chút nào. Dĩ nhiên là lớp Student cần có phương thức toString() để hiển thị các thông tin liên quan.

* + - 1. Đa hình

Từ điển định nghĩa đa hình liên quan đến nguyên lý trong sinh học đó là một cá thể có thể có nhiều hình dạng và trạng thái khác nhau. Ví dụ như con tắc kè có thể có nhiều màu da khác nhau. Ông cha ta cũng đã tổng kết tính đa hình của con người thông qua câu nói trong dân gian “đi với bụt thì mặc áo cà sa, đi với ma thì mặc áo giấy”. Tính đa hình của thế giới sinh vật cũng được áp dụng vào ngôn ngữ lập trình hướng đối tượng Java, có nghĩa là một hành động có thể được thực hiện theo nhiều cách khác nhau. Có hai kiểu đa hình trong Java là đa hình tại thời điểm biên dịch chương trình (complie time) hay còn gọi là gắn kết tĩnh (static binding) nhờ kỹ thuật nạp chồng phương thức đã xét trong phần 3.2 của giáo trình và đa hình tại thời điểm chạy chương trình (run time) hay còn gọi là gắn kết động (dynamic binding) nhờ kỹ thuật ghi đè phương thức. Đối với đa hình tại thời điểm biên dịch chương trình thì dựa vào nguyên mẫu của phương thức giúp cho máy ảo Java gọi đúng phương thức cần. Đa hình tại thời điểm thi hành chương trình có nghĩa là một biến tham chiếu có kiểu dữ liệu lớp cha thì nó có quyền tham chiếu đến bất kỳ đối tượng nào được tạo ra từ lớp con của nó. Trong các lớp con có ghi đè phương thức của lớp cha thì máy ảo Java sẽ gọi đúng phiên bản phương thức này của lớp con tùy thuộc vào biến tham chiếu tham chiếu đến đối tượng của lớp con nào.

* 1. Lớp trừu tượng (abstract class) và giao diện (interface)
     1. Lớp trừu tượng

Lớp trừu tượng gắn chặt với tính đa hình trong ngôn ngữ lập trình Java. Giả sử bạn đã định nghĩa lớp Person, trong lớp này có phương thức tính thu nhập hàng tháng cho mỗi người getIncome(). Nhưng việc tính thu nhập hàng tháng sẽ tùy thuộc vào từng người: người lao động (employee), sinh viên (student), giáo viên (teacher), bác sỹ (doctor), … Do đó, lớp Person không tự định nghĩa phương thức getIncome()cho tất cả mọi người. Việc tính thu nhập hàng tháng chỉ có thể được thực hiện bởi các lớp con của Person. Tuy nhiên, chúng ta vẫn phải khai báo phương thức getIncome() trong lớp Person để đảm bảo tính đa hình và tính khái quát hóa. Thật vậy, nếu chúng ta không khai báo phương thức getIncome() trong lớp Person thì khi chúng ta khai bảo một mảng có kiểu Person và sử dụng tính đa hình để cho mảng này lưu các đối tượng tạo ra từ lớp Student, Teacher, Doctor, …khi đó nếu chúng ta gọi phương thức getIncome() để in ra thông tin thu nhập của các đối tượng lưu trong mảng sẽ gặp lỗi khi biên dịch vì trình biên dịch của Java không tìm thấy phương thức getIncome() được khai báo trong lớp Person.

Như vậy, chúng ta vẫn cần khai báo phương thức getIncome() trong lớp Person nhưng chỉ là nguyên mẫu của phương thức mà không có thân. Phương thức khai báo như vậy là một phương thức trừu tượng (abstract). Lớp chứa phương thức trừu tượng cũng phải được khai báo là lớp trừu tượng. Lớp trừu tượng chưa được định nghĩa đầy đủ và cần phải được bổ sung trong các lớp con của nó. Nếu một lớp kế thừa từ lớp trừu tượng mà không thực thi tất cả các phương thức trừu tượng của lớp này thì nó cũng khải khai báo là trừu tượng. Cú pháp khai báo lớp và phương thức trừu tượng là sử dụng từ khóa **abstract** như sau:

public abstract class Person{

//Các thuộc tính của lớp khai báo ở đây

//Các phương thức tạo định nghĩa ở đây

public abstract double getIncome();//khai báo phương

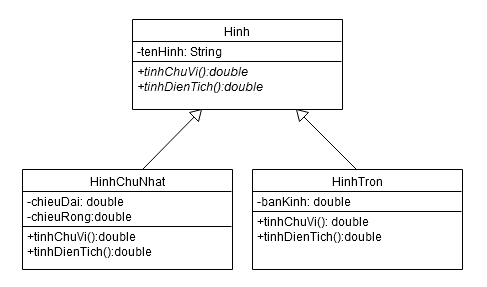
// thức trừu tượng

//Các phương thức khác định nghĩa ở đây

}

Ưu điểm của kỹ thuật bên trên là nếu bạn có một đối tượng là thể hiện của lớp con của lớp Person, bạn chỉ cần gọi phương thức getIncome() để tính thu nhập hàng tháng của một người. Tính đa hình sẽ biết lớp con nào của Person mà đối tượng thuộc về và đảm bảo gọi phương thức getIncome() một cách phù hợp. Việc làm trên có ưu điểm là nếu chúng ta bổ sung thêm lớp con của Person thì chúng ta không cần thay đổi chút gì với lớp cha mà chỉ cần cài đặt phương thức getIncome() một cách thích hợp tương ứng với lớp này.

Ví dụ sau minh họa lớp trừu tượng: Xét biểu đồ lớp như hình 3.7 sau. Trong biểu đồ lớp này lớp cha Hinh chứa hai phương thức trừu tượng tinhChuVi() và tinhDienTich() bởi vì mỗi hình có cách tính diện tích và chu vi khác nhau nên tại lớp cha hai phương thức này chỉ là nguyên mẫu, nó sẽ được ghi đè tại các lớp con HinhChuNhat và HinhTron.



Biểu đồ lớp minh họa lớp trừu tượng

Chương trình cài đặt cho ví dụ trên như sau:

package hinh;

public abstract class Hinh {

private String tenHinh;

public abstract double tinhChuVi();

public abstract double tinhDienTich();

public Hinh() {

}

public Hinh(String tenHinh) {

this.tenHinh = tenHinh;

}

public String getTenHinh() {

return tenHinh;

}

public void setTenHinh(String tenHinh) {

this.tenHinh = tenHinh;

}

}

package hinh;

import java.text.DecimalFormat;

public class HinhChuNhat extends Hinh {

private double chieuDai;

private double chieuRong;

public HinhChuNhat() {

super();

}

public HinhChuNhat(double chieuDai, double   
 chieuRong, String tenHinh) {

super(tenHinh);

this.chieuDai = chieuDai;

this.chieuRong = chieuRong;

}

public double getChieuDai() {

return chieuDai;

}

public double getChieuRong() {

return chieuRong;

}

public void setChieuDai(double chieuDai) {

this.chieuDai = chieuDai;

}

public void setChieuRong(double chieuRong) {

this.chieuRong = chieuRong;

}

@Override

public double tinhChuVi() {

double cv;

cv = (chieuDai+chieuRong)\*2;

return cv;

}

@Override

public double tinhDienTich() {

double dt;

dt = chieuDai\*chieuRong;

return dt;

}

@Override

public String toString() {

DecimalFormat df = new DecimalFormat("0.00");

return "Chu vi cua hinh "+getTenHinh()+":   
 "+df.format(tinhChuVi())+ "\nDien   
 tich cua hinh "+getTenHinh()+":   
 "+df.format(tinhDienTich());

}

}

package hinh;

import java.text.DecimalFormat;

public class HinhTron extends Hinh{

private double banKinh;

public HinhTron() {

super();

}

public HinhTron(double banKinh, String tenHinh) {

super(tenHinh);

this.banKinh = banKinh;

}

public double getBanKinh() {

return banKinh;

}

public void setBanKinh(double banKinh) {

this.banKinh = banKinh;

}

@Override

public double tinhChuVi() {

double cv;

cv = 2\*Math.PI\*banKinh;

return cv;

}

@Override

public double tinhDienTich() {

double dt;

dt = banKinh\*banKinh\*Math.PI;

return dt;

}

@Override

public String toString() {

DecimalFormat df = new DecimalFormat("0.00");

return "Chu vi cua hinh "+getTenHinh()+":  
 "+df.format(tinhChuVi())+

"\nDien tich cua hinh "+getTenHinh()+":   
 "+df.format(tinhDienTich());

}

}

package hinh;

public class Test {

public static void main(String[] args) {

Hinh h[] = new Hinh[2];

HinhChuNhat hcn = new HinhChuNhat(4, 3, "chu   
 nhat");

HinhTron ht = new HinhTron(2.3,"tron");

h[0] = hcn;

h[1] = ht;

for(int i=0;i<h.length;i++){

System.out.println(h[i]);

}

}

}

Kết quả khi chạy chương trình trên như sau:

*Chu vi cua hinh chu nhat: 14.00*

*Dien tich cua hinh chu nhat: 12.00*

*Chu vi cua hinh tron: 14.45*

*Dien tich cua hinh tron: 16.62*

* + 1. Interface

Interface trong Java hỗ trợ cho việc đa thừa kế tức là một lớp có thể thực thi (implements) nhiều interface hoặc một interface có thể kế thừa (extends) nhiều interface khác như minh họa trên hình 3.8 sau.

Interface giống như lớp trừu tượng nhưng nó không chứa biến mà chỉ có thể chứa hằng số và tất cả các phương thức phải là trừu tượng. Vì mặc định tất cả các phương thức của interface là công cộng, trừu tượng nên chúng ta không cần sử dụng từ khóa abstract khi khai báo các phương thức của interface. Trong Java một lớp chỉ được phép kế thừa từ một lớp khác nhưng được phép thực thi nhiều interface. Lớp thực thi interface phải định nghĩa tất cả các phương thức của interface hoặc định nghĩa một phần. Nếu định nghĩa một phần thì nó phải khai báo là lớp trừu tượng.

Cú pháp định nghĩa interface như sau:

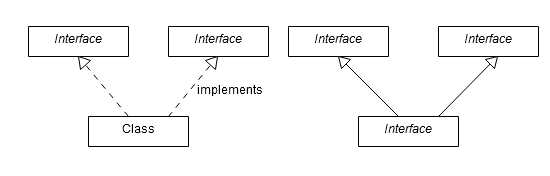
[quyền truy cập] <tên interface> {

//Khai báo các hằng số

//Khai báo các nguyên mẫu phương thức

}

Trong đó: quyền truy cập là tùy chọn, nếu không có thì mặc định quyền truy cập là public. Tên interface là bắt buột và đặt giống cú pháp như đặt tên lớp.



Minh họa đa thừa kế với Interface

Ví dụ minh họa interface. Trong ví dụ này chúng ta xây dựng 1 interface PhepToanSoHoc có 4 phương thức trừu tượng là cong, tru, nhan và chia, tiếp theo chúng ta xây dựng lớp PhepToan thực thi interface này để thực hiện các phép toán số học cơ bản. Cuối cùng, chúng ta viết lớp KiemTra để thử các phương thức chúng ta đã xây dựng.

package interfacedemo;

public interface PhepToanSoHoc {

public double cong(double n1,double n2);

public double tru(double n1,double n2);

public double nhan(double n1,double n2);

public double chia(double n1,double n2);

}

package interfacedemo;

public class PhepToan implements PhepToanSoHoc {

@Override

public double cong(double n1, double n2) {

double temp;

temp = n1 + n2;

return temp;

}

@Override

public double tru(double n1, double n2) {

double temp;

temp = n1 - n2;

return temp;

}

@Override

public double nhan(double n1, double n2) {

double temp;

temp = n1 \* n2;

return temp;

}

@Override

public double chia(double n1, double n2) {

double temp = 0;

if (n2 == 0) {

System.out.println("Phep chia cho 0");

} else {

temp = n1 / n2;

}

return temp;

}

}

*package interfacedemo;*

*public class KiemTra {*

*public static void main(String[] args) {*

*double n1 = 10;*

*double n2 = 20;*

*PhepToan pt = new PhepToan();*

*System.out.println("Tong cua "+n1+" va "+n2+" la:"+pt.cong(n1, n2));*

*System.out.println("Hieu cua "+n1+" va "+n2+" la:"+pt.tru(n1, n2));*

*System.out.println("Tich cua "+n1+" va "+n2+" la:"+pt.nhan(n1, n2));*

*System.out.println("Thuong cua "+n1+" va "+n2+" la:"+pt.chia(n1, n2));*

*}*

*}*

Ví dụ minh họa đa thừa kế với interface :

interface Printable{

void print();

}

interface Showable{

void show();

}

class MultInhDemo implements Printable,Showable{

public void print()

{

System.out.println("Hello");

}

public void show(){

System.out.println("Welcome");

}

public static void main(String args[]){

MultInhDemo obj = new MultInhDemo ();

obj.print();

obj.show();

}

}

Kết quả hiển thị lên màn hình khi thực hiện chương trình trên là:

Hello

Welcome

BAI TẬP CHƯƠNG 3

**Bài 1**: Viết chương trình Java cài đặt cho sơ đồ lớp như hình vẽ 3.9 bên dưới. Viết lớp Test có phương thức main() để tạo đối tượng từ lớp Person, sau đó kiểm tra các phương thức của lớp này (setter(), getter(), toString()).

Person

* name: String
* dob: String (ngày sinh)
* pob: String (nơi sinh)
* gender: char (giới tính: Nam là ‘M’ và nữ là ‘F’)
* email: String
* phone: String

+ Person()

+ Person(name:String, …, phone:String)

+ setName(name:String):void

+…

+ setPhone(phone:String):void

+ getName():String

+ …

+ getPhone():String

+ toString():String

Biểu đồ lớp Person

**Bài 2**: Viết chương trình Java cài đặt cho lớp DiaChi (địa chỉ) có biểu đồ lớp như hình vẽ 3.10 sau. Viết lớp Test trong đó có phương thức main() để tạo một đối tượng thuộc lớp Address sau đó kiểm tra các phương thức của đối tượng này.

**Bài 3**: Viết chương trình Java cài đặt cho lớp NhanVien có biểu đồ lớp như hình 3.11 bên dưới. Viết lớp Test chứa phương thức main() để tạo đối tượng từ lớp NhanVien và kiểm tra các phương thức của lớp này. Biết rằng kiểu dữ liệu DiaChi được mô tả như hình 3.10.

**Bài 4:** Viết chương trình Java cài đặt cho biểu đồ lớp như hình 3.12 bên dưới. Viết lớp Test chứa phương thức main() để tạo đối tượng từ lớp BenhNhan và kiểm tra các phương thức của lớp này.

**Bài 5:** Viết chương trình Java cài đặt cho biểu đồ lớp như hình 3.13 bên dưới. Viết lớp Test chứa phương thức main() để tạo đối tượng từ lớp SinhVien và kiểm tra các phương thức của lớp này.

**DiaChi**

* thon: String (thôn hoặc tổ dân phố)
* xa: String (xã hoặc phường)
* huyen:String (quận hoặc huyện)
* tinh:String (tỉnh hoặc thành phố)

+ DiaChi()

+ DiaChi(thon:String, xa:String, huyen:String, tinh:String)

+ setThon(thon:String):void

+ …

+ setTinh(tinh:String):void

+ getThon():String

+ …

+ getTinh():String

+ toString():String

Biểu đồ lớp DiaChi

NhanVien

* name:String
* address:DiaChi
* dob:String
* gender:char

+ NhanVien()

+ NhanVien(full parameters)

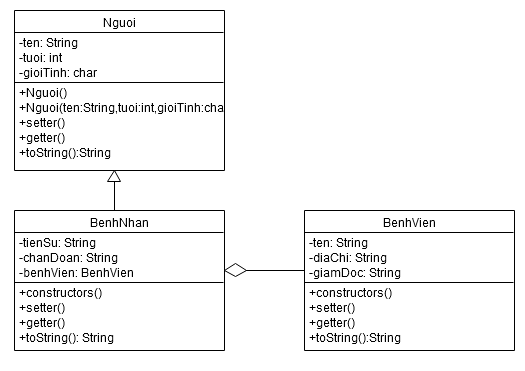
+ setter()

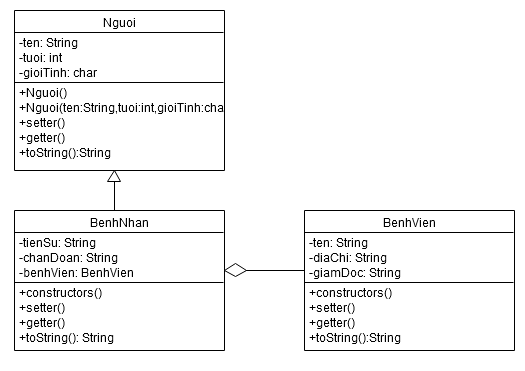
+ getter()

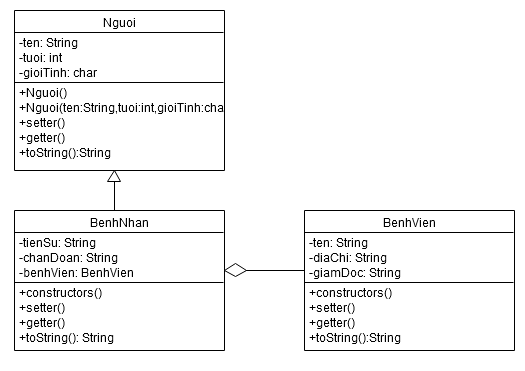
+ toString():String

DiaChi

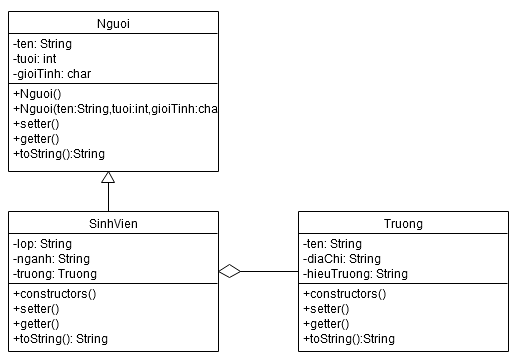
Biểu đồ lớp NhanVien





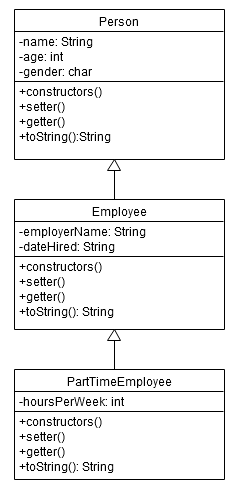


Biều đồ lớp BenhNhan



Biểu đồ lớp SinhVien

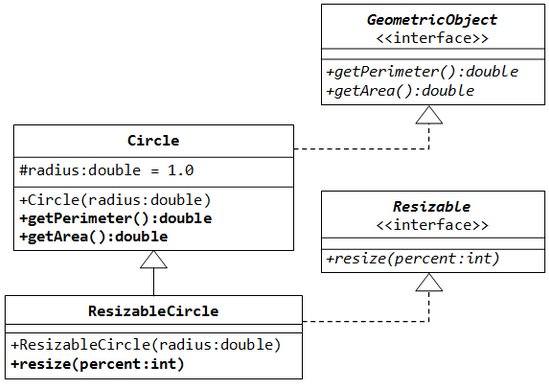
**Bài 6:** Viết chương trình Java cài đặt cho biểu đồ lớp như hình 3.14 sau. Tạo lớp Test có chứa phương thức main() để kiểm tra các phương thức của đối tượng tạo tự lớp PartTimeEmployee.

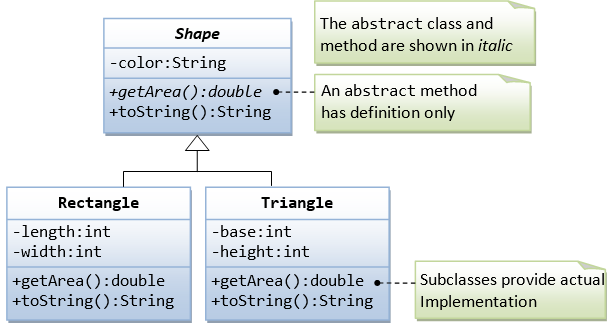


Biểu đồ lớp PartTimeEmployee

**Bài 7**: Viết chương trình Java cài đặt cho biểu đồ lớp như hình 3.15 sau. Tạo lớp Test có chứa phương thức main() để kiểm tra các phương thức của đối tượng được tạo từ lớp ResizableCircle.

**Bài 8:** Viết chương trình Java cài đặt cho biểu đồ lớp như hình 3.16 bên dưới. Tạo lớp Test có chứa phương thức main() để kiểm tra các phương thức của các đối tượng được tạo từ lớp Rectangle và Triangle. Biết rằng base là cạnh đáy còn height là đường cao của tam giác (Triangle).

Biểu đồ lớp của ResizableCircle

Biểu đồ lớp Rectangle và Triangle

**Bài 9:** Viết chương trình Java tạo mảng các đối tượng từ lớp Rectangle và Triangle có biểu đồ lớp như hình 3.14. Hiển thị thông tin của các đối tượng được tạo ra từ lớp Rectangle ra màn hình.

Gợi ý: Để kiểm tra xem đối tượng được tạo ra từ lớp nào chúng ta sử dụng toán tử **instanceof** với cú pháp như sau:

Object **instanceof** Class

Toán tử này trả về **true** nếu đối tương **Object** được tạo ra từ lớp **Class** và ngược lại.

**Bài 10:** Viết chương trình quản lí Học viên và mức thu học phí của một Trung tâm Tin học với các mức giá theo từng chương trình đăng kí học và các đối tượng ưu tiên như sau:

+ Đồ họa: học phí = số buổi\*đơn giá ĐH – ưu tiên

Trong đó: ưu tiên = 1.000.000 (VNĐ) nếu là loại 1

ưu tiên = 500.000 (VNĐ) nếu là loại 2

+ Lập trình: học phí = số buổi\*đơn giá LT – ưu tiên

Trong đó: ưu tiên = 1.000.000 (VNĐ) nếu là loại 1

ưu tiên = 800.000 (VNĐ) nếu là loại 2

**Yêu cầu:**

1. Xây dựng lớp trừu tượng **Hocvien** bao gồm:

+ Thuộc tính: Họ tên, địa chỉ, loại chương trình (Đồ họa, Lập trình).

+ Phương thức:

- **nhapThongTin** để nhập thông tin cho Học viên đăng kí gồm: Họ tên, địa chỉ, loại ưu tiên, loại chương trình đăng kí.

- Khởi tạo thông tin ban đầu cho học viên.

- **hocPhi** để tính học phí cho học viên (trừu tượng).

- **inThongTin** để in thông tin cho học viên gồm:

\*Họ tên: …

\*Địa chỉ: …

\*Loại ưu tiên:…

\*Loại chương trình đăng kí học: …

\*Học phí: …

2. Xây dựng hai lớp **HocvienDH** và **HocvienLT** kế thừa từ lớp **Hocvien** (bổ sung thêm các thuộc tính cần thiết cho từng lớp)

- Khởi tạo giá trị ban đầu về thông tin cho từng loại Hocvien (mỗi loại đối tượng).

- Cài đặt phương thức **Hocphi** cho từng loại chương trình đăng kí.

- Nhập giá trị cho các thuộc tính mới được bổ sung.

3. Trong phương thức **main** thực hiên:

- Khởi tạo các đối tượng (mỗi loại học viên một đối tượng)

- Nhập thông tin cho đối tượng

- In thông tin đối tượng

Bài 11: Một trường Đại học X muốn tính lương cho Giảng viên gồm 2 loại: Giảng viên cơ hữu và giảng viên thỉnh giảng (trả lương theo tháng).

Với cách tính lương tháng như sau:

+ Giảng viên thỉnh giảng: Lương tháng = số tiết dạy\* đơn giá – % thu nhập

Trong đó: % thu nhập = 15% (số tiết dạy\*đơn giá).

+ Giảng viên cơ hữu: Lương tháng = lương cơ bản (LCB)\* Hệ số lương + phụ cấp

Trong đó: Phụ cấp = 0%\*LCB. Nếu thâm niên công tác nhỏ hơn 5 năm

Phụ cấp =5%\*LCB. Nếu thâm niên công tác bằng 5 năm và sau 5 năm trở đi thì cứ mỗi năm được tính thêm 1%\*LCB nữa. (Ví dụ 5 năm công tác là 5%\*LCB, 6 năm là 6%\*LCB, 7 năm là 7%\*LCB…).

**Yêu cầu:**

1. Xây dựng lớp trừu tượng **GiangVien** bao gồm:

+ Thuộc tính: Họ tên, địa chỉ, loại giảng viên (Giảng viên cơ hữu và giảng viên thỉnh giảng).

+ Phương thức:

- **nhapThongTin** để nhập thông tin cho Giảng viên gồm: Họ tên, địa chỉ, loại giảng viên

- Khởi tạo thông tin ban đầu cho Giảng viên.

- **tinhLuong** để tính lương tháng cho Giảng viên (trừu tượng).

- **inThongTin** để in thông tin người Giảng viên gồm:

\*Họ tên: …

\*Địa chỉ: …

\*Loại Giảng viên: …

\*Lương tháng: …

2. Xây dựng hai lớp **GVCoHuu** và **GVThinhGiang** kế thừa từ lớp **GiangVien** (bổ sung thêm các thuộc tính cần thiết cho từng lớp)

- Khởi tạo giá trị ban đầu về thông tin cho từng loại Giảng viên(mỗi loại đối tượng).

- Cài đặt phương thức **tinhLuong** cho từng loại Giảng viên.

- Nhập giá trị cho các thuộc tính mới được bổ sung.

3. Trong phương thức **main** thực hiện:

- Khởi tạo các đối tượng (mỗi loại Giảng viên một đối tượng)

- Nhập thông tin cho đối tượng; - In thông tin đối tượng

Bài 12: Một trường Cao Đẳng X muốn tính điểm cho Sinh viên gồm 2 loại: SV cao đẳng nghề (SVCĐN) và SV cao đẳng chính quy (SVCĐCQ).

Với cách tính Điểm như sau:

+ SVCĐN: ĐTB = TĐKT / SMKT

+ SVCĐCQ: ĐTB= (TĐKT / SMKT + ĐTKTHP)/3.

Trong đó: ĐTB: Điểm trung bình

TĐKT: Tổng điểm các môn kiểm tra

SMKT: số môn kiểm tra

ĐTKTHP: Điểm thi kết thúc học phần

**Yêu cầu:**

1. Xây dựng lớp trừu tượng **Nguoi** bao gồm:

+ Thuộc tính: Họ tên, ngày sinh, địa chỉ, loại sinh viên (SVCĐN và SVCĐCQ).

+ Phương thức:

- **nhapThongTin** để nhập thông tin cho Sinh viên gồm: Họ tên, ngày sinh, địa chỉ, loại sinh viên

- Khởi tạo thông tin ban đầu cho sinh viên.

- **tinhDiem** để tính ĐTB cho sinh viên (trừu tượng).

- **inThongTin** để in thông tin sinh viên gồm:

\*Họ tên: …

\*Địa chỉ: …

\*Ngày sinh:…

\*Loại Sinh viên: …

\*ĐTB: …

2. Xây dựng hai lớp **SVCĐN** và **SVCĐCQ** kế thừa từ lớp **Nguoi** (bổ sung thêm các thuộc tính cần thiết cho từng lớp)

- Khởi tạo giá trị ban đầu về thông tin cho từng loại sinh viên (mỗi loại đối tượng).

- Cài đặt phương thức **tinhDiem** cho từng loại sinh viên.

- Nhập giá trị cho các thuộc tính mới được bổ sung.

3. Trong phương thức **main** thực hiên:

- Khởi tạo các đối tượng (mỗi loại sinh viên một đối tượng)

- Nhập thông tin cho đối tượng

- In thông tin đối tượng

Bài 13: Một công ty X sản xuất cần tính lương cho 2 loại lao động gồm: Lao động phổ thông và kĩ sư (trả lương theo tháng).

Tính lương tháng của công nhân như sau:

+ Lao động phổ thông: Lương tháng = số ngày công \* đơn giá ngày công + thưởng

Trong đó: Thưởng = 1.000.000 (VNĐ) nếu số ngày công lơn hơn hoặc bằng 25.

Thưởng = 700.000 (VNĐ) nếu số ngày công nhỏ hơn 25 và lớn hơn 15.

Thường =0 nếu số ngày công nhỏ hơn hoặc bằng 15.

+ Kĩ sư: Lương tháng = lương cơ bản \* Hệ số lương + thưởng

Trong đó: Thưởng = 2.000.000 (VNĐ) nếu số ngày công lơn hơn hoặc bằng 25.

Thưởng = 1.000.000 (VNĐ) nếu số ngày công nhỏ hơn 25 và lớn hơn 15.

Thường =500.000 nếu số ngày công nhỏ hơn hoặc bằng 15.

**Yêu cầu:**

1. Xây dựng lớp trừu tượng **NguoiLaoDong** bao gồm:

+ Thuộc tính: Họ tên, địa chỉ, loại lao động (Lao động phổ thông và kĩ sư).

+ Phương thức:

- **nhapThongTin** để nhập thông tin cho người lao động gồm: Họ tên, địa chỉ, loại lao động

- Khởi tạo thông tin ban đầu cho người lao động.

- **tinhLuong** để tính lương tháng cho người lao động (trừu tượng).

- **inThongTin** để in thông tin người lao động gồm:

\*Họ tên: …

\*Địa chỉ: …

\*Loại lao động: …

\*Lương tháng: …

2. Xây dựng hai lớp **LaoDongPhothong** và **Kisu** kế thừa từ lớp **NguoiLaoDong** (bổ sung thêm các thuộct tính cần thiết cho từng lớp)

- Khởi tạo giá trị ban đầu về thông tin cho từng loại lao động (mỗi loại đối tượng).

- Cài đặt phương thức **tinhLuong** cho từng loại lao động.

- Nhập giá trị cho các thuộc tính mới được bổ sung.

3. Trong phương thức **main** thực hiện:

- Khởi tạo các đối tượng (mỗi loại lao động một đối tượng)

- Nhập thông tin cho đối tượng

- In thông tin đối tượng.

Bài 14: Một Công ty sản xuất X cần tính lương cho 2 loại công nhân: Công nhân khoán sản phẩm (CNKSP) và công nhân tính công nhật (CNTCN) (trả lương theo tháng). Tính lương tháng của công nhân như sau:

+ **Công nhân khoán sản phẩm:** Lương tháng = số sản phẩm \* đơn giá sản phẩm + thưởng

Trong đó:

Thưởng = 0 đ nếu số sản phẩm ít hơn hoặc bằng số sản phẩm mức quy định.

Thưởng = 1.000.000đ nếu số sản phẩm bằng hoặc vượt mức quy định là 100 sản phẩm.

Thưởng = 1.500.000đ nếu số sản phẩm bằng hoặc vượt mức quy định là 150 sản phẩm.

*Biết rằng: Mức quy định số sản phẩm của CNKSP là 1000 sản phẩm/1 tháng*

+ **Công nhân tính công nhật:** Lương tháng = Lương cơ bản \* Hệ số lương + thưởng

Trong đó:

Thưởng = 0 nếu số ngày công nhỏ hơn 20 ngày.

Thưởng = 1.2\*LCB nếu số ngày công lớn hơn hoặc bằng 20 ngày.

**Yêu cầu:**

1. Xây dựng lớp trừu tượng **CongNhan** bao gồm:

+ Thuộc tính: Họ tên, địa chỉ, loại công nhân (CNKSP, CNTCN).

+ Phương thức:

- **nhapThongTin** để nhập thông tin cho công nhân gồm: Họ tên, ngày sinh, địa chỉ, loại công nhân.

- Khởi tạo thông tin ban đầu cho công nhân.

- **tinhLuong** để tính lương tháng cho công nhân (trừu tượng).

- **inThongTin** để in thông tin công nhân gồm:

\*Họ tên: …

\*Địa chỉ: …

\*Ngày sinh:…

\*Loại công nhân: …

\*Lương tháng: …

2. Xây dựng hai lớp **CNKSP** và **CNTCN** kế thừa từ lớp **CongNhan** (bổ sung thêm các thuộc tính cần thiết cho từng lớp)

- Khởi tạo giá trị ban đầu về thông tin cho từng loại công nhân (mỗi loại đối tượng).

- Cài đặt phương thức **tinhLuong** cho từng loại công nhân.

- Nhập giá trị cho các thuộc tính mới được bổ sung.

3. Trong phương thức **main** thực hiện:

- Khởi tạo các đối tượng (mỗi loại lao động một đối tượng)

- Nhập thông tin cho đối tượng.; - In thông tin đối tượng.

1. XỬ LÝ NGOẠI LỆ (EXCEPTION HANDLING)

Sau khi đọc xong chương này sinh viên có thể định nghĩa một ngoại lệ (Exception), hiểu được mục đích của việc xử lý ngoại lệ, các kiểu ngoại lệ khác nhau và mô hình xử lý ngoại lệ trong Java. Sinh viên cũng có thể tự tạo ra các ngoại lệ.

* 1. Giới thiệu

Một chương trình bất kỳ thường chứa các lỗi trong khi thiết kế logic hoặc lúc cài đặt và các lỗi này gọi là bug. Ngoài ra, môi trường thực thi chương trình không mong muốn cũng có thể xảy ra và làm cho chương trình bị lỗi, ví dụ như đĩa bị đầy, kết nối mạng bị mất, tập tin có thể bị xóa, …những lỗi kiểu như vậy gọi là ngoại lệ (exception). Như vậy, ngoại lệ là một loại lỗi đặc biệt. Lỗi này xuất hiện vào lúc thực thi chương trình. Các trạng thái không bình thường xảy ra trong khi thi hành chương trình tạo ra các ngoại lệ. Những trạng thái này không được biết trước trong khi ta đang xây dựng chương trình. Nếu bạn không xử lý các trạng thái này thì chương trình có thể bị kết thúc đột ngột. Ví dụ, việc chia cho 0 sẽ tạo một lỗi trong chương trình. Ngôn ngữ Java cung cấp cơ chế dùng để xử lý ngoại lệ rất hiệu quả. Việc xử lý này làm hạn chế tối đa trường hợp hệ thống bị hỏng (crash) hay hệ thống bị ngắt đột ngột. Tính năng này làm cho Java trở thành một ngôn ngữ lập trình mạnh.

Một chương trình nên có cơ chế xử lý ngoại lệ thích hợp. Nếu không, chương trình sẽ bị ngắt khi một ngoại lệ xảy ra. Trong trường hợp đó, tất cả các nguồn tài nguyên mà hệ thống đã cấp không được giải phóng. Điều này gây lãng phí tài nguyên. Để tránh trường hợp này, tất cả các nguồn tài nguyên mà hệ thống cấp nên được thu hồi lại. Tiến trình này đòi hỏi cơ chế xử lý ngoại lệ thích hợp.

Ví dụ, xét thao tác vào ra (I/O) trong một tập tin. Nếu việc chuyển đổi kiểu dữ liệu không thực hiện đúng, một ngoại lệ sẽ xảy ra và chương trình bị hủy mà không đóng tập tin lại. Lúc đó tập tin dễ bị hư hại và các nguồn tài nguyên được cấp phát cho tập tin không được trả lại cho hệ thống.

* 1. Xử lý ngoại lệ với java

Trong Java, mô hình xử lý ngoại lệ giám sát việc thực thi mã để phát hiện ngoại lệ. Mô hình xử lý ngoại lệ của Java được gọi là ‘catch and throw’. Trong mô hình này người lập trình **try** (cố) thực hiện một đoạn lệnh. Nếu có lỗi xảy ra khi thực thi đoạn lệnh này thì một ngoại lệ có thể được **throw** (ném) ra. Khối lệnh xử lý lỗi có thể được viết để **catch** (bắt) ngoại lệ này. Khi khối lệnh **try** được thực thi xong hoặc ngoại lệ đã được bắt thì khối **finally** sẽ thực thi việc giải phóng tài nguyên. Như vậy, các ngoại lệ phải được xử lý, hoặc thoát khỏi chương trình khi nó xảy ra.

Ngôn ngữ Java cung cấp 5 từ khoá sau để xử lý các ngoại lệ: try, catch, throw, throws, finally và cấu trúc của mô hình xử lý ngoại lệ như đoạn mã sau:

try

{

// đoạn mã có khả năng gây ra ngoại lệ

}

catch(Exception e)

{

// Xử lý ngoại lệ ở đây

// Đối tượng ngoại lệ bị bắt được truyền là tham số

// của catch

}

finally

{

// Các câu lệnh giải phóng tài nguyên ở đây

// khối lệnh nay luôn được thực hiện cho dù ngoại lệ có

// xảy ra hay không.

}

Chú ý rằng một khối **try** có thể có nhiều khối **catch,** mỗi khối catch bắt một loại ngoại lệ xác định. Cú pháp bắt nhiều ngoại lệ như sau:

try

{

// đoạn mã có khả năng gây ra ngoại lệ

}

catch(Exception\_Name e1)

{

// Xử lý ngoại lệ ở đây

// Đối tượng ngoại lệ bị bắt được truyền là tham số

// của catch

}

catch(Exception\_Name e2)

{

// Xử lý ngoại lệ ở đây

// Đối tượng ngoại lệ bị bắt được truyền là tham số

// của catch

}

//tiếp tục các khối catch khác ở đây

finally

{

// Các câu lệnh giải phóng tài nguyên ở đây

// khối lệnh nay luôn được thực hiện cho dù ngoại lệ có

// xảy ra hay không.

}

Chương trình Java sau minh họa cho việc xử lý ngoại lệ sử dụng khối try và catch:

class TryClass

{

public static void main(String args[])

{

int demo=0;

try

{

System.out.println(20/demo);

}

catch(ArithmeticException a)

{

System.out.println(“Cannot Divide by zero”);

}

}

}

Chương trình Java sau minh họa chương trình có nhiều khối catch để bắt ngoại lệ.

public class TestMultipleCatchBlock{

  public static void main(String args[]){

   try{

    int a[]=new int[5];

    a[5]=30/0;

   }

   catch(ArithmeticException e){

System.out.println("Lỗi về số học");

}

   catch(ArrayIndexOutOfBoundsException e){

System.out.println("Lỗi về chỉ số mảng");

}

   catch(Exception e){

System.out.println("Lỗi tổng quát");

}

  System.out.println("Phần còn lại của chương trình...");

 }

}

Kết quả hiển thị lên màn hình khi chạy chương trình trên như sau:

Lỗi về số học

Phần còn lại của chương trình …

* 1. Phân loại ngoại lệ

Tất cả các ngoại lệ là thể hiện của lớp kế thừa từ lớp Throwable của Java. Chúng ta có thể tạo ra lớp xử lý ngoại lệ riêng của mình nếu như các lớp xử lý ngoại lệ có sẵn của Java không đáp ứng được yêu cầu. Hình vẽ 4.1 bên dưới minh họa biểu đồ lớp xử lý ngoại lệ của Java



Biểu đồ lớp xử lý ngoại lệ của Java

* + 1. Lỗi (Error)

Ngoại lệ được tạo ra mà kế thừa từ lớp ***Error*** sẽ xuất hiện bởi các lỗi nội tại hoặc thiếu tài nguyên trong hệ thống thực thi của Java. Bạn có rất ít cơ hội để xử lý các lỗi kiểu như thế này do đó chúng ta chỉ có thể làm là cố gắng kết thúc chương trình một cách an toàn. Các lỗi kiểu như thế này rất hiếm khi xảy ra.

* + 1. Ngoại lệ (Exception)

Các ngoại lệ tạo ra trong chương trình của bạn mà kế thừa từ lớp Exception sẽ xuất hiện do các lỗi chương trình hoặc do yếu tố môi trường, ví dụ như đĩa bị đầy, mở một file không tồn tại, mất kết nối mạng, ...

Các ngoại lệ được tạo ra bởi các lỗi chương trình xuất hiện tại thời điểm thi hành chương trình được kế thừa từ lớp RuntimeException. Các ngoại lệ kế thừa từ lớp Error hoặc RuntimeException được gọi là các ngoại lệ không tường minh (implicit exceptions hoặc unchecked). Những ngoại lệ này là do viết chương trình chưa chuẩn. Java không yêu cầu bạn xử lý các ngoại lệ không tường minh. Một số ngoại lệ không tường mình có thể được liệt kê như sau:

* Chuyển đổi giữa các kiểu dữ liệu không hợp lệ (bad cast)
* Truy cập ra ngoài biên giới của mảng (array index out of bound)
* Truy xuất vào đối tượng null (chưa được cấp phát)

Chúng ta có thể tránh được các ngoại lệ này bằng cách kiểm tra kiểu dữ liệu trước khi chuyển, kiểm tra chỉ số phần tử của mảng có vượt ra ngoài biên giới của mảng không trước khi truy xuất chúng, kiểm tra đối tượng có tham chiếu đến null không trước thì truy xuất vào đối tượng này.

Tất cả các ngoại lệ không phân vào nhóm ngoại lệ không tường minh được gọi là ngoại lệ tường minh (explicit exceptions hoặc checked). Những lỗi này yêu cầu lập trình viên phải xử lý khi viết chương trình. Java bắt buộc bạn xác định cách xử lý các ngoại lệ tường mình. Ví dụ khi chúng ta có các thao tác với file thì bắt buộc chúng ta phải xử lý ngoại lệ IOException. Nếu chúng ta không xử lý ngoại lệ này thì chương trình của chúng ta sẽ không được biên dịch. Các ngoại lệ tường minh có thể được xử lý tại khối **catch** trong phương thức hoặc có thể thêm phần **throws** (bỏ qua ngoại lệ) vào nguyên mẫu của phương thức như sau:

public static void textInputFile() **throws IOException**

Như vậy trong phương thức textInputFile() chúng ta có thể sử dụng các câu lệnh thao tác với file trên đĩa mà không cần phải xử lý ngoại lệ. Việc xử lý ngoại lệ sẽ được thực hiện trong phương thức có lời gọi đến phương thức này.

* + 1. Thao tác với ngoại lệ

Một ngoại lệ có thể xuất hiện khi:

* Gọi một phương thức mà phương thức này ném ra một ngoại lệ
* Hệ thống phát hiện lỗi và ném ra một ngoại lệ
* Có lỗi trong lập trình (truy xuất phần tử của mảng có chỉ số nằm ngoài biên giới của mảng, …
* Lỗi nội tại do hệ thống thi hành của Java

Có 2 cách để một phương thức có thể thực hiện khi một ngoại lệ được ném ra từ nó hoặc bởi nó. Chúng ta có thể xử lý ngoại lệ bên trong nó hoặc chuyển việc xử lý ngoại lệ này cho phương thức gọi nó. Nếu chuyển cho phương thức gọi nó thì nguyên mẫu của phương thức này cần bổ sung thêm từ khóa **throws** sau đó là tên lớp xử lý ngoại lệ.

* 1. Tự tạo lớp xử lý ngoại lệ

Đôi khi, bạn muốn ném ra một ngoại lệ từ một phương thức mà ngoại lệ này được tạo ra từ lớp do chính bạn tự định nghĩa bởi vì các lớp xử lý ngoại lệ có sẵn của Java không đủ để mô tả cho lỗi xảy ra.

Chúng ta có thể tạo một lớp xử lý ngoại lệ bằng cách kế thừa lớp Exception hoặc kế thừa từ một lớp con của Exception. Ví dụ bên dưới minh họa việc tự tạo ra lớp ArraySizeException kế thừa từ lớp NegativeArraySizeException như sau:

class ArraySizeException extends NegativeArraySizeException

{

ArraySizeException() // constructor

{

//implicite call to superclass constructor

}

ArraySizeException(String msg) // constructor

{

super(msg);

}

}

class ThrowDemo

{

int size, array[];

ThrowDemo(int s)

{

size=s;

try

{

checkSize();

}

catch(ArraySizeException e)

{

System.out.println(e);

}

}

void checkSize() throws ArraySizeException

{

if (size < 0)

throw new ArraySizeException(“You have passed   
 and Illegal array size”);

else

System.out.println(“The array size is ok.”);

array = new int[3];

for (int i=0; i<3; i++)

array[i] = i+1;

}

public static void main(String arg[])

{

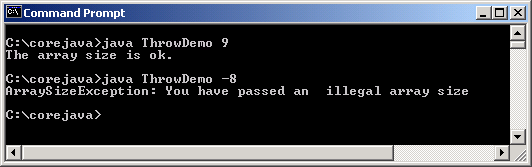
new ThrowDemo(Integer.parseInt(arg[0]));

}

}

Lớp ‘ArraySizeException’ được định nghĩa bởi người dùng là lớp con của lớp ‘NegativeArraySizeException’. Khi một đối tượng được tạo từ lớp này, thông báo về ngoại lệ được in ra. Phương thức ‘checkSize()’ được gọi để tạo ra ngoại lệ ‘ArraySizeException’ được chỉ ra bởi lệnh ‘throws’. Kích thước của mảng được kiểm tra trong cấu trúc ‘if’. Nếu kích thước là số âm thì đối tượng của lớp ‘ArraySizeException’ được tạo.

Kết quả khi chạy chương trình với một số tham số dòng lệnh như hình vẽ 4.2 bên dưới



Minh họa kết quả của ngoại lệ tự định nghĩa

* 1. Cơ chế gom và xử lý rác trong Java

Quá trình thu gom rác (garbage collection) trong máy ảo Java (JVM) là quá trình xác định và loại bỏ các đối tượng không được sử dụng (unreferenced) khỏi bộ nhớ Heap. Không gian trống này sẽ được cấp phát cho những đối tượng mới. Với các ngôn ngữ như C thì việc giải phóng bộ nhớ được thực hiện một cách thủ công (bằng những lệnh khởi tạo, giải phóng bộ nhớ). Với Java thì việc giải phóng bộ nhớ được thực hiện một cách tự động.

Trình thu gom rác (garbage collector) là chương trình chạy nền, nó theo dõi toàn bộ các đối tượng trong bộ nhớ (Heap) và tìm ra những đối tượng nào không được dùng nữa (không có đối tượng nào tham chiếu đến). Toàn bộ những đối tượng không có tham chiếu sẽ bị xóa.

Quá trình thu gom rác cơ bản thông qua 3 bước sau:

* **Marking (đánh dấu)**: Là bước đánh dấu những đối tượng còn sử dụng và những đối tượng không còn sử dụng.
* **Normal deleting (xóa bình thường)**: Trình thu gom rác sẽ xóa các đối tượng không còn sử dụng.
* **Deletion with Compacting (xóa và dồn)**: Sau khi những đối tượng không còn được sử dụng bị xóa, những đối tượng còn được sử dụng sẽ được dồn lại gần nhau. Điều đó làm tăng hiệu xuất sử dụng bộ nhớ trống để cấp phát cho những đối tượng mới.

Tuy nhiên, trình thu gom rác chỉ thực hiện nhiệm vụ khi các đối tượng không còn được tham chiếu gây nên giảm hiệu năng của hệ thống (thiếu bộ nhớ để cấp phát cho các đối tượng mới tạo). Để yêu trình gom rác thực hiện chức năng của mình bất cứ lúc nào thì chúng ta thực hiện 2 câu lệnh sau:

System.gc();

System.runFinalization();

BÀI TẬP CHƯƠNG 4

Bài 1: Viết chương trình Java nhập vào 2 số nguyên từ bàn phím. Tính và hiện thị ra màn hình phép chia 2 số vừa nhập. Yêu cầu có xử lý ngoại lệ khi số chia bằng 0.

Bài 2: Viết lớp xử lý ngoại lệ **InvalidDateException,** lớp nàyhiển thị thông báo lỗi khi thời gian không hợp lệvà lớp **ThoiGian** có 3 thuộc tính: ngày, tháng và năm kiểu số nguyên. Yêu cầu: phương thức tạo đầy đủ của lớp ThoiGian sẽ ném ra ngoại lệ **InvalidDateException** khi đối số ngày, tháng và năm không hợp lệ (ngày, tháng, năm là số âm, tháng lớn hơn 12, ngày lớn hơn 31). Viết lớp Test chứa phương thức main() để kiểm tra việc xử lý ngoại lệ.

Bài 3: Viết lớp xử lý ngoại lệ **StringTooLongException**, lớp này hiển thị thông báo lỗi khi một chuỗi nào đó có quá nhiều kí tự. Viết lớp **Test** chứa hàm **main()** cho phép người dùng nhập các chuỗi kí tự từ bàn phím, kết thúc nhập khi gặp chuỗi “DONE”. Mỗi chuỗi nhập vào sẽ được kiểm tra xem có vượt quá 30 kí tự không, nếu vượt quá thì thông báo cho người dùng biết và kết thúc chương trình. Yêu cầu sử dụng lớp StringTooLongException để xử lý lỗi này.

Bài 4: Thực hiện lại bài tập 3 với yêu cầu, nếu người dùng nhập vào một chuỗi quá 20 kí tự thì chương trình hiển thị thông báo lỗi và vẫn tiếp tục thực hiện.

Bài 5: Viết chương trình Java thực hiện công việc sau:

* Tạo một lớp xử lý ngoại lệ với tên là **OutOfRangeException** để hiển thị thông báo lỗi khi người sử dụng nhập vào một số nằm ngoài phạm vi cho phép.
* Tạo lớp **Test** chứa hàm **main()** cho phép người dùng nhập vào lương của một người. Nếu lương của người vừa nhập không nằm trong phạm vi từ 5.000.000 đến 20.000.000 thì hiển thị thông báo lỗi. Yêu cầu sử dụng lớp **OutOfRangeException** để xử lý ngoại lệ.

1. CÁC LỚP CƠ SỞ VÀ CẤU TRÚC DỮ LIỆU

Chương này cung cấp các lớp tiện ích mà Java đã xây dựng sẵn như lớp Math để thực hiện các thao tác toán học cơ bản, lớp Random để tạo các số ngẫu nhiên, lớp Calendar để thao tác với dữ liệu kiểu thời gian, lớp String, String Builder để thao tác với dữ liệu kiểu chuỗi và đặc biệt là lớp Collection cho phép lưu trữ và xử lý một tập hợp các đối tượng.

* 1. Các lớp cơ bản của Java
     1. Lớp Math

Lớp Math cung cấp các phương thức để thực hiện các thao tác toán học cơ bản như tính logarit, căn bậc hai, trị tuyệt đối, các hàm lượng giác, …

Lớp Math cung cấp 2 hằng số thường được sử dụng sau:

* statich double E chứa giá trị cơ số tự nhiên e
* statich double PI chứa giá trị hằng số pi

Một số hàm toán học thông dụng của lớp Math được liệt kê trong bảng 5.1 sau:

* + - * 1. Một số hàm toán học thông dụng của lớp Math

|  |  |
| --- | --- |
| **Tiền tố và kiểu dữ liệu trả về** | **Phương thức và mô tả** |
| static double | abs(double a) trả về kết quả trị tuyệt đối của đối số a kiểu double |
| static float | abs(float a) trả về kết quả trị tuyệt đối của đối số a kiểu float |
| static int | abs(int a) trả về kết quả trị tuyệt đối của đối a số kiểu int |
| static long | abs(long a) trả về kết quả trị tuyệt đối của đối số a kiểu long |
| static double | cos(double a) trả về kết quả là hàm lượng giác cosine của một góc a tính bằng radian |
| static double | sin(double a) trả về kết quả là hàm lượng giác sine của một góc tính a bằng radian |
| static double | tan(double a) trả về kết quả là hàm lượng giác tang của một góc a tính bằng radian |
| static double | acos(double a) trả về kết quả là hàm lượng giác arc cosine của một góc a tính bằng radian |
| static double | asin(double a) trả về kết quả là hàm lượng giác arc sine của một góc a tính bằng radian |
| static double | atan(double a) trả về kết quả là hàm lượng giác arc tang của một góc a tính bằng radian |
| static double | exp(double a) trả về kết quả là ea |
| static double | pow(double a, double b) trả về kết quả là ab |
| static double | sqrt(double a) trả về kết quả là căn bật 2 của a |
| static double | max(double a, double b) trả về kết quả là giá trị lớn nhất trong 2 số a và b |
| static double | min(double a, double b) trả về kết quả là giá trị nhỏ nhất trong 2 số a và b |
| static double | random() trả về kết quả là số ngẫu nhiên nằm trong phạm vi từ 0 đến 1 |
| static int | round(float a) trả về kết quả là giá trị làm tròn của đối số a |
| static double | ceil(double a) trả về kết quả là số thực tròn gần nhất với a và lớn hơn hoặc bằng a. |
| static double | floor(double a) trả về kết quả là số thực tròn gần nhất với a và nhỏ hơn hoặc bằng a. |

Chương trình Java sau minh họa việc sử dụng một số phương thức của lớp Math để tính giá trị các hàm toán học của đối số nhập vào từ bàn phím.

package mathdemo;

import java.util.\*;

public class MathDemo {

public static void main(String[] args) {

double a = 0;

Scanner sn = new Scanner(System.in);

System.out.print("Mời bạn nhập vào một số   
 thực: ");

a = sn.nextDouble();

//Tính và hiển thị ra màn hình trị tuyệt đối

// của số vừa nhập

System.out.println("Giá trị tuyệt đối của "+a  
 +" la:"+Math.abs(a));

//Tính và hiển thị ra màn hình sine của số

// vừa nhập

System.out.println("Giá trị sine của "+a  
 +" la:"+Math.sin(a));

//Tính và hiển thị ra màn hình sine của số vừa

// nhập

System.out.println("Giá trị e mũ của "+a  
 +" la:"+Math.exp(a));

//Giá trị trội trên của a

System.out.println("Giá trị trội trên của "+a  
 +" la:"+Math.ceil(a));

//Giá trị trội dưới của a

System.out.println("Giá trị trội dưới của "+a   
 +" la:"+Math.floor(a));

//Giá trị căn bậc 3 của a

System.out.println("Giá trị căn bậc 3 của "+a  
 +" la:"+Math.pow(a,1.0/3));

}

}

Kết quả hiển thị lên màn hình sẽ như hình vẽ 5.1 sau:

Mời bạn nhập vào một số thực: 3.14

Giá trị tuyệt đối của 3.14 la:3.14

Giá trị sine của 3.14 la:0.0015926529164868282

Giá trị e mũ của 3.14 la:23.103866858722185

Giá trị trội trên của 3.14 la:4.0

Giá trị trội dưới của 3.14 la:3.0

Giá trị căn bậc 3 của 3.14 la:1.4643443505031195

Kết quả của chương trình mô phỏng các hàm toán học thông dụng

* + 1. Lớp Calendar

Calendar là một lớp trừu tượng cung cấp phương thức tĩnh getInstance() để tạo ra đối tượng Calendar với các thuộc tính được thiết lập bằng thời gian hiện tại.

Câu lệnh Java sau tạo ra đối tượng thuộc lớp Calendar với các thuộc tính được thiết lập bằng thời gian hiện tại.

Calendar ct = Calendar.getInstance();

Sau khi tạo đối tượng thuộc lớp Calendar chúng ta có thể sử dụng các phương thức get() để lấy ra các thông tin về thời gian hiện tại:

* Lấy ngày hiện tại (từ 1 đến 31) sử dụng:

ct.get(Calendar.DATE)

* Lấy tháng hiện tại (từ 0 đến 11) như vậy để tháng tính từ 1 đến 12 thì bạn cần cộng thêm 1 vào kết quả thu được:

ct.get(Calendar.MONTH)

* Lấy năm hiện tại:

ct.get(Calendar.YEAR)

* Lấy giờ hiện tại:

ct.get(Calendar.HOUR)

* Lấy phút hiện tại:

ct.get(Calendar.MINUTE)

* Lấy giây hiện tại:

ct.get(Calendar.SECOND)

Chúng ta sử dụng phương thức set() để thiết lập giá trị cho các thuộc tính giống như phương thức get(). Ví dụ để thiết lập thuộc tính năm là 2019 chúng ta sử dụng câu lệnh set() như sau:

ct.set(Calendar.YEAR,2019);

Để trả về đối tượng Date biểu diễn thời gian hiện tại chúng ta sử dụng phương thức getTime() như sau:

Date cd = ct.getTime();

Để trả về số ms tính từ 1 tháng 1 năm 1970 đến thời điểm hiện tại chúng ta sử dụng phương thức getTimeInMillis() như sau:

long lt = ct. getTimeInMillis();

Để so sánh thời gian mà đối tượng Calendar đang lưu với một đối tượng thời gian khác chúng ta sử dụng 2 phương thức after() và before() có nguyên mẫu như sau: boolean after(Object when) và boolean before(Object when).

Để cộng hoặc trừ một trường thời gian (ngày, tháng, năm, giờ, phút, giây, …) đi một giá trị chúng ta sử dụng phương thức add() với nguyên mẫu như sau: void add(int field, int amount).

Ví dụ sau đây minh họa các phương thức hay sử dụng của đối tượng tạo từ lớp Calendar.

package calendardemo;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

public class CalendarDemo {

public static void main(String[] args) {

Calendar ct = Calendar.getInstance();

int ngay = ct.get(Calendar.DATE);

int thang = ct.get(Calendar.MONTH);

int nam = ct.get(Calendar.YEAR);

int gio = ct.get(Calendar.HOUR);

int phut = ct.get(Calendar.MINUTE);

int giay = ct.get(Calendar.SECOND);

System.out.println("Ngay hiện tại: "+ngay);

System.out.println("Tháng hiện tại: "+thang);

System.out.println("Năm hiện tại: "+nam);

System.out.println("Giờ tại: "+gio);

System.out.println("Phút hiện tại: "+phut);

System.out.println("Giây hiện tại: "+giay);

SimpleDateFormat sdf = new   
 SimpleDateFormat("dd:MM:yyyy");

Date cd = ct.getTime();

System.out.println("Ngày hiện tại là:   
 "+sdf.format(cd));

Calendar ct1 = Calendar.getInstance();

ct1.add(Calendar.DATE, 5);

Date cd1 = ct1.getTime();

System.out.println("Sau 5 ngày hiện tại là:   
 "+sdf.format(cd1));

//kiểm tra xem ngày hiện tại có trước hay sau

// cd1 không

if(ct.before(ct1)){

System.out.println("Ngày hiện tại trước   
 ngày: "+sdf.format(cd1));

}

else{

System.out.println("Ngày hiện tại sau  
 ngày: "+sdf.format(cd1));

}

}

}

Kết quả khi thực hiện chương trình trên như hình 5.2 sau:

*Ngày hiện tại: 26*

*Tháng hiện tại: 0*

*Năm hiện tại: 2018*

*Giờ tại: 4*

*Phút hiện tại: 57*

*Giây hiện tại: 30*

*Ngày hiện tại là: 26:01:2018*

*Sau 5 ngày hiện tại là: 31:01:2018*

*Ngày hiện tại trước ngày: 31:01:2018*

Kết quả chương trình minh họa sử dụng Calendar

* + 1. Lớp Random

Lớp Random dùng để tạo các số giả ngẫu nhiên. Để tạo ra đối tượng Random chúng ta có thể sử dụng các phương thức tạo sau: Random() và Random(long seed). Một số phương thức hay sử dụng của lớp Random được tổng hợp trong bảng 5.2 sau:

* + - * 1. Một số phương thức chính của lớp Random

|  |  |
| --- | --- |
| **Tiền tố và kiểu dữ liệu trả về** | **Phương thức và mô tả** |
| double | nextDouble(): tạo ra số ngẫu nhiên kiểu double trong phạm vi từ 0.0 đến 1.0 |
| float | nextFloat(): tạo ra số ngẫu nhiên kiểu float trong phạm vi từ 0.0 đến 1.0 |
| int | nextInt(): tạo ra số ngẫu nhiên kiểu số nguyên trong phạm vi từ -2147483648 đến 2147483647 |
| int | nextInt(int n): tạo ra các số nguyên ngẫu nhiên từ 0 đến nhỏ hơn n |
| long | nextLong(): tạo ra số nguyên ngẫu nhiên kiểu long nằm trong phạm vi của từ -9223372036854775808 đến 9223372036854775807 |

Ví dụ sau minh họa sử dụng các phương thức của lớp Random để tạo ra các số ngẫu nhiên.

package randomdemo;

import java.util.Random;

public class RandomDemo {

public static void main(String[] args) {

Random r = new Random();

double rd = r.nextDouble();

float rf = r.nextFloat();

int ri = r.nextInt();

int rri = r.nextInt(10);//tạo số nguyên ngẫu

// nhiên từ 0 đến nhỏ hơn 10

System.out.println("Số ngẫu nhiên kiểu double   
 tạo được là: " + rd);

System.out.println("Số ngẫu nhiên kiểu float   
 tạo được là: " + rf);

System.out.println("Số ngẫu nhiên kiểu nguyên   
 tạo được là: " + ri);

System.out.println("Số ngẫu nhiên kiểu kiểu   
 nguyên từ 0 đến 10 là: " + rri);

}

}

Kết quả khi chạy chương trình trên như hình 5.3 bên dưới:

*Số ngẫu nhiên kiểu double tạo được là: 0.9119818195023914*

*Số ngẫu nhiên kiểu float tạo được là: 0.3702464*

*Số ngẫu nhiên kiểu nguyên tạo được là: 908827598*

*Số ngẫu nhiên kiểu kiểu nguyên từ 0 đến 10 là: 7*

Minh họa kết quả chạy chương trình demo lớp Random

* + 1. Lớp String

Đối với ngôn ngữ lập trình Java thì String (xâu) là đối tượng biểu diễn cho một chuỗi kí tự. Đối tượng String được tạo ra từ lớp java.lang.String. Để tạo ra đối tượng String trong Java bạn có thể sử dụng một trong hai cách sau:

* Sử dụng hằng chuỗi kí tự
* Sử dụng từ khóa new

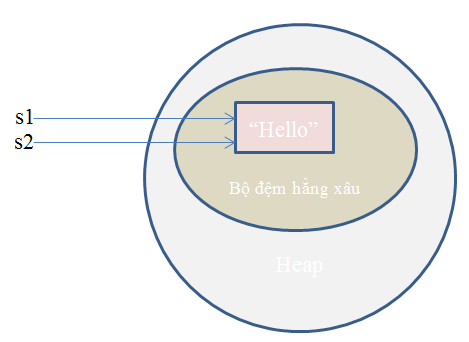
1. ***Sử dụng hằng chuỗi kí tự***

Hằng xâu kí tự trong Java được tạo ra bằng cách bao chuỗi kí tự trong cặp dấu ngáy kép.

String s = “Hello”;

Mỗi khi bạn tạo ra hằng xâu thì máy ảo java (JVM) sẽ kiểm tra trong bộ đệm hằng xâu kí tự trước, nếu hằng xâu này đã tồn tại trong bộ đệm rồi thì một tham chiếu đến hằng xâu này sẽ được trả lại. Nếu hằng xâu này chưa tồn tại thì nó sẽ được tạo và đặt trong bộ đệm hằng xâu. Ví dụ sau câu lệnh thứ hai không tạo ra xâu mới mà chỉ tạo ra một tham chiếu thêm đến xâu đã tạo ra bởi câu lệnh thứ nhất.

String s1 = “Hello”;

String s2 = “Hello”;

Minh họa hoạt động của hằng xâu kí tự.

Trong ví dụ trên chỉ một đối tượng được tạo ra. Đầu tiên máy ảo Java không tìm thấy đối tượng chuỗi nào trong vùng đệm hằng xâu có giá trị bằng “Hello”, do đó nó tạo ra một đối tượng mới cho hằng xâu này. Sau đó, nó sẽ tìm thấy một xâu có giá trị bằng “Hello” ở trong bộ đệm hằng xâu nên sẽ không tạo ra đối tượng mới mà chỉ trả về tham chiếu đến đối tượng đã tạo. Kết quả là cả s1 và s2 cùng tham chiếu đến một đối tượng xâu có giá trị là “Hello” như minh họa trên hình 5.4.

1. ***Sử dụng từ khóa new***

String s = new String(“Hello”);

Trong trường hợp này máy ảo Java sẽ tạo ra một đối tượng xâu đặt trong vùng nhớ heap bình thường (không phải vùng đệm hằng xâu) và một hằng xâu “Hello” được đặt trong vùng đệm và biến *s* tham chiếu đến đối tượng ở vùng nhớ heap.

1. ***Một số phương thức thông dụng của lớp String***

Bảng 5.3 sau tổng hợp một số phương thức thông dụng của lớp String

* + - * 1. Một số phương thức thông dụng của lớp String

|  |  |  |
| --- | --- | --- |
| **STT** | **Phương thức** | **Mô tả** |
| 1 | char charAt(int index) | Trả về kí tự tại vị trí **index** xác định. Vị trí bắt đầu tính từ 0. |
| 2 | int length() | Trả về chiều dài của chuỗi |
| 3 | String substring(int beginIndex) | Trả về xâu con trích từ vị trí **beginIndex** |
| 4 | String substring(int beginIndex,int endIndex) | Trả về xâu con trích từ vị trí **beginIndex** đến **endIndex** (không lấy kí tự tại vị trí endIndex) |
| 5 | boolean contains(CharSequence s) | Kiểm tra xem xâu có chứa chuỗi kí tự s hay không. Trả về true nếu có chữa và false ngược lại. |
| 6 | boolean equals(Object o) | Kiểm tra xem hai xâu có bằng nhau hay không. Trả về true nếu bằng nhau, false ngược lại. |
| 7 | boolean isEmpty() | Kiểm tra xem xâu có rỗng không. Trả về true nếu xâu rỗng và false nếu ngược lại. |
| 8 | String toLowerCase() | Trả về xâu chữ thường |
| 9 | String toUpperCase() | Trả về xâu chữ hoa |
| 10 | String[] split(String regex) | Phân tách xâu thành các xâu con, dấu hiệu phân tách là regex. |
| 11 | String trim() | Xóa tất cả các dấu cách ở đầu và cuối xâu |

Ví dụ minh họa sử dụng một số phương thức của lớp String.

public class StringExample{

public static void main(String args[]){

String s1="java";//Tạo xâu sử dụng hằng chuỗi kí tự

char ch[]={'s','t','r','i','n','g','s'};

String s2=new String(ch);//chuyển mảng kí tự thàng xâu

String s3=new String("example");//Tạo xâu bằng sử dụng từ khóa new

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

System.out.println(“Chiều dài của xâu thứ nhất là: “+s1.length());

System.out.println(“Xâu thứ 2 được chuyển sang chữ hoa là: ”+s2.toUpperCase());

System.out.println(“Kí tự đầu tiên của xâu thứ ba la: ”+s3.charAt(0));

}

}

Kết quả hiển thị lên màn hình khi chạy chương trình trên là:

java

strings

example

Chiều dài của xâu thứ nhất là: 4

Xâu thứ 2 được chuyển sang chữ hoa là: STRINGS

Kí tự đầu tiên của xâu thứ ba là: e

* + 1. Lớp StringBuffer

Trong khi lớp String tạo ra đối tượng chuỗi không thể thay đổi được, điều này có nghĩa là bạn chỉ cần bổ sung thêm một kí tự vào xâu được tạo ra từ lớp String cũng sẽ dẫn đến tạo ra đối tượng mới. Do đó chương trình sẽ chạy lâu và tốn bộ nhớ nếu chúng ta sử dụng đối tượng String để ghép nhiều xâu lại với nhau bởi vì mỗi lần ghép một xâu lại tạo ra đối tượng mới. Để khắc phục nhược điểm này thì ngôn ngữ lập trình Java cung cấp thêm lớp StringBuffer tương tự như lớp String nhưng cho phép tạo xâu kí tự có thể thay đổi được.

Một số phương thức tạo quan trọng của lớp StringBuffer được tổng hợp trong bảng 5.4 sau:

* + - * 1. Một số phương thức tạo quan trọng của lớp StringBuffer

|  |  |
| --- | --- |
| Phương thức tạo | Mô tả |
| StringBuffer() | Tạo ra một đối tượng StringBuffer rỗng với kích thước ban đầu là 16. |
| StringBuffer(String s) | Tạo ra một đối tượng StringBuffer với giá trị ban đầu là xâu s. |
| StringBuffer(int capacity) | Tạo ra một đối tượng StringBuffer rỗng với kích thước ban đầu là capacity. |

Một số phương thức quan trọng của lớp StringBuffer được tổng hợp trong bảng 5.5 sau:

* + - * 1. Một số phương thức quan trọng của lớp StringBuffer

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| public synchronized StringBuffer append(String s) | Bổ sung thêm chuỗi **s** vào cuối xâu. |
| public synchronized StringBuffer insert(int offset, String s) | Chèn chuỗi **s** vào xâu tại vị trí **offset** (vị trí tính bắt đầu từ 0). |
| public synchronized StringBuffer replace(int startIndex, int endIndex, String s) | Thay chuỗi con bắt đầu từ **startIndex** đến **endIndex-1** bằng chuỗi **s**. |
| public synchronized StringBuffer delete(int startIndex,int endIndex) | Xóa chuỗi con từ vị trí **startIndex** đến **endIndex-1**. |
| public synchronized StringBuffer reverse() | Đảo ngước xâu |
| public int capacity() | Trả về dung lượng hiện tại của xâu |
| public int length() | Trả về kích thước của xâu (số kí tự trong xâu) |
| public char charAt(int index) | Trả về kí tự tại vị trí xác định **index** trong xâu. |
| public String substring(int startIndex,int endIndex) | Trả về xâu con tính từ **startIndex** đến **endIndex-1**. |
| public String substring(int beginIndex) | Trả về xâu con từ vị trí **beginIndex** đến cuối chuỗi. |

Ví dụ minh họa sử dụng một số phương thức của lớp StringBuffer.

public class StringBufferDemo {

public static void main(String[] args) {

StringBuffer s = new StringBuffer("Hello");

System.out.println("Xâu ban đầu: "+s);

s.append(" Java");

System.out.println("Xâu sau khi chèn thêm: "+s);

s.replace(0, 5, "Welcome");

System.out.println("Xâu sau khi thay thế: "+s);

}

}

Kết quả hiển thị lên màn hình khi chạy chương trình trên như sau:

Xâu ban đầu: Hello

Xâu sau khi chèn thêm: Hello Java

Xâu sau khi thay thế: Welcome Java

* + 1. Lớp StringBuilder

Giống như lớp StringBuffer, StringBuilder cũng tạo đối tượng xâu có thể thay đổi được nhưng các phương thức của StringBuilder là không đồng bộ (non – synchronized). Điều này có nghĩa là hai Thread không được phép gọi đồng thời đến phương thức của lớp StringBuffer còn các phương thức của StringBuilder cho phép nhiều Thread gọi đồng thời. Ngoài ra, các phương thức của lớp StringBuilder chạy nhanh hơn các phương thức của lớp StringBuffer. Do đó, bạn nên sử dụng lớp StringBuilder cho lập trình đơn luồng và StringBuffer cho lập trình đa luồng (multi threading). Lớp StringBuilder được bổ sung bắt đầu từ phiên bản JDK 1.5.

Một số phương thức tạo quan trọng của lớp StringBuilder được tổng hợp trong bảng 5.6 sau:

* + - * 1. Một số phương thức tạo chính của lớp StringBuilder

|  |  |
| --- | --- |
| **Phương thức tạo** | **Mô tả** |
| StringBuilder() | Tạo đối tượng rỗng StringBuilder với dụng lượng mặc định là 16 |
| StringBuilder(String s) | Tạo đối tượng StringBuilder khởi tạo giá trị ban đầu là xâu **s**. |
| StringBuilder(int capacity) | Tạo đối tượng rỗng StringBuilder với dung lương ban đầu là **capacity** |

Một số phương thức quan trọng của lớp StringBuilder tổng hợp trong bảng 5.7 sau:

* + - * 1. Một số phương thức quan trọng của lớp StringBuilder

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| public StringBuilder append(String s) | Bổ sung thêm chuỗi **s** vào cuối xâu. |
| public StringBuilder insert(int offset, String s) | Chèn chuỗi s vào xâu tại vị trí **offset** (vị trí tính bắt đầu từ 0). |
| public StringBuilder replace(int startIndex, int endIndex, String s) | Thay chuỗi con bắt đầu từ startIndex đến **endIndex-1** bằng chuỗi **s**. |
| public StringBuilder delete(int startIndex,int endIndex) | Xóa chuỗi con từ vị trí **startIndex** đến **endIndex-1**. |
| public StringBuilder reverse() | Đảo ngước xâu |
| public int capacity() | Trả về dung lượng hiện tại của xâu |
| public int length() | Trả về kích thước của xâu (số kí tự trong xâu) |
| public char charAt(int index) | Trả về kí tự tại vị trí xác định index trong xâu. |
| public String substring(int startIndex,int endIndex) | Trả về xâu con tính từ **startIndex** đến **endIndex-1**. |
| public String substring(int beginIndex) | Trả về xâu con từ vị trí **beginIndex** đến cuối chuỗi. |

Ví dụ minh họa sử dụng một số phương thức của lớp StringBuilder:

public class StringBuilderDemo {

public static void main(String[] args) {

StringBuilder s = new StringBuilder("Hello");

System.out.println("Xâu ban đầu: "+s);

s.append(" Java");

System.out.println("Xâu sau khi chèn thêm: "+s);

s.replace(0, 5, "Welcome");

System.out.println("Xâu sau khi thay thế: "+s);

}

}

Kết quả hiển lên màn hình sau khi thực hiện chương trình trên như sau:

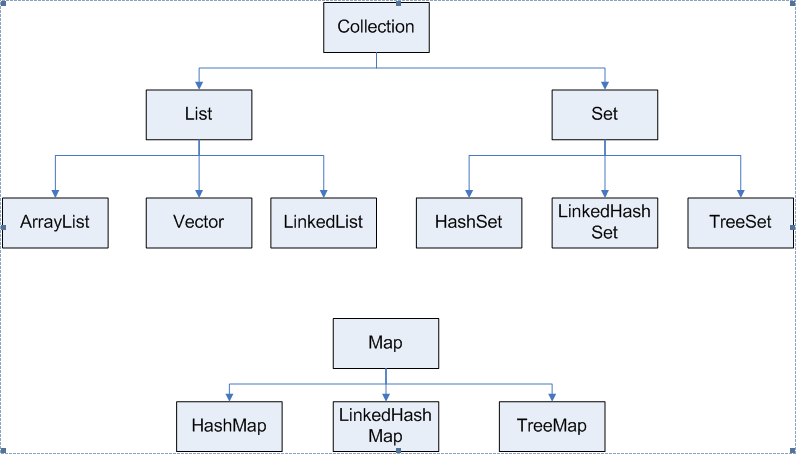
Xâu ban đầu: Hello

Xâu sau khi chèn thêm: Hello Java

Xâu sau khi thay thế: Welcome Java

* 1. Cấu trúc tuyển tập đối tượng (collection framework)

Collection là kiểu dữ liệu tập hợp cho phép lưu trữ và thao tác với một nhóm các đối tượng. Hình vẽ 5.4 bên dưới là biểu đồ lớp của Collection và Map trong Java.

Biểu đồ lớp của Collection và Map

* + 1. List interface
       1. Lớp ArrayList

Lớp ArrayList thực thi List interface dùng để cài đặt danh sách bằng mảng, tuy nhiên có 2 đặc điểm khác mảng:

* Không cần khai báo trước kiểu phần tử
* Không cần xác định trước số lượng phần tử (kích thước mảng)
* Phương thức khởi tạo
* ArrayList()
* ArrayList(Collection c)
* ArrayList(int initialCapactity)
* Các phương thức chính:
  + boolean add(Object o) dùng để thêm một đối tượng vào cuối của mảng.
  + void add(int index, Object o) dùng để thêm một đối tượng vào vị trí xác định trong mảng.
  + Object remove(int intdex) dùng để xóa một đối tượng ra khỏi mảng tại vị trí xác định và trả về đối tượng được xóa.
  + boolean remove(Object o) dùng để xóa một đối tượng ra khỏi mảng và trả về true nếu xóa được đối tương o, false nếu ngược lại.
  + Object get(int index) dùng để lấy đối tượng tại vị trí index (phần tử đầu tiên có vị trí là 0).
  + int size() trả về số phần tử được lưu trữ trong mảng.
  + boolean isEmpty()kiểm tra xem mảng có rỗng không.
  + boolean contains(Object o)kiểm tra xem mảng có chứa đối tượng o không?
  + void clear()xóa tất cả các phần tử khỏi mảng.

Ví dụ minh họa sử dụng ArrayList:

public static void main(String[] args) {

ArrayList al = new ArrayList();

Point p = new Point(1,1);

Integer i = new Integer(2);

Double d = new Double(3);

al.add(p);

al.add(i);

al.add(d);

// Cấu trúc for – each trong java: duyệt tất

// cả các phần tử của arrayList

for (Object o:al){

System.out.println(o);

}

}

Kết quả hiển thị ra màn hình khi chạy chương trình trên sẽ như sau:

1.0 1.0

2

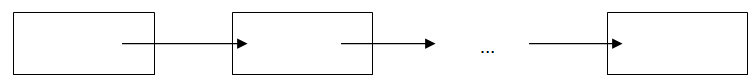
3.0

5.2.1.2 Lớp Vector

Lớp Vector giống như lớp ArrayList về chức năng chỉ khác biệt là các phương thức của Vector là Synchronized. VD: Khi có nhiều thread cùng thêm (add) hoặc xóa (remove) 1 phần tử trong Vector thì chỉ 1 thread được làm việc đó, các thread khác bị lock. Khi không làm việc với thread thì nên sử dụng ArrayList vì tốc độ nhanh hơn Vector.

* + - 1. Lớp LinkedList

Về cấu trúc, các object trong LinkedList chứa liên kết đến object kế sau nó trong list. Khi duyệt list, không truy nhập trực tiếp được mà phải đi từ phần tử đầu tiên. Do đó, việc truy cập các phần tử của danh sách liên kết chậm hơn rất nhiều so với ArrayList hay Vector. Tuy nhiên khi thêm hay xóa 1 phần tử thì rất nhanh vì không phải thực hiện dồn các phần tử. Hình vẽ 5.5 minh họa liên kết giữa các phần tử của danh sách liên kết.

Minh họa danh sách liên kết

Trong Java, vẫn có thể dùng phương thức *get(n)* để lấy phần tử thứ *n* trong 1 danh sách liên kết, tuy nhiên về bản chất Java vẫn phải duyệt qua *n-1* phần tử khác phía trước khi tìm được đến phần tử thứ *n*.

Ngoài các phương thức như ArrayList, LinkedList bổ sung thêm các phương thức sau:

* void addFirst(E e): Thêm phần tử e vào đầu danh sách
* void addLast(E e): Thêm phần tử e vào cuối danh sách
* E getFirst(): Trả về phần tử đầu tiên của danh sách
* E getLast(): Trả về phần tử cuối cùng của danh sách
* E removeFirst(): Xóa phần tử đầu tiên ra khỏi danh sách và trả về phần tử bị xóa.
* E removeLast(): Xóa phần tử cuối cùng ra khỏi danh sách và trả về phần tử bị xóa.
* E peek(): Trả về phần tử đầu tiên trong danh sách. Tương tự như phương thức getFirst().
* E poll(): Trả về phần tử đầu tiên của danh sách, đồng thời xóa phần tử này ra khỏi danh sách. Phương thức này tương tự như phương thức removeFirst().

Ví dụ minh họa sử dụng LinkedList:

package listdemo;

import java.awt.Point;

import java.util.LinkedList;

public class ListDemo {

public static void main(String[] args) {

LinkedList ll = new LinkedList();

Point p = new Point(1, 1);

Integer i = new Integer(2);

Double d = new Double(3);

ll.add(p);

ll.addFirst(i);

ll.addLast(d);

System.out.println("Phần tử đầu tiên của danh   
 sách: " + ll.getFirst());

System.out.println("Phần tử cuối cùng của   
 danh sách: " + ll.getLast());

System.out.println("Danh sách ban đầu:");

for(Object o:ll){

System.out.println(o);

}

ll.removeFirst();

System.out.println("Danh sách sau khi xóa   
 phần tử đầu tiên:");

for(Object o:ll){

System.out.println(o);

}

}

}

Kết quả khi chạy chương trình như sau:

Phần tử đầu tiên của danh sách: 2

Phần tử cuối cùng của danh sách: 3.0

Danh sách ban đầu:

2

java.awt.Point[x=1,y=1]

3.0

Danh sách sau khi xóa phần tử đầu tiên:

java.awt.Point[x=1,y=1]

3.0

**Khi nào dùng ArrayList, khi nào dùng LinkedList?**

* Nếu chương trình thường xuyên phải truy nhập trực tiếp dữ liệu trong list thì sử dụng **ArrayList**
* Nếu chương trình thường xuyên phải thêm hoặc xóa 1 phần tử trong list (đặc biệt là thêm hoặc xóa ở giữa) thì nên sử dụng **LinkedList**
  + 1. Set interface

Cũng là kiểu tập hợp như list. Điểm khác biệt giữa Set và List là trong Set không bao giờ có 2 phần tử giống nhau, có nghĩa là nếu thêm 2 lần cùng 1 đối tượng thì lần thêm thứ 2 không có tác dụng.

Ví dụ minh họa:

Point p = new Point(1,2);

HashSet hs = new HashSet();

hs.add(p);

hs.add(p);//Lần add này không có tác dụng.

* + - 1. Lớp HashSet

Lớp **HashSet** được sử dụng để tạo tập đối tượng sử dụng bảng băm (hash table) để lưu trữ. Nó kế thừa từ lớp trừu tượng **AbstractSet** và thực thi interface **Set** có các phương thức tạo và các phương thức chính sau:

***Phương thức tạo***

* HashSet(): Tạo một HashSet rỗng
* HashSet(Collection c): Tạo một HashSet mà các phần tử lấy từ một tuyển tập c
* HashSet(int capactity): Tạo một HashSet có dung lượng là capacity.

***Phương thức chính***

* boolean add(Object o): Thêm đối tượng vào HashSet nếu đối tượng này chưa tồn tại trong HashSet. Phương thức trả về true nếu thêm thành công và ngược lại.
* boolean remove(Object o): Phương thức xóa đối tượng o ra khỏi tập tập hợp nếu đối tượng này tồn tại.
* boolean contains(Object o): Phương thức kiểm tra xem đối tượng o có tồn tại trong set hay không. Phương thức trả về true nếu đối tượng o tồn tại trong set và ngược lại.
* int size(): Phương thức trả về số phần tử của set.
* boolean isEmpty(): Phương thức kiểm tra xem set có rỗng không. Phương thức trả về true nếu set là rỗng (không có phần tử nào) và trái lại.
* void clear(): Xóa tất cả các phần tử của set.

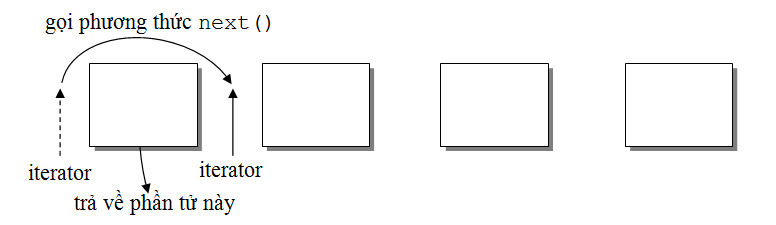
Tuy nhiên, chúng ta không thể sử dụng phương thức get() để lấy ra các phần tử của set mà chúng ta phải sử dụng con trỏ iterator để duyệt set.

***Con trỏ iterator để duyệt set***

Iterator dùng để duyệt các phần tử của Collection. Để sử dụng iterator để duyệt các phần tử của tuyển tập chúng ta thực hiện qua các bước sau:

* Bước 1: Tạo con trỏ iterator trỏ vào đầu tuyển tập bằng cách sử dụng phương thức iterator của collection.
* Bước 2: Sử dụng vòng lặp while gọi phương thức hasNext() của iterator để kiểm tra xem đã đến cuối của collection chưa. Vòng lặp kết thúc khi phương thức hasNext() trả về false.
* Bước 3: Trong thân vòng lặp while để nhận mỗi phần tử chúng ta gọi phương thức next().

Hình vẽ 5.6 minh họa hoạt động của iterator. Khi mới tạo thì iterator đứng trước phần tử đầu tiên của collection, nếu chúng ta gọi phương thức hasNext() sẽ trả về true, nên câu lệnh next() trong thân vòng lặp while được thực hiện sẽ làm con trỏ iterator nhảy quả phần tử đầu tiên để đến trước phần tử thứ 2 và trả về là phần tử đầu tiên mà nó đã nhảy qua.

Minh họa hoạt động của iterator

Ví dụ minh họa sử dụng HashSet và iterator để duyệt các phần tử của nó.

package hashsetdemo;

import java.util.HashSet;

import java.util.Iterator;

public class HashSetDemo {

public static void main(String[] args) {

HashSet hs = new HashSet();

hs.add("A");

hs.add("B");

hs.add(1);

hs.add(3.2);

Iterator ir = hs.iterator();

while(ir.hasNext()){

System.out.println(ir.next().toString());

}

}

}

Kết quả hiển thị khi chạy chương trình trên như sau:

A

1

B

3.2

Mặc dù “B” được thêm (add) vào trước nhưng khi duyệt lại xuất hiện sau. Nguyên nhân là HashSet không duy trì thứ tự các phần tử khi add.

* + - 1. Lớp LinkedHashSet

Giống như HashSet ngoại trừ thứ tự các phần tử trong set là thứ tự khi add vào.

Ví dụ minh họa LinkedHashSet như sau:

package hashsetdemo;

import java.util.Iterator;

import java.util.LinkedHashSet;

public class HashSetDemo {

public static void main(String[] args) {

LinkedHashSet hs = new LinkedHashSet();

hs.add("A");

hs.add("B");

hs.add(1);

hs.add(3.2);

Iterator ir = hs.iterator();

while(ir.hasNext()){

System.out.println(ir.next().toString());

}

}

}

Kết quả hiển thị thi thực hiện chương trình trên như sau:

A

B

1

3.2

5.2.2.3 Lớp TreeSet

Lớp TreeSet thực thi interface Set sử dụng cấu trúc cây (tree) để lưu trữ dữ liệu. Các đối tượng của TreeSet được lưu trữ theo thứ tự tăng dần. Do đó yêu cầu các đối tượng thêm vào TreeSet phải chỉ ra tiêu chí so sánh. Nói cách khác, các đối tượng thêm vào TreeSet phải được tạo ra từ lớp thực thi interface Comparable và viết đè phương thức compareTo(). Ví dụ, khi thêm vào set các đối tượng kiểu String thì các đối tượng này sẽ được sắp theo thứ tự alphabet vì lớp String thực thi interface Comparable và đã viết đè phương thức compareTo().

Ví dụ minh họa sử dụng TreeSet để lưu trữ các đối tượng kiểu String:

package treesetdemo;

import java.util.Iterator;

import java.util.TreeSet;

public class TreeSetDemo {

public static void main(String[] args) {

TreeSet ts = new TreeSet();

ts.add("B");

ts.add("A");

ts.add("C");

Iterator ir = ts.iterator();

while(ir.hasNext()){

System.out.println(ir.next().toString());

}

}

}

Kết quả hiển thị lên màn hình khi thực thi chương trình trên

A

B

C

Ví dụ minh họa khi lưu các đối tượng kiểu Book vào TreeSet thì yêu cầu lớp Book phải thực thị interface Comparable như sau:

package treesetbookdemo;

public class Book implements Comparable<Book> {

private int isbn;

private String ten;

private String tacGia;

private int namXuatBan;

private String nhaXuatBan;

public Book(int isbn, String ten, String tacGia, int namXuatBan, String nhaXuatBan) {

this.isbn = isbn;

this.ten = ten;

this.tacGia = tacGia;

this.namXuatBan = namXuatBan;

this.nhaXuatBan = nhaXuatBan;

}

@Override

public String toString() {

return isbn + "\t" + ten + "\t" + tacGia +  
 "\t" + namXuatBan + "\t" + nhaXuatBan;

}

@Override

public int compareTo(Book o) {

if(this.isbn>o.isbn){

return 1;

}

else if(this.isbn<o.isbn){

return -1;

}

else{

return 0;

}

}

}

package treesetbookdemo;

import java.util.Iterator;

import java.util.TreeSet;

public class TreeSetBookDemo {

public static void main(String[] args) {

TreeSet ts = new TreeSet();

Book b1 = new Book(100,"Lập trình Java căn   
 bản","Nguyễn Bá Nghiễn",2018,"Giáo dục");

Book b2 = new Book(101,"Lập trình mạng","Ngô   
 Văn Bình",2018,"Giáo dục");

Book b3 = new Book(102,"Tính toán   
 mềm","Nguyễn Bá Nghiễn",2018,"Giáo dục");

ts.add(b3);

ts.add(b1);

ts.add(b2);

Iterator bir = ts.iterator();

while(bir.hasNext()){

System.out.println(bir.next());

}

}

}

Kết quả hiển thị ra màn hình khi thực thi chương trình trên:

100 Lập trình Java căn bản Nguyễn Bá Nghiễn 2018 Giáo dục

101 Lập trình mạng Ngô Văn Bình 2018 Giáo dục

102 Tính toán mềm Nguyễn Bá Nghiễn 2018 Giáo dục

* + 1. Map interface

Map interface lưu dữ liệu theo kiểu từ điển, mỗi phần tử của map là một cặp key (khóa) và value (giá trị). Giá trị của khóa phải là duy nhất (không có 2 phần tử trùng khóa). Map sử dụng hiệu quả trong các bài toán tìm kiếm, cập nhật, xóa dựa vào khóa.

* + - 1. Lớp HashMap

Lớp HashMap thực thi interface Map lưu trữ dữ liệu sử dụng bảng băm.

Các phương thức tạo của HashMap được tổng hợp trong bảng 5.7 sau

* + - * 1. Các phương thức tạo của lớp HashMap

|  |  |
| --- | --- |
| **Phương thức tạo** | **Mô tả** |
| HashMap() | Sử dụng để tạo HashMap mặc định. |
| HashMap(Map m) | Khởi tạo HashMap với các phần tử là của một Map khác m. |
| HashMap(int capacity) | Tạo HashMap có dung lượng là capacity. |
| HashMap(int capacity, float fillRatio) | Tạo HashMap có dung lượng và hệ số tải. |

Các phương thức chính:

* Object put(Object key, Object value): Dùng để thêm một phần tử vào map.
* Object get(Object key): Trả về giá trị tương ứng với khóa.
* Object remove(Object key): Xóa phần tử có khóa (key) ra khỏi map nếu tồn tại và trả về giá trị của phần tử này.
* boolean containsKey(Object key): Trả về true nếu trong map có chứa phần tử có khóa bằng với khóa cần tìm và ngược lại.
* boolean containsValue(Object value): Trả về true nếu trong map có chứa phần tử có giá trị bằng với value cần tìm và ngước lại.
* int size(): Trả về số phần tử được lưu trong map (kích thước của map).
* boolean isEmpty(): Trả về true nếu map là rỗng (không chứa phần tử nào) và ngược lại.

Ví dụ minh họa sử dụng HashMap

package hashmapdemo;

import java.util.HashMap;

import java.util.Map;

public class HashMapDemo {

public static void main(String[] args) {

HashMap<Integer, String> hm = new HashMap();

hm.put(100, "Hoàng");

hm.put(102, "Hồng");

hm.put(101, "Quỳnh");

for (Map.Entry m : hm.entrySet()) {

System.out.println(m.getKey() + " " + m.getValue());

}

}

}

Kết quả hiển thị lên màn hình khi thực thi chương trình trên như sau:

100 Hoàng

101 Quỳnh

102 Hồng

* + - 1. Lớp LinkedHashMap

Lớp LinkedHashMap tương tự như lớp HashMap nhưng các phần tử được duy trì thứ tự như đúng thứ tự chèn vào map. Ví dụ sau minh họa sử dụng LinkedHashMap.

package linkedhashmapdemo;

import java.util.LinkedHashMap;

import java.util.Map;

public class HashMapDemo {

public static void main(String[] args) {

LinkedHashMap<Integer, String> hm = new LinkedHashMap();

hm.put(100, "Hoàng");

hm.put(102, "Hồng");

hm.put(101, "Quỳnh");

for (Map.Entry m : hm.entrySet()) {

System.out.println(m.getKey() + " " +   
 m.getValue());

}

}

}

Kết quả hiển thị lên màn hình khi thực thi chương trình trên như sau:

100 Hoàng

102 Hồng

101 Quỳnh

* + - 1. Lớp TreeMap

Lớp TreeMap thực thi interface Map và sử dụng cấu trúc cây để lưu trữ các phần tử. Như vậy TreeMap lưu các phần tử theo chiều tăng dần của giá trị khóa. Ví dụ sau minh họa sử dụng TreeMap.

package treemapdemo;

import java.util.Map;

import java.util.TreeMap;

public class HashMapDemo {

public static void main(String[] args) {

TreeMap<Integer, String> hm = new TreeMap();

hm.put(102, "Hoàng");

hm.put(101, "Hồng");

hm.put(100, "Quỳnh");

for (Map.Entry m : hm.entrySet()) {

System.out.println(m.getKey() + " " + m.getValue());

}

}

}

Kết quả hiển thị lên màn hình khi thực thi chương trình trên như sau:

100 Quỳnh

101 Hồng

102 Hoàng

* + - 1. Lớp HashTable

Lớp HashTable cài đặt cấu trúc dữ liệu bảng băm, nó thực thi interface Map. HashTable chính là mảng của các danh sách. Mỗi danh sách gọi là bucket, vị trí của bucket được xác định dựa vào việc gọi phương thức hashCode(). HashTable chứa giá trị dựa vào khóa.

Phương thức tạo của HashTable như sau:

* HashTable(): Tạo một bảng băm mặc định
* HashTable(int size): Tạo bảng băm có kích thước size.
* HashTable(int size,float ratio): Tạo bảng băm có kích thước size và hệ số tải ratio.

Lớp HashTable giống như lớp HashMap về chức năng chỉ khác biệt là các phương thức của HashTable là Synchronized. VD: Khi có nhiều thread cùng thêm (put) hoặc xóa (remove) 1 phần tử trong HashTable thì chỉ 1 thread được làm việc đó, các thread khác bị lock. Khi không làm việc với thread thì nên sử dụng HashMap vì tốc độ nhanh hơn HashTable.

Ví dụ sau minh họa sử dụng HashTable

package hashmapdemo;

import java.util.Map;

import java.util.\*;

public class HashMapDemo {

public static void main(String[] args) {

Hashtable<Integer,String> hm=new Hashtable();

hm.put(100, "Hoàng");

hm.put(102, "Hồng");

hm.put(101, "Quỳnh");

for (Map.Entry m : hm.entrySet()) {

System.out.println(m.getKey() + " " + m.getValue());

}

}

}

Kết quả hiển thị lên màn hình khi thực thi chương trình trên như sau:

102 Hồng

101 Quỳnh

100 Hoàng

* 1. Phương thức tìm kiếm và sắp xếp của Collections
     1. Sắp xếp

Để sắp xếp các đối tượng trong một danh sách (ArrayList hoặc LinkedList) theo tiêu chí nào đó Java cung cấp phương thức sort() có 2 nguyên mẫu sau:

* public static  void sort(List<T> list); Sử dụng phương thức này yêu cầu các phần tử T của List phải thực thi interface comparable.
* public static void sort(List<T> list,Comparator c); Sử dụng phương thức này chúng ta phải truyền một đối tượng c thực thi interface Comparator và phải viết đè phương thức compare().

Ví dụ sau minh họa việc sắp xếp danh sách sinh viên theo điểm sử dụng cả 2 cách liết kê bên trên.

* + - * 1. **Cách 1**: Đối tượng cần sắp xếp thực thi interface Comparable

package liststudentsort;

public class Student implements Comparable<Student>{

private String code;

private String name;

private int yob;//nam sinh

private int mark;

public Student(String code, String name, int yob, int mark) {

this.code = code;

this.name = name;

this.yob = yob;

this.mark = mark;

}

@Override

public String toString() {

return code + " " + name + " " + yob + " "   
 + mark;

}

@Override

public int compareTo(Student o) {

if(this.mark>o.mark){

return 1;

}

else if(this.mark<o.mark){

return -1;

}

else return 0;

}

}

package liststudentsort;

import java.util.ArrayList;

import java.util.Collections;

public class ListStudentSort {

public static void main(String[] args) {

ArrayList<Student> al = new ArrayList();

Student s1 = new Student("100","Nguyễn Thị   
 Lan",1997,9);

Student s2 = new Student("101","Nguyễn Văn   
 Hùng",1997,10);

Student s3 = new Student("102","Nguyễn Thị   
 Hằng",1997,8);

al.add(s1);

al.add(s2);

al.add(s3);

System.out.println("Danh sách sinh viên trước   
 khi sắp xếp");

for(Student s:al){

System.out.println(s);

}

Collections.sort(al);

System.out.println("Danh sách sinh viên sắp   
 xếp theo điểm tăng dần");

for(Student s:al){

System.out.println(s);

}

}

}

Kết quả khi thực hiện chương trình trên như sau:

Danh sách sinh viên trước khi sắp xếp

100 Nguyễn Thị Lan 1997 9

101 Nguyễn Văn Hùng 1997 10

102 Nguyễn Thị Hằng 1997 8

Danh sách sinh viên sắp xếp theo điểm tăng dần

102 Nguyễn Thị Hằng 1997 8

100 Nguyễn Thị Lan 1997 9

101 Nguyễn Văn Hùng 1997 10

* + - * 1. **Cách 2: Tạo lớp theo tiêu chí cần sắp xếp thực thi interface Comparator**
* Lớp Student là đối tượng cần sắp xếp

package liststudentsort;

public class Student{

private String code;

private String name;

private int yob;//nam sinh

private int mark;

public Student(String code, String name, int yob, int mark) {

this.code = code;

this.name = name;

this.yob = yob;

this.mark = mark;

}

public String getCode() {

return code;

}

public String getName() {

return name;

}

public int getYob() {

return yob;

}

public int getMark() {

return mark;

}

@Override

public String toString() {

return code + " " + name + " " + yob + " "   
 + mark;

}

}

* Lớp SortByMark là xác định tiêu chí sắp xếp theo điểm

package liststudentsort;

import java.util.Comparator;

public class SortByMark implements Comparator<Student> {

@Override

public int compare(Student o1, Student o2) {

if(o1.getMark()>o2.getMark()){

return 1;

}

else if(o1.getMark()<o2.getMark()){

return -1;

} else{

return 0;

}

}

}

* Lớp kiểm tra phương thức sort của Collections

package liststudentsort;

import java.util.ArrayList;

import java.util.Collections;

public class ListStudentSort {

public static void main(String[] args) {

ArrayList<Student> al = new ArrayList();

Student s1 = new Student("100","Nguyễn Thị  
 Lan",1997,9);

Student s2 = new Student("101","Nguyễn Văn   
 Hùng",1997,10);

Student s3 = new Student("102","Nguyễn Thị   
 Hằng",1997,8);

al.add(s1);

al.add(s2);

al.add(s3);

System.out.println("Danh sách sinh viên trước khi sắp xếp");

for(Student s:al){

System.out.println(s);

}

SortByMark sm = new SortByMark();

Collections.sort(al,sm);

System.out.println("Danh sách sinh viên sắp   
 xếp theo điểm tăng dần");

for(Student s:al){

System.out.println(s);

}

}

}

Kết quả hiển thị lên màn hình khi thực thi chương trình trên.

Danh sách sinh viên trước khi sắp xếp

100 Nguyễn Thị Lan 1997 9

101 Nguyễn Văn Hùng 1997 10

102 Nguyễn Thị Hằng 1997 8

Danh sách sinh viên sắp xếp theo điểm tăng dần

102 Nguyễn Thị Hằng 1997 8

100 Nguyễn Thị Lan 1997 9

101 Nguyễn Văn Hùng 1997 10

Nếu chúng ta thực thi interface Comparable thì chúng ta chỉ sắp xếp được theo một tiêu chí. Nếu chúng ta muốn sắp xếp theo nhiều tiêu chí thì chúng ta phải tiến hành theo cách 2, tức là chúng ta phải tạo lớp thực thi interface Comparator. Thật vậy, với bài toán trên nếu yêu cầu sắp xếp sinh viên theo tên khi đó nếu thực hiện theo cách 2 chúng ta chỉ cần bổ sung thêm lớp SortByName và chỉnh sửa một chút trong phương thức main() như sau là chúng ta đã có thể sắp xếp danh sách sinh viên theo tên.

package liststudentsort;

import java.util.Comparator;

public class SortByName implements Comparator<Student> {

@Override

public int compare(Student o1, Student o2) {

// chúng ta sử dụng phương thức compareTo có

// sẵn của lớp String

return o1.getName().compareTo(o2.getName());

}

}

package liststudentsort;

import java.util.ArrayList;

import java.util.Collections;

public class ListStudentSort {

public static void main(String[] args) {

ArrayList<Student> al = new ArrayList();

Student s1 = new Student("100","Lan",1997,9);

Student s2 = new Student("101","Hung",1997,10);

Student s3 = new Student("102","Hang",1997,8);

al.add(s1);

al.add(s2);

al.add(s3);

System.out.println("Danh sách sinh viên trước   
 khi sắp xếp");

for(Student s:al){

System.out.println(s);

}

SortByMark sm = new SortByMark();

Collections.sort(al,sm);

System.out.println("Danh sách sinh viên sắp xếp  
 theo điểm tăng dần");

for(Student s:al){

System.out.println(s);

}

SortByName sn = new SortByName();

Collections.sort(al,sn);

System.out.println("Danh sách sinh viên sắp xếp   
 theo tên tăng dần");

for(Student s:al){

System.out.println(s);

}

}

}

Kết quả hiển thị ra màn hình khi thực hiện chương trình trên như sau:

Danh sách sinh viên trước khi sắp xếp

100 Lan 1997 9

101 Hung 1997 10

102 Hang 1997 8

Danh sách sinh viên sắp xếp theo điểm tăng dần

102 Hang 1997 8

100 Lan 1997 9

101 Hung 1997 10

Danh sách sinh viên sắp xếp theo tên tăng dần

102 Hang 1997 8

101 Hung 1997 10

100 Lan 1997 9

* + 1. Tìm kiếm

Để tìm kiếm Java cung cấp sẵn phương thức tĩnh binarySearch() của lớp Collections với 2 nguyên mẫu (signature) như sau:

* public static int binarySearch(List slist, T key); Sử dụng phương thức này khi các đối tượng trong list đã thực thi interface comparable (đã chỉ ra tiêu chí so sánh giữa các đối tượng)
* public static int binarySearch(List slist, T key, Comparator c); Sử dụng phương thức này khi các đối tượng trong list chưa chỉ ra tiêu chí so sánh nên chúng ta cần truyền tham số thứ 3 chính là tiêu chí so sánh giữa 2 đối tượng.

Cả hai phương thức trên đều trả về vị trí của đối tượng tìm được (phần tử đầu tiên trong danh sách có vị trí là 0), nếu không tìm thấy thì trả về kết quả là (-vị trí chèn phần tử - 1).

Ví dụ sau đây minh họa cách sử dụng phương thức thứ nhất để tìm sinh viên theo id.

* + - * 1. Định nghĩa lớp Student

package binarysearchdemo;

public class Student implements Comparable<Student> {

private int id;

private String name;

public Student(int id, String name) {

this.id = id;

this.name = name;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

@Override

public String toString() {

return id + " " + name ;

}

@Override

public int compareTo(Student o) {

if(this.id>o.getId()){

return 1;

}

else if(this.id<o.getId()){

return -1;

}

else{

return 0;

}

}

}

* + - * 1. Lớp kiểm tra phương thức tìm kiếm

package binarysearchdemo;

import java.util.ArrayList;

import java.util.Collections;

public class BinarySearchDemo {

public static void main(String[] args) {

ArrayList<Student> al = new ArrayList();

al.add(new Student(100,"Nguyễn Văn Anh"));

al.add(new Student(103,"Nguyễn Thị Hồng"));

al.add(new Student(102,"Trần Hồng Hạnh"));

//Cần tìm sinh viên có id là 103

int pos = Collections.binarySearch(al,new   
 Student(103,null));

if(pos>=0){

System.out.println("Tên của sinh viên có id  
 103 la: "+al.get(pos).getName());

}

else{

System.out.println("Không có sinh viên có id   
 là 103 trong danh sách");

}

}

}

Kết quả hiển thị lên màn hình khi thực hiện chương trình trên như sau:

Tên của sinh viên có id 103 la: Nguyễn Thị Hồng

Ví dụ minh họa tìm kiếm sinh viên theo id sử dụng phương thức thứ 2:

* + - * 1. Lớp Student được định nghĩa lại như sau:

package binarysearchdemo;

public class Student {

private int id;

private String name;

public Student(int id, String name) {

this.id = id;

this.name = name;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

@Override

public String toString() {

return id + " " + name ;

}

}

* + - * 1. Bổ sung thêm lớp CompareById để đưa ra tiêu chí so sánh giữa 2 sinh viên theo id như sau:

package binarysearchdemo;

import java.util.Comparator;

public class CompareById implements Comparator<Student>{

@Override

public int compare(Student o1, Student o2) {

if(o1.getId()>o2.getId()){

return 1;

}

else if(o1.getId()<o2.getId()){

return -1;

}

else{

return 0;

}

}

}

* + - * 1. Lớp kiểm tra phương thức binarySearch() như sau:

package binarysearchdemo;

import java.util.ArrayList;

import java.util.Collections;

public class BinarySearchDemo {

public static void main(String[] args) {

ArrayList<Student> al = new ArrayList();

al.add(new Student(100,"Nguyễn Văn Anh"));

al.add(new Student(103,"Nguyễn Thị Hồng"));

al.add(new Student(102,"Trần Hồng Hạnh"));

//Cần tìm sinh viên có id là 103

int pos = Collections.binarySearch(al,new   
 Student(103,null),new CompareById());

if(pos>=0){

System.out.println("Tên của sinh viên có id   
 103 la: "+al.get(pos).getName());

}

else{

System.out.println("Không có sinh viên có   
 id là 103 trong danh sách");

}

}

}

Kết quả hiển thị lên màn hình khi thực thi chương trình trên như sau:

Tên của sinh viên có id 103 la: Nguyễn Thị Hồng

BÀI TẬP CHƯƠNG 5:

Bài 1:

1. Viết lớp Circle, biết lớp có một thuộc tính là radius (bán kính). Viết phương thức getArea() để tính diện tích của hình tròn, viết đè phương thức toString() để trả về một chuỗi chứa bán kính và diện tích của hình tròn. Viết lớp TestCircle để kiểm tra các phương thức của lớp Circle.
2. Viết lớp CircleCollection, trong đó có thuộc tính và các phương thức:

* ArrayList<Circle> al; danh sách chứa các hình tròn.
* void addCircle(Circle c): thêm một hình tròn vào danh sách
* Circle getCircle(int pos): lấy hình tròn tại ví trí pos trong danh sách
* void setCircle(int pos,Circle c): thiết lập giá trị cho phần tử thứ pos trong danh sách.
* String toString(): Trả về thông tin của tất cả các hình tròn trong danh sách
* double calcSumArea(): tính tổng diện tích của tất cả các hình tròn trong danh sách.
* double findMinArea(): trả về diện tích hình tròn nhỏ nhất trong danh sách.
* double findMaxArea(): trả về diện tích hình tròn lớn nhất trong danh sách.

1. Viết chương trình nhập vào một danh sách các hình tròn với bán kính được tạo ngẫu nhiên. Sau đó thực hiện các yêu cầu sau:

* Hiển thị thông tin của các hình tròn đã nhập trong danh sách
* Tính tổng diện tích của các hình tròn trong danh sách
* Hiển thị diện tích lớn nhất
* Hiển thị thông tin của hình tròn có diện tích nhỏ nhất (vị trí trong danh sách, bán kính).

Bài 2: Xây dựng ứng dụng đổi tiền từ loại tiền tệ này sang loại tiền tế khác. Chương trình nhập vào mã tiền tệ thứ nhất (code1), mã tiền tế thứ 2 (code2) và lượng tiền cần chuyển (amount) sau đó hiển thị lên màn hình lượng tiền sau khi đã chuyển.

Chương trình tạo một mảng các chuỗi **MonArr** sau:

* USD;VND;17000
* EUR;USD;1.2
* USD;IDN;15789
* USD;EUR;0.83
* CAD;IDN;16869

Chương trình cần có 3 lớp:

* lớp Rate với 3 trường: code1(String), code2(String), rate(double)
* lớp Exchange
  + có thuộc tính kiểu ArrayList hoặc Vector để lưu các đối tượng Rate, mỗi đối tượng Rate tương ứng một một phần tử trong mảng MonArr.
  + có phương thức:   
    public double convert(String code1, String code2, double amount)  
    trong đó:
    - code1: mã tiền tệ thứ nhất
    - code2: mã tiền tệ thứ hai
    - amount: lượng tiền cần chuyển đổi
* lớp Test chứa phương thức main() thực hiện nhiệm vụ sau:
  + đọc thông tin người dùng nhập vào từ bàn phím
  + hiển thị lượng tiền sau khi chuyển đổi.

Ví dụ kết quả khi chạy chương trình:

Mời nhập vào mã tiền tệ thứ nhất: USD

Mời nhập mã tiền tệ thứ hai: VND

Mời nhập lượng tiền cần chuyển đổi: 100

Lượng tiền sau khi đổi: 1700000

Gợi ý: Đọc mảng MonArr sau đó sử dụng phương thức split() để phân tách chuỗi để lấy thông tin của mã tiền thứ nhất, thứ 2 và tỷ giá.

Bài 3: Viết chương trình đọc một mảng các chuỗi sau đó hiển thị số lần xuất hiện của mỗi ký tự và tổng số ký tự trong các chuỗi của mảng.

Ví dụ khi chạy chương trình với mảng các chuỗi ban đầu như sau:

learn java by example

guide to advance java

example of distributed in java

thì kết quả sẽ hiển thị lên màn hình như sau:

learn : 1

java: 3

by: 1

example: 2

guide: 1

to : 1

advance: 1

of: 1

distributed: 1

in: 1

Total words: 13

Gợi ý: Sử dụng HashMap hoặc HashTable với key là các từ đọc được từ chuỗi, còn value là tần xuất suất hiện của từ tương ứng.

Bài 4: Viết ứng dụng tra từ điển Anh – Việt đơn giản, nội dung của từ điển được lưu trong một mảng các chuỗi ký tự như sau:

* apple: qua tao
* ball: qua bong
* cat : con meo
* dog : con cho
* elephant: con voi
* fish: con ca
* gift: mon qua
* home: nha

…

Viết lớp Dictionary với phương thức lookup() để tìm nghĩa của một từ mới:

String lookup(String word);

Phương thức này trả về nghĩa của một từ. Trong trường hợp từ cần tìm chưa có thì phương thức này trả về null.

Phương thức main() nhận từ cần tra mà người sử dụng nhập vào từ bàn phím và hiển thị nghĩa của từ cho tới khi người sử dụng nhập vào chuỗi rỗng. Ví dụ:

Moi nhap tu can tra: cat

Nghia cua tu la: con meo

Moi nhap tu can tra: bear

Nghia cua tu la: khong tim thay

Moi nhap mot tu can tra:

Gợi ý: Sử dụng HashMap hoặc HashTable để chứa các phần tử, mỗi phần tử gồm có khóa là từ tiếng Anh còn value là nghĩa tiếng Việt tương ứng.

1. CÁC LUỒNG NHẬP XUẤT FILE
   1. Giới thiệu

File là tập hợp các thông tin có liên quan với nhau, thường được lưu trữ trên các ổ đĩa. Mỗi file có tên và phần mở rộng dùng để phân biệt chúng với nhau. Các dữ liệu thường được nhóm lại thành các trường (field). Các trường thường có tên để phân biệt với nhau và phán ánh dữ liệu mà nó chứa. Ví dụ trường có tên là số điện thoại thì có giá trị là số điện thoại bất kỳ, trường địa chỉ để chứa địa chỉ bất kỳ. Một tập các trường có mối quan hệ với nhau hình thành nên các bản ghi (record). Tập các bản ghi có cùng cấu trúc và kiểu dữ liệu của các trường tạo nên file.

Trong lập trình chúng ta thường sử dụng file với mục đích tách dữ liệu ra khỏi chương trình. Việc tách này cho phép dữ liệu có thể được thay đổi hoặc cập nhật mà không ảnh hưởng đến chương trình và cho phép nhiều chương trình có thể dùng chung dữ liệu.

Ngôn ngữ lập trình Java coi file là một dòng các bytes (do đó từ stream xuất hiện trong các tài liệu để nói về file). Do đó, khi bạn lập trình bằng Java thì bạn coi file là một chuỗi các byte chứ không phải là tập các bản ghi mặc dầu về mặt logic thì file có thể chứa các bản ghi.

Để đánh dấu kết thúc của một file thì tất cả các file dữ liệu đều chứa dấu hiệu kết thúc file gọi ngắn gọn là EOF (End Of File). Đây là một đánh dấu vật lý xuất hiện trực tiếp ngay sau byte cuối cùng. Nếu một file đã được tạo ra nhưng chưa ghi bất kỳ dữ liệu gì vào thì file này chỉ chứa dấu hiệu kết thúc EOF.

Các bước để sử dụng một file dữ liệu trong chương trình là: Đầu tiên chương trình phải chuẩn bị file để sử dụng, công việc này thường được gọi là mở file. Công việc này liên quan đến rất nhiều thao tác ngầm của máy tính. Ví dụ nếu file đã tồn thì tại giai đoạn này hệ điều hành sẽ phải thực hiện việc tìm file trên thiết bị lưu trữ phụ (thường là trên đĩa). Bước tiếp theo chương trình sẽ đọc hoặc ghi dữ liệu vào file. Bởi vì, một chương trình có thể làm việc với nhiều file dữ liệu nên khi chương trình muốn ghi hoặc đọc một file thì nó phải có cơ chế xác định file mà nó muốn làm việc với. Cuối cùng trước khi chương trình kết thúc nó phải báo cho máy tính là nó vừa kết thúc thao tác với file dữ liệu có nghĩa là file phải được đóng một cách thích hợp. Việc này lại được thực hiện bởi rất nhiều thao tác ngầm, ví dụ như phải lưu lại vị trí EOF một cách chính xác.

Dòng (Streams) dữ liệu trong Java là một luồng thông tin liên tục. Có nhiều loại dòng nhập và xuất khác nhau cho hai kiểu dữ liệu ký tự và byte với các tính chất khác nhau. Dòng có thể kết hợp chuỗi với nhau để cung cấp một tính chất mới. Một dòng dữ liệu có thể đến hoặc đi từ file, một tiến trình khác, một máy tính hoặc một luồng (thread)**.** Trong phạm vi của giáo trình chúng ta chỉ xét dòng đến và đi từ file.

File được chia làm 2 loại là file nhị phân và file văn bản. File nhị phân chứa tập các byte biểu diễn dữ liệu dưới dạng nhị phân như là dự liệu lưu ở trong RAM. Chúng ta có thể đọc và ghi giá trị của các biến với các kiểu dữ liệu khác nhau mà không cần biết chính xác khuôn dạng của nó mà chỉ cần biết kiểu dữ liệu. Xử lý file nhị phân thì nhanh hơn nhiều so với xử lý file văn bản. File văn bản thực sự cũng là file nhị nhân nhưng chúng chỉ chữa mã UNICODE (ASCII). Ví dụ khi ghi một số nguyên vào file văn bản thì các chữ số của nó sẽ được chuyển sang mã UNICODE tương ứng và các mã này được lưu vào file. Làm việc với file văn bản chậm vì khi ghi dữ liệu vào file phải có sự chuyển đổi sang mã UNICODE tương ứng và khi đọc dữ liệu từ file lại phải có sự chuyển đổi ngược lại thành giá trị ban đầu. Ưu điểm lớn nhất khi làm việc với file văn bản là chúng ta có thể dễ dàng xem được nội dung của nó bằng một chương trình soạn thảo văn bản đơn giản ví dụ như Notepad.

Có 2 phương pháp truy xuất file là tuần tự và ngẫu nhiên. File truy cập tuần tự là một tập các byte mà chỉ có thể đọc hoặc ghi như là một dòng liên tục tuần tự từ đầu đến cuối. File tuần tự là một file không có cấu trúc. File truy cập ngẫu nhiên là file nhị phân có cấu trúc, mỗi bản ghi được lưu với kích thước giống nhau. Chúng ta có thể đọc hoặc ghi bất kỳ bản ghi nào khi file được mở. Chúng ta có thể di chuyển con trỏ bản ghi đến bản ghi bất kỳ ở trong file. Điều này có nghĩa là chúng ta có thể tạo chỉ số cho file truy cập ngẫu nhiên để tăng tốc việc tìm kiếm dữ liệu trong file.

* 1. Luồng nhập xuất văn bản

Java có hai lớp chuyên dụng để xử lý file văn bản là lớp Writer và Reader. Các lớp này được giới thiệu cùng với phiên bản 1.1 của JDK để đơn giản hóa cho việc ghi và đọc file văn bản. Có các phương pháp khác để đọc và ghi file văn bản nhưng phương pháp đề cập trong giáo trình này là đơn giản nhất.

* + 1. Ghi vào file văn bản

Sau đấy là các lớp kế thừa từ lớp Writer và để ghi vào file văn bản chúng ta sử dụng lớp FileWriter và PrintWriter vì chúng sử dụng phương thức print() và println() để ghi dữ liệu vào file giống như các phương thức chúng ta thường sử dụng của lớp System.out khi hiển thị kết quả ra màn hình. Để ghi vào file văn bản đầu tiến chúng ta cần tạo đối tượng FileWriter để mở file cần ghi, sau đó chúng ta tạo đối tượng PrintWriter để gắn với file đã mở bằng cách truyền đối số cho phương thức tạo của nó là đối tượng FileWriter đã tạo được. Tiếp theo chúng ta sử dụng các phương thức print() hoặc println() của đối tượng PrintWriter để ghi dữ liệu vào file. Cuối cùng là chúng ta đóng file bằng cách sử dụng phương thức close() của đối tượng này.

class java.io.Writer

class java.io.BufferedWriter

class java.io.CharArrayWriter

class java.io.FilterWriter

class java.io.OutputStreamWriter

**class java.io.FileWriter**

class java.io.PipedWriter

**class java.io.PrintWriter**

class java.io.StringWriter

Ví dụ sau minh họa việc ghi dữ liệu vào file văn bản:

package textfiledemo;

import java.io.\*;

public class TextFileDemo {

public static void main(String[] args) {

FileWriter outFileWriter;

PrintWriter out;

try{

outFileWriter = new FileWriter("vidu.txt");

out = new PrintWriter(outFileWriter);

out.println("Hello from Java core");

out.println(100);

out.println(30.123);

out.println(false);

out.close();

}

catch(IOException e){

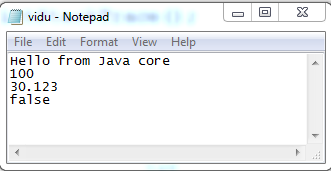
e.printStackTrace();

}

}

}

Kết quả khi mở file vidu.txt như hình 6.1 sau:



Nội dung file văn bản vidu.txt

* + 1. Đọc file văn bản

Để đọc file văn bản chúng ta thường sử dụng lớp BufferedReader và FileReader kế thừa từ lớp Reader. Đầu tiên chúng ta sử dụng đối tượng FileReader để mở file văn bản sau đó tạo đối tượng BufferedReader để kết nối với file thông qua đối tượng FileReader đã tạo được bằng cách truyền đối số cho phương thức tạo BufferedReader. Tiếp theo chúng ta dùng phương thức readLine để đọc lần lượt từng dòng của file văn bản. Cuối cùng chúng ta sử dụng phương thức close để đóng file sau khi đã đọc xong.

Ví dụ sau minh họa đọc nội dung của file văn bản vidu.txt đã tạo được ở ví dụ trên:

package textfiledemo;

import java.io.\*;

public class TextFileDemo {

public static void main(String[] args) {

FileReader inFileReader;

BufferedReader in;

try{

inFileReader = new FileReader("vidu.txt");

in = new BufferedReader(inFileReader);

String s = in.readLine();

System.out.println("Chuỗi đã ghi vào file:  
 "+s);

String ii = in.readLine();

int i = Integer.parseInt(ii);

System.out.println("Số nguyên đã ghi vào file:   
 "+i);

String id = in.readLine();

double d = Double.parseDouble(id);

System.out.println("Số thực đã ghi vào file:   
 "+d);

String ib = in.readLine();

boolean b = Boolean.parseBoolean(ib);

System.out.println("Giá trị logic đã ghi vào   
 file: "+b);

in.close();

}

catch(IOException e){

e.printStackTrace();

}

}

}

Kết quả hiển thị ra màn hình khi thực hiện chương trình trên như sau:

Chuỗi đã ghi vào file: Hello from Java core

Số nguyên đã ghi vào file: 100

Số thực đã ghi vào file: 30.123

Giá trị logic đã ghi vào file: false

* 1. Luồng nhập xuất nhị phân

Với file nhị phân thì cách đơn giản nhất để đọc và ghi dữ liệu kiểu nguyên thủy là sử dụng phương thức read và write có sẵn của lớp DataInputStream và DataOutputStream.

* + 1. Ghi vào file nhị phân

Cũng giống như file văn bản để ghi vào file nhị phân chúng ta cần đối tượng DataOutputStream gắn với file thông qua đối tượng FileOutputStream. DataOutputStream cung cấp các phương thức để ghi các kiểu dữ liệu nguyên thủy vào file nhị phân. Như vậy, để ghi dữ liệu nguyên thủy vào file nhị phân thì cách đơn giản nhất là chúng ta sử dụng hai lớp: DataOutputStream và FileOutputStream. Đầu tiên chúng ta tạo đối tượng FileOutputStream với đối số truyền vào cho phương thức tạo là file nhị phân cần ghi. Tiếp theo, chúng ta tạo đối tượng DataOutputStream gắn với file cần ghi thông qua việc truyền tham số cho phương thức tạo là đối tượng tạo ra ở bước trước. Ví dụ sau đây mình họa việc ghi các kiểu dữ liệu cơ bản vào file nhị phân.

package binaryfiledemo;

import java.io.\*;

public class BinaryFileDemo {

public static void main(String[] args) {

FileOutputStream fo;

DataOutputStream out;

try{

fo = new FileOutputStream("vidu.dat");

out = new DataOutputStream(fo);

out.writeInt(120);

out.writeUTF("Hello from java core");

out.writeDouble(123.45);

out.writeBoolean(false);

out.close();

}

catch(IOException e){

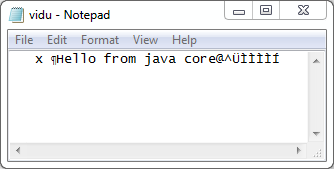
e.printStackTrace();

}

}

}

Nội dung của file vidu.dat nếu dùng Notepad để xem sẽ như hình 6.2 bên bên dưới. Vì đây là file nhị phân nên chúng ta không thể sử dụng chương trình soạn thảo văn bản đơn giản để xem được nội dung, muốn đọc nội dung của file chúng ta cần viết chương trình để đọc dữ liệu từ file ra.



Nội dung của file vidu.dat mở bằng Notepad

* + 1. Đọc file nhị phân

Muốn đọc file nhị phân chúng ta cần đối tượng tạo ra từ lớp DataInputStream gắn với file cần đọc thông qua đối tượng tạo từ lớp FileInputStream. Đầu tiên chúng ta tạo đối tượng FileInputStream với tham số truyền vào cho phương thức tạo là tên file cần đọc, tiếp theo chúng ta tạo đối tượng DataInputStream gắn với file vừa tạo ở trên bằng cách truyền tham số cho phương thức tạo là đối tượng FileInputStream đã tạo được. Cuối cùng, chúng ta sử dụng các phương thức read để đọc các kiểu dữ liệu cơ bản tương ứng. Ví dụ sau minh họa đọc dữ liệu từ file vidu.dat đã tạo được bên trên.

package binaryfiledemo;

import java.io.\*;

public class BinaryFileDemo {

public static void main(String[] args) {

FileInputStream fi;

DataInputStream in;

try{

fi = new FileInputStream("vidu.dat");

in = new DataInputStream(fi);

int i;

i = in.readInt();

System.out.println("Số nguyên đọc được từ file:  
 "+i);

String s;

s = in.readUTF();

System.out.println("Chuỗi đọc được từ file:   
 "+s);

double d;

d = in.readDouble();

System.out.println("Số thực đọc được từ file:   
 "+d);

boolean b;

b = in.readBoolean();

System.out.println("Giá trị logic đọc được từ   
 file: "+b);

in.close();

}

catch(IOException e){

e.printStackTrace();

}

}

}

Kết quả hiển thị lên màn hình khi thực hiện chương trình trên như sau:

Số nguyên đọc được từ file: 120

Chuỗi đọc được từ file: Hello from java core

Số thực đọc được từ file: 123.45

Giá trị logic đọc được từ file: false

* 1. Luồng nhập xuất đối tượng

Để đọc và ghi đối tượng vào file thì yêu cầu đối tượng phải được tạo ra từ lớp thực thi interface Serializable. File chứa đối tượng là file nhị phân.

* + 1. Ghi đối tượng vào file

Để ghi đối tượng vào file nhị phân chúng ta cần tạo đối tượng từ lớp ObjectOutputStream gắn với file cần ghi thông qua đối tượng tạo từ lớp FileOutputStream. Đầu tiên chúng ta tạo đối tượng FileOutputStream với đối số truyền vào là tên file cần ghi, tiếp theo chúng ta tạo đối tượng ObjectOutputStream với đối số truyền vào là đối tượng FileOutputStream đã được tạo ra ở bên trên. Tiếp theo, chúng ta sử dụng phương thức writeObject để ghi các đối tượng vào file. Cuối cùng chúng ta đóng luồng ghi đối tượng vào file. Ví dụ sau minh họa ghi các đối tượng tạo ra từ lớp Person vào file.

* Nội dung của lớp Person

package objectfiledemo;

import java.io.Serializable;

public class Person implements Serializable {

private int id;

private String name;

private String address;

private char gender;

public Person() {

}

public Person(int id, String name, String address,   
 char gender) {

this.id = id;

this.name = name;

this.address = address;

this.gender = gender;

}

@Override

public String toString() {

return "id: "+id + "\nname: " + name +   
 "\naddress: " + address + "\ngender: " + gender;

}

}

* Nội dung của lớp ghi đối tượng Person vào file

package objectfiledemo;

import java.io.\*;

public class ObjectFileDemo {

public static void main(String[] args) {

FileOutputStream fo;

ObjectOutputStream out;

try{

fo = new FileOutputStream("exp.dat");

out = new ObjectOutputStream(fo);

Person p1 = new Person(100,"Nguyen Van An","Nam   
 Dinh",'M');

out.writeObject(p1);

Person p2 = new Person(101,"Nguyen Thi Anh","Hai  
 Phong",'F');

out.writeObject(p2);

out.close();

}

catch(Exception e){

e.printStackTrace();

}

}

}

* + 1. Đọc đối tượng từ file

Để đọc đối tượng từ file nhị phân chúng ta cần tạo đối tượng từ lớp ObjectInputStream gắn với file cần ghi thông qua đối tượng tạo từ lớp FileInputStream. Đầu tiên chúng ta tạo đối tượng FileInputStream với đối số truyền vào là tên file cần đọc, tiếp theo chúng ta tạo đối tượng ObjectInputStream với đối số truyền vào là đối tượng FileInputStream đã được tạo ra ở bên trên. Tiếp theo, chúng ta sử dụng phương thức readObject để đọc các đối tượng vào file. Cuối cùng chúng ta đóng luồng đọc đối tượng từ file. Ví dụ sau minh họa đọc các đối tượng đã ghi vào file ở bài trên.

* Nội dung của lớp Person

package objectfiledemo;

import java.io.Serializable;

public class Person implements Serializable {

private int id;

private String name;

private String address;

private char gender;

public Person() {

}

public Person(int id, String name, String address,   
 char gender) {

this.id = id;

this.name = name;

this.address = address;

this.gender = gender;

}

@Override

public String toString() {

return "id: "+id + "\nname: " + name +  
 "\naddress: " + address + "\ngender: " + gender;

}

}

* Nội dung của lớp đọc đối tượng Person từ file

package objectfiledemo;

import java.io.\*;

public class ObjectFileDemo {

public static void main(String[] args) {

FileInputStream fi;

ObjectInputStream in;

try{

fi = new FileInputStream("exp.dat");

in = new ObjectInputStream(fi);

Person p1 = (Person)in.readObject();

System.out.println("Thông tin người thứ nhất\n   
 "+p1);

Person p2 = (Person)in.readObject();

System.out.println("Thông tin người thứ hai\n   
 "+p2);

in.close();

}

catch(Exception e){

e.printStackTrace();

}

}

}

Kết quả hiển thị lên màn hình khi thực thi chương trình như sau:

Thông tin người thứ nhất

id: 100

name: Nguyen Van An

address: Nam Dinh

gender: M

Thông tin người thứ hai

id: 101

name: Nguyen Thi Anh

address: Hai Phong

gender: F

BÀI TẬP CHƯƠNG 6

Bài 1: Viết chương trình Java nhập một chuỗi ký tự, một ký tự, một số nguyên, một số thực rồi ghi vào file văn bản với tên file là exp.txt. Đọc dữ liệu từ file vừa ghi được và hiển thị lên màn hình.

Bài 2: Viết chương trình Java nhập một chuỗi ký tự, một ký tự, một số nguyên và một số thực rồi ghi vào file nhị phân với tên file là exp.dat. Đọc dữ liệu từ file vừa ghi được và hiển thị kết quả lên màn hình.

Bài 3: Viết chương trình Java định nghĩa lớp Student có các thuộc tính code (mã sinh viên), name (tên sinh viên), dob (ngày tháng năm sinh của sinh viên), gender (giới tính của sinh viên), các phương thức tạo không đối số và đầy đủ đối số, phương thức toString trả về thông tin các thuộc tính của sinh viên. Sử dụng lớp Student đã định nghĩa được tạo 2 đối tương sinh viên với thông tin khác nhau rồi ghi vào file student.dat. Đọc thông tin của các sinh viên từ file đã lưu và hiển thị nội dung lên màn hình.

TÀI LIỆU THAM KHẢO

[1]. Douglas Mawson Institute text book, Program technique B (Java), 2004

[2]. Cay S. Horstmann, Gary Cornell, Core Java 2: Volumn I – Fundamentals, Prentice Hall, 2002

[3]. Kathy Sierra, Bert Bates, Head First Java 2nd edition, O’Reilly, 2010

[4]. <https://www.javatpoint.com/java-tutorial>