|  |
| --- |
| TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG  **KHOA CÔNG NGHỆ THÔNG TIN**  **Logo  Description automatically generated**  TS. Nguyễn Văn Núi  TS. Tô Hữu Nguyên  ThS. Trần Hải Thanh  ThS. Phạm Thị Thương  ThS. Nguyễn Thị Dung  ThS. Đỗ Thị Loan  ThS. Nguyễn Văn Việt  **BÀI GIẢNG**  **LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**  **Tài liệu lưu hành nội bộ** |

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

**KHOA CÔNG NGHỆ THÔNG TIN**

TS. Nguyễn Văn Núi

TS. Tô Hữu Nguyên

ThS. Trần Hải Thanh

ThS. Phạm Thị Thương

ThS. Nguyễn Thị Dung

ThS. Đỗ Thị Loan

ThS. Nguyễn Văn Việt

**BÀI GIẢNG**

**LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

**Thái Nguyên, tháng năm 2022**

# Mục lục

[Mục lục 3](#_Toc111103914)

[Mở đầu 10](#_Toc111103915)

[Chương I: TỔNG QUAN 11](#_Toc111103916)

[Bài 1: Tổng quan (Số tiết: 3 tiết) 11](#_Toc111103917)

[1.1. Tổng quan lập trình hướng chức năng và lập trình hướng đối tượng 11](#_Toc111103918)

[1.1.1. Lập trình tuyến tính 11](#_Toc111103919)

[1.1.2. Lập trình có cấu trúc 11](#_Toc111103920)

[1.1.3. Lập trình hướng đối tượng 14](#_Toc111103921)

[1.2. Các khái niệm cơ bản của lập trình hướng đối tượng 15](#_Toc111103922)

[1.2.1. Đối tượng 15](#_Toc111103923)

[1.2.2. Lớp đối tượng 17](#_Toc111103924)

[1.2.3. Trừu tượng hóa 18](#_Toc111103925)

[1.2.4. Bao bọc và che dấu thông tin 19](#_Toc111103926)

[1.2.5. Kế thừa và mở rộng 20](#_Toc111103927)

[1.2.6. Đa xạ và nạp chồng 21](#_Toc111103928)

[1.2.7. Liên kết động 22](#_Toc111103929)

[1.2.8. Truyền thông điệp 23](#_Toc111103930)

[1.3. Các ngôn ngữ lập trình hướng đối tượng 23](#_Toc111103931)

[Bài tập cuối chương 25](#_Toc111103932)

[Chương II: TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG JAVA 27](#_Toc111103933)

[Nội dung chính của chương 27](#_Toc111103934)

[Mục tiêu cần đạt được của chương 27](#_Toc111103935)

[Bài 2: Sơ lược về java (Số tiết: 03 tiết) 27](#_Toc111103936)

[2.1 Lịch sử phát triển của java 27](#_Toc111103937)

[2.2 Ngữ cảnh các ứng dụng của ngôn ngữ java. 28](#_Toc111103938)

[2.3. Cấu trúc chung của chương trình java 30](#_Toc111103939)

[2.4. Biên dịch và thông dịch chương trình java 30](#_Toc111103940)

[Bài tập cuối chương 31](#_Toc111103941)

[Bài 3: Đặc trưng của chương trình ứng dụng Java (Số tiết: 03 tiết) 33](#_Toc111103942)

[2.5. Các đặc trưng của chương trình ứng dụng Java 33](#_Toc111103943)

[2.5.1. Máy ảo Java (JVM - Java Virtual Machine) 33](#_Toc111103944)

[2.5.2. Thông dịch 35](#_Toc111103945)

[2.5.3. Độc lập nền 35](#_Toc111103946)

[2.5.4. Hướng đối tượng 35](#_Toc111103947)

[2.5.5. Đa nhiệm - đa luồng (MultiTasking - Multithreading) 35](#_Toc111103948)

[2.5.6. Khả chuyển (portable) 35](#_Toc111103949)

[2.5.7. Hỗ trợ mạnh cho việc phát triển ứng dụng 36](#_Toc111103950)

[2.6. Bộ công cụ phát triển JDK 36](#_Toc111103951)

[2.7 Các môi trường tích hợp phát triển ứng dụng Java 37](#_Toc111103952)

[2.8. Các dạng chương trình ứng dụng của Java 38](#_Toc111103953)

[2.8.1. Chương trình ứng dụng độc lập 39](#_Toc111103954)

[2.8.2. Chương trình ứng dụng nhúng 40](#_Toc111103955)

[Bài tập cuối chương 42](#_Toc111103956)

[Chương III: CÁC THÀNH PHẦN CƠ SỞ CỦA JAVA 44](#_Toc111103957)

[Nội dung chính của chương 44](#_Toc111103958)

[Mục tiêu cần đạt được của chương 44](#_Toc111103959)

[Bài 4: Các phần tử cơ sở của Java (Số tiết: 03 tiết) 44](#_Toc111103960)

[3.1. Các phần tử cơ sở của Java 44](#_Toc111103961)

[3.1.1. Các phần tử cơ sở 44](#_Toc111103962)

[3.1.2. Các kiểu dữ liệu trong Java 45](#_Toc111103963)

[3.1.3. Khai báo biến 46](#_Toc111103964)

[3.1.4. Biểu thức trong Java 47](#_Toc111103965)

[3.1.5. Các phép toán trong Java 48](#_Toc111103966)

[3.2. Các câu lệnh nhập xuất dữ liệu cơ bản qua các thiết bị vào ra chuẩn 51](#_Toc111103967)

[3.2.1. Nhập dữ liệu từ bàn phím 51](#_Toc111103968)

[3.2.2. Xuất dữ liệu ra màn hình 52](#_Toc111103969)

[3.3. Các câu lệnh điều khiển 52](#_Toc111103970)

[3.3.1. Câu lệnh rẽ nhánh 52](#_Toc111103971)

[Bài tập cuối chương 55](#_Toc111103972)

[Bài 5: Các câu lệnh điều khiển (Số tiết: 03 tiết) 57](#_Toc111103973)

[3.3.2. Các câu lệnh lặp 57](#_Toc111103974)

[3.3.3. Các lệnh chuyển vị trong java 59](#_Toc111103975)

[3.4. Ngoại lệ và xử lý ngoại lệ 62](#_Toc111103976)

[3.4.1. Cấu trúc phân cấp của các lớp xử lý ngoại lệ 64](#_Toc111103977)

[3.4.2. Câu lệnh try, catch và finally 65](#_Toc111103978)

[3.4.3. Ném ngoại lệ bằng lệnh ‘throw’ 68](#_Toc111103979)

[3.4.4. Mệnh đề throws 69](#_Toc111103980)

[3.4.5. Tự định nghĩa ngoại lệ 70](#_Toc111103981)

[3.4.6. Lần vết ngoại lệ StackTrace 71](#_Toc111103982)

[Bài tập cuối chương 72](#_Toc111103983)

[Chương IV: LỚP VÀ CÁC THÀNH PHẦN CỦA LỚP ĐỐI TƯỢNG 74](#_Toc111103984)

[Nội dung chính của chương 74](#_Toc111103985)

[Mục tiêu cần đạt được của chương 74](#_Toc111103986)

[Bài 6: Lớp đối tượng (Số tiết: 03 tiết) 74](#_Toc111103987)

[4.1. Cấu trúc lớp và khai báo các thành phần của lớp 74](#_Toc111103988)

[4.1.1. Định nghĩa lớp 74](#_Toc111103989)

[4.1.2. Các thuộc tính dữ liệu của lớp 75](#_Toc111103990)

[4.2. Định nghĩa hàm thành phần và cơ chế nạp chồng, viết đè trong Java 76](#_Toc111103991)

[4.2.1. Các hàm thành phần của lớp 76](#_Toc111103992)

[4.2.2 Truyền tham số và gọi phương thức(hàm) 79](#_Toc111103993)

[4.2.3. Cơ chế nạp chồng, viết đè trong Java 82](#_Toc111103994)

[4.2.4. Nạp chồng các toán tử tạo lập 85](#_Toc111103995)

[4.2.4. Cơ chế che bóng của các biến 87](#_Toc111103996)

[4.3. Các thuộc tính kiểm soát truy nhập các thành phần của lớp 88](#_Toc111103997)

[Bài tập cuối chương 90](#_Toc111103998)

[Bài 7: Lớp và các thành phần của lớp đối tượng (tiếp theo) (Số tiết: 03 tiết) 93](#_Toc111103999)

[4.4. Toán tử tạo lập đối tượng 93](#_Toc111104000)

[4.4.1. Khai báo toán tử tạo lập 93](#_Toc111104001)

[4.4.2. Toán tử this() và super() 94](#_Toc111104002)

[4.5. Kế thừa giữa các lớp đối tượng 96](#_Toc111104003)

[4.5.1. Cú pháp 96](#_Toc111104004)

[4.5.2. Các kiểu kế thừa trong java 97](#_Toc111104005)

[4.6. Giao diện và sự mở rộng quan hệ kế thừa 101](#_Toc111104006)

[4.6.1. Các bước tạo interface 101](#_Toc111104007)

[4.6.2. Khai báo interface 103](#_Toc111104008)

[4.6.3. Sử dụng interface 104](#_Toc111104009)

[4.6.4. Đa kế thừa trong Java bởi Interface 105](#_Toc111104010)

[Bài tập cuối chương 107](#_Toc111104011)

[Chương V: CÁC LỚP CƠ SỞ VÀ CÁC CẤU TRÚC DỮ LIỆU 109](#_Toc111104012)

[Nội dung chính của chương 109](#_Toc111104013)

[Mục tiêu cần đạt được của chương 109](#_Toc111104014)

[Bài 8: Mảng và các lớp cơ sở trong gói java.lang (Số tiết: 03 tiết) 109](#_Toc111104015)

[5.1. Cấu trúc mảng 109](#_Toc111104016)

[5.1.1. Khai báo mảng 109](#_Toc111104017)

[5.1.2. Tạo lập mảng 109](#_Toc111104018)

[5.1.3. Truy cập các phần tử của mảng 110](#_Toc111104019)

[5.1.4. Duyệt các phần tử của mảng trong Java 112](#_Toc111104020)

[5.1.5. Mảng nhiều chiều 113](#_Toc111104021)

[5.2. Các lớp cơ sở trong gói java.lang 114](#_Toc111104022)

[5.2.1. Lớp Object 114](#_Toc111104023)

[5.2.2. Các lớp nguyên thủy 116](#_Toc111104024)

[5.2.3. Lớp Math 120](#_Toc111104025)

[5.2.4. Lớp String 123](#_Toc111104026)

[5.2.5. Lớp StringBuffer 130](#_Toc111104027)

[Bài tập cuối chương 131](#_Toc111104028)

[Bài 9: Mảng và các lớp cơ sở trong gói java.lang (tiếp theo) (Số tiết: 03 tiết) 134](#_Toc111104029)

[5.3. Một số cấu trúc dữ liệu phổ dụng 134](#_Toc111104030)

[5.3.1. Enumeration 134](#_Toc111104031)

[5.3.2. BitSet 135](#_Toc111104032)

[5.3.3. Vector 137](#_Toc111104033)

[5.3.4. Stack 144](#_Toc111104034)

[5.4. Một số bài tập về cấu trúc dữ liệu 146](#_Toc111104035)

[Bài tập cuối chương 147](#_Toc111104036)

[Chương VI: CÁC LUỒNG VÀO RA DỮ LIỆU CƠ BẢN 149](#_Toc111104037)

[Nội dung chính của chương 149](#_Toc111104038)

[Mục tiêu cần đạt được của chương 149](#_Toc111104039)

[Bài 10: Các luồng vào ra dữ liệu cơ bản (Số tiết: 03 tiết) 149](#_Toc111104040)

[6.1. Khái niệm 149](#_Toc111104041)

[6.1.1. Khái niệm về stream 149](#_Toc111104042)

[6.1.2. Các lớp OutputStream và InputStream 149](#_Toc111104043)

[6.2. Các luồng vào/ra 150](#_Toc111104044)

[6.2.1. Standard Stream trong Java 150](#_Toc111104045)

[6.2.2. Lớp OutputStream 151](#_Toc111104046)

[6.2.3. Lớp InputStream 152](#_Toc111104047)

[6.3. Lớp File 154](#_Toc111104048)

[6.4. Truy cập tệp tuần tự và truy cập tệp ngẫu nhiên 160](#_Toc111104049)

[6.4.1. Truy cập tệp tuần tự 160](#_Toc111104050)

[6.4.2. Truy cập tệp ngẫu nhiên 164](#_Toc111104051)

[Tài liệu tham khảo 165](#_Toc111104052)

[Các câu hỏi thường gặp 167](#_Toc111104053)

[Bài tập thực hành 173](#_Toc111104054)

[Bài thực hành số 1 (Số tiết: 03 tiết) 173](#_Toc111104055)

[Bài thực hành số 2 (số tiết: 03 tiết) 175](#_Toc111104056)

[Bài thực hành số 3 (số tiết: 03 tiết) 177](#_Toc111104057)

[Bài thực hành số 4 (số tiết: 03 tiết) 179](#_Toc111104058)

[Bài thực hành số 5 (số tiết: 03 tiết) 182](#_Toc111104059)

[Bài thực hành số 6 (số tiết: 03 tiết) 191](#_Toc111104060)

[Bài thực hành số 7 (số tiết: 03 tiết) 194](#_Toc111104061)

[Bài thực hành số 8 (số tiết: 03 tiết) 201](#_Toc111104062)

[Bài thực hành số 9 (số tiết: 03 tiết) 206](#_Toc111104063)

[Bài thực hành số 10 (số tiết: 03 tiết) 211](#_Toc111104064)

# Mở đầu

Bài giảng Lập trình hướng đối tượng được tập thể giảng viên thuộc bộ môn Công nghệ phần mềm biên soạn nhằm phục vụ cho việc giảng dạy của giảng viên và học tập của sinh viên Trường Đại học Công nghệ thông tin và Truyền thông - Đại học Thái Nguyên. Tập bài giảng này được biên soạn theo nội dung đề cương chi tiết học phần lập trình hướng đối tượng ở trình độ đại học.

Nội dung tài liệu gồm 6 chương:

Chương 1. Tổng quan.

Chương 2. Tổng quan về ngôn ngữ lập trình hướng đối tượng Java.

Chương 3. Các thành phần cơ sở của Java.

Chương 4. Lớp và các thành phần của lớp đối tượng.

Chương 5. Các lớp cơ sở và các cấu trúc dữ liệu.

Chương 6. Các luồng vào ra dữ liệu cơ bản.

Mặc dù tập thể tác giả đã dành nhiều thời gian và công sức để biên soạn, song khó tránh khỏi thiếu sót. Vậy, chúng tôi kính mong quý thầy cô và các bạn sinh viên đóng góp ý kiến để cuốn bài giảng được hoàn thiện hơn. Xin trân trọng cảm ơn.

# Chương I: TỔNG QUAN

Nội dung chính của chương

* Tổng quan lập trình hướng chức năng và lập trình hướng đối tượng
* Các khái niệm cơ bản của lập trình hướng đối tượng

Mục tiêu cần đạt được của chương

Kết thúc chương, học viên có thể:

* Hiểu được vai trò của lập trình hướng đối tượng trong việc phát triển phần mềm.
* Định nghĩa Lập trình hướng Đối tượng (Object-oriented Programming).
* Nắm được các nguyên lý cơ bản trong lập trình hướng đối tượng:
* Liệt kê những thuận lợi của phương pháp hướng đối tượng

## Bài 1: Tổng quan (Số tiết: 3 tiết)

## 1.1. Tổng quan lập trình hướng chức năng và lập trình hướng đối tượng

### 1.1.1. Lập trình tuyến tính

Lập trình tuyến tính có 2 đặc trưng cơ bản sau:

- Đơn giản: chương trình được tiến hành đơn giản theo lối tuần tự, không phức tạp.

- Đơn luồng: Chỉ có một luồng công việc duy nhất, các công việc được thực hiện tuần tự trong các luồng đó.

Ưu điểm: Chương trình đơn giản, dễ hiểu.

Nhược điểm: Không thể dùng lập trình tuyến tính để giải quyết các bài toán phức tạp.

Ngày nay, lập trình tuyến tính chỉ tồn tại trong phạm vi các module nhỏ nhất của các phương pháp lập trình khác.

### 1.1.2. Lập trình có cấu trúc

Cách tiếp cận theo hướng thủ tục dựa vào chức năng, nhiệm vụ là chính và Phân rã chức năng làm mịn dần theo cách từ trên xuống (Top/Down). Các đơn thể chức năng trao đổi với nhau bằng cách truyền tham số hay sử dụng dữ liệu chung.

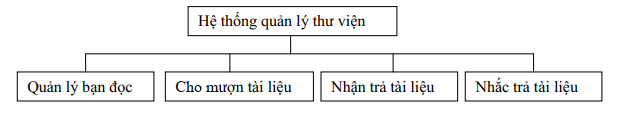
* **Dựa vào chức năng nhiệm vụ**: Khi khảo sát, phân tích một hệ thống chúng ta thường tập trung vào các nhiệm vụ mà nó cần thực hiện. Chúng ta tập trung trước hết nghiên cứu các yêu cầu của bài toán để xác định các chức năng chính của hệ thống.

Ví dụ khi cần xây dựng “hệ thống quản lý thư viện” thì trước hết chúng ta thường đi nghiên cứu, khảo sát trao đổi và phỏng vấn xem những người thủ thư, bạn đọc cần phải thực hiện những công việc gì để phục vụ được bạn đọc và quản lý tốt được các tài liệu.

Qua nghiên cứu “hệ thống quản lý thư viện”, chúng ta xác định được các nhiệm vụ chính của hệ thống như: quản lý bạn đọc, cho mượn sách, nhận trả sách, thông báo nhắc trả sách, v.v

Hệ thống phần mềm được xem như là tập các chức năng, nhiệm vụ cần tổ chức thực thi.

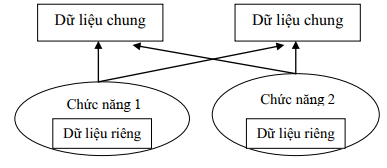
* **Phân rã chức năng làm mịn từ trên xuống**: Khả năng của con người là có giới hạn khi khảo sát, nghiên cứu để hiểu và thực thi những gì mà hệ thống thực tế đòi hỏi. Để thống trị (quản lý được) độ phức tạp của những vấn đề phức tạp trong thực tế thường chúng ta phải sử dụng nguyên lý chia để trị (divide and conquer), nghĩa là phân tách nhỏ các chức năng chính thành các chức năng đơn giản hơn theo cách từ trên xuống. Qui trình này được lặp lại cho đến khi thu được những đơn thể chức năng tương đối đơn giản, hiểu được và thực hiện cài đặt chúng mà không làm tăng thêm độ phức tạp để liên kết chúng trong hệ thống. Độ phức tạp liên kết các thành phần chức năng của hệ thống thường là tỉ lệ nghịch với độ phức tạp của các đơn thể. Vì thế một vấn đề đặt ra là có cách nào để biết khi nào quá trình phân tách các đơn thể chức năng hay còn gọi là quá trình làm mịn dần này kết thúc. Thông thường thì quá trình thực hiện phân rã các chức năng của hệ thống phụ thuộc nhiều vào độ phức hợp của bài toán ứng dụng và vào trình độ của những người tham gia phát triển phần mềm. Một hệ thống được phân tích dựa trên các chức năng hoặc quá trình sẽ được chia thành các hệ thống con và tạo ra cấu trúc phân cấp các chức năng. Ví dụ, hệ thống quản lý thư viện có thể phân chia từ trên xuống như sau:



Hình 1.1: Sơ đồ chức năng của Hệ thống quản lý thư viện

Các chức năng của hầu hết các hệ thống thông tin quản lý đều có thể tổ chức thành sơ đồ chức năng theo cấu trúc phân cấp có thứ bậc.

* Các đơn thể chức năng trao đổi với nhau bằng cách truyền tham số hay sử dụng dữ liệu chung. Một hệ thống phần mềm bao giờ cũng phải được xem như là một thể thống nhất, do đó các đơn thể chức năng phải có quan hệ trao đổi thống tin, dữ liệu với nhau. Trong một chương trình gồm nhiều hàm (thực hiện nhiều chức năng khác nhau) muốn trao đổi dữ liệu được với nhau thì nhất thiết phải sử dụng dữ liệu chung hoặc liên kết với nhau bằng cách truyền tham biến. Mỗi đơn thể chức năng không những chỉ thao tác, xử lý trên những dữ liệu cục bộ (Local Variables) mà còn phải sử dụng các biến chung, thường đó là các biến toàn cục (Global Variable).



Hình 1.2: Mối quan hệ giữa các chức năng trong hệ thống

**Nhận xét:**

* Hệ thống được xây dựng dựa vào chức năng là chính mà trong thực tế thì chức năng, nhiệm vụ của hệ thống lại hay thay đổi. Để đảm bảo cho hệ thống thực hiện được công việc theo yêu cầu, nhất là những yêu cầu về mặt chức năng đó lại bị thay đổi là công việc phức tạp và rất tốn kém.

Ví dụ: giám đốc thư viện yêu cầu thay đổi cách quản lý bạn đọc hoặc hơn nữa, yêu cầu bổ sung chức năng theo dõi những tài liệu mới mà bạn đọc thường xuyên yêu cầu để đặt mua, v.v. Khi đó vấn đề bảo trì hệ thống phần mềm không phải là vấn đề dễ thực hiện. Nhiều khi có những yêu cầu thay đổi cơ bản mà việc sửa đổi không hiệu quả và vì thế đòi hỏi phải phân tích, thiết kế lại hệ thống thì hiệu quả hơn.

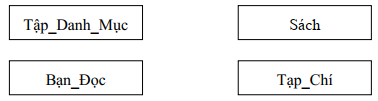
* Các bộ phận của hệ thống phải sử dụng biến toàn cục để trao đổi với nhau, do vậy khả năng thay đổi, mở rộng của chúng và của cả hệ thống là bị hạn chế. Như trên đã phân tích, những thay đổi liên quan đến các dữ liệu chung sẽ ảnh hưởng tới các bộ phận liên quan. Do đó, một thiết kế tốt phải rõ ràng, dễ hiểu và mọi sửa đổi chỉ có hiệu ứng cục bộ.
* Khả năng tái sử dụng (Reuse) bị hạn chế và không hỗ trợ cơ chế kế thừa (Inheritance). Để có độ thích nghi cao thì mỗi thành phần phải là tự chứa. Muốn là tự chứa hoàn toàn thì một thành phần không nên dùng các thành phần ngoại lai. Tuy nhiên, điều này lại mâu thuẫn với kinh nghiệm nói rằng các thành phần hiện có nên là dùng lại được. Vậy là cần có một sự cân bằng giữa tính ưu việt của sự dùng lại các thành phần (ở đây chủ yếu là các hàm) và sự mất mát tính thích ứng được của chúng. Các thành của hệ thống phải kết dính (Cohension) nhưng phải tương đối lỏng để dễ thích nghi. Một trong cơ chế chính hỗ trợ để dễ có được tính thích nghi là kế thừa thì cách tiếp cận hướng chức năng lại không hỗ trợ. Đó là cơ chế biểu diễn tính tương tự của các thực thể, đơn giản hoá định nghĩa những khái niệm tương tự từ những sự vật đã được định nghĩa trước trên cơ sở bổ sung hay thay đổi một số các đặc trưng hay tính chất của chúng. Cơ chế này giúp chúng ta thực hiện được nguyên lý tổng quát hoá và chi tiết hoá các thành phần của hệ thống phần mềm.

### 1.1.3. Lập trình hướng đối tượng

Dựa trên nền tảng là các đối tượng: Đặt trọng tâm vào dữ liệu, xem hệ thống là tập các thực thể và tập các đối tượng.

- Đặt trọng tâm vào dữ liệu (thực thể). Khi khảo sát, phân tích một hệ thống chúng ta tìm hiểu xem nó gồm những thực thể nào. Thực thể hay còn gọi là đối tượng, là những gì như người, vật, sự kiện, v.v. mà chúng ta đang quan tâm, hay cần phải xử lý. Ví dụ, khi xây dựng “Hệ thống quản lý thư viện” thì trước hết chúng ta tìm hiểu xem nó gồm những lớp đối tượng hoặc những khái niệm nào.

- Xem hệ thống như là tập các thực thể, các đối tượng. Để hiểu rõ về hệ thống, chúng ta phân tách hệ thống thành các đơn thể đơn giản hơn. Quá trình này được lặp lại cho đến khi thu được những đơn thể tương đối đơn giản, dễ hiểu và thực hiện cài đặt chúng mà không làm tăng thêm độ phức tạp khi liên kết chúng trong hệ thống. Xét “Hệ thống quản lý thư viện”, chúng ta có các lớp đối tượng sau:



Hình 1.3 Tập các lớp đối tượng của hệ thống

- Các lớp đối tượng trao đổi với nhau bằng các thông điệp (Message). Theo nghĩa thông thường thì lớp (Class) là nhóm một số người, vật có những đặc tính tương tự nhau hoặc có những hành vi ứng xử giống nhau. Trong mô hình đối tượng, khái niệm lớp là cấu trúc mô tả hợp nhất các thuộc tính (Attributes), hay dữ liệu thành phần (Data Member) thể hiện các đặc tính của mỗi đối tượng và các phương thức (Methods), hay hàm thành phần (Member Function) thao tác trên các dữ liệu riêng và là giao diện trao đổi với các đối tượng khác để xác định hành vi của chúng trong hệ thống. Khi có yêu cầu dữ liệu để thực hiện một nhiệm vụ nào đó, một đối tượng sẽ gửi một thông điệp (gọi một phương thức) cho đối tượng khác. Đối tượng nhận được thông điệp yêu cầu sẽ phải thực hiện một số công việc trên các dữ liệu mà nó sẵn có hoặc lại tiếp tục yêu cầu những đối tượng khác hỗ trợ để có những thông tin trả lời cho đối tượng yêu cầu. Với phương thức xử lý như thế thì một chương trình hướng đối tượng thực sự có thể không cần sử dụng biến toàn cục nữa.

## 1.2. Các khái niệm cơ bản của lập trình hướng đối tượng

### 1.2.1. Đối tượng

- Đối tượng là thực thể của hệ thống được xác định thông qua định danh ( tên gọi).

- Mỗi đối tượng bao gồm dữ liệu thành phần hay các thuộc tính mô tả các tính chất và phương thức thao tác trên dữ liệu để xác định hành vi của đối tượng.

Đối tượng = Thuộc tính (dữ liệu) + phương thức (hàm).

- Theo quan điểm của người lập trình đối tượng được xem như vùng nhớ được phân chia trong máy tính để lưu dữ liệu và tập các hàm tác động lên dữ liệu gắn với chúng.

Ví dụ: Quyển sách = thông tin về sách ( khổ sách, độ dày, nội dung…) + Đọc ( mở sách, lật trang, đọc 1 đoạn, đọc một trang, tìm mục lục…)

(đối tượng) = Thuộc tính + phương thức

- Ta có thể đọc sách dưới ánh đèn.

Đèn = Thông tin trạng thái (sáng, tắt) + bật, tắt đèn.

- Quyển sách chứa nội dung thông tin, nó cũng bao gồm các thông tin về trang thái (sách đang mở, đóng) và có thể chứa các đối tương nhỏ hơn (các trang sách) . Phương thức cúa sách cho phép truy cập đến trang và phương thức của trang truy cập đến thông tin trong trang đó.

- Đối tượng có thể là một thực thể trực quan ( người, sự vật) hay là một khái niệm (phòng ban, bộ phận, đăng ký…)

- Ta sẽ gọi tập hợp giá trị các thuộc tính của một đối tượng cụ thể tại một thời điểm nào đó là trạng thái (state) của nó.

- Một hành vi là cái mà đối tượng có thể thực hiện, và sẽ bằng cách nào đó tác động tới một hoặc nhiều thuộc tính của đối tượng (do đó ảnh hưởng đến trạng thái của đối tượng). *Xét một người có một thuộc tính tư thế - một trong các giá trị {đứng, ngồi, nằm, quỳ}¨ Một hành vi “đứng” có thể dẫn đến kết quả là người đó đứng dậy (nếu người đó đang không ở tư thế đứng)*

- Trao đổi thông điệp: Các đối tượng liên lạc với nhau bằng các thông điệp (message). Một thông điệp là một yêu cầu một đối tượng thực hiện một hành vi.

Ví dụ: nếu ta muốn một người nào đó đứng dậy, ta có thể gửi một thông điệp đến cho người đó để đề nghị người đó đứng dậy. Đối với con người, phần lớn việc “truyền thông điệp” (messagepassing) của ta được thực hiện qua đường tiếng nói. Ta sẽ thấy rằng cơ chế “truyền thông điệp” đó xảy ra trong lập trình hướng đối tượng qua các lời gọi hàm.

- Trong mô hình hướng đối tượng: các đối tượng tương tác với nhau để thực hiện nhiệm vụ.

\* Phân loại đối tượng:

Khả năng phân loại bẩm sinh của con người, khả năng trừu tượng hoá này cho phép ta đơn giản hoá và tổ chức cuộc sống của mình, và hiểu về một thế giới rất phức tạp. Có hai cách phân loại đối tượng mà ta hay dùng nhất: phân loại các đối tượng tương tự và phân loại các đối tượng theo thành phần.

+ Phân loại các đối tượng tương tự: nhóm các đối tượng có cùng các thuộc tính và hành vi lại với nhau. Ví dụ, trong loại “ô tô”, ta có thể đưa vào đó các đối tượng có các thuộc tính mầu sắc, nhãn hiệu, kiểu, và các hành vi lái, dừng, rẽ.

Lưu ý: phân loại theo "có hay không một thuộc tính", không phân loại theo giá trị của thuộc tính. Ở đây, ta không lập nhóm các xe ô tô có giá trị “đỏ” cho thuộc tính mầu và giá trị “Honda” cho thuộc tính nhãn hiệu đối tượng chỉ cần có các thuộc tính đó là đủ để thuộc nhóm.

+ Phân loại theo cấu tạo: Thuộc tính của một lớp nào đó có thể có giá trị là một đối tượng thuộc một lớp nào đó.

+ Phân loại đối tượng theo cấu tạo là một trong các kiểu *quan hệ* có thể tồn tại giữa các đối tượng. Kiểu quan hệ này thường được gọi là quan hệ chứa “has-a”.

\* Mô hình hóa: Bước đầu tiên khi mô hình hoá một hệ thống gồm các lớp đối tượng, ta xác định xem:

* Có các lớp đối tượng nào?
* Chúng có các thuộc tính gì?
* Chúng có liên quan đến nhau như thế nào?
* Việc xác định hành vi của các lớp sẽ để sau

Ổn định tập thuộc tính cho mỗi lớp trước khi xác định các hành vi dùng để truy nhập các thuộc tính này.

Khi đã xác định được tất cả các lớp đối tượng cần thiết, các thuộc tính và quan hệ giữa chúng, bước tiếp theo là chỉ ra các hành vi của các lớp đối tượng đó.

* Có những hành vi nào?
* Các lớp đối tượng có thể cần những hành vi nào để có thể sử dụng các thuộc tính của mình?
* Các lớp đối tượng cần thực hiện những nhiệm vụ nào cho các đối tượng khác?

### 1.2.2. Lớp đối tượng

- Ta sẽ gọi một nhóm các đối tượng tương tự là một lớp đối tượng – object class. Một lớp là “một tập các đối tượng có cùng thuộc tính và hành vi”

- Mỗi đối tượng thuộc một lớp là một thể hiện (instance) của lớp đó. *Mỗi xe ô tô là một thể hiện của lớp ô tô.*

- Các thể hiện của một lớp phân biệt nhau bởi một số thuộc tính nhất định có giá trị duy nhất trong toàn lớp - định danh duy nhất *¨ định danh của sinh viên là mã sinh viên – không có hai sinh viên nào có mã giống nhau đôi khi không dễ tìm được cách phân biệt giữa các thể hiện*

- Lớp định nghĩa kiểu dữ liệu và phương thức dùng để truy cập dữ liệu của đối tượng.

*Ví dụ: Lớp đơn xin việc = trường cần điền vào đơn + ghi/đọc đơn*

*(đối tượng) = (dữ liệu) + (phương thức)*

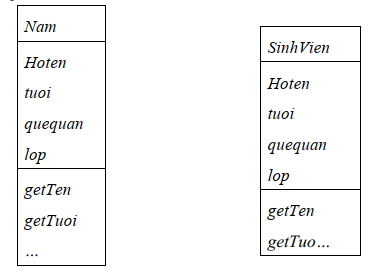
*Lớp này là mẫu đơn xin việc.*

*Những người đến xin việc họ phải được cung cấp một thực thể duy nhất của đơn, những thực thể tạo ra bằng cách lấy bản mẫu làm bản chính, sao chép tạo ra nhiều thực thể. Người xin việc phải điền các thông tin vào đơn bằng phương thức ghi đơn.*

- Lớp được đặc tả bởi tên lớp: tập thuộc tính, tập các hàm.

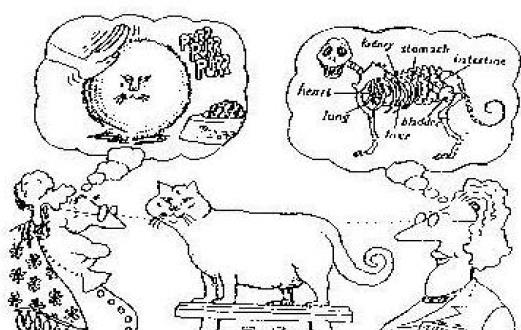
*Ví dụ: lớp sinh viên có 4 thuộc tính, 2 hàm (phương thức)*

*Đối tượng: sinh viên Nam thuộc lớp sinh viên nên nó mang đầy đủ thuộc tính và phương thức như lớp sinh viên.*

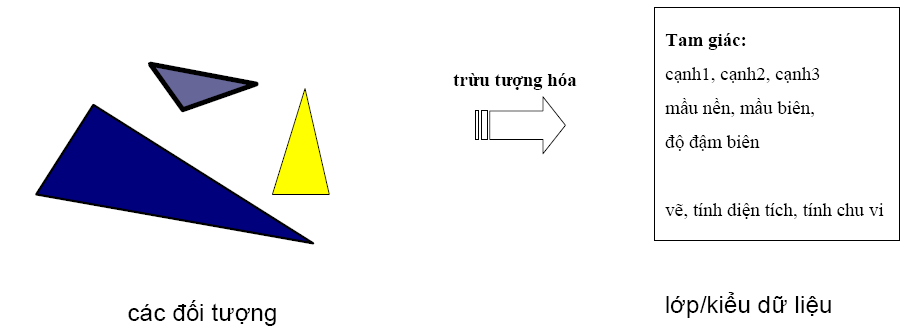


### 1.2.3. Trừu tượng hóa

Trừu tượng hóa là cách biểu diễn đặc tính chính và bỏ qua những chi tiết.



Hình 1.4 Trừu tượng hoá

Trừu tượng hóa là cách nhìn đơn giản hóa về một đối tượng mà trong đó chỉ bao gồm các đặc điểm được quan tâm và bỏ qua những chi tiết không cần thiết. 

Hình 1.5 Sử dụng trừu tượng hoá để xây dựng lớp

- Trìu tượng hóa là sự mở rộng các khái niệm kiểu dữ liệu và cho phép định nghĩa những phép toán trìu tượng trên các kiểu dữ liệu trìu tượng.

- Để xác định lớp ta phải sử dụng khái niệm trìu tượng hóa.

### 1.2.4. Bao bọc và che dấu thông tin

- Đóng gói: Nhóm những gì có liên quan với nhau vào làm một, để sau này có thể dùng một cái tên để gọi đến: Các hàm/ thủ tục đóng gói các câu lệnh. Các đối tượng đóng gói dữ liệu của chúng và các thủ tục có liên quan.

Chi tiết về các thành viên của một đối tượng cần được đóng gói bên trong đối tượng, để cách duy nhất truy nhập đến chúng là qua các hành vi của đối tượng. Tương tự, chi tiết về cách thực hiện các hành vi của một đối tượng cũng nên được đóng gói để bên ngoài không biết chi tiết.

*Tại sao đóng gói đối tượng? Đóng gói bắt chước thế giới thực: nếu tôi muốn anh đứng dậy ( thay đổi thuộc tính “vị trí”) khả năng lớn là tôi sẽ đề nghị anh đứng dậy thay vì đến kéo anh dậy. ¨ điều này làm đơn giản hoá cuộc sống, ta thường không quan tâm đến chi tiết. Trong lập trình hướng đối tượng, lợi ích của việc đóng gói: ¨ khi che dấu chi tiết cài đặt, người dùng bên ngoài không phải bận tâm đến chuyện cái gì được làm như thế nào. Điều này đem lại lợi ích đáng kể trong việc bảo trì phần mềm – do người dùng không bao giờ biết chi tiết bên trong đối tượng, ta có thể thay đổi các chi tiết đó mà họ không cần biết (miễn là giao diện với bên ngoài không đổi)*

- Xác định các vùng: riêng (private) công khai (public) hay bảo vệ ( protected) nhằm điều khiển, hay han chế những truy nhập tùy tiện của những đối tượng khác.

- Việc đóng gói dữ liệu và hàm vào cùng một đơn vị cấu trúc (lớp) được xem là nguyên tắc bao bọc thông tin.

- Nguyên tắc bao bọc dữ liệu để cấm sự truy nhập trực tiếp gọi là sự che dấu thông tin.

Che dấu thông tin: đóng gói để che một số thông tin và chi tiết cài đặt nội bộ để bên ngoài không nhìn thấy. mục tiêu là để khách hàng của ta (thường là các lập trình viên khác) coi các đối tượng của ta là các hộp đen.

### 1.2.5. Kế thừa và mở rộng

- Nguyên lý kế thừa cho phép các đối tượng của lớp này được quyền sử dụng lại một số tính chất (thuộc tính và phương thức) của lớp khác.

- A: lớp cha; B: lớp con. Lớp B sẽ kế thừa các thuộc tính và các hàm có khai báo public hoặc protected của lớp A.

- Trong lớp B có thể bổ sung thêm một số tính chất, phương thức truy cập đẻ thu hẹp phạm vi xác định đối tượng trong lớp mới.

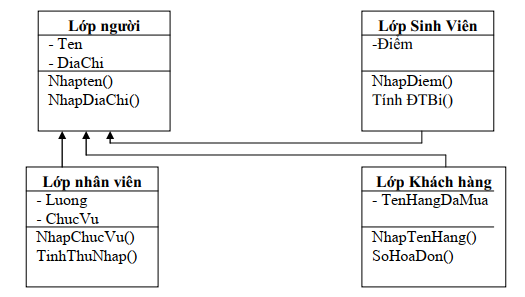
- Những thuộc tính hay hàm được khai báo private thì không kế thừa lại được.

Đây là sự kế từa đơn, trong java không có sự kế thừa bội. Để tận dụng lợi ích của kế thừa bội java xây dựng khái niệm interface (giao diện).

Ví dụ: Trong bài toán có 3 lớp:

\* Lớp sinh viên: - Thuộc tính: tên, địa chỉ, điểm và các hàm: nhapTen(), nhapDiaChi(), nhapDiem().

\* Lớp khách hàng: - Thuộc tính: tên, địa chỉ, tên hàng đã mua, và các hàm: nhapTen(), nhapDiaChi(), nhapTenHang(), nhapHoaDon()



Hình 1.6 Ví dụ kế thừa

### 1.2.6. Đa xạ và nạp chồng

* **Đa hình:**

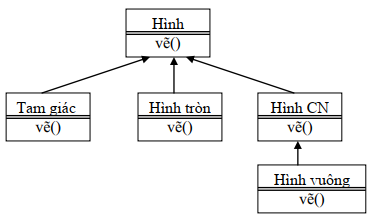
- Là kỹ thuật sử dụng để mô tả khả năng gửi một thông điệp chung tới nhiều đối tượng mà mỗi đối tượng lại có cách xử lý riêng theo ngữ cảnh của mình.

- Khả năng của đối tượng có nhiều phương thức cùng tên nhưng nội dung thực hiện khác nhau. Một thông điệp (lời gọi hàm) được hiểu theo các cách khác nhau tuỳ theo danh sách tham số của thông điệp. Ví dụ: nhận được cùng một thông điệp “nhảy”, một con kangaroo và một con cóc nhảy theo hai kiểu khác nhau: chúng cùng có hành vi “nhảy” nhưng các hành vi này có nội dung khác nhau.



Hình 1.7 Đa hình

Ví dụ: Hàm vẽ là hàm đa trị. Nó được xác định tùy ngữ cảnh khi sử dụng. Khi sử dụng làm vẽ với đối tượng tam giác thì thực hiện việc vẽ tam giác. Khi sử dụng hàm vẽ với đối tượng hình chữ nhật thì thực hiện việc vẽ hình chữ nhật...



Hình 1.8 Ví dụ đa hình

* **Nạp chồng:**

- Nạp chồng là trường hợp đặc biệt của đa trị. Dùng một tên hàm cho nhiều định nghĩa hàm, nhưng khác nhau ở kiểu của tham số hoặc số lượng tham số thì khi đó nội dung thực trong mỗi hàm là khác nhau.

*Ví dụ: hàm cộng() int cong(int, int)*

*float (float, float)*

*string cong (string, string)*

*int cong(int a, int b, int c)*

*int cong(int a[])// danh sách đầu vào là một mảng*

### 1.2.7. Liên kết động

- Liên kết tĩnh: lời gọi hàm (phương thức) được quyết định khi biên dịch, do đó chỉ có một phiên bản của chương trình con được thực hiện. Ưu điểm về tốc độ

- Liên kết động: lời gọi hàm được quyết định khi thực hiện, phiên bản của phương thức phù hợp với đối tượng được gọi. Java mặc định sử dụng liên kết động.

*Ví dụ: khi chương trình gặp hàm vẽ cho đối tượng hình tròn thì hệ thống sẽ liên kết tới hàm vẽ được định nghĩa trong lớp hình tròn.*

### 1.2.8. Truyền thông điệp

Chương trình hướng đối tượng bao gồm tập các đối tượng và mối quan hệ giữa các đối tượng đó với nhau. Lập trình trong ngôn ngữ hướng đối tượng bao gồm các bước sau:

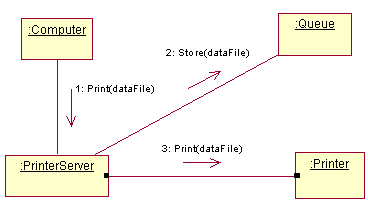
1. Tạo ra các lớp đối tượng và mô tả chúng bằng các thuộc tính và hành vi của chúng

2. Tạo ra các đối tượng theo định nghĩa của các lớp.

3. Xác định sự trao đổi thông tin giữa các đối tượng trong hệ thống.

Truyền thông điệp cho một đối tượng chính là để yêu cầu thực hiện một công việc cụ thể nào đó, nghĩa là sử dụng những hàm tương ứng để xử lý dữ liệu đã được khai báo trong lớp đối tượng đó.

*Ví dụ, khi hệ thống máy tính muốn in một tệp dataFile thì máy tính hiện thời (:Computer) gửi đến cho đối tượng :PrinterServer một yêu cầu Print(dataFile).*



Hình 1.9 Truyền thông điệp giữa các đối tượng

Mỗi đối tượng chỉ tồn tại trong thời gian nhất định. Đối tượng tạo ra khi nó được báo và sẽ đươc loại bỏ khi chương trình ra khỏi miền xác đinh của đối tượng đó. Sự trao đổi thông điệp chỉ thực hiện được trong thời gian tồn tại của đối tượng.

## 1.3. Các ngôn ngữ lập trình hướng đối tượng

Các ngôn ngữ lập trình thời kỳ đầu như Basic, Fortran... không có cấu trúc và cho phép viết những đoạn mã rối rắm (spaghetti code). Lập trình viên sử dụng các lệnh goto” và “gosub” để nhảy đến mọi nơi trong chương trình. Đoạn trình trên khó theo dõi, khó hiểu, dễ gây lỗi, khó sửa đổi.

Kiểu lập trình rối rắm trên dẫn tới phong cách lập trình mới: lập trình cấu trúc, với các ngôn ngữ Algol, Pascal, C... Đặc điểm của lập trình cấu trúc hay lập trình thủ tục (Procedural Programming- PP) là:

* Sử dụng các cấu trúc vòng lặp: for, while, repeat, do-while
* Chương trình là một chuỗi các hàm/ thủ tục
* Mã chương trình tập trung thể hiện thuật toán: làm như thế nào

Hạn chế của lập trình thủ tục:

* Dữ liệu và phần xử lý tách biệt
* Dữ liệu thụ động, xử lý chủ động
* Không đảm bảo được tính nhất quán và các ràng buộc của dữ liệu
* Khó cấm mã ứng dụng sửa dữ liệu của thư viện
* Khó bảo trì code
* Phần xử lý có thể nằm rải rác và phải hiểu rõ cấu trúc dữ liệu

Lập trình hướng đối tượng: *Lập trình hướng đối tượng cho phép khắc phục các hạn chế nói trên*

Các ngôn ngữ lập trình hướng đối tượng không mới: ¨ Simula (1967) là ngôn ngữ đầu tiên, có lớp, thừa kế, liên kết động (hay còn gọi là hàm ảo). *Nhưng các ngôn ngữ hướng đối tượng chậm hơn các ngôn ngữ thời kỳ đầu nên chúng chỉ được dùng rộng rãi khi máy tính bắt đầu chạy nhanh (khoảng thời gian chiếc máy Pentium đầu tiên ra đời).*

*Dựa vào khả năng đáp ứng các khái niệm về hướng đối tượng, ta chia ra làm hai loại:*

\* Ngôn ngữ lập trình dựa trên đối tượng (object-based):

Lập trình dựa trên đối tượng là kiểu lập trình hỗ trợ chính cho việc bao bọc, che giấu thông tin và định danh các đối tượng. Lập trình dựa trên đối tượng có những đặc tính sau:

* Bao bọc dữ liệu,
* Cơ chế che giấu và hạn chế truy nhập dữ liệu,
* Tự động tạo lập và huỷ bỏ các đối tượng,
* Tính đa xạ.

Ngôn ngữ hỗ trợ cho kiểu lập trình trên được gọi là ngôn ngữ lập trình dựa trên đối tượng. Ngôn ngữ trong lớp này không hỗ trợ cho việc thực hiện kế thừa và liên kết động. Ada là ngôn ngữ lập trình dựa trên đối tượng.

\* Ngôn ngữ lập trình hướng đối tượng (object-oriented)

Lập trình hướng đối tượng là kiểu lập trình dựa trên đối tượng và bổ sung thêm nhiều cấu trúc để cài đặt những quan hệ về kế thừa và liên kết động. Vì vậy đặc tính của LTHĐT có thể viết một cách ngắn gọn như sau:

Các đặc tính dựa trên đối tượng + kế thừa + liên kết động.

Ngôn ngữ hỗ trợ cho những đặc tính trên được gọi là ngôn ngữ LTHĐT, ví dụ như Java, C++, Smalltalk, Object Pascal hay Eiffel, v.v... *Mức độ hướng đối tượng của các ngôn ngữ không giống nhau: Eiffel (tuyệt đối), Java (rất cao), C++ (nửa nọ nửa kia)*

## Bài tập cuối chương

1.1 Nêu các đặc trưng cơ bản của cách tiếp cận lập trình hướng chức năng và lập trình hướng đối tượng.

1.2 Tại sao lại gọi khái niệm lớp là kiểu dữ liệu trừu tượng trong lập trình hướng đối tượng.

1.3 Nêu nguyên lý hoạt động của khái niệm đa xạ, tương ứng bội trong lập trình hướng đối tượng, khái niệm này thực hiện được trong lập trình hướng chức năng hay không, tại sao?

1.4 Khái niệm kế thừa và sử dụng lại trong lập trình hướng đối tượng là gì?, ngôn ngữ lập trình C++ và Java hỗ trợ quan hệ kế thừa như thế nào?.

1.5 Liên kết động là gì?, tại sao lại phải cần liên kết động?

1.6 Các tính chất của hướng đối tượng là gì ?

1.7 Thế nào là lớp?

1.8 Thế nào là đối tượng?

1.9 Thế nào là tính đóng gói?

1.10 Thế nào là tính đa hình?

1.11. Đối tượng trong phần mềm là gì?

A. Là một bó phần mềm gồm các hành vi và trạng thái có liên quan với nhau.

B. Là vật thể xác định của thế giới thực.

C. Là vật thể gồm hành vi và trạng thái.

D. Là các đối tượng được biểu diễn trong phần mềm gồm có 2 thuộc tính trường dữ liệu và các cách xử lý dữ liệu.

# Chương II: TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG JAVA

### Nội dung chính của chương

- Các thành phần cơ sở của ngôn ngữ lập trình Java

- Các kiểu dữ liệu kiểu nguyên thủy

- Các phép toán và các biểu thức tính toán

- Các cấu trúc cơ bản: Cấu trúc rẽ nhánh, cấu trúc lặp, các câu lệnh chuyển vị

### Mục tiêu cần đạt được của chương

Trong chương 2 sinh viên sẽ nắm được tổng quan về ngôn ngữ java: lịch sử của java, java là gì? Cấu trúc chung của một chương trình java và các loại chương trình java.

## Bài 2: Sơ lược về java (Số tiết: 03 tiết)

## 2.1 Lịch sử phát triển của java

Java là ngôn ngữ lập trình hướng đối tượng (tựa C++) do Sun Microsystem đưa ra vào giữa thập niên 90. Chương trình viết bằng ngôn ngữ lập trình java có thể chạy trên bất kỳ hệ thống nào có cài máy ảo java

Năm 1990: James Gosling và các công sự của Công ty Sun Microsystem tham gia dự án Green Team - xây dựng công nghệ mới cho ngành điện tử tiêu dung (cho các thiết bị điện dân dụng). Để giải quyết vấn đề này nhóm nghiên cứu phát triển đã xây dựng một ngôn ngữ lập trình mới đặt tên là Oak tương tự như C++ nhưng loại bỏ một số tính năng nguy hiểm của C++ và có khả năng chạy trên nhiều nền phần cứng khác nhau.

Năm 1993 world wide web bắt đầu phát triển. Năm 1994, Sun đưa ra trình duyệt web viết bằng ngôn ngữ oak là webrunner, sau đổi tên thành hostJava.

Sau đó, Sun đổi tên oak thành Java và được giới thiệu năm 1995 tại Sunworld 1995.

Java là tên gọi của một hòn đảo ở Indonexia, đây là nơi nhóm nghiên cứu phát triển đã chọn để đặt tên cho ngôn ngữ lập trình Java trong một chuyến đi tham quan và làm việc trên hòn đảo này. Hòn đảo Java này là nơi rất nổi tiếng với nhiều khu vườn trồng cafe, đó chính là lý do chúng ta thường thấy biểu tượng ly café trong nhiều sản phẩm phần mềm, công cụ lập trình Java của Sun cũng như một số hãng phần mềm khác đưa ra.

## 2.2 Ngữ cảnh các ứng dụng của ngôn ngữ java.

Ngôn ngữ Java có những đặc trưng cơ bản sau:

**Đơn giản:** phát triển trên một ngôn ngữ dễ học và quen thuộc với đa số người lập trình là C và Java loại bỏ các phức tạp của C và C++ như: con trỏ, đa kế thừa, định nghĩa chồng toán tử , không sử dụng lệnh “goto” cũng như file header (.h), loại bỏ cấu trúc “struct” và “union”.

**Hướng đối tượng** Java là ngôn ngữ lập trình hoàn toàn hướng đối tượng:

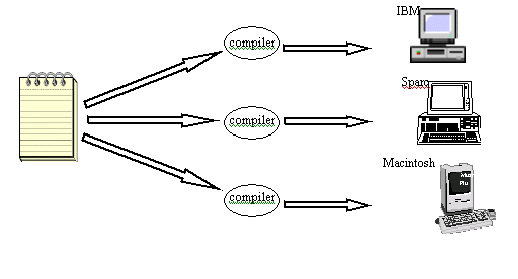
• Mọi thực thể trong hệ thống đều được coi là một đối tượng, tức là một thể hiện cụ thể của một lớp xác định.

• Tất cả các chương trình đều phải nằm trong một class nhất định.

• Không thể dùng Java để viết một chức năng mà không thuộc vào bất kì một lớp nào. Tức là Java không cho phép định nghĩa dữ liệu và hàm tự do trong chương trình.

**Độc lập phần cứng và hệ điều hành**

Đối với các ngôn ngữ lập trình truyền thống như C/C++, phương pháp biên dịch được thực hiện như sau:



Hình 2.1 Chương trình java chạy trên mọi hệ điều hành

Với một nền phần cứng khác nhau, có một trình biên dịch khác nhau để biên dịch mã nguồn chương trình cho phù hợp với nền phần cứng ấy. Do vậy, khi chạy trên một nền phần cứng khác, bắt buộc phải biên dịch lại mà nguồn.

Đối các chương trình viết bằng Java, chỉ cần cài máy ảo java là có thể chạy được.

**Máy ảo Java** (Java Virtual Machine).

• Là một phần mềm dựa trên cơ sở máy tính ảo

• Là tập hợp các lệnh logic để xác định hoạt động của máy tính

• Được xem như là một hệ điều hành thu nhỏ

• Nó thiết lập lớp trừu tượng cho:

– Phần cứng bên dưới

– Hệ điều hành

– Mã đã biên dịch

**Mạnh mẽ** Java là ngôn ngữ yêu cầu chặt chẽ về kiểu dữ liệu:

• Kiểu dữ liệu phải được khai báo tường minh.

• Java không sử dụng con trỏ và các phép toán con trỏ.

• Java kiểm tra việc truy nhập đến mảng, chuỗi khi thực thi để đảm bảo rằng các truy nhập đó không ra ngoài giới hạn kích thước mảng.

• Quá trình cấp phát, giải phóng bộ nhớ cho biến được thực hiện tự động, nhờ dịch vụ thu nhặt những đối tượng không còn sử dụng nữa (garbage collection).

• Cơ chế bẫy lỗi của Java giúp đơn giản hóa quá trình xử lý lỗi và hồi phục sau lỗi.

**Bảo mật** Java cung cấp một môi trường quản lý thực thi chương trình với nhiều mức để kiểm soát tính an toàn:

• Ở mức thứ nhất, 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.

• Ở mức thứ hai, trình biên dịch kiểm soát để đảm bảo mã là an toàn, và tuân theo các nguyên tắc của Java.

• Mức 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.

• Mức thứ tư kiểm soát việc nạp các lớp vào bộ nhớ để giám sát việc vi phạm giới hạn truy xuất trước khi nạp vào hệ thống.

**Phân tán** Java được thiết kế để hỗ trợ các ứng dụng chạy trên mạng bằng các lớp Mạng (java.net). Hơn nữa, Java hỗ trợ nhiều nền chạy khác nhau.

**Đa luồng** Java cung cấp giải pháp đa luồng (Multithreading) để thực thi các công việc cùng đồng thời và đồng bộ giữa các luồng.

**Linh động** Java được thiết kế như một ngôn ngữ động để đáp ứng cho những môi trường mở. Các chương trình Java chứa rất nhiều thông tin thực thi nhằm kiểm soát và truy nhập đối tượng lúc chạy.

## 2.3. Cấu trúc chung của chương trình java

Cấu trúc chung:

*package packageName; // Khai báo tên gói, nếu có*

*import java.awt.\*; // Khai báo tên thư viện sẵn có, nếu cần dùng*

*class className // Khai báo tên lớp*

*{*

*/\* Đây là dòng ghi chú \*/*

*int var; // Khai báo biến*

*public void methodName() // Khai báo tên phương thức*

*{*

*/\* Phần thân của phương thức \*/*

*statement (s); // Lệnh thực hiện}*

*}*

*interface …// Khai báo giao diện nếu có*

**Ví dụ minh họa**

*public class HelloWorldApp{*

*public static void main(String[] args){*

*System.out.println(“HelloWorld”);*

*}*

*}*

## 2.4. Biên dịch và thông dịch chương trình java

Quá trình dịch chương trình Java:

• Trình biên dịch chuyển mã nguồn thành tập các lệnh không phụ thuộc vào phần cứng cụ thể.

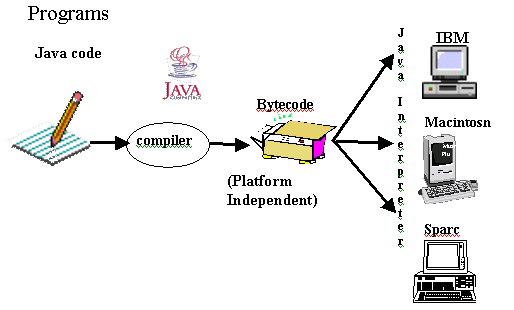
• Trình thông dịch trên mỗi máy chuyển tập lệnh này thành chương trình thực thi

• Máy ảo tạo ra một môi trường để thực thi các lệnh bằng cách:

– Nạp các file .class

– Quản lý bộ nhớ

– Dọn “rác”



Hình 2.2 Chạy và biên dịch chương trình java

Chạy java chỉ cần cài máy ảo và khi chạy chỉ cần chạy file class, không cần phải dịch lại nên nó độc lập với phần cứng máy tính: “viết một lần chạy ở mọi nơi”

## Bài tập cuối chương

2.1. Đâu là câu SAI về ngôn ngữ Java?

A. Ngôn ngữ Java có phân biệt chữ hoa – chữ thường

B. Java là ngôn ngữ lập trình hướng đối tượng

C. Dấu chấm phẩy được sử dụng để kết thúc lệnh trong java

D. Chương trình viết bằng Java chỉ có thể chạy trên hệ điều hành win

2.2. Tên đầu tiên của Java là gì?

A. Java

B. Oak

C. Cafe

D. James golings

2.3. Muốn chạy được chương trình java, chỉ cần cài phần mền nào sau đây?

A. Netbeans

B. Eclipse

C. JDK

D. Java Platform

2.4. API là gì?

A. Thư viện mã nguồn của Java.

B. Là thư viện chứa các thành phần phần mềm tạo sẵn cung cấp các chức năng cho chương trình Java.

C. Thư viện cung cấp giao diện đồ họa cho chương trình Java.

D. Tất cả đều sai

## Bài 3: Đặc trưng của chương trình ứng dụng Java (Số tiết: 03 tiết)

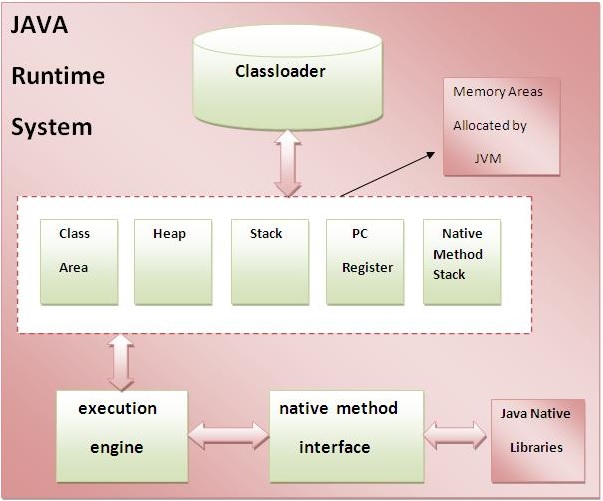
## 2.5. Các đặc trưng của chương trình ứng dụng Java

### 2.5.1. Máy ảo Java (JVM - Java Virtual Machine)

Tất cả các chương trình muốn thực thi được thì phải được biên dịch ra mã máy. Mã máy của từng kiến trúc CPU của mỗi máy tính là khác nhau (tập lệnh mã máy của CPU Intel, CPU Solarix, CPU Macintosh … là khác nhau), vì vậy trước đây một chương trình sau khi được biên dịch xong chỉ có thể chạy được trên một kiến trúc CPU cụ thể nào đó. Đối với CPU Intel chúng ta có thể chạy các hệ điều hành như Microsoft Windows, Unix, Linux, OS/2, …

Chương trình thực thi được trên Windows được biên dịch dưới dạng file có đuôi .EXE còn trên Linux thì được biên dịch dưới dạng file có đuôi .ELF, vì vậy trước đây một chương trình chạy được trên Windows muốn chạy được trên hệ điều hành khác như Linux chẳng hạn thì phải chỉnh sửa và biên dịch lại.

Ngôn ngữ lập trình Java ra đời, nhờ vào máy ảo Java mà khó khăn nêu trên đã được khắc phục. Một chương trình viết bằng ngôn ngữ lập trình Java sẽ được biên dịch ra mã của máy ảo java (mã java bytecode). Sau đó máy ảo Java chịu trách nhiệm chuyển mã java bytecode thành mã máy tương ứng. Sun Microsystem chịu trách nhiệm phát triển các máy ảo Java chạy trên các hệ điều hành trên các kiến trúc CPU khác nhau.



Hình 2.3 Kiến trúc máy ảo Java

Trong đó:

* **Classloader:** Là một hệ thống con của JVM được sử dụng để tải class file.
* **Class (method) Area:** Lưu trữ cấu trúc mỗi lớp, chẳng hạn như hằng, trường, dữ liệu phương thức, code của phương thức, ...
* **Heap:** Nó là khu vực dữ liệu runtime mà trong đó đối tượng được cấp phát.
* **Stack:** Stack trong Java lưu giữ các Frame. Nó giữ các biến cục bộ và các kết quả cục bộ, và thực hiện một phần nhiệm vụ trong phần triệu hồi và trả về phương thức. Mỗi Thread có một Stack riêng, được tạo tại cùng thời điểm với Thread.  
  Một Frame mới được tạo mỗi khi một phương thức được triệu hồi và bị hủy khi lời triệu hồi phương thức là kết thúc.
* **Program Counter Register:** Nó chứa địa chỉ của chỉ lệnh JVM hiện tại đang được thực thi.
* **Native Method Stack:** Bao gồm tất cả các phương thức tự nhiện được sử dụng trong ứng dụng.
* **Execution Engine:** Phần này bao gồm: Một bộ xử lý ảo Virtual Processor  
  Một trình thông dịch Interpreter. Đọc Bytecode Stream sau đó thực thi các chỉ thị.
* **Just-In-Time (JIT) Compiler:** được sử dụng để cải thiện hiệu suất. JIT biên dịch các phần của Bytecode mà có cùng tính năng tại cùng một thời điểm, và vì thế giảm lượng thời gian cần thiết để biên dịch. Ở đây khái niệm Compiler là một bộ biên dịch tập chỉ thị của JVM thành tập chỉ thị của một CPU cụ thể.

### 2.5.2. Thông dịch

Java là một ngôn ngữ lập trình vừa biên dịch vừa thông dịch. Chương trình nguồn viết bằng ngôn ngữ lập trình Java có đuôi \*.java đầu tiên được biên dịch thành tập tin có đuôi \*.class và sau đó sẽ được trình thông dịch thông dịch thành mã máy.

### 2.5.3. Độc lập nền

Một chương trình viết bằng ngôn ngữ Java có thể chạy trên nhiều máy tính có hệ điều hành khác nhau (Windows, Unix, Linux, …) với điều kiện ở đó có cài đặt máy ảo java (Java Virtual Machine).

### 2.5.4. Hướng đối tượng

Hướng đối tượng trong Java tương tự như C++ nhưng Java là một ngôn ngữ lập trình hướng đối tượng hoàn toàn. Tất cả mọi thứ đề cập đến trong Java đều liên quan đến các đối tượng được định nghĩa trước, thậm chí hàm chính của một chương trình viết bằng Java (đó là hàm main) cũng phải đặt bên trong một lớp. Hướng đối tượng trong Java không có tính đa kế thừa (multi inheritance) như trong C++ mà thay vào đó Java đưa ra khái niệm interface để hỗ trợ tính đa kế thừa.

### 2.5.5. Đa nhiệm - đa luồng (MultiTasking - Multithreading)

Java hỗ trợ lập trình đa nhiệm, đa luồng cho phép nhiều tiến trình, các tiến trình có thể chạy song song cùng một thời điểm và có thể tương tác với nhau.

### 2.5.6. Khả chuyển (portable)

Chương trình ứng dụng viết bằng ngôn ngữ Java chỉ cần chạy được trên máy ảo Java là có thể chạy được trên bất kỳ máy tính, hệ điều hành nào có máy ảo Java. “Viết một lần, chạy mọi nơi” (Write Once, Run Anywhere).

### 2.5.7. Hỗ trợ mạnh cho việc phát triển ứng dụng

Công nghệ Java phát triển mạnh mẽ nhờ vào “đại gia Sun Microsystem” cung cấp nhiều công cụ, thư viện lập trình phong phú hỗ trợ cho việc phát triển nhiều loại hình ứng dụng khác nhau cụ thể như:

- J2SE (Java 2 Standard Edition) hỗ trợ phát triển những ứng dụng đơn, ứng dụng client-server.

- J2EE (Java 2 Enterprise Edition) hỗ trợ phát triển các ứng dụng thương mại.

- J2ME (Java 2 Micro Edition) hỗ trợ phát triển các ứng dụng trên các thiết bị di động, không dây,...

## 2.6. Bộ công cụ phát triển JDK

JDK – Java Develop Kit – được dịch ra tiếng Việt là bộ công cụ phát triển ngôn ngữ lập trình bằng ứng dụng Java. JDK chính là một công nghệ và trong đó là tập hợp một loạt các công cụ nhằm hỗ trợ cho người dùng trong việc viết lên các chương trình ứng dụng dưới dạng ngôn ngữ lập trình. JDK dùng để viết ứng dụng mà trong đó gồm có JRE – dùng để chạy những ứng dụng mà JDK cần đến, để góp thêm phần làm ra những ứng dụng trong một hệ ngôn từ lập trình mà trong đó JDK có được sự tương hỗ từ những chương trình có sẵn trong JRE.

Từ khi ngôn từ lập trình Java sinh ra cũng đồng nghĩa tương quan với sự hình thành và tăng trưởng của bộ công cụ tương hỗ này ngay sau đó. Đây là một trong những bộ công cụ thông dụng và được sử dụng thoáng rộng và thông dụng bậc nhất đi cùng với sự tăng trưởng của ngôn từ lập trình JAVA. JDK cũng là một công nghệ tiên tiến được tăng trưởng bởi Sun Microsystem và được công bố cấp phép bởi GNU vào ngày 11/7/2006 – giấy phép về hội đồng GNU – một dạng giấy phép về ứng dụng tự do được cho phép người dùng hoàn toàn có thể san sẻ, sửa đổi và điều tra và nghiên cứu về những loại ứng dụng và được sử dụng rất thông dụng. JDK đã được cung ứng nguồn mở và trở thành một ứng dụng tự do. Nguồn mở không lấy phí – Openjdk được phát hành vào năm 2007 ngay sau đó.

**Bộ dụng cụ trong JDK gồm:**

– Trình tương hỗ thực thi những chương trình Java hay hiểu cách khác đây chính là môi trường tự nhiên để cho Java chạy và tăng trưởng. Nó chính là thiên nhiên và môi trường được tạo dựng nên để đem lại sự tăng trưởng cho những chương trình khác. Với tên gọi JRE ( Java Runtime Environment ).

– Trình biên dịch tương hỗ trong việc biên dịch ngôn từ lập trình từ ngôn từ nguồn ( mã nguồn ) sang ngôn từ đích. Thông thường những ngôn từ nguồn thường sẽ đơn thuần hơn những ngôn từ được biên dịch ra ngôn từ đích. ( Java Compile ).

– Trình tạo bản phân phối được cho phép toàn bộ những tệp thành một bản duy nhất việc làm này có tính năng rất lớn trong việc tiết kiệm chi phí dung tích trong quy trình sử dụng những ứng dụng và việc tàng trữ tài liệu hay gửi đi một nơi khác trong thực tiễn trở nên nhỏ gọn và thuận tiện hơn. hay người ta gọi trình tạo bản phân phối này là Java Achieved – nén những file lại thành một bản duy nhất với những đuôi Jav một cách đồng nhất hóa.

– Trình sửa lỗi – Debug – chuyên đi nghiên cứu và phân tích và gỡ lỗi cho những chương trình Java.

– Trình tạo tài liệu – Java Doc

– Thư viện lập trình trong thực tiễn JDK = JRE + những công cụ tăng trưởng ứng dụng bên trong JRE như : Trình thông dịch, biên dịch, tàng trữ, trình sửa lỗi, ..

**JRE** - Java Runtime Environment – là môi trường để các ứng dụng thực thi trên đó trong khoảng thời gian lập trình. JRE có chứa các thư viện lớp Java, trình tải lớp và máy ảo Java.

**Máy ảo Java**, tiếng Anh là Java virtual machine (viết tắt: JVM) là một máy ảo cho phép chạy các chương trình Java cũng như các chương trình khác được viết bằng các ngôn ngữ khác mà được biên dịch sang mã byte Java. JVM được mô tả chi tiết bởi một đặc tả mô tả chính thức những gì cần cho một hiện thực JVM. Đặc tả đó để bảo đảm khả năng tương tác của các chương trình Java trên những hiện thực khác nhau để tác giả các chương trình đó có thể sử dụng Bộ công phụ Phát triển Java (JDK) mà không cần phải lo lắng về đặc tính của nền tảng phần cứng bên dưới.

## 2.7 Các môi trường tích hợp phát triển ứng dụng Java

**Eclipse** là một môi trường phát triển tích hợp dùng cho lập trình máy tính.[6], Nó chứa một không gian làm việc cơ sở và một hệ thống plug-in để mở rộng để tùy chỉnh môi trường. Eclipse được viết chủ yếu bằng Java và nó được dùng chủ yếu cho lập trình ứng dụng Java, nhưng nó cũng có thể dùng để lập trình ứng dụng bằng các ngôn ngữ khác thông qua plug-ins, bao gồm Ada, ABAP, C, C++, C#, Clojure, COBOL, D, Erlang, Fortran, Groovy, Haskell, HTML, JavaScript, Julia[7] Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby (Bao gồm Ruby on Rails framework), Rust, Scala, và Scheme. Nó cũng có thể dùng để phát triển các tài liệu bằng LaTeX (thông qua một plug-in TeXlipse) và các gói tin cho phần mềm Mathematica. Môi trường phát triển bao gồm Eclipse Java development tools (JDT) cho Java và Scala, Eclipse CDT cho C/C++, và Eclipse PDT for PHP, và những gói khác.

**NetBeans** là một môi trường phát triển tích hợp (IDE) cho Java. NetBeans cho phép các ứng dụng được phát triển từ một tập hợp các thành phần phần mềm được gọi là modules. NetBeans chạy trên Windows, macOS, Linux và Solaris. Ngoài việc phát triển Java, nó còn có các phần mở rộng cho các ngôn ngữ khác như PHP, C, C++, HTML5, và JavaScript. Các ứng dụng dựa trên NetBeans, bao gồm NetBeans IDE, có thể được mở rộng bởi các nhà phát triển bên thứ ba.

**jGRASP** là phần mềm được cung cấp hỗ trợ cho môi trường phát triển tích hợp có thể tự động tạo các bản trình bày trực quan về phần mềm của nhà phát triển, cung cấp thông tin quan trọng liên quan đến cấu trúc, kích thước và các thuộc tính khác của phần mềm. Phần mềm có thể áp dụng cho môi trường Java, tương thích với mọi nền tảng Java. jGRASP có nhiều phiên bản khác nhau, dựa trên Hệ điều hành được cài đặt trên máy. Các biểu diễn trực quan do phần mềm tạo ra là Sơ đồ cấu trúc điều khiển, nhằm mục đích trực quan hóa mã nguồn; Sơ đồ lớp UML, nhằm mục đích trực quan hóa cấu trúc kiến ​​trúc; và trình xem động, dành cho các chế độ xem thời gian chạy của các đối tượng như cây nhị phân; cũng như các chế độ xem thời gian chạy ban đầu. Điểm đặc biệt của phần mềm là Object Workbench, về cơ bản cho phép người dùng thêm một đối tượng vào workbench và phân tích các thay đổi trên đối tượng cho mọi phương thức mà người dùng gọi. Phần mềm cũng bao gồm một hướng dẫn có thể hướng dẫn người dùng mới làm quen với toàn bộ khái niệm về việc tạo tệp nguồn Java. Người dùng có thể học cách sử dụng đầy đủ từng tính năng của phần mềm và tìm hiểu tất cả các kiến ​​thức cơ bản về tạo tệp nguồn Java.

## 2.8. Các dạng chương trình ứng dụng của Java

### 2.8.1. Chương trình ứng dụng độc lập

Ví dụ: gõ đoạn mã sau:

*public class HelloWorldApp{*

*public static void main(String[] args){*

*System.out.println(“HelloWorld”);*

*} }*

Lưu lại với tên HelloWorldApp.java

Để biên dịch mã nguồn, ta sử dụng trình biên dịch javac.

- Mở cửa sổ Command Prompt.

- Chuyển đến thư mục chứa tập tin nguồn vừa tạo ra.

- Thực hiện câu lệnh: **javac HelloWordApp.java**

Trình dịch javac tạo ra file **HelloWordApp**.class chứa các mã “bytecodes”. Để chương trình thực thi được ta cần dùng trình thông dịch:

**java HelloWordApp**

- Nếu chương trình đúng ta sẽ thấy dòng chữ HelloWord trên màn hình Console.

- Nếu nhận được lỗi “Exception in thread "main

java.lang.NoClassDefFoundError:HelloWorldApp” có nghĩa là Java không thể tìm được tập tin mã bytecode tên HelloWorldApp.class.

Cấu trúc của chương trình **java HelloWordApp**

*public static void main(String args[ ])*

Đây là phương thức chính, từ đây chương trình bắt đầu việc thực thi của mình. Tất cả các ứng dụng java đều sử dụng một phương thức main này.

• Từ khoá public là một chỉ định truy xuất. Nó cho biết thành viên của lớp có thể được truy xuất từ bất cứ đâu trong chương trình.

• Từ khoá static cho phép main được gọi tới mà không cần tạo ra một thể hiện (instance) của lớp. Nó không phụ thuộc vào các thể hiện của lớp được tạo ra.

• Từ khoá void thông báo cho máy tính biết rằng phương thức sẽ không trả lại bất cứ giá trị nào khi thực thi chương trình.

• String args[] là tham số dùng trong phương thức main. Khi không có một thông tin nào được chuyển vào main, phương thức được thực hiện với các dữ liệu rỗng – không có gì trong dấu ngoặc đơn.

• System.out.println(“Hello World”); Dòng lệnh này hiển thị chuỗi “Hello World” trên màn hình. Lệnh println() cho phép hiển thị chuỗi được truyền vào lên màn hình.

### 2.8.2. Chương trình ứng dụng nhúng

Applet là một chương trình Java có thể chạy trong các trình duyệt web có hỗ trợ Java. Tất cả các applet đều là các lớp con của lớp Applet. Để tạo applet, ta cần import hai gói sau:

*import java.applet.\*;*

*import java.awt.\*;*

* Cấu trúc của một Applet:

*public class <Tên lớp applet> extends Applet{*

*… // Các thuộc tính*

*public void init(){…}*

*public void start(){…}*

*public void stop(){…}*

*public void destroy(){…}*

*… // Các phương thức khác*

*}*

Các phương thức cơ bản của một applet:

• init(): Khởi tạo các tham số, nếu có, của applet.

• start(): Applet bắt đầu hoạt động

• stop(): Chấm dứt hoạt động của applet.

• destroy(): Thực hiện các thao tác dọn d p trước khi thoát khỏi applet.

**Lưu ý:**

• Không phải tất cả các applet đều phải cài đặt đầy đủ 4 phương thức cơ bản trên. Applet còn có thể cài đặt một số phương thức tuỳ chọn (không bắt buộc) sau:

• paint(Graphics): Phương thức vẽ các đối tượng giao diện bên trong applet. Các thao tác vẽ này được thực hiện bởi đối tượng đồ hoạ Graphics (là tham số đầu vào).

• repaint(): Dùng để vẽ lại các đối tượng trong applet. Phương thức này sẽ tự động gọi phương thức update().

• update(Graphics): Phương thức này được gọi sau khi thực hiện phương thức paint nhằm tăng hiệu quả vẽ. Phương này sẽ tự động gọi phương thức paint().

Ví dụ: cài đặt một applet đơn giản vẽ ra chuỗi “hello word!”

*import java.awt.\*;*

*import java.applet.\*;*

*public class SimpleApplet extends Applet{*

*public void paint(Graphics g){*

*g.drawString( “ Hello world!”,50,50);*

*} }*

**\* Sử dụng applet**

Applet không thể chạy như một ứng dụng Java độc lập (nó không có hàm main), mà nó chỉ chạy được khi được nhúng trong một trang HTML (đuôi .htm, .html) và chạy bằng một trình duyệt web thông thường.

Các bước xây dựng và sử dụng một applet bao gồm:

• Biên dịch mã nguồn thành lớp .class

• Nhúng mã .class của applet vào trang html.

Để nhúng một applet vào một trang html, ta dùng thẻ (tag) <Applet>:

Trong trang myHtml.htm có chứa nội dung như sau:

*<HTML>*

*<HEAD>*

*<TITLE> A simple applet </TITLE>*

*</HEAD>*

*<BODY>*

*This is the output of applet:*

*<APPLET CODE = “SimpleApplet.class” WIDTH=200 HEIGHT=20>*

*</APPLET>*

*</BODY>*

*</HTML>*

Mở trang myHtml trên các trình duyệt thông thường hoặc dùng lệnh appletviewer để chạy

## Bài tập cuối chương

2.5. Gói nào trong java chứa lớp Scanner dùng để nhập dữ liệu từ bàn phím?

A. java.net

B. java.io

C. java.util

D. java.awt

2.6. Trong câu lệnh sau: public static void main(String[] agrs) thì phần tử agrs[0] chứa giá trị gì?

A. Tên của chương trình

B. Số lượng tham số

C. Tham số đầu tiên của danh sách tham số

D. Không câu nào đúng

2.7. Tên đầu tiên của Java là gì?

A. Java

B. Oak

C. Cafe

D. James golings

2.8. Java Virtual Machine là gì?

A. Là một thành phần của Java platform dùng để đọc mã bytecode trong file .class

B. Là chương trình biên dịch của java dùng để biên dịch file nguồn java thành mã bytecode.

C. Là chương trình chạy cho java

D. Tất cả các đáp án đều đúng

2.9. Java chạy trên hệ điều hành nào sau đây:

A. Microsoft Windows

B. Linux

C. Sun Solaris OS

D. Tất cả các đáp án đều đúng.

2.10. Đối tượng trong phần mềm là gì?

A. Là một bó phần mềm gồm các hành vi và trạng thái có liên quan với nhau.

B. Là vật thể xác định của thế giới thực.

C. Là vật thể gồm hành vi và trạng thái.

2.11. Một chương trình gồm 2 class sẽ có bao nhiêu cách main?

A. 1

B. 2

C. 3

D. 4

# Chương III: CÁC THÀNH PHẦN CƠ SỞ CỦA JAVA

### Nội dung chính của chương

- Các thành phần cơ sở của ngôn ngữ lập trình Java

- Các kiểu dữ liệu nguyên thủy

- Các phép toán và các biểu thức tính toán

- Các cấu trúc cơ bản: Cấu trúc rẽ nhánh, cấu trúc lặp, các câu lệnh chuyển vị

### Mục tiêu cần đạt được của chương

Giúp sinh viên sẽ nắm được các thành phần cơ sở của ngôn ngữ java, cách xây dựng một chương trình Java đơn giản.

## Bài 4: Các phần tử cơ sở của Java (Số tiết: 03 tiết)

## 3.1. Các phần tử cơ sở của Java

### 3.1.1. Các phần tử cơ sở

**\* Định danh (Tên gọi)**

Trong Java định danh là một dãy các ký tự gồm các chữ cái, chữ số và một số các ký hiệu như: ký hiệu gạch dưới nối câu „\_‟, các ký hiệu tiền tệ $, Ơ, Ê, Â, và không được bắt đầu bằng chữ số.

Java phân biệt chữ thường và chữ hoa. Độ dài (số ký tự) của định danh trong Java về lý thuyết là không bị giới hạn.

Quy ước(lời khuyên):

+ Định danh được đặt tên cho các lớp, các đối tượng: chữ cái đầu của mỗi từ viết hoa.

+ Định danh cho các biến, phương thức, tên phương thức thường bằng động từ (viết in thường, và các danh từ phía sau thì chữ cái đầu viết chữ in hoa) và , đối tượng: chữ cái đầu của mỗi từ trong định danh đều viết hoa trừ từ đầu tiên(“náo”).

**\* Chú thích (Comment)**

// Chú thích trên một dòng

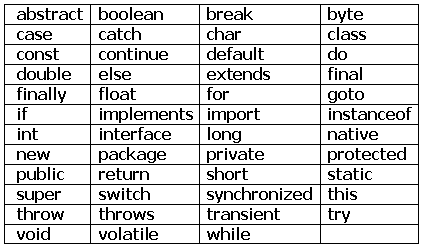
/\* Chú thích trên nhiều dòng

… \*/

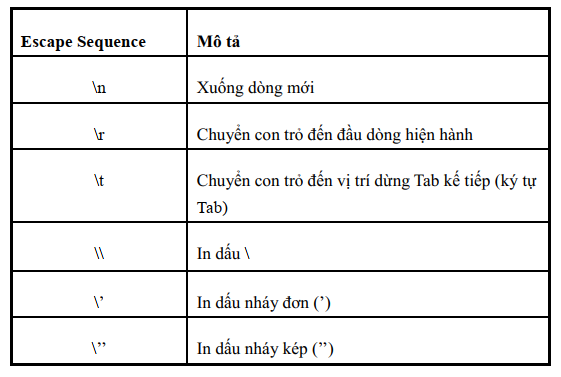
/\*\* Chú thích trong tư liệu (javadoc)

…\*/

**\* Các từ khóa của java**



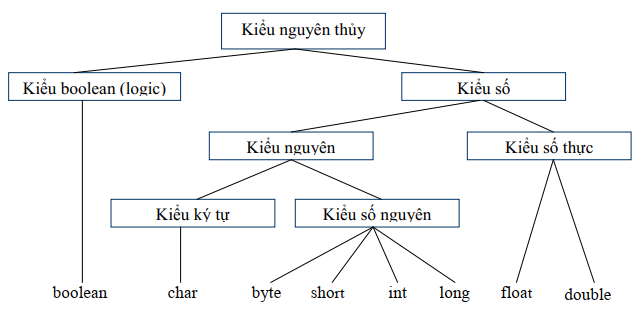
**\* Các kí tự định dạng xuất dữ liệu (Escape Sequences)**



### 3.1.2. Các kiểu dữ liệu trong Java

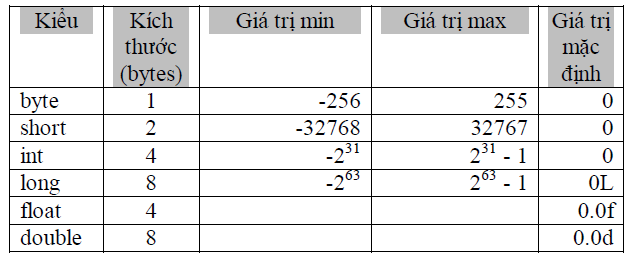
Java có 2 dạng dữ liệu: Dữ liệu nguyên thủy và dữ liệu tham chiếu đối tượng.

Các kiểu dữ liệu nguyên thủy là các kiểu dữ liệu có sẵn và định nghĩa trước của java:



Hình 2.4 Các kiểu dữ liệu

Mỗi kiểu nguyên thủy có một lớp bao bọc (Wrapper hay còn gọi lớp nguyên thủy) tương ứng để sử dụng các giá trị nguyên thủy như là các đối tượng. Ví dụ ứng với kiểu int có lớp Integer, ứng với char là Char vv.. các lớp này sẽ cung cấp một số phương thức hỗ trợ trên kiểu tương ứng,



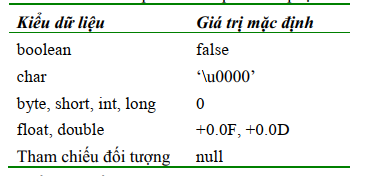
### 3.1.3. Khai báo biến

[phạm vi] <Kiểu dữ liệu> <tên biến>;

hoặc khai báo đồng thời khởi tạo giá trị ban đầu luôn:

[phạm vi]<Kiểu dữ liệu> <tên biến> = <giá trị>;

**Lưu ý:** chỉ khai báo biến thành phần của lớp mới có phạm vi truy cập.



*Khi khai báo:* Nếu các biến không được khởi tạo giá trị tường minh thì:

* Các biến tĩnh luôn được khởi tạo với các giá trị mặc định.
* Các biến thành phần được khởi tạo mặc định mỗi khi đối tượng của lớp có thành phần đó được khởi tạo.
* Biến tham chiếu được gán mặc định là null nếu không tạo lập tường minh theo toán tử new và toán tử tạo lập (constructor).

**Phạm vi hoạt động của biến**

Một biến có phạm vi hoạt động là sau khi khai báo và trong toàn bộ khối lệnh mà nó được khai báo. Một khối lệnh bắt đầu bằng dấu “{” và kết thúc bằng dấu “}”:

* Nếu biến được khai báo trong một cấu trúc lệnh điều khiển, biến đó có phạm vi hoạt động trong khối lệnh tương ứng.
* Nếu biến được khai báo trong một phương thức (Không nằm trong khối lệnh nào), biến đó có phạm vi hoạt động trong phương thức tương ứng: có thể được sử dụng trong tất cả các khối lệnh của phương thức.
* Nếu biến được khai báo trong một lớp (Không nằm trong trong một phương thức nào), biến đó có phạm vi hoạt động trong toàn bộ lớp tương ứng: có thể được sử dụng trong tất cả các phương thức của lớp.

### 3.1.4. Biểu thức trong Java

|  |  |  |  |
| --- | --- | --- | --- |
| Số ưu tiên | Tên gọi | Các phép toán | Qui tắc kết hợp thực hiện |
| 1 | Phép toán 1 ngôi hậu tố (postfix) | [] . (tham\_so) exp++ exp-- | Từ trái qua phải |
| 2 | Phép toán 1 ngôi tiền tố (prefix) | ++exp --exp +exp - exp ~ ! | Từ phải qua trái |
| 3 | Tạo lập đối tượng, ép kiểu | new() (type) | Từ phải qua trái |
| 4 | Loại phép nhân | \* / % | Từ trái qua phải |
| 5 | Loại phép cộng | + - | Từ trái qua phải |
| 6 | Chuyển dịch | << >> >> | Từ trái qua phải |
| 7 | Phép toán quan hệ | < <= > >= instanceof | Từ trái qua phải |
| 8 | Phép so sánh đẳng thức | == != | Từ trái qua phải |
| 9 | Phép AND | & | Từ trái qua phải |
| 10 | Phép XOR | ^ | Từ trái qua phải |
| 11 | Phép OR | | | Từ trái qua phải |
| 12 | Phép AND | && | Từ trái qua phải |
| 13 | Phép hoặc (OR) logic | || | Từ trái qua phải |
| 14 | Phép toán điều kiện | ?: | Từ trái qua phải |
| 15 | Các phép gán | = += -= \*= /= %= <<= >>= >>>= &= ^= |= | Từ phải qua trái |

### 3.1.5. Các phép toán trong Java

* Phép chia nguyên (2 toán hạng đều là kiểu nguyên) đòi hỏi số chia phải khác 0.
* Phép chia số thực (ít nhất một toán hạng kiểu số thực) cho phép chia cho 0 và kết quả phép chia cho 0.0 là INF (số lớn vô cùng) hoặc -INF (số âm vô cùng), hai hằng đặc biệt trong Java.
* Trong Java, phép chia lấy số dư % thực hiện được cả đối với số thực. (float m = 11.5 % 2.5; // Cho m là 1.5)
* Khi sử dụng các phép toán đơn nguyên đối với các đối số kiểu byte, short, hoặc char thì trước tiên toán hạng phải được tính rồi chuyển về kiểu int và kết quả là kiểu int.

*Ví dụ:*

*short h = 30; // OK: 30 kiểu int chuyển về short (mặc định đối với hằng nguyên)*

*h = h + 4; // Lỗi vì h + 4 cho kết quả kiểu int không thể gán cho h kiểu short phải ép kiểu: h = (short) (h+4);*

* Các hằng số thực trong Java được xem là giá trị (mặc định) kiểu *double*.

float t = 3.14; *// Lỗi vì không tương thích kiểu*

Vì thế, hoặc phải thông báo tường minh là số thực kiểu *float* (3.14F):

float t = 3.14F; hoặc float t = (float)3.14;

* Các phép chuyển dịch <<, >>, >>> thực hiện dịch dạng biểu diễn nhị phân của toán hạng thứ nhất sang trái, sang phải số lần bằng giá trị số nguyên của toán hạng thứ 2.
* Biểu diễn nhị phân của các số nguyên: Java sử dụng *phần bù 2 để lưu trữ các giá trị nguyên.*

Cho trước giá trị nguyên dương, ví dụ 41. Biểu diễn nhị phân của -41 được tính như  
sau:

|  |  |  |
| --- | --- | --- |
|  | Biểu diễn nhị phân | Giá trị thập phân |
| Cho trước giá trị | 00101001 | 41 |
| Lấy phần bù 1 | 11010110 |  |
| Cộng thêm 1 | 00000001 |  |
| Kết quả là phần bù | 211010111 | -41 |

**Phép dịch trái**: <<

a << n Dịch tất cả các bit của a sang trái n lần, điền số 0 vào bên phải. (a << n tương đương với a \* 2n ).

**Phép dịch phải và điền bit dấu**: >>

a >> n Dịch tất cả các bit của a sang phải n lần, điền bit dấu vào bên trái.

**Phép dịch phải và điền bit 0**: >>>

a >>> n Dịch tất cả các bit của a sang phải n lần, điền 0 vào bên trái. A >>> n tương đương với a\* 1/2n

* Giá trị của đối số bên phải ( ở trên là n) luôn là số nguyên (dương) do vậy đối số bên trái ( ở trên là a) nếu là *byte* hoặc *short* thì phải đổi sang kiểu *int*. Kết quả của các phép chuyển dịch vì vậy sẽ luôn là *int* (hoặc là kiểu *long* nếu đối số thứ nhất là *long*).
* Các phép gán số học mở rộng tương đương ngữ nghĩa với lệnh sau:

<var> = (<type>) (<var> <op> (<exp>));

Trong đó <type> là các kiểu số và <op> là phép toán số học +, -, \*, /, %.

* So sánh đẳng thức trên các giá trị kiểu nguyên thủy: ==, !=

Cho trước hai toán hạng a, b có kiểu dữ liệu nguyên thủy.

a == b (a, b có giá trị kiểu nguyên thủy bằng nhau thì cho kết quả true, ngược lại cho false).

a != b (a, b có các giá trị kiểu nguyên thủy không bằng nhau thì cho kết quả true, ngược lại cho false).

* So sánh đẳng thức trên các tham chiếu đối tượng: ==, !=

Cho trước r và s là hai biến tham chiếu.

r == s Cho giá trị true nếu r, s cùng tham chiếu tới cùng một trị (đối tượng), ngược lại sẽ cho giá trị false.

r != s Cho giá trị true nếu r, s không cùng tham chiếu tới cùng một trị (đối tượng), ngược lại sẽ cho giá trị false.

**Chuyển đổi kiểu**

* + **ép kiểu**

Qui tắc ép kiểu có dạng: **(<type>) <exp>**

Chuyển kết quả tính toán của biểu thức <exp> sang kiểu được ép là <type>.

Ví dụ: float f = (float) 100.15D; // Chuyển số 100.15 dạng kiểu *double* sang *float*

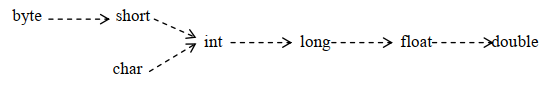
**Lưu ý:**

Không cho phép chuyển đổi giữa các kiểu nguyên thủy với kiểu tham chiếu, ví dụ kiểu *double* không thể ép sang các kiểu lớp như *HocSinh* được.

Kiểu giá trị *boolean* (logic) không thể chuyển sang các kiểu dữ liệu số và ngược lại.

* + **Mở rộng và thu hẹp kiểu**

Giá trị của kiểu h p hơn (chiếm số byte ít hơn) có thể được chuyển sang những kiểu rộng hơn (chiếm số byte nhiều hơn) mà không tổn thất thông tin. Cách chuyển kiểu đó được gọi là mở rộng kiểu.



Ví dụ: char c = “A”;

int k = c; // mở rộng kiểu *char* sang kiểu *int* (mặc định)

Chuyển đổi kiểu theo chiều ngược lại, từ kiểu rộng về kiểu h p hơn được gọi là *thu hẹp kiểu*. Lưu ý là *thu hẹp kiểu có thể dẫn tới mất thông tin*.

## 3.2. Các câu lệnh nhập xuất dữ liệu cơ bản qua các thiết bị vào ra chuẩn

### 3.2.1. Nhập dữ liệu từ bàn phím

Java là một ngôn ngữ lập trình thuần hướng đối tượng, nên cách nhập dữ liệu từ bàn phím cũng phải sử dụng theo hướng đổi tượng. Tức là ta phải khai báo một đối tượng, sau đó gọi hàm nhập, rồi gán giá trị cho biến.

**Scanner** là một lớp trong thư viện Java.util giúp bạn nhập và lưu data từ bàn phím. Ngoài scanner, còn có nhiều lớp khác nữa, nhưng đây là lớp được sử dụng nhiều nhất.

Các phương thức thường dùng trong lớp **Scanner**:

Tên phương thức Tác dụng

nextBoolean Nhập vào kiểu Boolean ( true – false) từ bàn phím

nextByte Nhập vào kiểu dữ liệu Byte

nextShort Nhập vào kiểu Short ( số nguyên từ -32768 đến 32767)

nextInt Nhập vào kiểu số nguyên từ bàn phím

nextFloat Nhập vào kiểu số thực

nextDouble Nhập vào kiểu Double

nextLine Nhập vào kiểu String

nextLong Nhập vào số nguyên lớn

**Cú pháp**

Scanner ip = new Scanner(System.in);

ip.nextInt(); *// Gọi hàm nhập số nguyên. Các hàm khác cú pháp tương tự*

### 3.2.2. Xuất dữ liệu ra màn hình

**Cú pháp**

System.out.println(“Noi dung can in” + biểu thức);

System.out.print(“Noi dung can in” + biểu thức);

System.out.printf(“Noi dung can in” + biểu thức);

* Với *print*: Xuất kết quả ra màn hình nhưng con trỏ chuột không xuống dòng.
* Với *println*: Xuất kết quả ra màn hình đồng thời con trỏ chuột nhảy xuống dòng tiếp theo.
* Với *printf*: Xuất ra màn hình kết quả đồng thời có thể định dạng được kết quả đó nhờ vào các đối số thích hợp.

System.out.printf(local, format, arguments1, arguments2,…, argumentsn);

Trong đó:

* *local*: Nếu khác null sẽ được tự động định dạng theo khu vực.
* *format*: Quy định chuẩn định dạng đầu ra cho các đối số
* Các *argument*: Đối số cần định dạng.

Các bộ định dạng có sẵn trong printf:

%c: Ký tự

%d: Số thập phân (số nguyên) (cơ số 10)

%e: Dấu phẩy động theo cấp số nhân

%f: Dấu phẩy động

%i: Số nguyên (cơ sở 10)

%o: Số bát phân (cơ sở 8)

%s: Chuỗi

%u: Số thập phân (số nguyên) không dấu

%x: Số trong hệ thập lục phân (cơ sở 16)

%t: Định dạng ngày / giờ

%%: Dấu phần trăm

\%: Dấu phần trăm

## 3.3. Các câu lệnh điều khiển

### 3.3.1. Câu lệnh rẽ nhánh

**Câu lệnh if**

*if ( <Biểu thức điều kiện>)*

*<lệnh 1>*

*[else <lệnh 2> ]*

Nếu biểu thức điều kiện đúng thì thực hiện lệnh 1 nếu không thì thực hiện lệnh 2 (nếu có *else*)

Ví dụ:

*public class Test {*

*public static void main(String[] args) {*

*int number = 13;*

*if (number % 2 == 0) {*

*System.out.println("Số " + number + " là số chẵn.");*

*} else {*

*System.out.println("Số " + number + " là số lẻ.");*

*}*

*}*

*}*

Kết quả:

*Số 13 là số lẻ.*

**Câu lệnh switch**

*switch (<Biểu thức nguyên>) {*

*case nhan\_1: <Câu lệnh 1>*

*case nhan\_2: <Câu lệnh 2>*

*. . .*

*case nhan\_n: <Câu lệnh n>*

*[default: <Câu lệnh >]*

*}*

*switch* sẽ kiểm tra giá trị của *Biểu thức nguyên* và so sánh với từng giá trị *nhan1, nhan2, …*, lần lượt từ trên xuống dưới, mỗi giá trị cần so sánh gọi là một *case*. Khi một trường hợp đúng (true), khối lệnh ở trong *case* đó sẽ được thực thi.

Nếu tất cả các trường hợp đều sai (tức là các *nhan* ở *case* không bằng với *Biểu thức nguyên*), thì câu lệnh ở trong *default* sẽ được thực thi.

Ví dụ:

*public class SwitchExample {*

*public static void main(String[] args) {*

*int number = 20;*

*switch (number) {*

*case 10:*

*System.out.println("10");*

*break;*

*case 20:*

*System.out.println("20");*

*break;*

*case 30:*

*System.out.println("30");*

*break;*

*default:*

*System.out.println("Not in 10, 20 or 30");*

*}*

*}*

*}*

Kết quả:

*20*

**Lưu ý**:

* Khối *default* là không bắt buộc có ở cấu trúc *switch case* trong Java
* Trong một *switch* có thể có nhiều *case*
* Nếu không gặp lệnh *break* trong khối lệnh này, thì chương trình sẽ thực hiện tiếp các *case* bên dưới cho tới khi nó gặp lệnh *break* thì nó sẽ thoát ra khỏi *switch*, hoặc cho đến khi thực hiện hết câu lệnh.

Ví dụ: mệnh đề *Switch-case* khi không sử dụng *'break'*

*public class SwitchExample2 {*

*public static void main(String[] args) {*

*int number = 20;*

*switch (number) {*

*case 10:*

*System.out.println("10");*

*case 20:*

*System.out.println("20");*

*case 30:*

*System.out.println("30");*

*default:*

*System.out.println("Not in 10, 20 or 30");*

*}*

*}*

*}*

Kết quả:

*20*

*30*

*Not in 10, 20 or 30*

## Bài tập cuối chương

3.1. Câu nào sau đây là sai?

A. /\*\* chú thích \*/

B. /\* chú thích \*/

C. /\* chú thích

D. // chú thích

3.2. Phương thức nextLine() thuộc lớp nào ?

A. String

B. Scanner

C. Integer

D. System

3.3. Kiểu dữ liệu nào trong Java chứa giá trị bao gồm cả chữ và số?

A. int

B. byte

C. char

D. String

3.4. Đâu là khai báo biến hợp lệ?

A. theOne

B. the One

C. 1the\_One

D. $the One

3.5. Biểu thức nào có giá trị khác các biểu thức còn lại trong các biểu thức sau? Cho x=true thuộc kiểu boolean.

A. true

B. x==true;

C. 1==1

D. !x

3.6. Có bao nhiêu kiểu dữ liệu cơ sở trong Java?

A. 7

B. 8

C. 9

D. 5

3.7. Có bao nhiêu kiểu số nguyên trong Java?

A. 1

B. 2

C. 3

D. 4

3.8. Khối lệnh sau có kết quả bao nhiêu?

String greeting = “Hello”;

int k = greeting.length();

System.out.print(k);

A. 4

B. 5

C. 6

D. 7

## Bài 5: Các câu lệnh điều khiển (Số tiết: 03 tiết)

### 3.3.2. Các câu lệnh lặp

**\* Vòng lặp for**

*for (<Biểu thức bắt đầu>; <Điều kiện lặp>; <Biểu thức gia tăng>)*

*<Khối lệnh>*

Trong đó:

* *Biểu thức bắt đầu:* được thực thi đầu tiên, và chỉ một lần. Bước này cho phép khai báo và khởi tạo bất kỳ biến điều khiển vòng lặp. Tất cả các biến được khai báo trong *<Biểu thức bắt đầu>* đều là cục bộ trong khối thân của chu trình *for*.
* *Điều kiện lặp:* Nếu nó là *true*, phần *Khối lệnh* được thực thi. Nếu nó là *false*, phần *Khối lệnh* không thực thi và luồng điều khiển nhảy tới lệnh tiếp theo sau vòng lặp for.
* *Biểu thức gia tăng:* Thay đổi (tăng hoặc giảm) giá trị của biến sau mỗi lần lặp

Ví dụ:

*public class Main {*

*public static void main(String args[]) {*

*for (int i = 0; i < 5; i++) {*

*System.out.println("Vị trí thứ: " + i);*

*}*

*}*

*}*

Kết quả:

*Vị trí thứ 0*

*Vị trí thứ 1*

*Vị trí thứ 2*

*Vị trí thứ 3*

*Vị trí thứ 4*

**Lưu ý:**

* Các thành phần của chu trình for là tùy chọn. Một trong <Biểu thức bắt đầu>, <Biểu thức gia tăng>, <Điều kiện kết thúc> có thể trống. Trường hợp <Điều kiện kết thúc> là trống thì điều kiện lặp của chu trình được xem là true.
* for (; ; ) được sử dụng để xây dựng chu trình lặp vô điều kiện.

**\* Vòng lặp while**

while (<điều kiện>)

<khối lệnh>

Trong đó:

* *điều kiện:* Biểu thức *Boolean*, nó trả về giá trị *True* hoặc *False*. Vòng lặp sẽ tiếp tục cho đến khi nào giá trị *True* được trả về
* *khối lệnh:* Các câu lệnh được thực hiện nếu *điều kiện* nhận giá trị True

Ví dụ:

*public class Main {*

*public static void main(String args[]) {*

*int i=0;*

*while(i<5){*

*System.out.println("Vị trí thứ "+i);*

*i++;*

*}*

*}*

*}*

Vòng lặp *while* kiểm tra điều kiện nếu *i*<5 thì in ra vị trí *i*. Mỗi lần lặp, chúng ta tăng *i* thêm 1 đơn vi, do đó, sau 5 lần lặp, *i* có giá trị là 5 và vòng lặp sẽ dừng lại.

Kết quả:

*Vị trí thứ 0*

*Vị trí thứ 1*

*Vị trí thứ 2*

*Vị trí thứ 3*

*Vị trí thứ 4*

**\* Vòng lặp do...while**

Vòng lặp do … while là tương tự như Vòng lặp while, ngoại trừ rằng phần thân của vòng lặp do…while được bảo đảm thực thi ít nhất một lần. Nói cách khác, vòng lặp do ... while thực hiện phần thân vòng lặp trước khi kiểm tra điều kiện.

*Do{*

*<khối lệnh>*

*}while (<điều kiện>)*

Trong đó:

* *điều kiện:* Biểu thức *Boolean*, nó trả về giá trị *True* hoặc *False*. Vòng lặp sẽ tiếp tục cho đến khi nào giá trị *True* được trả về
* *khối lệnh:* Các câu lệnh được thực hiện 1 lần sau đó kiểm tra điều kiện, nếu đúng thì thực hiện tiếp và ngược lại.

Ví dụ:

*public class Main {*

*public static void main(String args[]) {*

*int i = 0;*

*do {*

*System.out.println("Vị trí thứ : " + i);*

*i++;*

*} while (i < 5);*

*}*

*}*

Kết quả:

*Vị trí thứ 0*

*Vị trí thứ 1*

*Vị trí thứ 2*

*Vị trí thứ 3*

*Vị trí thứ 4*

### 3.3.3. Các lệnh chuyển vị trong java

**Câu lệnh break:**

Câu lệnh *break* được sử dụng trong các khối được gắn nhãn, trong các chu trình

lặp (*for, while, do-while*) và câu lệnh *switch* để chuyển điều khiển thực hiện chương

trình ra khỏi khối trong cùng chứa nó.

* Trong gắn nhãn: phần còn lại của khối bị cho qua
* Trong chu trình lặp: khi gặp lệnh break thì phần còn lại của thân chu trình được bỏ qua và kết thúc chu trình đó,
* Trong lệnh switch: phần còn lại của lệnh switch bị bỏ qua và tiếp tục thực hiện lệnh đứng sau lệnh switch đó.

Ví dụ:

*public class BreakExample {*

*public static void main(String[] args) {*

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

*if (i == 5) {*

*break;*

*}*

*System.out.println(i);*

*}*

*}*

*}*

Kết quả:

*1*

*2*

*3*

*4*

Ví dụ sử dụng break với vòng lặp bên trong vòng lặp for khác:

*public class BreakExample2 {*

*public static void main(String[] args) {*

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

*for (int j = 1; j <= 3; j++) {*

*if (i == 2 && j == 2) {*

*break;*

*}*

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

*}*

*}*

*}*

*}*

Kết quả:

*1 1*

*1 2*

*1 3*

*2 1*

*3 1*

*3 2*

*3 3*

**Câu lệnh continue**

Câu lệnh continue được sử dụng trong các chu trình lặp for, while, do-while để dừng sự thực hiện của lần lặp hiện thời và bắt đầu lặp lại lần tiếp theo nếu điều kiện lặp còn thoả mãn (*true*).

Ví dụ sử dụng Continue trong java với vòng lặp for:

*public class ContinueExample {*

*public static void main(String[] args) {*

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

*if (i == 5) {*

*continue;*

*}*

*// Khi i == 5 thì không in i = 5 ra màn hình*

*System.out.println(i);*

*}*

*}*

*}*

Kết quả:

*1*

*2*

*3*

*4*

*6*

*7*

*8*

*9*

*10*

Ví dụ sử dụng Continue với vòng lặp bên trong vòng lặp for khác:

*public class ContinueExample2 {*

*public static void main(String[] args) {*

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

*for (int j = 1; j <= 3; j++) {*

*if (i == 2 && j == 2) {*

*continue;*

*}*

*// Không in trường hợp i=2 và j=2 ra màn hình*

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

*}*

*}*

*}*

*}*

Kết quả:

*1 1*

*1 2*

*1 3*

*2 1*

*2 3*

*3 1*

*3 2*

*3 3*

**Câu lệnh return**

Câu lệnh return được sử dụng để kết thúc thực hiện của hàm hiện thời và chuyển điều khiển chương trình về lại sau lời gọi hàm đó.

|  |  |  |
| --- | --- | --- |
| Dạng lệnh return | Hàm khai báo void | Hàm có kiểu trả lại khác void |
| return | Tùy chọn | Không cho phép |
| return <Biểu thức> | Không cho phép | Bắt buộc |

## 3.4. Ngoại lệ và xử lý ngoại lệ

Mọi đoạn chương trình đều tiềm ẩn khả năng sinh lỗi. Lỗi chủ quan: do lập trình sai, lỗi khách quan: do dữ liệu, do trạng thái của hệ thống.

Ngoại lệ: các trường hợp hoạt động không bình thường của chương trình ví dụ: đọc 1 file không tồn tại, lỗi chia cho 0 với số nguyên, truy cập vượt quá phạm vi xác định của mảng vv... Xử lý ngoại lệ như thế nào? Làm thế nào để có thể tiếp tục thực hiện?

Cách xử lý lỗi truyền thống: Cài đặt mã xử lý tại nơi phát sinh ra lỗi, làm cho chương trình trở nên khó hiểu, không phải lúc nào cũng đầy đủ thông tin để xử lý, không nhất thiết phải xử lý, truyền trạng thái lên mức trên, thông qua tham số, giá trị trả lại hoặc biến tổng thể (flag). Dễ nhầm, khó kiểm soát được hết các trường hợp, lỗi số học, lỗi bộ nhớ,… Lập trình viên thường quên không xử lý lỗi do bản chất con người, thiếu kinh nghiệm, cố tình bỏ qua.

Ví dụ, việc chia cho 0 sẽ tạo một lỗi trong chương trình hoặc khi xét thao tác nhập xuất (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 lại tập tin. Lúc đó tập tin dễ bị hư hại và các nguồn tài nguyên được cấp phát cho tập tin không được thu hồi lại cho hệ thống.

Ngôn ngữ Java cung cơ chế tự động dùng cho bắt và xử lý ngoại lệ. Có hai cách để xử lý ngoại lệ: Bắt, xử lý ngoại lệ và *ném* ngoại lệ (chuyển hướng hay ủy quyền xử lý ngoại lệ đến một vị trí khác của chương trình).

**Ưu điểm của bắt ngoại lệ**

- Dễ sử dụng

- Dễ dàng chuyển điều khiển đến nơi có khả năng xử lý ngoại lệ

- Có thể ném nhiều loại ngoại lệ

- Tách xử lý ngoại lệ khỏi thuật toán

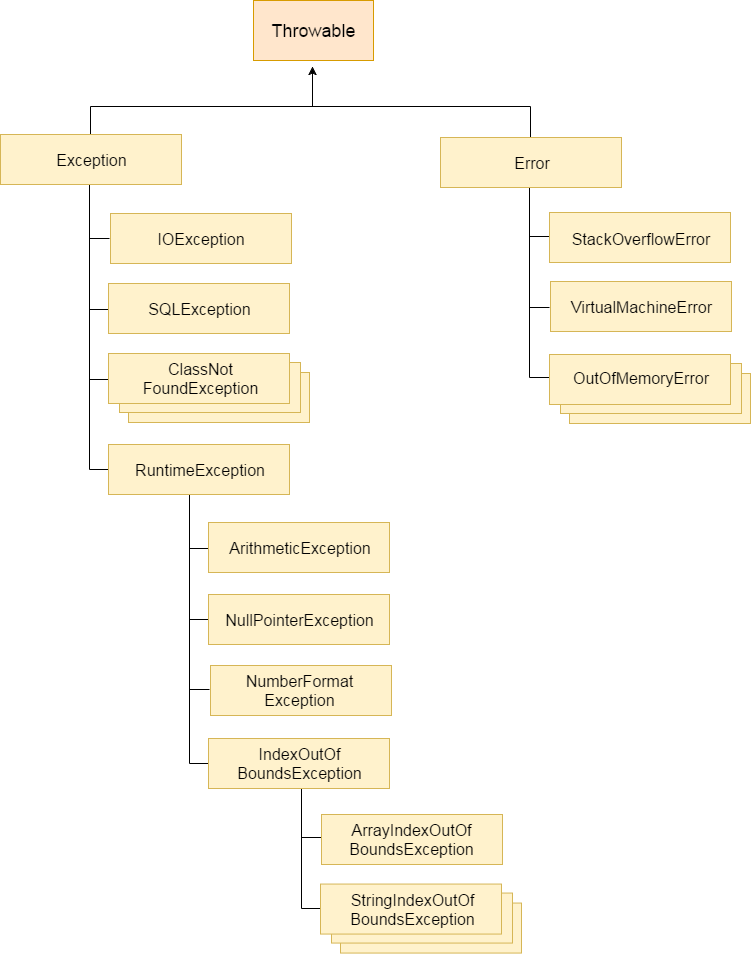
- Tách mã xử lý

- Sử dụng cú pháp khác

- Không bỏ sót ngoại lệ (ném tự động)

- Làm chương trình dễ đọc hơn, an toàn hơn

### 3.4.1. Cấu trúc phân cấp của các lớp xử lý ngoại lệ



Hình 2.5 Ngoại lệ

|  |  |
| --- | --- |
| **Ngoại lệ** | **Lớp cha của thứ tự phân cấp ngoại lệ** |
| RuntimeException | Lớp cơ sở cho nhiều ngoại lệ java.lang |
| ArthmeticException | Trạng thái lỗi về số, ví dụ như „chia cho 0‟ |
| IllegalAccessException | Lớp không thể truy cập |
| IllegalArgumentException | Phương thức nhận một đối số không hợp lệ |
| ArrayIndexOutOfBoundsExeption | Kích thước của mảng lớn hơn 0 hay lớn hơn kích thước thật sự của mảng |
| NullPointerException | Khi muốn truy cập đối tượng null |
| SecurityException | Việc thiết lập cơ chế bảo mật không được hoạt động |
| ClassNotFoundException | Không thể nạp lớp yêu cầu |
| NumberFormatException | Việc chuyển đối không thành công từ chuỗi sang số thực |
| AWTException | Ngoại lệ về AWT |
| IOException | Lớp cha của các ngoại lệ I/O |
| FileNotFoundException | Không thể định vị tập tin |
| EOFException | Kết thúc một tập tin |
| NoSuchMethodException | Phương thức yêu cầu không tồn tại |
| InterruptedException | Khi một luồng bị ngắt |

Bảng. 2.1 Ngoại lệ

### 3.4.2. Câu lệnh try, catch và finally

*try { // Khối try*

*<Các câu lệnh> }*

*catch (<Kiểu ngoại lệ 1> <Tham biến 1>) { // Khối catch*

*<Các câu lệnh xử lý khi xuất hiện kiểu ngoại lệ 1> }*

*. . .*

*catch (<Kiểu ngoại lệ n> <Tham biến n>) { // Khối catch*

*<Các câu lệnh xử lý khi xuất hiện kiểu ngoại lệ n>*

*}*

*finally { < Các câu lệnh phải thực hiện đến cùng>// Khối finally}*

*Trong đó:*

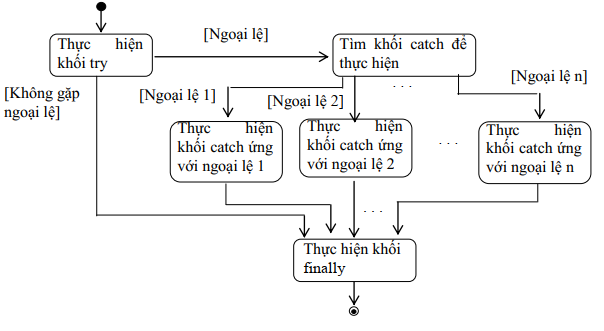
*- <Các câu lệnh> là các câu lệnh có khả năng gây ra ngoại lệ*

*-<Kiểu ngoại lệ> là các kiểu ngoại lệ tương ứng.*

*<tham biến i> là đối tượng mang thông tin về ngoại lệ*

Các ngoại lệ sẽ được cho qua trong quá trình thực hiện khối *try* và sẽ bị tóm lại để xử lý ở các khối *catch* tương ứng nếu có. Khối *finally* phải thực hiện đến cùng, bất luận có gặp phải ngoại lệ hay không.

Hoạt động của các khối trên được minh hoạ như sau:



Hình 2.6. Try .... catch

**Khối try**

Một chương trình cũng có thể chứa các khối *try* lồng nhau. Khi sử dụng các “*try*” lồng nhau, khối *try* bên trong được thi hành đầu tiên. Bất kỳ ngoại lệ nào bị chặn trong khối *try* sẽ bị bắt giữ trong các khối *catch* theo sau. Nếu khối *catch* thích hợp không được tìm thấy thì các khối *catch* của các khối *try* bên ngoài sẽ được xem xét. Nếu không, Java Runtime Environment xử lý các ngoại lệ.

**Khối catch**

Khối này chỉ được sử dụng để xử lý ngoại lệ nếu có. Khi một khối *catch* được thực hiện thì các khối *catch* còn lại sẽ bị bỏ qua.

Có thể bắt nhiều loại ngoại lệ khác nhau bằng cách sử dụng nhiều khối lệnh *catch* đặt kế tiếp

Khối lệnh *catch* sau không thể bắt ngoại lệ là lớp dẫn xuất của ngoại lệ được bắt trong khối lệnh *catch* trước.

**Khối finally**

Khi một ngoại lệ xuất hiện, phương thức đang được thực thi có thể bị dừng mà không được thi hành toàn vẹn. Nếu điều này xảy ra, thì các đoạn mã sẽ không bao giờ được gọi. Khối *finally* thực hiện tất cả các việc thu dọn khi một ngoại lệ xảy ra. Khối *finally* bảo đảm lúc nào cũng được thực thi, bất chấp có ngoại lệ xảy ra hay không. Khối *finally* là tuỳ ý, không bắt buộc.

Khối *finally* có thể dùng để:

- Đóng tập tin.

- Đóng lại bộ kết quả ( sử dụng trong chương trình cơ sở dữ liệu).

- Đóng lại các kết nối được tạo trong cơ sở dữ liệu.

Ví dụ minh họa cách sử dụng các khối *try* và *catch*.

class TryClass{

public static void main(String args[])

{

int demo=0;

try { System.out.println(20/demo); }

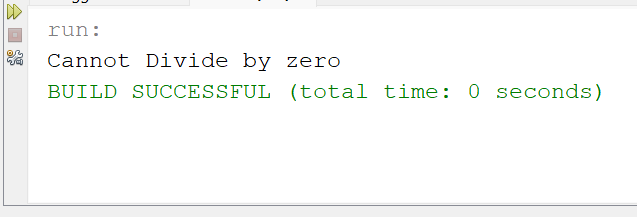
catch(ArithmeticException a) {

System.out.println(“Cannot Divide by zero”); }

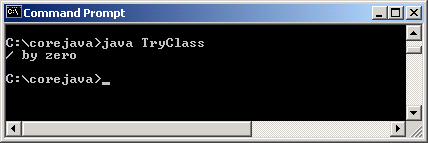
}

}

Kết quả:



*a* được sử dụng như một đối tượng của Arithmaticexception để in các chi tiết về các toán tử ngoại lệ mà hệ thống cung cấp. Nếu bạn thay thế lệnh *System.out.println* của khối *catch* bằng lệnh *System.out.println(a.getMessage())* thì kết quả của chương trình sẽ như sau:



### 3.4.3. Ném ngoại lệ bằng lệnh ‘throw’

Để tạm thời bỏ qua ngoại lệ chúng ta có thể sử dụng câu lệnh *throw*. Đoạn lệnh sau chỉ ra cách sử dụng của lệnh *throw*:

*try{ if (flag<0)*

*{throw new MyException();} // user-defined*

*}*

Ví dụ:

if (0==denominator) {

throw new Exception();

} else res = nominator / denominator;

Giả sử phương thức *x()* gọi phương thức *y()*. Phương thức *y()* chặn một ngoại lệ không được xử lý. Phương thức gọi *x()* nên khai báo việc chặn cùng một ngoại lệ với phương thức được gọi *y()*. Ta nên khai báo khối *try catch* trong phương thức *x()* để đảm bảo rằng ngoại lệ không được truyền cho các phương thức mà gọi phương thức này.

Phương thức được định nghĩa lại tại lớp dẫn xuất có thể không ném ngoại lệ. Nếu ném ngoại lệ, chỉ có thể ném ngoại lệ giống tại phương thức của lớp cơ sở hoặc ngoại lệ là lớp dẫn xuất của ngoại lệ được ném tại phương thức của lớp cơ sở.

Ví dụ:

*class A {public void methodA() throws RuntimeException {}*

*}*

*class B extends A {*

*public void methodA() throws ArithmeticException {}*

*}*

*class C extends A {*

*public void methodA() throws Exception { }*

*}*

*class D extends A {*

*public void methodA() { }*

*}*

*A a = new B();*

*try { a.methodA();}*

*catch (RuntimeException e) { ...}*

Ném lại ngoại lệ: Sau khi bắt ngoại lệ, nếu thấy cần thiết chúng ta có thể ném lại chính ngoại lệ vừa bắt được để cho chương trình mức trên tiếp tục xử lý.

*try {...}*

*catch (Exception e) {*

*System.out.println(e.getMessage());*

*throw e;}*

### 3.4.4. Mệnh đề throws

Khi thiết kế các hàm thành phần, chúng ta có thể sử dụng mệnh đề throws để tạm thời cho qua ngoại lệ mà thực hiện một số công việc cần thiết khác. Chúng ta ném ngoại lệ khỏi phương thức khi:

- Không nhất thiết phải xử lý ngoại lệ trong phương thức

- Không đủ thông tin để xử lý

- Không đủ thẩm quyền

Định nghĩa hàm với mệnh đề *throws* có dạng:

<Thuộc tính của hàm> <tên hàm>(<Danh sách các tham biến>) throws <Danh sách các kiểu ngoại lệ> { /\* ... \*/}

Trong đó *<Danh sách các kiểu ngoại lệ>* là các lớp xử lý ngoại lệ được kế thừa từ lớp Exception và được phân tách bởi dấu ‘,’.

**Ném ngoại lệ từ main()**

– Nếu không có phương thức nào bắt ngoại lệ, ngoại lệ sẽ được truyền lên phương thức main() và được cần được xử lý tại đây.

– Nếu vẫn không muốn xử lý ngoại lệ, chúng ta có thể để ngoại lệ truyền lên mức điều khiển của máy ảo bằng cách khai báo main() ném ngoại lệ

– chương trình sẽ bị dừng và hệ thống sẽ in thông tin về ngoại lệ trên Console (printStackTrace())

### 3.4.5. Tự định nghĩa ngoại lệ

Chúng ta có thể tạo lớp ngoại lệ để phục vụ các mục đích riêng

• Lớp ngoại lệ mới phải kế thừa từ lớp Exception hoặc lớp dẫn xuất của lớp này.

• Có thể cung cấp hai constructor

• constructor nhận một tham số String và truyền tham số này cho phương thức khởi tạo của lớp cơ sở.

Ví dụ:

*class SimpleException extends Exception { }*

*class MyException extends Exception {*

*public MyException() {}*

*public MyException(String msg) {*

*super(msg);}*

*}*

Minh họa ngoại lệ được định nghĩa bởi người dùng *ArraySizeException*:

*class* ***ArraySizeException*** *extends NegativeArraySizeException*

*{ ArraySizeException() // constructor*

*{ super(“You have passed an illegal array size”);*

*}*

*}*

*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();*

*else*

*System.out.println(“The array size is ok.”);*

*array = new int[3];*

*for (int i=0; i<3; i++) array[i] = i+1;*

*}*

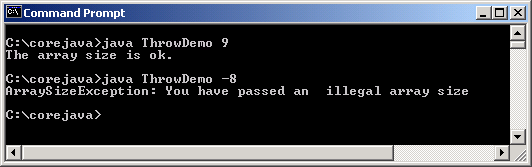
*public static void main(String arg[])*

*{ new ThrowDemo(Integer.parseInt(arg[0])); }*

*}*

Lớp được định nghĩa bởi người dùng *ArraySizeException* 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 chặn ngoại lệ *ArraySizeException* mà được chỉ ra bởi mệnh đề *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. Phương thức *call()* được bao quanh trong khối *trycatch*, là nơi mà giá trị của đối tượng được in ra. Phương thức *call()* cần được bao trong khối *try*, để cho khối *catch* tương ứng có thể in ra giá trị.

Kết xuất của chương trình:



### 3.4.6. Lần vết ngoại lệ StackTrace

Có thể sử dụng phương thức *printStackTrace()* để lần vết vị trí phát sinh ngoại lệ debug chương trình

*public class Test4 {*

*void methodA() throws Exception {*

*methodB();*

*throw new Exception();*

*}*

*void methodB() throws Exception {*

*methodC();*

*throw new Exception();}*

*void methodC() throws Exception {*

*throw new Exception();}*

*public static void main(String[] args) {*

*Test4 t = new Test4();*

*try { t.methodA(); }*

*catch(Exception e) {*

*e.printStackTrace();}*

*} }*

## Bài tập cuối chương

3.9. Kết quả in ra của đoạn mã bên dưới là gì? Chọn một câu trả lời

*public class Test {*

*public static int switchIt(int x)*

*{*

*int j = 1;*

*switch (x) {*

*case 1:*

*j++;*

*case 2:*

*j++;*

*case 3:*

*j++;*

*case 4:*

*j++;*

*default:*

*j++;*

*}*

*return j + x;*

*}*

*public static void main(String[] args) {*

*System.out.println("value = " + switchIt(4));*

*}*

*}*

A. value = 3

B. value = 4

C. value = 5

D. value = 7

3.10. Đâu là khai báo đúng về lớp Cat?

A. Class Cat{}

B. class public Cat(){}

C. class Cat{}

D. public Cat class{}

3.11. Giá trị của x sau khi thực hiện lệnh trên sẽ cho kết quả gì?

Khi thực hiện lệnh:

*A a = new A();*

*a.calTotal(a);*

*class A {*

*int x = 10;*

*public void calTotal(A a) {*

*a.x = 12;*

*System.out.println(a.x);*

*}*

*}*

A. 10

B. 12

C. 22

D. 44

# Chương IV: LỚP VÀ CÁC THÀNH PHẦN CỦA LỚP ĐỐI TƯỢNG

### Nội dung chính của chương

- Cấu trúc lớp và khai báo các thành phần của lớp

- Định nghĩa hàm thành phần và cơ chế nạp chồng, viết đè trong Java

- Các thuộc tính kiểm soát truy cập và kế thừa các thành phần của lớp

### Mục tiêu cần đạt được của chương

Giúp sinh viên sẽ nắm được cấu trúc, các thành phần của lớp đối tượng trong ngôn ngữ java, hiểu được phương pháp lập trình hướng đối tượng và thiết kế chương trình bằng việc sử dụng các lớp và đối tượng.

## Bài 6: Lớp đối tượng (Số tiết: 03 tiết)

## 4.1. Cấu trúc lớp và khai báo các thành phần của lớp

### 4.1.1. Định nghĩa lớp

Lớp là mô hình khái quát đặc tính của một họ thực thể vật lý hoặc trừu tượng trong chương trình.

Định nghĩa lớp là đặc tả một kiểu dữ liệu mới và mô tả cách cài đặt kiểu dữ liệu đó.

Mỗi đối tượng là một thể hiện của lớp.

**\* Cú pháp định nghĩa lớp**

*[<Phạm vi hoặc thuộc tính kiểm soát truy nhập>] class <Tên lớp> [extends <Tên lớp cha>] [implements <Tên giao diện>]*

*{ <Các thành phần của lớp> }*

**\* Giải thích:**

*Phạm vi*: Xác định phạm vi hoặc kiểm soát truy nhập:

* + - *absstract*: Không thể tạo ra một đối tượng cụ thể của lớp, lớp loại này là khung cơ sở nhất để dẫn xuất ra các lớp con cháu.
    - *public*: (công khai) Cho phép sử dụng mọi nơi trong hệ thống.
    - *protected*: cho phép truy cập trong gói và trong các lớp thuộc gói khác nhưng được kế thừa.
    - *private*: chỉ được sử dụng bên trong lớp, không cho phép truy cập và kế thừa đối với các lớp khác ở trong cùng gói và khác gói.
    - *final*: (hằng) nếu nó đứng trước khai báo lớp thì nó không được sử dụng để kế thừa cho các lớp con, nếu nó đứng trước khai báo biến thì nó là “hằng” (chỉ được gán giá trị một lần duy nhất), nếu nó đứng trước các phương thức thì nó sẽ không được phép viết đè ở các lớp con.
    - Nếu không khai báo ( public, protected, private) thì gọi là các thành phần “mặc định”, khi đó phạm vi là truy cập và kế thừa là trong gói chứa lớp đó.

*tên\_lớp\_cha*: Tên lớp cha được kế thừa

*DS\_tên\_các\_giao\_diện*: Tên các giao diện (được viết phân cách nhau bới dấu “,”) được phép sử dụng các phương thức và các hằng.

*class, extends, implements* là các từ khoá.

Những phần trong cặp [ và ] là tùy chọn( có thể có hoặc có thể không).

<Các thành phần của lớp> bao gồm:

* Các biến dữ liệu thành phần(attributes) : Xác định các thuộc tính (dữ liệu) của lớp, thông thường dùng để mô tả các đặc tính của đối tượng thuộc lớp đó
* Các toán tử tạo lập(constructor): Xác định cách khởi tạo một đối tượng thuộc lớp
* Các phương thức(method): Xác định hành vi của các đối tượng thuộc lớp

### 4.1.2. Các thuộc tính dữ liệu của lớp

Thuộc tính dữ liệu của lớp là các biến thành phần có thể thuộc kiểu nguyên thuỷ, kiểu tham chiếu.

**\* Khai báo:**

*<Phạm\_vi> <Kiểu> <tên\_biến>;*

*<Phạm\_vi> <Kiểu> <tên\_biến> = <Giá\_trị\_khởi\_tạo>;*

*Phạm vi*: xác định phạm vi truy cập của các đối tượng đến hàm. có thể chọn một trong các phạm vi:

• *static*: Khi một thuộc tính được khao báo static thì khi đó thuộc tính đó là “sở hữu” của tất cả các đối tượng của lớp đó chứ không phải nó là của riêng của từng đối tượng thuộc lớp đó.

• *public*: Có thể truy cập mọi nơi trong hệ thống (các đối tượng khác lớp

có thể truy cập được)

• *protected*: cho phép truy cập trong gói và trong các lớp thuộc gói khác nhưng được kế thừa.

• *private*: của riêng đối tượng chí bên trong đối tượng.

• *final*: Hằng (chỉ được khở tạo giá trị một lần duy nhất

• *volatile*: là biến được lưu trữ trên bộ nhớ chính, được dùng với luồng kết hợp với từ khóa synchronized.

## 4.2. Định nghĩa hàm thành phần và cơ chế nạp chồng, viết đè trong Java

### 4.2.1. Các hàm thành phần của lớp

Hành vi của các đối tượng thuộc một lớp được xác định bởi các hàm thành phần của lớp đó, hành vi thể hiện “vai trò” của đối tượng đó trong hệ thống.

Là các hàm, thủ tục thực hiện các phép xử lý trên các biến thành phần.

**\* Khai báo**

*<Phạm\_vi> <Kiểu\_giá\_tri> <tên\_hàm>(<danh sách các tham số hình thức>) <các\_mệnh\_đề\_throws> {//Thân hàm}*

*Phạm\_vi:*

• *public*: Cho phép truy cập mọi nơi trong hệ thống

• *private*: Chỉ được truy xuất (gọi) bên trong lớp chứa nó

• *protected*: lớp đó và lớp con của nó được truy xuất

• *final*: Không cho phép nạp chồng và ghi đè

• *static*: Nó có ý nghĩa tương tự như với thuộc tính *static*, thông thường chỉ những phương thức chung và không dùng cho một đối tượng cụ thể thì khi đó phương thức đó nó mới là phương thức tĩnh (*static*).

• *abstract*: Phương thức trừu tượng(là phương thức “mẫu”, nó chỉ nói lên là nó làm được gì mà không mô tả nó làm như thế nào), do đó nó không cài đặ nội dung ì ở lớp khai báo nó

• *native*: dùng cho phương thức khi cài đặt phụ thuộc môi trường trong một ngôn ngữ khác, như C hay hợp ngữ.

• *synchronized*: dùng trong lập trình đa luồng, nhằm ngăn các tác động của các phương thức khác của các luồng khác khi phương thức đang được thực hiện.

• Giá trị mặc định là *public*.

*<Kiểu trả lại>* có thể là kiểu nguyên thủy, kiểu lớp hoặc không có giá trị trả lại

(*kiểu void*).

*<Danh sách tham biến hình thức>* bao gồm dãy các tham biến (kiểu và tên)

phân cách với nhau bởi dấu phẩy.

**\* Sử dụng phương thức:**

- Nếu phương thức không phải là static thì cần phải tạo ra một đối tượng cụ thể và gọi phương thức thông qua tên đối tượng như cấu trúc sau:

**<tên đối tượng>.<tên\_hàm>(< danh sách các tham số hiện thời>)**

Khai báo đối tượng: <tên lớp> <tên đối tượng> ;

Khởi tạo giá trị thuộc tính cho đối tượng khi sử dụng hàm tạo lập(constructor) tự định nghĩa :

**<tên đối tượng> = new <tên lớp> (<danh sách tham số hiện thời >) ;**

Khai báo và khởi tạo đối tượng gíá trị:

**<tên lớp> <tên đối tượng> = new <tên lớp> (<danh sách tham số hiện thời >) ;**

- Nếu phương thức là static thì gọi thông qua tên lớp hoặc thông qua tên đối tượng (nên gọi các phương thức tĩnh thông qua tên lớp)

**<tên\_lớp>.<tên\_hàm\_tĩnh> (<danh sách tham số hiện thời >)**

- Các tham số trong các lời gọi hàm cung cấp cách thức trao đổi thông tin giữa đối tượng gửi và đối tượng nhận thông điệp.

* Số các tham biến của danh sách hiện thời phải bằng số các tham biến của danh sách tham biến hình thức.
* Kiểu của các tham biến hiện thời phải tương thích với kiểu của tham biến hình thức tương ứng.

|  |  |
| --- | --- |
| Kiểu của tham biến hiện thời | Giá trị được truyền |
| Các kiểu nguyên thủy  Kiểu lớp (class)  Kiểu mảng (array) | Giá trị kiểu nguyên thủy  Giá trị tham chiếu  Giá trị tham chiếu |

Ví dụ1: Tệp Demo.java

*public class Demo {*

*public static void main(String args[]) {*

*int i = 5;*

*if (i = = 0) {*

*return;*

*}*

*output(checkValue(args.length));*

*}*

*static void output(int value) {*

*System.out.println(value);*

*}*

*static int checkValue(int i) {*

*if (i > 3) {*

*return 1;*

*} else {*

*return 2;*

*}*

*}*

*}*

Ví dụ2: Tệp Demo2.java

*public class Demo2 {*

*public static void main(String args[]) {*

*int i = 5;*

*Demo2 d = new Demo2();*

*if (i = = 0) {*

*return;*

*}*

*output(d.checkValue(args.length));*

*}*

*static void output(int value) {*

*System.out.println(value);*

*}*

*public int checkValue(int i) {*

*if (i > 3) {*

*return 1;*

*} else {*

*return 2;*

*}*

*}*

*}*

**\* Phương thức và thuộc tính static:** Phương thức và thuộc tính độc lập với đối tượng, có thể sử dụng mà không cần thông qua đối tượng, Phương thức tĩnh: không sử dụng được thuộc tính và phương thức “thông thường” (non-static) mà nó chỉ được gọi đến các phương thức và thuộc tính tĩnh khác

### 4.2.2 Truyền tham số và gọi phương thức(hàm)

**Truyền các giá trị kiểu nguyên thủy**

Trong Java, khi gọi một phương thức và truyền các giá trị nguyên thủy cho phương thức, được gọi là truyền tham trị. Hiệc thay đổi giá trị chỉ có hiệu lực trong phương thức được gọi, không có hiệu lực bên ngoài phương thức.

Các tham số có kiểu dữ liệu nguyên thủy là byte, short, int, long, float, double, boolean, char.

Ví dụ: *Truyền các giá trị nguyên thủy*

*class KhachHang1 { // Lớp khách hàng*

*public static void main(String[] arg) {*

*HangSX banh = new HangSX(); // Tạo ra một đối tượng*

*int giaBan = 20;*

*double tien = banh.tinh(10, giaBan);*

*System.out.println("Gia ban: " + giaBan);// giaBan không đổi*

*System.out.println("Tien ban duoc : " + tien);*

*}*

*}  
// Lớp Hãng sản xuất*

*class HangSX {*

*double tinh(int num, double gia) {*

*gia = gia / 2;*

*return num \* gia;// Thay đổi gia nhưng không ảnh hưởng tới giaBan, số tiền bị thay đổi theo*

*}*

*}*

**Truyền các giá trị tham chiếu đối tượng.**

Trong Java, khi gọi một phương thức và truyền một tham chiếu cho phương thức, được gọi là truyền tham chiếu. Việc thay đổi giá trị của biến tham chiếu bên trong phương thức làm thay đổi giá trị của nó.

Trong Java, tất các phương thức có tham số là biến có kiểu là các lớp (class) đều là kiểu tham chiếu

//KhachHang2.java

*class KhachHang2 { // Lớp khách hàng*

*public static void main(String[] arg) {*

*Banh banhMoi = new Banh(); // Tạo ra một đối tượng (1)*

*System.out.println("Nhoi thit vao banh truoc khi nuong:" + banhMoi.thit);*

*nuong(banhMoi); // (2)*

*System.out.println("Thit cua banh sau khi nuong:" + banhMoi.thit);*

*}*

*public static void nuong(Banh banhNuong) { // (3)*

*banhNuong.thit = “Thit vit*

*”; // đổi nhân thành thịt vit*

*banhNuong = null; // (4)*

*}*

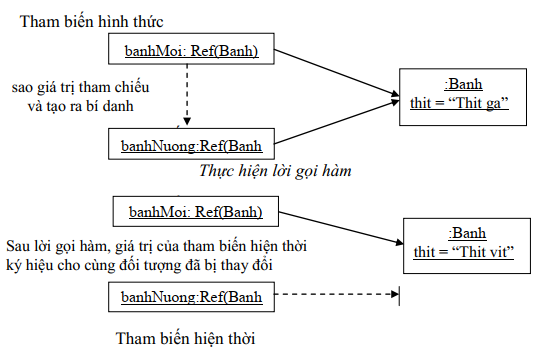
*}*

*class Banh { // Lớp Banh (5)*

*String thit = “Thit ga*

*”; // Qui định của hãng làm nhân bánh bằng thịt gà*

*}*



Truyền tham chiếu: đối với đối tượng thì nội dung của tham chiếu (LValue) được copy lên stack.

**Truyền đối số trong dòng lệnh**

*class Pass {*

*public static void main(String parameters[]) {*

*System.out.println("This is what the main method received");*

*System.out.println(parameters[0]);*

*System.out.println(parameters[1]); System.out.println(parameters[2]);*

*}*

*}*

Biên dịch chương trình: **javac PassArgumet.java**

Thực thi chương trình với dòng lệnh: **java PassArgument A 123 B1**

Sẽ thu được trên màn hình kết quả:

This is what the main method received

*A*

*123*

*B1*

**Các tham biến final**

Tham biến loại này được gọi là biến cuối “trắng”, nghĩa là nó không được khởi tạo giá trị (là trắng) cho đến khi nó được gán một trị nào đó và khi đã được gán trị thì giá trị đó là cuối cùng, không thay đổi được.

Ví dụ *Sử dụng tham biến final*

//*KhachHang3.java*

*class KhachHang3 { // Lớp khách hàng*

*public static void main(String[] arg) {*

*HangSX banh = new HangSX(); // Tạo ra 1 đối tượng*

*int giaBan = 20;*

*double tien = banh.tinh(10, giaBan);*

*System.out.println("Gia ban: " + giaBan);// giaBan không đổi*

*System.out.println("Tien ban duoc : " + tien);*

*}*

*}*

*// Lớp Hãng sản xuất*

*class HangSX {*

*double tinh(int num, final double gia) { // (1)*

*gia = gia / 2.0; // (2)*

*return num \* gia;// Thay đổi gia nhưng không ảnh hưởng tới giaBan, số tiền vẫn bị thay đổi theo*

*}*

*}*

### 4.2.3. Cơ chế nạp chồng, viết đè trong Java

**Nạp chồng**

Nạp chồng là cho phép sử dụng cùng tên hàm nhưng định nghĩa nhiều nội dung thực hiện khác nhau.

Các hàm nạp chồng phải khác nhau về danh sách tham biến (khác nhau về kiểu tham biến hoặc số lượng tham biến hay là có định danh khác nhau)

Ví dụ: Xây dựng lớp các hình học, trong đó xây dựng các hàm nạp chồng tính diện tích

*public class HinhHoc {*

*static int dienTich(int a) {*

*System.out.println("Dien tich hinh vuong: ");*

*return a \* a;*

*}*

*static int dienTich(int a, int b) {*

*System.out.println("Dien tich chu nhat: ");*

*return a \* b;*

*}*

*static double dienTich(int a, double b) {*

*System.out.println("Dien tich tron: ");*

*return a \* a \* b;*

*}*

*public static void main(String[] arg) {*

*int a = 3;*

*int b = 4;*

*double c = 3.14;*

*System.out.println(" " + dienTich(a));*

*System.out.println(" " + dienTich(a, c));*

*System.out.println(" " + dienTich(a, b));*

*}*

*}*

JDK API đã xây dựng rất nhiều hàm được nạp chồng.

Lưu ý là danh sách tham biến của các hàm nạp chồng phải khác nhau về số lượng hoặc về thứ tự các kiểu của các tham biến.

Ví dụ:

|  |  |
| --- | --- |
| **Hàm** | **Định danh** |
| public void methodA(int a, double b){/\* ...\*/} | methodA(int, double) |
| public int methodA(int a){return a} | methodA(int) |
| public int methodA(){return 1} // (3) | methodA() |
| public long methodA(double a,int b){return a\*b} | methodA(double, int) |
| public long methodA(int a, double b){return a} //**NoOK** | methodA(int, double) không đúng do định danh giống (1‟) |

**Viết chồng (viết đè)**

Một lớp con có thể viết đè, thay đổi nội dung thực hiện của những hàm thừa kế từ một lớp cha gọi là viết đè các hàm thành phần.

Khi những hàm này được gọi để thực thi đối với những đối tượng của lớp con thì nội dung thực hiện được định nghĩa ở lớp con sẽ được thực thi.

Ví dụ:

- Xây dựng lớp Meo có hàm thành phần speak()

- Xây dựng lớp MeoCon kế thừa lớp Meo và viết đè hàm speak()

- Gọi hàm speak() của lớp MeoCon

- Gọi hàm speak() của lớp Meo đối với đối tượng thuộc lớp MeoCon

|  |
| --- |
| *class Meo{*  *protected static void speak(){*  *System.out.print("Meo Meo!"); }*  *}*  *class MeoCon extends Meo{*  *protected static void speak(){*  *System.out.print("Miu Miu!"); }*  *public static void main(String []arg){*  *Meo meo=new Meo();*  *MeoCon meocon=new MeoCon();*  *MeoCon.speak();*  *System.out.println();*  *meo.speak();*  *System.out.println(); }*  *}* |

**\* Lưu ý:**

* Định nghĩa mới của hàm viết đè phải có cùng định danh (tên gọi và danh sách tham biến) và cùng kiểu trả lại giá trị.
* Định nghĩa mới của hàm viết đè trong lớp con chỉ có thể xác định tất cả hoặc tập con các lớp ngoại lệ được kể ra trong mệnh đề cho qua ngoại lệ
* Các hàm final, static không được phép viết đè.

Phân biệt rõ hai cơ chế viết đè và nạp chồng là khác nhau trong Java.

* Viết đè yêu cầu cùng định danh hàm (cùng tên gọi, cùng danh sách tham số) và cùng kiểu trả lại kết quả đã được định nghĩa tại lớp cha.
* Nạp chồng yêu cầu khác nhau về định danh, nhưng giống nhau về tên gọi của hàm, vì thế chúng sẽ khác nhau về số lượng, kiểu, hay thứ tự của các tham biến.
* Hàm có thể nạp chồng ở trong cùng lớp hoặc ở các lớp con cháu.
* Từ những lớp con khi muốn gọi tới các hàm ở lớp cha mà bị viết đè thì phải gọi qua toán tử đại diện cho lớp cha, đó là super(). Đối với hàm nạp chồng thì lại không cần như thế. Lời gọi hàm nạp chồng được xác định thông qua danh sách các đối số hiện thời sánh với đối số hình thức để xác định nội dung tương ứng.

### 4.2.4. Nạp chồng các toán tử tạo lập

Giống như các hàm thành phần, các toán tử tạo lập có thể nạp chồng với nhiều nội dung thực hiện khác nhau.

Khi tạo lập đối tượng sử dụng toán tử tạo lập nào thì toán tử đó được gọi.

Ví dụ:

*class BongDen{*

*// Biến thành phần (1)*

*private int soWatts;*

*private boolean batTat;*

*private String viTri;*

*// Định nghĩa toán tử tạo lập số 1 (2)*

*BongDen(){*

*this(40, true);*

*System.out.println(“Toán tử số 1”); }*

*// Định nghĩa toán tử tạo lập số 2 (3)*

*BongDen(int w, boolean s){*

*this(w, s, “XX”);*

*System.out.println(“Toán tử số 2”); }*

*//Định nghĩa toán tử tạo lập số 3 (4)*

*BongDen(int soWatts, boolean batTat, String viTri){*

*this.soWatts = soWatts;*

*this.batTat = batTat;*

*this.viTri = new String(viTri);*

*System.out.println(“Toán tử số 3”); }*

*}*

*class DenTuyp extends BongDen {*

*private int doDai;*

*private int mau;*

*DenTuyp(int leng, int colo){*

*this(leng,colo,100,true,“Chua biet”);*

*}*

*DenTuyp(int leng, int colo, int soWatt, boolean bt, String noi){*

*super(soWatt, bt, noi);*

*this.doDai = leng;*

*this.mau = colo;*

*}*

*}*

*public class NhaKho{*

*public static void main(String args[]){*

*System.out.println(“Tao ra bong đèn tuyp”);*

*DenTuyp d = new DenTuyp(20, 5);*

*}*

### 4.2.4. Cơ chế che bóng của các biến

Về nguyên tắc, một lớp con không được phép viết đè các biến thành phần của lớp cha, nhưng có thể che khuất chúng tương tự như biến cục bộ trong lớp con.

Ví dụ *Viết đè và nạp chồng các hàm thành phần*

*// KhachHang.java*

*import java.io.\*;*

*class Den {*

*protected String loaiHoaDon = “Hoa don nho: ”; // (1)*

*protected double docHoaDon(int giaDien) throws Exception { // (2)*

*double soGio = 10.0, hoaDonNho = giaDien \* soGio;*

*System.out.println(loaiHoaDon + hoaDonNho);*

*return hoaDonNho;*

*}*

*}*

*class DenTuyp extends Den {*

*public String loaiHoaDon =“Hoa don lon:”;// Bị che bóng (3)*

*public double docHoaDon(int giaDien) throws Exception { // Viết đè hàm (4)*

*double soGio = 100.0;*

*hoaDonLon = giaDien \* soGio;*

*System.out.println(loaiHoaDon + hoaDonLon);*

*return hoaDonLon;*

*}*

*public double docHoaDon() {*

*System.out.println(“Khong co hoa don!”);*

*return 0.0;*

*}*

*}*

*public class KhachHang {*

*public static void main(String args[]) throws Exception { // (6)*

*DenTuyp den1 = new DenTuyp(); // (7)*

*Den den2 = den1; // (8)*

*Den den3 = new Den(); // (9)*

*// Gọi các hàm đã viết đè*

*den1.docHoaDon(1000); // (10)*

*den2.docHoaDon(1000); // (11)*

*den3.docHoaDon(1000); // (12)*

*// Truy nhập tới các biến thành phần đã bị viết đè (bị che bóng)*

*System.out.println(den1.loaiHoaDon); // (13)*

*System.out.println(den2.loaiHoaDon); // (14)*

*System.out.println(den3.loaiHoaDon); // (15)*

*// Gọi các hàm nạp chồng*

*den1.docHoaDon();*

*}*

*}*

Kết quả thực hiện của chương trình KhachHang:

*Hóa đơn lớn: 100000.00*

*Hóa đơn lớn: 100000.00*

*Hóa đơn nhỏ: 10000.00*

*Hóa đơn lớn:*

*Hóa đơn nhỏ:*

*Hóa đơn nhỏ:*

*Không có Hóa đơn!*

## 4.3. Các thuộc tính kiểm soát truy nhập các thành phần của lớp

Khi xây dựng một lớp ta có thể hạn chế sự truy cập đến các thành viên của lớp, từ một đối tượng khác. Ta tóm tắt qua bảng sau:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Từ khoá | Truy cập trong chính lớp đó | Truy cập  trong lớp con cùng gói | Truy cập trong lớp con khác gói | Truy cập trong lớp khác cùng gói | Truy cập trong lớp khác khác gói |
| private | X | - | - | - | - |
| protected | X | X | X | X | - |
| public | X | X | X | X | X |
| default | X | X | - | X | - |

Trong bảng trên thì X thể hiện cho sự truy cập hợp lệ còn – thể hiện không thể truy cập vào thành phần này.

**\*Ví dụ:** Các thành phần private

Các thành viên private chỉ có thể sử dụng bên trong lớp, ta không thể truy cập các thành viên private từ bên ngoài lớp này.

*class Alpha {*

*private int iamprivate;*

*private void privateMethod() {*

*System.out.println("privateMethod");*

*}*

*}*

*class Beta {*

*void accessMethod() {*

*Alpha a = new Alpha();*

*a.iamprivate = 10;// không hợp lệ*

*a.privateMethod();// không hợp lệ }*

*}*

\* Ví dụ: Các thành phần public

Các thành viên public có thể truy cập từ bất cứ đâu, ta se xem ví dụ sau:

*Package Greek;*

*public class Alpha {*

*public int iampublic;*

*public void publicMethod() {*

*System.out.println("publicMethod");*

*}*

*}*

*package Roman;*

*import Greek.\*;*

*class Beta {*

*void accessMethod() {*

*Alpha a = new Alpha();*

*a.iampublic = 10;// hợp lệ*

*a.publicMethod();// hợp lệ*

*}*

*}*

\*Ví dụ các thành phần có mức truy xuất gói:

Khi ta khai báo các thành viên mà không sử dụng một trong các từ public, private, protected thì java mặc định thành viên đó có mức truy cập gói.

Ví dụ:

*package Greek;*

*class Alpha {*

*int iampackage;*

*void packageMethod() {*

*System.out.println("packageMethod");*

*}*

*}*

*package Greek;*

*class Beta {*

*void accessMethod() {*

*Alpha a = new Alpha();*

*a.iampackage = 10;// hợp lệ*

*a.packageMethod();// hợp lệ*

*}*

*}*

## Bài tập cuối chương

4.1. Đâu KHÔNG phải là cách thức để khởi tạo giá trị cho thuộc tính name có kiểu chuỗi của lớp Cat?

A. class Cat {String name = "noname";}

B. public class Cat{ String name; public Cat(){name = "noname";} }

C. public class Cat{ String name; public Cat(String x){name = x;} }

D. public class Cat{ String name; public Cat(){ String name = "noname"; } }

4.2. Đâu là khai báo đúng về lớp Cat?

A. Class Cat{}

B. class public Cat(){}

C. class Cat{}

D. public Cat class{}

4.3. Thứ tự các từ khóa public và static khi khai bao như thế nào?

A. public đứng trước static

B. static đứng trước public

C. Thứ tự bất kỳ nhưng thông thường public đứng trước

D. Tất cả đều sai.

4.4. Đoạn mã sau có lỗi biên dịch gì?

*class Person {*

*protected String name;*

*public int age;*

*};*

*public class Main {*

*public static void main(String[] args) {*

*Person p = new Person();*

*p.name = "Tom";*

*}*

*}*

A. Không có lỗi biên dịch.

B. name has protected access.

C. age has protected access.

D. name has public access.

4.5. Đối tượng trong phần mềm là gì?

A. Là một bó phần mềm gồm các hành vi và trạng thái có liên quan với nhau.

B. Là vật thể xác định của thế giới thực.

C. Là vật thể gồm hành vi và trạng thái.

4.6. Khai báo lớp nào dưới đây là đúng?

A. public class default {}

B. protected inner class engine {}

C. final class outer {}

D. Tất cả đều sai

4.7. Phạm vi truy cập của một đối tượng khi được khai bao protected là gì?

A. Có thể được truy cập từ bất kỳ vị trí nào trong chương trình.

B. Có thể được truy cập từ các lớp trong cùng package.

C. Có thể được truy cập từ các lớp trong cùng package và lớp con nằm trong package khác.

D. Chỉ có thể truy cập từ các phương thức khác trong class đó.

4.8. Phạm vi truy cập của một đối tượng khi được khai báo public là gì?

A. Có thể được truy cập từ bất kỳ vị trí nào trong chương trình.

B. Có thể được truy cập từ các lớp trong cùng package.

C. Có thể được truy cập từ các lớp trong cùng package và lớp con nằm trong package khác.

D. Chỉ có thể truy cập từ các phương thức khác trong class đó.

## Bài 7: Lớp và các thành phần của lớp đối tượng (tiếp theo) (Số tiết: 03 tiết)

## 4.4. Toán tử tạo lập đối tượng

Đối tượng(Object) là một thể hiện của một lớp(Class). Lớp là một mẫu hoặc thiết kế từ đó các đối tượng được tạo ra. Vì vậy, đối tượng là các thể hiện của một lớp.

Toán tử tạo lập đối tượng được sử dụng để tạo ra thể hiện của lớp, nghĩa là đặt các giá trị khởi tạo cho đối tượng khi một đối tượng được tạo ra bằng toán tử *new*

Có 2 kiểu của constructor:

* Constructor mặc định (không có tham số truyền vào – default constructors).
* Constructor tham số (parameterized constructors), ta có thể tự định nghĩa toán tử tạo lập cho một lớp đối tượng.

### 4.4.1. Khai báo toán tử tạo lập

\* Khai báo:

**[phạm vi] <tênlớp>( <danh sách các tham biến>) { //Phần thân}**

Phạm\_vi có thể là: public, protected, private

Phần thân là các lệnh khởi tạo giá trị các thuộc tính của đối tượng, là thành phần tuỳ chọn.

Danh sách các tham số : tuỳ chọn

\* Phân loại:

* Toán tử tạo lập mặc định không tường minh: **Tên\_Lớp() {}**

Khi định nghĩa một lớp mà không xây dựng toán tử tạo lập nào thì hệ thống sẽ cung cập toán tử tạo lập mặc định không tường minh.

* Toán tử tạo lập mặc định tường minh:

TênLớp(<danh sách tham biến hình thức>) {

tên thuộc tính= giá trị;

...... }

Giá trị của các biến thành phần được khởi tạo không mặc định bằng các phép gán trong thân của toán tử bằng các giá trị được truyền vào.

Ví dụ: Minh hoạ một phương thức khởi tạo của lớp Person bằng cách gán giá trị cho các thuộc tính tên và tuổi.

*class Person {*

*public String name;*

*public int age;*

*// Phương thức tạo lập mặc định không tường minh*

*//public Person(){}*

*// Phương thức tạo lập mặc định tường minh*

*public Person(){*

*name = “”;*

*age = 16;*

*}*

*// Phương thức tạo lập không mặc định*

*public Person(String name1, int age1){*

*name = name1;*

*age = age1;*

*}*

*// Phương thức tạo lập không mặc định*

*public Person(String name1, int age1){*

*name = name1;*

*age = age1;*

*}*

*/\*public Person(String name, int age){*

*this.name = name;*

*this.age = age;*

*}\*/*

*public void show(){*

*System.out.println( name + “ is ” + age + “ years old!”);*

*}*

*}*

### 4.4.2. Toán tử this() và super()

|  |  |
| --- | --- |
| Toán tử tạo lập this()  Toán tử tạo lập này được sử dụng để tham  chiếu đến các thành phần(là các thuộc tính  và phương thức) của lớp hiện thời | Toán tử tạo lập super()  Được sử dụng tham chiếu đến các thành phần thuộc lớp cha trực tiếp |
| class BongDen{  // Biến thành phần (1)  private int soWatts;  private boolean batTat;  private String viTri;  // Toán tử tạo lập số 1 (2)  BongDen(){  this(40, true);  System.out.println(“Toán tử số 1”);  } // Toán tử tạo lập không mặc định số 2 (3) BongDen(int w, boolean s){  this(w, s, “XX”);  System.out.println(“Toán tử số 2”);  } // Toán tử tạo lập không mặc định số 3 (4) BongDen(int soWatts, Boolean batTat, String viTri){  this.soWatts = soWatts;  this.batTat = batTat;  this.viTri = new String(viTri);  System.out.println(“Toán tử số 3”);  }  } | class BongDen{  // Biến thành phần (1)  private int soWatts;  private boolean batTat;  private String viTri;  // Toán tử tạo lập số 2 (2)  BongDen(){  this(40, true);  System.out.println(“Toán tử số 1”);  } // Toán tử tạo lập không mặc định số 2 (3) *BongDen(int w, boolean s){*  this(w, s, “XX”);  System.out.println(“Toán tử  số 2”);  } // Toán tử tạo lập không mặc định số 3 (4) BongDen(int soWatts, Boolean  batTat, String viTri){  this.soWatts = soWatts;  this.batTat = batTat;  this.viTri = new  String(viTri);  System.out.println(“Toán tử số 3”);  }  } |
| public class NhaKho{  public static void main(String args[]){  BongDen d1=new BongDen();// OK  BongDen d2 = new BongDen(100, true, “Nha bep”); // OK  BongDen d3 = new BongDen(100, true); //OK } } | class DenTuyp extends BongDen {  private int doDai;  private int mau;  DenTuyp(int leng, int colo){ // (5)  this(leng, colo, 100, true, “Chua biet”);  }  DenTuyp(int leng, int colo, int soWatt, boolean bt, String noi){ // (6)  super(soWatt, bt, noi);  this.doDai = leng;  this.mau = colo;  } } public class NhaKho{  public static void main(String args[]){  System.out.println(“Tao ra bong đen tuyp”); DenTuyp d = new DenTuyp(20,5);  } } |

## 4.5. Kế thừa giữa các lớp đối tượng

Kế thừa trong java là một loại quan hệ giữa hai lớp (class) với nhau, trong đó có class cha (superclass) và class con (subclass). Khi kế thừa class con được sử dụng lại các phương thức và thuộc tính của class cha. Tuy nhiên, nó chỉ được kế thừa các thành phần public và protected của class cha. Nó không được phép truy cập đến thành phần private của class cha.

Tư tưởng của kế thừa trong java là có thể tạo ra một class mới được xây dựng trên các lớp đã xây dựng. Khi kế thừa từ một lớp ta có sử dụng lại các phương thức và thuộc tính của lớp cha, đồng thời có thể khai báo bổ sung thêm các phương thức và thuộc tính khác nhằm giới hạn phạm vi của đối tượng.

### 4.5.1. Cú pháp

Sử dụng từ khóa *extends* để kế thừa:

*class Subclass-name extends Superclass-name {*

*//methods and fields*

*}*

Ví dụ:

*class Employee {*

*float salary = 1000;*

*}*

*class Programmer extends Employee {*

*int bonus = 150;*

*}*

*public class InheritanceSample1 {*

*public static void main(String args[]) {*

*Programmer p = new Programmer();*

*System.out.println("Programmer salary is: " + p.salary);*

*System.out.println("Bonus of Programmer is: " + p.bonus);*

*}*

*}*

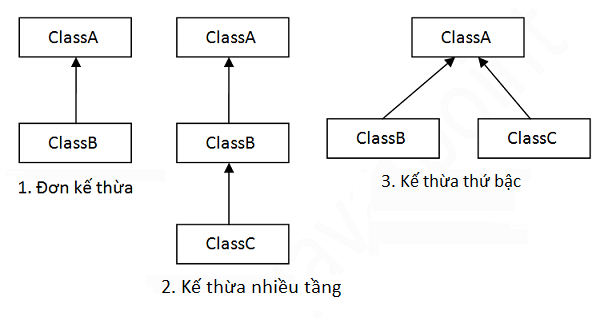
Kết quả:

*Programmer salary is: 1000.0*

*Bonus of Programmer is: 150*

### 4.5.2. Các kiểu kế thừa trong java

Có 3 kiểu kế thừa trong java đó là đơn kế thừa, kế thừa nhiều cấp, kế thừa thứ bậc. Khi một class được kế thừa từ nhiều class được gọi là đa kế thừa. Trong java, đa kế thừa chỉ được hỗ trợ thông qua giao diện(interface).



***Chú ý:*** *Đa kế thừa trong java không được support thông qua class.*

***Ví dụ:***

*class Animal {*

*void eat() {*

*System.out.println("eating...");*

*}*

*}*

*class Dog extends Animal {*

*void bark() {*

*System.out.println("barking...");*

*}*

*}*

*public class TestInheritance1 {*

*public static void main(String args[]) {*

*Dog d = new Dog();*

*d.bark();*

*d.eat();*

*}*

*}*

**Kết quả:**

*barking...*

*eating...*

**Ví dụ về kế thừa nhiều cấp**

*class Animal {*

*void eat() {*

*System.out.println("eating...");*

*}*

*}*

*class Dog extends Animal {*

*void bark() {*

*System.out.println("barking...");*

*}*

*}*

*class BabyDog extends Dog {*

*void weep() {*

*System.out.println("weeping...");*

*}*

*}*

*public class TestInheritance2 {*

*public static void main(String args[]) {*

*BabyDog d = new BabyDog();*

*d.weep();*

*d.bark();*

*d.eat();*

*}*

*}*

**Kết quả:**

*weeping...*

*barking...*

*eating...*

**Ví dụ về kế thừa thứ bậc**

*class Animal {*

*void eat() {*

*System.out.println("eating...");*

*}*

*}*

*class Dog extends Animal {*

*void bark() {*

*System.out.println("barking...");*

*}*

*}*

*class Cat extends Animal {*

*void meow() {*

*System.out.println("meowing...");*

*}*

*}*

*public class TestInheritance3 {*

*public static void main(String args[]) {*

*Cat c = new Cat();*

*c.meow();*

*c.eat();*

*// c.bark(); // compile error*

*}*

*}*

**Kết quả:**

*meowing...*

*eating...*

**Lưu ý:** Đa kế thừa không được hỗ trợ trong java.

Để giảm thiểu sự phức tạp và đơn giản hóa ngôn ngữ, đa kế thừa không được support trong java. Hãy xem xét kịch bản sau: Có 3 lớp A, B, C. Trong đó lớp C kế thừa từ các lớp A và B. Nếu các lớp A và B có phương thức giống nhau và bạn gọi nó từ đối tượng của lớp con, như vậy khó có thể xác đinh được việc gọi phương thức của lớp A hay B.

*class A {*

*void msg() {*

*System.out.println("Hello");*

*}*

*}*

*class B {*

*void msg() {*

*System.out.println("Welcome");*

*}*

*}*

*public class C extends A,B {*

*public static void main(String args[]) {*

*C obj = new C();*

*obj.msg();*

*}*

*}*

**Kết quả:**

*Compile Time Error*

## 4.6. Giao diện và sự mở rộng quan hệ kế thừa

* Chương trình Java chỉ có thể kế thừa từ 1 lớp duy nhất trong cùng một thời điểm, nhưng có thể dẫn xuất cùng lúc nhiều giao diện (Interfaces)
* Không được phép có những phương thức cụ thể (concrete methods)
* Interface cần phải được hiện thực (implements).

### 4.6.1. Các bước tạo interface

Một Interface trong Java là một bản thiết kế của một lớp. Nó chỉ có các phương thức trừu tượng. Interface là một kỹ thuật để thu được tính trừu tượng hoàn toàn và đa kế thừa trong Java. Interface trong Java cũng biễu diễn mối quan hệ IS-A. Nó không thể được khởi tạo giống như lớp trừu tượng. Nói cách khác, các thành phần của Interface là public, static và final theo mặc định và các phương thức là public và abstract.

Một Interface trong Java là một tập hợp các phương thức trừu tượng (abstract). Một class triển khai một interface, do đó kế thừa các phương thức abstract của interface.

Một interface không phải là một lớp. Viết một interface giống như viết một lớp, nhưng chúng có 2 định nghĩa khác nhau. Một lớp mô tả các thuộc tính và hành vi của một đối tượng. Một interface chứa các hành vi mà một class triển khai. Trừ khi một lớp triển khai interface là lớp trừu tượng abstract, còn lại tất cả các phương thức của interface cần được định nghĩa trong class.

Một interface tương tự với một class bởi những điểm sau đây:

* Một interface được viết trong một file với định dạng .java, với tên của interface giống tên của file.
* Bytecode của interface được lưu trong file có định dạng .class.
* Khai báo interface trong một package, những file bytecode tương ứng cũng có cấu trúc thư mục có cùng tên package.

Một interface khác với một class ở một số điểm sau đây:

* Interface không có thể hiện(instance) do đó một interface không chứa bất cứ hàm Contructor nào.
* Tất cả các phương thức của interface đều là abstract.
* Trong interface chỉ có các thuộc tính static final
* Một interface không thể kế thừa từ lớp, nó được triển khai bởi một lớp.
* Một interface có thể kế thừa từ nhiều interface khác.

Các bước để tạo một interface:

* Định nghĩa Interface
* Biên dịch Interface
* Hiện thực hay dẫn xuất (implements) Interface

**Tính chất của interface:**

* Tất cả phương thức trong interface phải là public.
* Các phương thức phải được định nghĩa trong lớp dẫn xuất giao diện đó.
* Không thể dẫn xuất từ lớp khác, nhưng có thể dẫn xuất từ những interface khác
* Khi định nghĩa một interface mới thì một kiểu dữ liệu tham chiếu cũng được tạo ra.

### 4.6.2. Khai báo interface

Cú pháp khai báo một interface như sau:

*[public] interface <tên interface> [extends <danh sách interface>]*

*{ … }*

- **Tính chất:** tính chất của một interface luôn là public. Nếu không khai báo tường minh thì giá trị mặc định cũng là public.

- Tên *interface*: tuân thủ theo quy tắc đặt tên biến của java.

- Danh sách các *interface*: danh sách các *interface* cha đã được định nghĩa để kế thừa, các interface cha được phân cách nhau bởi dấu phẩy. (Phần trong ngoặc vuông “[]” là tuỳ chọn).

**Lưu ý:** Một *interface* chỉ có thể kế thừa từ các *interface* khác mà không thể được kế thừa từ các lớp sẵn có.

**Khai báo phương thức của interface**

[public] <kiểu giá trị trả về> <tên phương thức> ([<các tham số>])  
[throws <danh sách ngoại lệ>];

**Tính chất:** tính chất của một thuộc tính hay phương thức của *interface* luôn là *public*. Nếu không khai báo tường minh thì giá trị mặc định cũng là *public*.

**Kiểu giá trị trả về:** có thể là các kiểu cơ bản của java, cũng có thể là kiểu do người dùng tự định nghĩa (kiểu đối tượng).

**Tên phương thức:** tuân thủ theo quy tắc đặt tên phương thức của lớp

**Các tham số:** nếu có thì mỗi tham số được xác định bằng một cặp <kiểu tham số> <tên tham số>. Các tham số được phân cách nhau bởi dấu phẩy.

**Các ngoại lệ:** nếu có thì mỗi ngoại lệ được phân cách nhau bởi dấu phẩy.

**Lưu ý:**

* Các phương thức của *interface* chỉ được khai báo dưới dạng mẫu mà không có cài đặt chi tiết (có dấu chấm phẩy ngay sau khai báo và không có phần cài đặt trong dấu “{}”). Phần cài đặt chi tiết của các phương thức chỉ được thực hiện trong các lớp (*class*) sử dụng *interface* đó.
* Các thuộc tính của *interface* luôn có tính chất là hằng (*final*), tĩnh (*static*) và *public*. Do đó, cần gán giá trị khởi đầu ngay khi khai báo thuộc tính của *interface*.

### 4.6.3. Sử dụng interface

Vì *interface* chỉ được khai báo dưới dạng các phương thức mẫu và các thuộc tính hằng nên việc sử dụng *interface* phải thông qua một lớp có cài đặt *interface* đó. Việc khai báo một lớp có cài đặt *interface* được thực hiện thông qua từ khoá *implements* như sau:

*<tính chất> class <tên lớp> implements <các interface>*

*{ …}*

*Tính chất* và *tên lớp* được sử dụng như trong khai báo lớp thông thường.

*Các interface:* một lớp có thể dẫn xuất từ nhiều *interface*. Các *interface* được phân cách nhau bởi dấu phẩy. Khi đó, lớp phải cài đặt cụ thể tất cả các phương thức của tất cả các *interface* mà nó sử dụng.

**Lưu ý:**

Một phương thức được khai báo trong *interface* phải được cài đặt cụ thể trong lớp có cài đặt *interface* nhưng không được phép khai báo chồng. Nghĩa là số lượng các tham số của phương thức trong *interface* phải được giữ nguyên khi cài đặt cụ thể trong lớp.

**Ví dụ:**

*interface printable {*

*void print();*

*}*

*class A6 implements printable {*

*public void print() {*

*System.out.println("Hello");*

*}*

*public static void main(String args[]){*

*A6 obj = new A6();*

*obj.print();*

*}*

*}*

Khi ghi đè các phương thức được định nghĩa trong interface, có một số qui tắc sau:

* Các *checked* *exception* không nên được khai báo trong phương thức *implements*, thay vào đó nó nên được khai báo trong phương thức *interface* hoặc các lớp phụ được khai báo bởi phương thức *interface*.
* *Signature* (ký số) của phương thức *interface* và kiểu trả về nên được duy trì khi ghi đè phương thức (*overriding* *method*).
* Một lớp triển khai chính nó có thể là *abstract* và vì thế các phương thức *interface* không cần được triển khai.

Khi triển khai *interface*, có các quy tắc sau:

* Một lớp có thể triển khai một hoặc nhiều *interface* tại một thời điểm.
* Một lớp chỉ có thể kế thừa một lớp khác, nhưng được triển khai nhiều *interface*.
* Một *interface* có thể kế thừa từ một *interface* khác, tương tự cách một lớp có thể kế thừa lớp khác.

### 4.6.4. Đa kế thừa trong Java bởi Interface

Nếu một lớp triển khai đa kế thừa, hoặc một Interface kế thừa từ nhiều *interface* thì đó là đa kế thừa.

*interface Printable {*

*void print();*

*}*

*interface Showable{*

*void show();*

*}*

*class A7 implements Printable,Showable {*

*public void print() {*

*System.out.println("Hello");*

*}*

*public void show() {*

*System.out.println("Welcome");*

*}*

*public static void main(String args[]){*

*A7 obj = new A7();*

*obj.print();*

*obj.show();*

*}*

*}*

Trong Java đa kế thừa không được hỗ trợ thông qua lớp. Nhưng nó được hỗ trợ bởi *Interface* bởi vì không có tính lưỡng nghĩa khi trình triển khai được cung cấp bởi lớp *Implementation*. Ví dụ:

*interface Printable{*

*void print();*

*}*

*interface Showable{*

*void print();*

*}*

*class TestTnterface1 implements Printable,Showable{*

*public void print() {*

*System.out.println("Hello");*

*}*

*public static void main(String args[]) {*

*TestTnterface1 obj = new TestTnterface1();*

*obj.print();*

*}*

*}*

Trong ví dụ trên, Printable và Showable interface có cùng các phương thức nhưng trình triển khai của nó được cung cấp bởi lớp TestInterface1, vì thế không có tính lưỡng nghĩa.

**Kế thừa Interface trong Java**

Một lớp chỉ được kế thừa từ một lớp cha, nhưng một interface lại cho phép được kế thừa từ nhiều *interface*. Ví dụ:

*interface Printable{*

*void print();*

*}*

*interface Printable1{*

*void print1();*

*}*

*interface Showable extends Printable,Printable1{*

*void show();*

*}*

*class Testinterface2 implements Showable{*

*public void print() {*

*System.out.println("Hello");*

*}*

*@Override*

*public void print1() {*

*System.out.println("Hello 1");*

*}*

*public void show() {*

*System.out.println("Welcome");*

*}*

*public static void main(String args[]){*

*Testinterface2 obj = new Testinterface2();*

*obj.print();*

*obj.print1();*

*obj.show();*

*}*

*}*

## Bài tập cuối chương

4.9. Chọn câu trả lời đúng nhất. Interface là gì?

A. Là lớp chứa các cách rỗng có liên quan với nhau.

B. Là một kiểu tham chiếu, tương tự như class, chỉ có thể chứa hằng giá trị, khai báo cách và kiểu lồng.

C. Là một cách thực hiện của lớp khác.

D. Là lớp nối giữa lớp cơ sở và lớp cha.

4.10. Để sử dụng giao diện Xedap cho lớp Xedap1, ta làm thế nào?

A. class Xedap1 implement Xedap {}

B. public class Xedap1 implement Xedap {}

C. class Xedap1 implements Xedap {}

D. public class Xedap1 extends Xedap {}

4.11. Để khai báo lớp Xedap1 kế thừa lớp Xedap phải làm như thế nào?

A. class Xedap1 extend Xedap {}

B. public classs Xedap1 extend Xedap {}

C. class Xedap1 extends Xedap {}

D. Tất cả đều sai.

# Chương V: CÁC LỚP CƠ SỞ VÀ CÁC CẤU TRÚC DỮ LIỆU

### Nội dung chính của chương

- Cấu trúc mảng

- Các lớp cơ sở trong gói java.lang

- Một số cấu trúc dữ liệu phổ dụng trong java

### Mục tiêu cần đạt được của chương

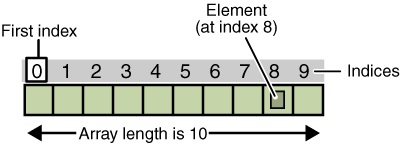
Giúp sinh viên sẽ nắm được cấu trúc, các thành phần của mảng đối tượng trong ngôn ngữ java, hiểu được cách sử dụng các lớp cơ sở trong gói java.lang và một số cấu trúc dữ liệu phổ dụng trong java.

## Bài 8: Mảng và các lớp cơ sở trong gói java.lang (Số tiết: 03 tiết)

## 5.1. Cấu trúc mảng

### 5.1.1. Khai báo mảng

Mảng (array) là tập hợp các phần tử có cùng kiểu dữ liệu và có số phần tử là cố định(không thay đổi được số phần tử sau khi tạo lập). Trong java mảng là kiểu tham chiếu do vậy ta phải khởi tạo mảng trước khi sử dụng.



Hình 5.1. Mảng

**Khai báo**

*<Kiểu các phần tử>[] <Tên mảng>;*

*<Kiểu các phần tử> <Tên mảng>[];*

Trong đó *<Kiểu các phần tử>* có thể là kiểu nguyên thủy hoặc là kiểu lớp (tên lớp).

Kích thước của mảng chưa xác định khi khai báo

### 5.1.2. Tạo lập mảng

Để tạo lập đối tượng mảng thì phải xác định số phần tử của mảng đó.

*<Tên mảng> = new <Kiểu các phần tử >[<Số phần tử>];*

Trong đó:

*<Kiểu các phần tử>* là kiểu tương thích với kiểu mà mảng đã khai báo. Giá trị cực tiểu của *<Số phần tử>* là 0.

Kết hợp cả khai báo với tạo lập mảng như sau:

*<Kiểu các phần tử 1> <Tên mảng>[] = new <Kiểu các phần tử 2>[<Số phần tử>];*

Khai báo và tạo lập mảng *<Tên mảng>* kiểu *<Kiểu các phần tử 1>* để chứa *<Số phần tử>* các phần tử có *<Kiểu các phần tử 2>*.

*<Kiểu các phần tử 1>* và *<Kiểu các phần tử 2>* là hai kiểu tương thích với nhau. Nếu mảng đó có kiểu nguyên thủy thì hai kiểu đó phải trùng nhau. Nếu là kiểu lớp thì *<Kiểu các phần tử 2>* là lớp con của *<Kiểu các phần tử 1>* ( một lớp cũng được xem là lớp con của chính nó).\

Khi một mảng được tạo lập thì tất cả các phần tử của nó được khởi tạo giá trị mặc định (0 hoặc 0.0 đối với kiểu số, false đối với kiểu boolean, null cho kiểu lớp).

**Ví dụ:**

*float mangFloat[] = new float[20];// Các phần tử được gán trị mặc định*

*Object[] dayDen = new BongDen[5];// Các phần tử được gán mặc định null khởi tạo các mảng.*

Khai báo, tạo lập và gán ngay các giá trị ban đầu:

*<Kiểu các phần tử >[] <Tên mảng> = {<Các giá trị ban đầu>};*

**Ví dụ:** int[] mangInt = {1, 3, 5, 7, 9};

Tạo ra *mangInt* có 5 phần tử với phần tử đầu có giá trị là 1 (mangInt[0] = 1), phần tử thứ hai là 3 (mangInt[1] = 3) , v.v.

*Object[] dayDT = {new BongDen(), new BongDen(), null};*

*char[] charArr = {“a”, “b”, “a”}; // mảng 3 ký tự và hoàn toàn khác với “aba”*

### 5.1.3. Truy cập các phần tử của mảng

Truy cập một phần tử mảng bằng cách sử dụng số chỉ số (index) trong đó *0≥ index< (số phần tử mảng)*. Phần tử thứ i của mảng là phần tử có chỉ số i - 1 (<Tên mảng>[ i -1]). Nếu truy cập vượt quá phạm vi xác định của mảng sẽ gặp ngoại lệ ArrayIndexOutOfBoundsException

Truy cập vào phần tử của mảng: *<tên mảng>[i]*

**Ví dụ:**

*public class TruyCapArray1 {*

*public static void main(String[] args) {*

*String[] cars = { "Honda", "BMW", "Ford", "Mazda" };*

*System.out.println(cars[0]);*

*}*

*}*

Kết quả:

*Honda*

Để thay đổi giá trị của một phần tử cụ thể trong mảng, ta cũng dùng chỉ số:

*public class TruyCapArray2 {*

*public static void main(String[] args) {*

*String[] cars = { "Honda", "BMW", "Ford", "Mazda" };*

*// thay đổi phần tử đầu tiên của mảng cars*

*cars[0] = "Morning";*

*// hiển thị phần tử đầu tiên của mảng cars*

*System.out.println("Phần tử đầu tiên: " + cars[0]);*

*}*

*}*

Kết quả:

*Phần tử đầu tiên: Morning*

Để biết có bao nhiêu phần tử trong một mảng, ta sử dụng thuộc tính **length**:

*public class DodaiArray1 {*

*public static void main(String[] args) {*

*String[] cars = { "Honda", "BMW", "Ford", "Mazda" };*

*System.out.println("Độ dài của mảng cars là: " + cars.length);*

*}*

*}*

Kết quả:

*4*

### 5.1.4. Duyệt các phần tử của mảng trong Java

Có 2 các để duyệt các phần tử của mảng:

* Sử dụng vòng lặp for
* Sử dụng foreach

**Sử dụng vòng lặp for**

Ta có thể lặp qua các phần tử mảng bằng vòng lặp for và sử dụng thuộc tính length để chỉ định số lần vòng lặp sẽ chạy.

Ví dụ:

*public class DuyetArray1 {*

*public static void main(String[] args) {*

*String[] cars = { "Honda", "BMW", "Ford", "Mazda" };*

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

*System.out.println(cars[i]);*

*}*

*}*

*}*

Kết quả:

*Honda*

*BMW*

*Ford*

*Mazda*

**Sử dụng foreach**

*public class DuyetArray2 {*

*public static void main(String[] args) {*

*String[] cars = { "Honda", "BMW", "Ford", "Mazda" };*

*for (String car : cars) {*

*System.out.println(car);*

*}*

*}*

*}*

Kết quả:

*Honda*

*BMW*

*Ford*

*Mazda*

**So sánh for với foreach**

|  |  |
| --- | --- |
| Vòng lặp for là một cấu trúc điều khiển để chỉ định phép lặp cho phép lệnh được thực thi nhiều lần. | Vòng lặp foreach là một cấu trúc điều khiển để duyệt các mục trong một mảng hoặc một tập hợp. |
| Vòng lặp for có thể được sử dụng để truy xuất một tập hợp các phần tử cụ thể. | Vòng lặp foreach không thể được sử dụng để truy xuất một tập hợp các phần tử cụ thể. |
| Vòng lặp for khó đọc và ghi hơn vòng lặp foreach. | Vòng lặp foreach dễ đọc hơn vòng lặp for. |
| Vòng lặp for được sử dụng như một vòng lặp mục đích chung. | Vòng lặp foreach được sử dụng cho mảng và tập hợp. |

### 5.1.5. Mảng nhiều chiều

**Định nghĩa:**

*<Kiểu các phần tử>[][]...[] <Tên mảng>;*

hoặc

*<Kiểu các phần tử> <Tên mảng>[][]...[];*

**Ví dụ:** *int[][] mang1; // Mảng hai chiều*

tương đương với *int mang1[][];*

tương đương với *int[] mang1[];*

Khai báo với thiết lập mảng nhiều chiều tương tự mảng đơn.

*int[][] mangA = new int[4][5]; // Ma trận có 4 hàng, 5 cột*

Truy nhập tới từng phần tử : *mangA[i][j].*

Kích thước của *mangA* là *mangA.length* = 4 và mỗi phần tử của nó lại là mảng có kích thước là mangA[i].length = 5, i = 1, 2, ..., 4.

*double[][] maTran = {*

*{1, 2, 3, 4}, // hàng 1*

*{0, 2, 0, 0}, // hàng 2*

*{0, 0, 3, 0}, // hàng 3*

*{0, 0, 0, 4}, // hàng 4*

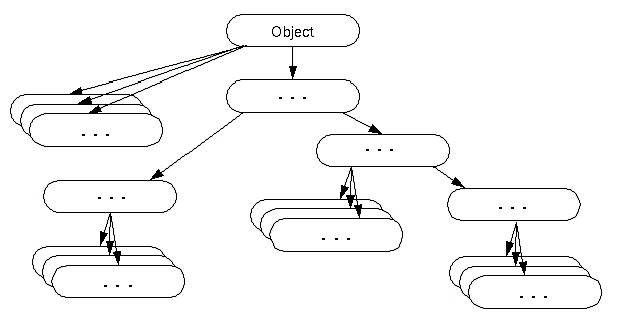
*};*

**Lưu ý:** Các mảng trong mảng nhiều chiều không nhất thiết phải có số phần tử giống nhau.

## 5.2. Các lớp cơ sở trong gói java.lang

### 5.2.1. Lớp Object

Tất cả các lớp được xây dựng trong các chương trình Java đều hoặc là trực tiếp hoặc gián tiếp được mở rộng từ lớp Object. Đây là lớp cơ sở nhất, định nghĩa hầu như tất cả những phương thức phần cơ bản để các lớp con cháu của nó sử dụng trực tiếp hoặc viết đè. Object cung cấp các phương thức sau: *int hashCode()*



Hình 5.2 Lớp object

Khi các đối tượng được lưu vào các bảng băm (hash table), hàm này có thể sử dụng để xác định duy nhất giá trị cho mỗi đối tượng. Điều này đảm bảo tính nhất quán của hệ thống khi thực hiện chương trình.

Sử dụng lớp Object là hữu ích nếu ta muốn tham chiếu bất kỳ đối tượng nào mà ta chưa biết kiểu dữ liệu của đối tượng đó. Chú ý rằng biến tham chiếu của lớp cha có thể tham chiếu đến đối tượng của lớp con được gọi là *upcasting*.

**Ví dụ:** giả sử phương thức getObject() trả về một đối tượng nhưng nó có thể là bất kỳ kiểu nào như Employee,Student, ... chúng ta có thể sử dụng biến tham chiếu của lớp Object để tham chiếu tới đối tượng đó.

*Object obj=getObject(); // chúng ta không biết đối tượng gì được trả về từ phương thức này.*

Lớp Object cung cấp các phương thức như trong bảng sau:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| public final Class getClass() | trả về đối tượng lớp Class của đối tượng hiện tại. Từ lớp Class đó có thể lấy được các thông tin metadata của class hiện tại. |
| public int hashCode() | trả về số hashcode cho đối tượng hiện tại. |
| public boolean equals(Object obj) | so sánh đối tượng đã cho với đối tượng hiện tại. |
| protected Object clone() throws CloneNotSupportedException | tạo và trả về bản sao chép (clone) của đối tượng hiện tại. |
| public String toString() | trả về chuỗi ký tự đại diện của đối tượng hiện tại. |
| public final void notify() | đánh thức một luồng, đợi trình giám sát của đối tượng hiện tại. |
| public final void notifyAll() | đánh thức tất cả các luồng. đợi trình giám sát của đối tượng hiện tại. |
| public final void wait(long timeout)throws InterruptedException | làm cho Thread hiện tại đợi trong khoảng thời gian là số mili giây cụ thể, tới khi Thread khác thông báo (gọi phương thức notify() hoặc notifyAll()). |
| public final void wait(long timeout,int nanos)throws InterruptedException | làm cho Thread hiện tại đợi trong khoảng thời gian là số mili giây và nano giây cụ thể, tới khi Thread khác thông báo (gọi phương thức notify() hoặc notifyAll()). |
| public final void wait()throws InterruptedException | làm Thread hiện tại đợi, tới khi Thread khác thông báo (gọi phương thức notify() hoặc notifyAll()). |
| protected void finalize()throws Throwable | Được gọi bởi Garbage Collector trước khi đối tượng bị dọn rác. |

### 5.2.2. Các lớp nguyên thủy

Các giá trị nguyên thủy không phải là đối tượng trong Java. Để có thể thao tác được trên các giá trị nguyên thủy (giá trị số, kí tự và logic) thì gói java.lang cung cấp các lớp bao gói (Wrapper) cho từng kiểu dữ liệu nguyên thủy (gọi tắt là lớp bao). Các lớp bao có những đặc tính chung sau:

**\* Các toán tử tạo lập chung:** Các lớp bao (trừ lớp Character chỉ có một toán tử tạo lập) đều có hai toán tử tạo lập:

* Toán tử tạo lập sử dụng giá trị nguyên thủy để tạo ra đối tượng tương ứng.

*Character charObj = new Character(„a‟);*

*Boolean boolObj = new Boolean(true);*

*Integer intObj = new Integer(2002);*

*Float floatObj = new Float(3.14F);*

*Double doubleObj = new Double(3.14);*

* Toán tử thứ hai: chuyển các đối tượng lớp String biểu diễn cho các giá trị nguyên thủy về các lớp tương ứng. Các toán tử này sẽ ném ra ngoại lệ NumberFormatException khi giá trị String truyền vào hàm tạo không hợp lệ.

*Boolean boolObj = new Boolean(“true”);*

*Integer intObj = new Integer(“2002”);*

*Float floatObj = new Float(“3.14F”);*

*Double doubleObj= new Double(“3.14”);*

- Các hàm tiện ích chung: *valueOf(String s), toString(), typeValue(), equals()*

**\* Mỗi lớp (trừ lớp Character)** đều định nghĩa hàm static valueOf(String s) trả lại đối tượng tương ứng. Các hàm này ném ra ngoại lệ NumberFormatException khi giá trị String truyền vào phương thức không hợp lệ.

*Boolean boolObj = Boolean.valueOf(“true”);*

*Integer intObj = Integer.valueOf(“2002”);*

*Float floatObj = Float.valueOf(“3.14F”);*

*Double doubleObj= Double.valueOf(“3.14”);*

**\* Các lớp viết đè hàm toString()** trả lại là các đối tượng String biểu diễn cho các giá trị nguyên thủy ở dạng xâu.

*String charStr String boolStr String intStr String doubleStr = doubleObj.toString;*

**\* Các lớp viết đè hàm equals()** để thực hiện so sánh bằng nhau của các đối tượng nguyên thủy.

*Character charObj = new Character(„a‟);*

*boolean charTest = charObj.equals(„b‟); // false*

*Integer intObj1 = Integer.valueOf(“2010”);*

*boolean intTest = intObj.equals(intObj1); // false*

**Lớp Boolean**

Lớp này định nghĩa hai đối tượng Boolean.TRUE, Boolean.FALSE biểu diễn cho hai giá trị nguyên thủy true và false tương ứng.

**Lớp Character**

Lớp Character định nghĩa hai giá trị cực tiểu, cực đại Character.MIN\_VALUE, Character.MAX\_VALUE và các giá trị kiểu ký tự Unicode. Ngoài ra lớp này còn định nghĩa một số hàm static để xử lý trên các ký tự:

*static boolean isLowerCase(char ch)// true nếu ch là ký tự thường*

*static boolean isUpperCase(char ch)// true nếu ch là ký tự viết hoa*

*static boolean isDigit(char ch) // true nếuch là chữ số*

*static boolean isLetter(char ch)// true nếu ch là chữ cái*

*static boolean isLetterOrDigit(char ch) // true nếu ch là chữ hoặc là số*

*static char toUpperCase(char ch)// Chuyển ch về chữ viết hoa*

*static char toLowerCase(char ch)// Chuyển ch về chữ viết thường*

*static char toTitleCase(char ch)// Chuyển ch về dạng tiêu đề.*

**Các lớp bao kiểu số**

Các lớp Byte, Short, Integer, Long, Float, Double là các lớp con của lớp Number. Trong các lớp này đều xác định hai giá trị:

<Lớp bao>.MIN\_VALUE

<Lớp bao>.MAX\_VALUE là các giới hạn của các số trong kiểu đó.

Ví dụ: *byte minByte = Byte.MIN\_VALUE;*

*int maxInt = Integer.MAX\_VALUE;*

Trong mỗi lớp bao có hàm typeValue() để chuyển các giá trị của các đối tượng nguyên thủy về giá trị số:

// -128

// 2147483647

*byte byteValue()*

*short shortValue()*

*int intValue()*

*long longValue()*

*float floatValue()*

*double doubleValue()*

Trong mỗi lớp bao còn có hàm static parseType(String s) để chuyển các giá trị được biểu diễn dưới dạng xâu về các giá trị số:

*byte value1 = Byte.parseByte(“16”);*

*int value2 = Integer.parseInt(“2002”);*

*double value3 = Double.parseDouble(“3.14”);*

**Ví dụ:** Viết chương trình để nhập vào một dãy số tùy ý và sắp xếp theo thứ tự tăng dần.

*import java.io.\*;*

*class SapXep {*

*static int[] day;*

*static void nhap() {*

*String str;*

*int n = day.length;*

*DataInputStream stream = new DataInputStream(System.in);*

*System.out.println("Nhap vao " + n + " so nguyen");*

*for (int k = 0; k < n; k++) {*

*try {*

*System.out.print(k + ": ”);*

*str = stream.readLine();*

*day[k] = Integer.valueOf(str).intValue();*

*} catch (IOException e) {*

*System.err.println("I/O Error!");*

*}*

*}*

*}*

*static void hienThi() {*

*int n = day.length;*

*for (int k = 0; k < n; k++) {*

*System.out.print(day[k] + " ");*

*}*

*System.out.println();*

*}*

*static void sapXep() {*

*int x, max, k;*

*for (int i = day.length - 1; i > 0; i--) {*

*max = day[i];*

*k = i;*

*for (int j = 0; j < i; j++) {*

*if (max < day[j]) {*

*max = day[j];*

*k = j;*

*}*

*}*

*day[k] = day[i];*

*day[i] = max;*

*}*

*}*

*public static void main(String[] args) {*

*String str;*

*int n;*

*DataInputStream stream = new DataInputStream(System.in);*

*System.out.print("\nCho biet bao nhieu so nhap vao: ");*

*try {*

*str = stream.readLine();*

*} catch (IOException e) {*

*System.err.println("I/O Error!");*

*str = "0";*

*}*

*n = Integer.valueOf(str).intValue();*

*SapXep.day = new int[n];*

*nhap();*

*sapXep();*

*System.out.println("Day so duoc sap xep: ");*

*hienThi();*

*}*

*}*

### 5.2.3. Lớp Math

Lớp Math nằm trong gói java.lang (gói mặc định) do vậy không cần phải thêm câu lệnh import ở đầu chương trình để có thể sử dụng lớp này. Các hàm này được viết là các phương thức tĩnh nên ta không cần phải tạo ra thể hiện của lớp Math. Bảng sau liệt kê một số phương thức tĩnh trong lớp Math:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên phương thức** | **Mô tả** | **Kiểu tham số** | **Kiểu trả về** |
| sin(arg) | tính sin của arg | arg là một biểu thức kiểu double thể hiện một cung theo radians | double |
| cos(arg) | tính cos của arg | arg là một biểu thức kiểu double thể hiện một cung theo radians | double |
| tan(arg) | tính tang của arg | arg là một biểu thức kiểu double thể hiện một cung theo radians | double |
| asin(arg) | tính sin-1 (arcsin) arg | arg là một biểu thức kiểu double thể hiện một cung theo radians | double trong hệ radians |
| acos(arg) | tính cos-1 (arccosin) của arg | arg là một biểu thức kiểu double thể hiện một cung theo radians | double trong hệ radians |
| atan(arg) | tính tan-1 (arctang) của arg | arg là một biểu thức kiểu double thể hiện một cung theo radians | double trong hệ radians |
| atan2 (arg1,arg2) | tính tan-1 (arctang) của arg1/arg2 | arg1,arg2 là các biểu thức kiểu double thể hiện một cung theo radians | double trong hệ radians |
| abs(arg) | tính trị tuyệt đối của arg | arg là một biểu thức kiểu int, long, float, hoặc double | The same type as the argument |
| max (arg1,arg2) | Nhận về giá trị lớn trong hai tham số | arg1, arg2 là một biểu thức kiểu int, long, float, hoặc double | Nhận về kiểu cùng Kiểu với tham số |
| min (arg1,arg2) | Nhận về giá trị nhỏ trong hai tham số | arg1, arg2 là một biểu thức kiểu int, long, float, double | Nhận về kiểu cùng kiểu với tham số |
| ceil(arg) | Nhận về giá trị nguyên nhỏ hơn hoặc bằng arg | arg là biểu thức kiểu float hoặc double | double |
| floor(arg) | Nhận về giá trị nguyên lớn hơn hoặc bằng arg | arg là biểu thức kiểu float hoặc double | double |
| round(arg) | Trả về giá trị nguyên gần arg nhất, giá trị này chính là giá trị của arg sau khi đã làm tròn | arg là biểu thức kiểu floathoặc double | Nhận về kiểu int nếu arg kiêu float, nhận về kiùu long nếu arg kiểu double |
| rint(arg) | Giống Như round(arg) | arg là biểu thức kiểu double | double |
| sqrt(arg) | tính căn bậc hai của arg | arg là biểu thức kiểu double | double |
| pow (arg1,arg2) | tính arg1arg2 | Cả arg1 và arg2 là các biểu thức kiểu double | double |
| exp(arg) | tính earg | arg là biểu thức kiểu double | double |
| log(arg) | tính logarithm cơ số e của arg | arg là biểu thức kiểu double | double |
| random() | Nhận về một số giả ngẫu nhiên nằm trong khoản [0, 1) | Không có tham số | double |

Bảng 5.1 Một số phương thức tĩnh trong lớp Math

### 5.2.4. Lớp String

Lớp String dựng cho những xâu ký tự bất biến (xâu chỉ đọc ), khi được khởi tạo giá trị đầu, giá trị đó không thay đổi được nữa.

Phương thức khởi tạo (Constructor):

*String(String s)*

*String(<Kieu>[] mang)*

*String(StringBuffer buf)*

Ví dụ:

*byte[] bytes = {97, 98,98,97};*

*char[] characters = {„a‟, „b‟, „b‟,„a‟};*

*StringBuffer buff = new StringBuffer(“abba”);*

*String byteStr = new String(bytes);// Chuyển mảng bytes về xâu*

*String charStr = new String(charactes);//charStr = “abba”*

*String buffStr = new String(buff);*

*String str = new String(“abba”);*

Có thể sử dụng: *String str = “Dung dong den toi”;*

Ví dụ: Khởi tạo chuỗi trong Java

*public class StringInitialization {*

*public static void main(String a[]) {*

*String objStr\_01 = "This is a string object";*

*String objStr\_02 = new String("This is also string object");*

*char[] arrChar = {'V', 'N', 'L', 'I', 'V', 'E', 'S'};*

*String objStr\_03 = new String(arrChar);*

*String objStr\_04 = objStr\_03 + " This is another String object";*

*System.out.println("objStr\_01: " + objStr\_01);*

*System.out.println("objStr\_02: " + objStr\_02);*

*System.out.println("objStr\_03: " + objStr\_03);*

*System.out.println("objStr\_04: " + objStr\_04);*

*}*

*}*

Kết quả chạy chương trình:

*objStr\_01: This is a string object*

*objStr\_02: This is also string object*

*objStr\_03: VNLIVES*

*objStr\_04: VNLIVES This is another String object*

**Đọc từng ký tự**

+ *int length():* kích thước xâu.

+ *char charAt(int index):* cho lại ký tự thứ index (bắt đầu từ 0) của xâu hiện thời *(0≤index < length()).* (ngoại lệ *StringIndexOutOfBoundsException*).

**So sánh các xâu**

*boolean equals(Object obj)*: kiểm tra xem đối tượng lớp String và đối tượng ở tham số có cùng tập ký hiệu hay không

*boolean equalsIgnoreCase(String str2)* kiểm tra xem đối tượng lớp String và đối tượng ở tham số có cùng tập ký hiệu hay không (phân biệt chữ hoa và chữ thường)

*boolean startWith(String str)*

*boolean endWith(String str)*

Hai hàm này cho kết quả true nếu xâu bắt đầu (kết thuc) bằng đối số str.

*int compareTo(String str2)*

*int compareTo(Object obj)*

+ 0 nếu xâu hiện thời bằng xâu đối số,

+ nhỏ hơn 0, nếu xâu đó nhỏ hơn xâu đối số theo thứ tự từ điển,

+ lớn hơn 0, nếu xâu đó lớn hơn xâu đối số theo thứ tự từ điển,

Hàm *compareTo()* thứ hai thực hiện tương tự như như hàm đầu nếu đối số là chuyển về được đối tượng xâu, ngược lại sẽ cho qua để xử lý ngoại lệ *ClassCastException*.

**Chuyển đổi xâu:**

*String toUpperCase()*

*String toLowerCase()*

**Ghép xâu**

*String concat(String str)*

**Tìm các ký tự và các xâu con** Nếu không tìm được các hàm này sẽ cho kết quả là -1:

*int indexOf(int ch)* Tìm chỉ số của lần xuất hiện đầu tiên của *ch* trong xâu.

*int indexOf(int ch, int fromIndex)* Tìm chỉ số của lần xuất hiện đầu tiên của *ch* trong xâu bắt đầu từ *fromIndex*.

*int indexOf(String str)* Tìm chỉ số của lần xuất hiện đầu tiên của xâu con *str* trong xâu hiện thời.

*int indexOf(String str, int fromIndex)* Tìm chỉ số của lần xuất hiện đầu tiên của xâu con *str* bắt đầu từ *fromIndex* trong xâu hiện thời.

*int lastindexOf( int ch)* Tìm chỉ số của lần xuất hiện cuối cùng của ký tự *ch* trong xâu.

*int lastindexOf(int ch, int fromIndex)* Tìm chỉ số của lần xuất hiện cuối cùng của *ch* bắt đầu từ *fromIndex* trong xâu.

*int lastindexOf(String str)* Tìm chỉ số của lần xuất hiện cuối cùng của xâu con *str* trong xâu hiện thời.

*int lastindexOf(String str, int fromIndex)* Tìm chỉ số của lần xuất hiện cuối cùng của xâu con *str* bắt đầu từ *fromIndex* trong xâu hiện thời.

*String replace(char cu, char moi)* Thay tất cả các lần xuất hiện của ký tự *cu* bằng ký tự *moi* trong xâu.

**Trích ra các xâu con**

**String trim()** tạo một xâu mới, các ký tự “trắng” (những ký tự có giá trị nhỏ hơn giá trị dấu cách „ „) ở trong xâu đều bị loại bỏ.

*String substring(int startIndex)* Cho kết quả là một xâu con được triết ra từ vị trí *startIndex* đến cuối của xâu.

*String substring(int startIndex, int endIndex)* Cho kết quả là một xâu con được triết ra từ vị trí *startIndex* đến vị trí *endIndex* của xâu.(ngoại lệ *StringIndexOutOfBoundsException*.)

**Chuyển các đối tượng của Object về String**

*static String valueOf(Object obj)*

*static String valueOf(char[] characters)*

Hai hàm này được nạp chồng để chuyển các đối tượng obj và mảng các ký tự characters về xâu ký tự.

*static String valueOf(boolean b)*

*static String valueOf(char c)*

Hàm được nạp chồng để chuyển hai giá trị boolean: true, false về xâu ký tự “true”, “false” tương ứng.

*static String valueOf(int i)*

*static String valueOf(long l)*

*static String valueOf(float f)*

*static String valueOf(double d)*

Các hàm này được nạp chồng để chuyển các giá trị số về dạng xâu ký tự.

**Ví dụ: Sử dụng các hàm cơ bản của chuỗi.**

*package javaandroidvn;*

*public class JavaAndroidVn {*

*public static void main(String[] args) {*

*String str = "Android.Vn Android.Vn";*

*System.out.println("str = " + str);*

*// Lấy từ vị trí số 8 tới cuối cùng của chuỗi*

*System.out.println("str.substring(8) = " + str.substring(8));*

*//Lấy từ vị trí số 3 tới vị trí số 9*

*System.out.println("str.substring(3,9) = " + str.substring(3, 9));*

*//Độ dài chuỗi:*

*System.out.println("Độ dài chuỗi: str.length() = " + str.length());*

*//Lấy ra ký tự trong chuỗi theo chỉ số*

*char ch;*

*ch = str.charAt(4);*

*System.out.println("str.charAt(4) = " + ch);*

*// Thay 1 ký tự bằng ký tự khác trong chuỗi:*

*System.out.println("Thay tất cả ký tự 'n' bằng ký tự 'x' = " + str.replace('n', 'x'));*

*//Tìm chuỗi "And" là chuỗi con của chuỗi str, thay kết quả đầu tiên bằng chuỗi "xxx"*

*System.out.println("Thay And đầu tiên bằng xxx = " + str.replaceFirst("And", "xxx"));*

*//Thay toàn bộ chuỗi "And" của chuỗi str bằng chuỗi "xxx"*

*System.out.println("Thay tất cả And bằng xxx = " + str.replaceAll("And", "xxx"));*

*//Chuyển thành chữ thường:*

*System.out.println("str chuyển về viết thường: " + str.toLowerCase());*

*//Chuyển thành chữ hoa:*

*System.out.println("str chuyển về viết hoa: " + str.toUpperCase());*

*//Loại bỏ khoảng trống 2 bên chuỗi*

*String str1 = " " + str + " ";*

*System.out.println(" Android.Vn Android.Vn --> " + str1.trim());*

*}*

*}*

*package javaandroidvn;*

*public class JavaAndroidVn {*

*public static void main(String[] args) {*

*String str = "Android.Vn Android.Vn";*

*System.out.println("str = " + str);*

*// Lấy từ vị trí số 8 tới cuối cùng của chuỗi*

*System.out.println("str.substring(8) = " + str.substring(8));*

*//Lấy từ vị trí số 3 tới vị trí số 9*

*System.out.println("str.substring(3,9) = " + str.substring(3, 9));*

*//Độ dài chuỗi:*

*System.out.println("Độ dài chuỗi: str.length() = " + str.length());*

*//Lấy ra ký tự trong chuỗi theo chỉ số*

*char ch;*

*ch = str.charAt(4);*

*System.out.println("str.charAt(4) = " + ch);*

*// Thay 1 ký tự bằng ký tự khác trong chuỗi:*

*System.out.println("Thay tất cả ký tự 'n' bằng ký tự 'x' = " + str.replace('n', 'x'));*

*//Tìm chuỗi "And" là chuỗi con của chuỗi str, thay kết quả đầu tiên bằng chuỗi "xxx"*

*System.out.println("Thay And đầu tiên bằng xxx = " + str.replaceFirst("And", "xxx"));*

*//Thay toàn bộ chuỗi "And" của chuỗi str bằng chuỗi "xxx":*

*System.out.println("Thay tất cả And bằng xxx = " + str.replaceAll("And", "xxx"));*

*//Chuyển thành chữ thường:*

*System.out.println("str chuyển về viết thường: " + str.toLowerCase());*

*//Chuyển thành chữ hoa:*

*System.out.println("str chuyển về viết hoa: " + str.toUpperCase());*

*//Loại bỏ khoảng trống 2 bên chuỗi*

*String str1 = " " + str + " ";*

*System.out.println(" Android.Vn Android.Vn --> " + str1.trim());*

*}*

*}*

*package javaandroidvn;*

*public class JavaAndroidVn {*

*public static void main(String[] args) {*

*String str = "Android.Vn Android.Vn";*

*System.out.println("str = " + str);*

*// Lấy từ vị trí số 8 tới cuối cùng của chuỗi*

*System.out.println("str.substring(8) = " + str.substring(8));*

*//Lấy từ vị trí số 3 tới vị trí số 9*

*System.out.println("str.substring(3,9) = " + str.substring(3, 9));*

*//Độ dài chuỗi:*

*System.out.println("Độ dài chuỗi: str.length() = " + str.length());*

*//Lấy ra ký tự trong chuỗi theo chỉ số*

*char ch;*

*ch = str.charAt(4);*

*System.out.println("str.charAt(4) = " + ch);*

*// Thay 1 ký tự bằng ký tự khác trong chuỗi:*

*System.out.println("Thay tất cả ký tự 'n' bằng ký tự 'x' = " + str.replace('n', 'x'));*

*//Tìm chuỗi "And" là chuỗi con của chuỗi str, thay kết quả đầu tiên bằng chuỗi "xxx"*

*System.out.println("Thay And đầu tiên bằng xxx = " + str.replaceFirst("And", "xxx"));*

*//Thay toàn bộ chuỗi "And" của chuỗi str bằng chuỗi "xxx":*

*System.out.println("Thay tất cả And bằng xxx = " + str.replaceAll("And", "xxx"));*

*//Chuyển thành chữ thường:*

*System.out.println("str chuyển về viết thường: " + str.toLowerCase());*

*//Chuyển thành chữ hoa:*

*System.out.println("str chuyển về viết hoa: " + str.toUpperCase());*

*//Loại bỏ khoảng trống 2 bên chuỗi*

*String str1 = " " + str + " ";*

*System.out.println(" Android.Vn Android.Vn --> " + str1.trim());*

*}*

*}*

### 5.2.5. Lớp StringBuffer

Lớp StringBuffer được sử dụng đối với những xâu ký tự động, cú thể thay đổi nội dung.

**Tạo lập đối tượng StringBuffer**

*StringBuffer(String str)* Tạo ra đối tượng mới nội dung giống như xâu đối số str kích thước bằng kích thước của xâu str + 16.

*StringBuffer(int length)* Tạo ra đối tượng mới chưa có nội dung và đặt kích thước bằng đối số length, nếu đối số lớn hơn 0.

*StringBuffer()* Tạo ra đối tượng mới chưa có nội dung và đặt kích thước của xâu buffer là 16.

**Đọc và thay đổi các ký tự trong StringBuffer**

*int length()* Cho số ký tự trong xâu buffer.

*char charAt(int index)*

*void setCharAt(int index, char ch)*

Hàm đầu đọc một ký tự và hàm thứ hai thay đổi ký tự ở vị trí index trong xâu buffer thành *ch*. Hệ thống sẽ cho qua ngoại lệ *StringIndexOutOfBoundsException* nếu các chỉ số khụng tương thích.

**Chuyển StringBuffer sang String**

Lớp *StringBuffer* nạp chồng hàm *toString*() để chuyển xâu buffer về xâu của lớp cố địng String.

Hàm *append*() bổ sung các ký tự vào cuối của xâu.

*StringBuffer append(Object obj)*

Đối tượng obj được chuyển về xâu (bằng hàm *String.valueOf()*) và sau đó bổ sung vào xâu buffer.

*StringBuffer append(String str)*

*StringBuffer append(char[] str)*

*StringBuffer append(char[] str, int offset, int len)*

*StringBuffer append(char c)*

Các mảng ký tự str và ký tự c được chuyển về xâu và sau đú bổ sung vào xâu *buffer*

*StringBuffer append(boolean b)*

*StringBuffer append(int i)*

*StringBuffer append(long l)*

*StringBuffer append(float f)*

*StringBuffer append(double d)*

Các giá trị nguyên thủy được chuyển về xâu và sau đó bổ sung vào xâu *buffer*.

*StringBuffer insert(int offset, Object obj)*

*StringBuffer insert(int offset, String str)*

*StringBuffer insert(int offset, char[] str )*

*StringBuffer insert(int offset, char c)*

*StringBuffer insert(int offset, boolean b)*

*StringBuffer insert(int offset, int i)*

*StringBuffer insert(int offset, long l)*

*StringBuffer insert(int offset, float f)*

*StringBuffer insert(int offset, double d)*

chèn đối số thứ hai (sau khi được chuyển về dạng xâu bằng cách sử dụng hàm *String.valueOf()*) vào từ vị trí *offset* trong xâu *buffer*.

*StringBuffer delete(int index)StringBuffer delete(int start, int end)*

Hàm đầu xóa đi ký tự ở vị trí *index*, hàm sau xóa đi một xâu con kể từ vị trí *start* đến *end*.

*StringBuffer reverse()* đảo ngược xâu

**Kiểm soát kích thước của StringBuffer**

*int capacity()*: Cho lại kích thước (khả năng chứa) của xâu *buffer*.

*void ensureCapacity(int minCap)*: Đảm bảo đủ chỗ cho lớt nhất *minCap* ký tự của xâu *buffer*.

*void setLength(int newLen):* Đặt độ dài của xâu *buffer* là *newLen*.

## Bài tập cuối chương

5.1. Có bao nhiêu lớp cơ sở trong Java?

A. 7

B. 8

C. 9

D. 5

5.2. Khối lệnh sau có kết quả bao nhiêu?

*String greeting = “Hello”;*

*int k = greeting.length();*

*System.out.print(k);*

A. 4

B. 5

C. 6

D. 7

5.3. Đoạn mã sau có lỗi biên dịch gì?

*public class Person {*

*private String name;*

*protected int age;*

*public static void main(String[] args) {*

*Person p = new Person();*

*p.name = "Tom";*

*System.out.println(p.name);*

*}*

*}*

A. Không có lỗi biên dịch

B. name has private access.

C. age has protected access.

D. name has public access.

5.5. Chương trình sau in ra màn hình xâu nào?

*public class Main {*

*public static void main(String[] args) {*

*String names[] = {*

*"John",*

*"Anna",*

*"Peter",*

*"Victor",*

*"David"*

*};*

*System.out.println(names[2]);*

*}*

*}*

A. Có lỗi biên dịch: use new keyword to create object

B. Peter

C. Anna

D. Victor

## Bài 9: Mảng và các lớp cơ sở trong gói java.lang (tiếp theo) (Số tiết: 03 tiết)

## 5.3. Một số cấu trúc dữ liệu phổ dụng

### 5.3.1. Enumeration

Interface Enumeration bản thân nó không phải là cấu trúc dữ liệu, nhưng rất quan trọng bên trong ngữ cảnh sử dụng các cấu trúc dữ liệu khác. Interface Enumeration định nghĩa để nhận các thành phần kế tiếp từ cấu trúc dữ liệu.

Các phương thức được khai báo bởi Enumeration:

*boolean hasMoreElements()* Khi được triển khai, nó phải trả về true trong khi vẫn còn nhiều phần tử để extract, và false khi tất cả phần tử đã được liệt kê

*Object nextElement( )* Trả về đối tượng kế tiếp trong Enumeration như là một tham chiếu generic Object

**Ví dụ:**

*import java.util.Vector;*

*import java.util.Enumeration;*

*public class EnumerationTester {*

*public static void main(String args[]) {*

*Enumeration days;*

*Vector dayNames = new Vector();*

*dayNames.add("Sunday");*

*dayNames.add("Monday");*

*dayNames.add("Tuesday");*

*dayNames.add("Wednesday");*

*dayNames.add("Thursday");*

*dayNames.add("Friday");*

*dayNames.add("Saturday");*

*days = dayNames.elements();*

*while (days.hasMoreElements()){*

*System.out.println(days.nextElement());*

*}*

*}*

*}*

Kết quả:

*Sunday*

*Monday*

*Tuesday*

*Wednesday*

*Thursday*

*Friday*

*Saturday*

### 5.3.2. BitSet

Lớp BitSet trong Java triển khai một nhóm các bit hoặc flag mà có thể được thiết lập và xóa một cách riêng rẽ.

Class này rất hữu dụng trong trường hợp bạn muốn lưu trữ một tập các giá trị Boolean và chỉ muốn gắn từng bit các giá trị và thiết lập hoặc xóa nó thích hợp.

BitSet định nghĩa hai constructor:

*BitSet()*: tạo một đối tượng mặc định.

*BitSet(int size)*: cho phép bạn xác định kích cỡ ban đầu của nó, ví dụ như số bit mà nó có thể giữ. Tất cả các bit được khởi tạo về 0.

*BitSet* triển khai *Cloneable* *Interface* và định nghĩa các phương thức được liệt kê sau:

*void and(BitSet bitSet)* AND nội dung của đối tượng BitSet đang triệu hồi với nội dung đã được xác định bởi bitSet. Kết quả được đặt trong đối tượng đang triệu hồi

*void andNot(BitSet bitSet):* Với mỗi bit trong bitSet, bit tương ứng trong BitSet đang triệu hồi bị xóa.

*int cardinality( ):* Trả về số bit được thiết lập trong đối tượng đang triệu hồi

*void clear( ):* Thiết lập tất cả bit về 0

*void clear(int index):* Thiết lập tất cả bit được xác định bởi *index* về 0

*void clear(int startIndex, int endIndex):* Thiết lập tất cả bit được xác định từ *startIndex* tới *endIndex* -1 về 0

*Object clone( ):* Sao chép đối tượng BitSet đang triệu hồi

*boolean equals(Object bitSet):* Trả về *true* nếu BitSet đang triệu hồi là tương đương với thiết lập bit trong bitSet. Nếu không là *false*

*void flip(int index):* Đảo ngược các bit được xác định bởi index

*void flip(int startIndex, int endIndex):* Đảo ngược các bit được xác định từ *startIndex* tới *endIndex* – 1

*BitSet get(int startIndex, int endIndex):* Trả về một BitSet mà chứa các bit từ startIndex tới endIndex-1. Đối tượng đang triệu hồi không bị thay đổi.

*int length( ):* Trả về số bit cần thiết để giữ nội dung của BitSet đang gọi. Giá trị này được xác định bởi vị trí của bit cuối cùng.

*void set(int index):* Thiết lập bit được xác định bởi index.

*int size( ):* Trả về số bit trong đối tượng BitSet đang gọi

*void xor(BitSet bitSet):* XOR nội dung của đối tượng BitSet đang gọi với nội dung được xác định bởi bitSet. Kết quả được đặt trong đối tượng đang gọi.

**Ví dụ:**

*import java.util.BitSet;*

*public class BitSetDemo {*

*public static void main(String args[]) {*

*BitSet bits1 = new BitSet(16);*

*BitSet bits2 = new BitSet(16);*

*// thiet la mot so bit*

*for (int i = 0; i < 16; i++) {*

*if ((i % 2) == 0) {*

*bits1.set(i);*

*}*

*if ((i % 5) != 0) {*

*bits2.set(i);*

*}*

*}*

*System.out.println("Pattern ban dau trong bits1: ");*

*System.out.println(bits1);*

*System.out.println("\nPattern ban dau trong bits2: ");*

*System.out.println(bits2);*

*// AND bits*

*bits2.and(bits1);*

*System.out.println("\nbits2 AND bits1: ");*

*System.out.println(bits2);*

*// OR bits*

*bits2.or(bits1);*

*System.out.println("\nbits2 OR bits1: ");*

*System.out.println(bits2);*

*// XOR bits*

*bits2.xor(bits1);*

*System.out.println("\nbits2 XOR bits1: ");*

*System.out.println(bits2);*

*}*

*}*

Kết quả:

*Pattern ban dau trong bits1:*

*{0, 2, 4, 6, 8, 10, 12, 14}*

*Pattern ban dau trong bits2:*

*{1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14}*

*bits2 AND bits1:*

*{2, 4, 6, 8, 12, 14}*

### 5.3.3. Vector

Lớp Vector trong Java là tương tự như các mảng dữ liệu Java truyền thống, ngoại trừ việc có thể tăng lưu trữ cho các thành phần mới. Giống như mảng, các thành phần trong đối tượng Vector có thể truy cập bởi index.

Lớp Vector trong Java hỗ trợ 4 constructor:

*Vector( ):* Form đầu tiên tạo một vector mặc định, mà có kích cỡ khởi tạo là 10

*Vector(int size):* Form thứ hai tạo một vector mà dung lượng khởi tạo được xác định bởi size.

*Vector(int size, int incr):* Form thứ ba tạo một vector mà dung lượng khởi tạo được xác định bởi size và lượng gia của nó được xác định bởi incr. Lượng gia xác định số phần tử để cấp phát mỗi khi một vector được resize.

*Vector(Collection c):* Form thứ tư tạo một vector mà chứa các phần tử của collection c.

Vector định nghĩa các phương thức sau:

|  |  |
| --- | --- |
| **STT** | **Phương thức và Miêu tả** |
| 1 | **void add(int index, Object element)**  Chèn *element* đã xác định tại vị trí đã cho trong Vector này |
| 2 | **boolean add(Object o)**  Phụ thêm phần tử đã cho vào cuối của Vector này |
| 3 | **boolean addAll(Collection c)**  Thêm tất cả phần tử trong Collection đã xác định vào cuối của Vector này, để mà chúng được trả về bởi Iterator của Collection đã cho đó |
| 4 | **boolean addAll(int index, Collection c)**  Chèn tất cả phần tử trong Collection đã xác định vào trong Vector này tại vị trí đã cho |
| 5 | **void addElement(Object obj)**  Thêm phần tử đã cho tới cuối của Vector này, tăng kích cỡ nó thêm 1 |
| 6 | **int capacity()**  Trả về dung lượng hiện tại của Vector này |
| 7 | **void clear()**  Gỡ bỏ tất cả phần tử từ Vector này |
| 8 | **Object clone()**  Trả về một mô phỏng của Vector này |
| 9 | **boolean contains(Object elem)**  Kiểm tra nếu đối tượng đã cho là một phần tử trong Vector này |
| 10 | **boolean containsAll(Collection c)**  Trả về true nếu Vector này chứa tất cả phần tử trong Collection đã cho |
| 11 | **void copyInto(Object[] anArray)**  Sao chép các thành phần của Vector này vào trong mảng đã cho |
| 12 | **Object elementAt(int index)**  Trả về phần tử tại index đã cho |
| 13 | **Enumeration elements()**  Trả về một bản liệt kê các phần tử của Vector này |
| 14 | **void ensureCapacity(int minCapacity)**  Tăng dung lượng của Vector này, nếu cần thiết, để bảo đảm rằng nó có thể giữ ít nhất số các phần tử được xác định bởi tham số minCapacity |
| 15 | **boolean equals(Object o)**  So sánh Object đã cho với Vector này về sự cân bằng |
| 16 | **Object firstElement()**  Trả về phần tử đầu tiên (tại chỉ mục 0) của Vector này |
| 17 | **Object get(int index)**  Trả về phần tử tại vị trí đã cho trong Vector này |
| 18 | **int hashCode()**  Trả về giá trị hash code cho Vector này |
| 19 | **int indexOf(Object elem)**  Tìm kiếm sự xuất hiện đầu tiên của tham số đã cho, kiểm tra tính cân bằng bởi sử dụng phương thức equals |
| 20 | **int indexOf(Object elem, int index)**  Tìm kiếm sự xuất hiện đầu tiên của tham số đã cho, bắt đầu tìm kiếm tại index, kiểm tra tính cân bằng bởi sử dụng phương thức equals |
| 21 | **void insertElementAt(Object obj, int index)**  Chèn đối tượng đã cho như là một phần tử vào Vector này tại index đã cho |
| 22 | **boolean isEmpty()**  Kiểm tra nếu Vector này không có phần tử |
| 23 | **Object lastElement()**  Trả về phần tử cuối cùng của Vector này |
| 24 | **int lastIndexOf(Object elem)**  Trả về chỉ mục của sự xuất hiện cuối cùng của đối tượng đã cho trong Vector này |
| 25 | **int lastIndexOf(Object elem, int index)**  Tìm kiếm ngược về sau cho đối tượng đã cho, bắt đầu từ index đã xác định, và trả về một chỉ mục |
| 26 | **Object remove(int index)**  Gỡ bỏ phần tử tại vị trí đã cho trong Vector này |
| 27 | **boolean remove(Object o)**  Gỡ bỏ sự xuất hiện đầu tiên của phần tử đã cho trong Vector này. Nếu Vector này không chứa phần tử đó, nó không bị thay đổi |
| 28 | **boolean removeAll(Collection c)**  Gỡ bỏ tất cả phần tử, mà chứa trong Collection đã cho, từ Vector này |
| 29 | **void removeAllElements()**  Gỡ bỏ tất cả phần tử từ Vector này và thiết lập kích cỡ về 0 |
| 30 | **boolean removeElement(Object obj)**  Gỡ bỏ sự xuất hiện đầu tiên (chỉ mục thấp nhất) của tham số từ Vector này |
| 31 | **void removeElementAt(int index)**  removeElementAt(int index) |
| 32 | **protected void removeRange(int fromIndex, int toIndex)**  Gỡ bỏ từ danh sách này tất cả phần tử mà có index từ fromIndex tới toIndex |
| 33 | **boolean retainAll(Collection c)**  Chỉ giữ lại phần tử, mà ở trong Collection đã cho, trong Vector này |
| 34 | **Object set(int index, Object element)**  Thay thế phần tử tại vị trí đã cho trong Vector này với phần tử đã xác định |
| 35 | **void setElementAt(Object obj, int index)**  Thiết lập phần tử tại index đã cho của Vector này thành đối tượng đã xác định |
| 36 | **void setSize(int newSize)**  Thiết lập kích cỡ của Vector này |
| 37 | **int size()**  Trả về số phần tử trong Vector này |
| 38 | **List subList(int fromIndex, int toIndex)**  Trả về một danh sách phụ từ fromIndex tới toIndex |
| 39 | **Object[] toArray()**  Trả về một mảng chứa tất cả phần tử trong Vector này theo đúng thứ tự |
| 40 | **Object[] toArray(Object[] a)**  Trả về một mảng chứa tất cả phần tử trong Vector này theo đúng thứ tự; kiểu runtime của mảng trả về là mảng đã xác định |
| 41 | **String toString()**  Trả về một biểu diễn chuỗi của Vector này, chứa biểu diễn chuỗi của mỗi phần tử |
| 42 | **void trimToSize()**  Trim dung lượng của Vector này về kích cỡ hiện tại của vector |

**Ví dụ:**

*import java.util.\*;*

*public class VectorDemo {*

*public static void main(String args[]) {*

*// capacity ban dau la 3, incr la 2*

*Vector v = new Vector(3, 2);*

*System.out.println("Size ban dau: " + v.size());*

*System.out.println("Capacity ban dau: "+ v.capacity());*

*v.addElement(new Integer(1));*

*v.addElement(new Integer(2));*

*v.addElement(new Integer(3));*

*v.addElement(new Integer(4));*

*System.out.println("Capacity sau 4 lan cong la: "+ v.capacity());*

*v.addElement(new Double(5.45));*

*System.out.println("Capacity hien tai: "+ v.capacity());*

*v.addElement(new Double(6.08));*

*v.addElement(new Integer(7));*

*System.out.println("Capacity hien tai: "+ v.capacity());*

*v.addElement(new Float(9.4));*

*v.addElement(new Integer(10));*

*System.out.println("Capacity hien tai: "+ v.capacity());*

*v.addElement(new Integer(11));*

*v.addElement(new Integer(12));*

*System.out.println("Phan tu dau tien: "+ (Integer) v.firstElement());*

*System.out.println("Phan tu cuoi cung: "+ (Integer) v.lastElement());*

*if (v.contains(new Integer(3))) {*

*System.out.println("Vector chua 3.");*

*}*

*// tinh toan so phan tu trong vector.*

*Enumeration vEnum = v.elements();*

*System.out.println("\nCac phan tu trong Vector:");*

*while (vEnum.hasMoreElements()) {*

*System.out.print(vEnum.nextElement() + " ");*

*}*

*System.out.println();*

*}*

*}*

**Kết quả:**

*Size ban dau: 0*

*Capacity ban dau: 3*

*Capacity sau 4 lan cong la: 5*

*Capacity hien tai: 5*

*Capacity hien tai: 7*

*Capacity hien tai: 9*

*Phan tu dau tien: 1*

*Phan tu cuoi cung: 12*

*Vector chua 3.*

*Cac phan tu trong Vector:*

*1 2 3 4 5.45 6.08 7 9.4 10 11 12*

### 5.3.4. Stack

Lớp Stack trong Java triển khai một last-in-first-out (LIFO) stack các phần tử.

Stack chỉ định nghĩa constructor mặc định, mà tạo một stack trống. Lớp Stack bao gồm tất cả phương thức được định nghĩa bởi lớp Vector, và một số phương thức khác của riêng nó.

Lớp Stack định nghĩa các phương thức sau:

|  |  |
| --- | --- |
| **STT** | **Phương thức và Miêu tả** |
| 1 | **boolean empty()**  Kiểm tra nếu Stack này là trống. Trả về true nếu nó trống và false nếu stack chứa các phần tử |
| 2 | **Object peek( )**  Trả về phần tử trên cùng của Stack, nhưng không gỡ bỏ nó |
| 3 | **Object pop( )**  Trả về phần tử trên cùng của Stack, gỡ bỏ nó |
| 4 | **Object push(Object element)**  Đẩy phần tử lên trên cùng của Stack. Cũng trả về phần tử đó |
| 5 | **int search(Object element)**  Tìm kiếm phần tử trong Stack. Nếu tìm thấy, offset của nó từ trên cùng của Stack được trả về. Nếu không, nó trả về -1 |

**Ví dụ:**

*import java.util.\*;*

*public class StackDemo {*

*static void showpush(Stack st, int a) {*

*st.push(new Integer(a));*

*System.out.println("push(" + a + ")");*

*System.out.println("stack: " + st);*

*}*

*static void showpop(Stack st) {*

*System.out.print("pop -> ");*

*Integer a = (Integer) st.pop();*

*System.out.println(a);*

*System.out.println("stack: " + st);*

*}*

*public static void main(String args[]) {*

*Stack st = new Stack();*

*System.out.println("stack: " + st);*

*showpush(st, 42);*

*showpush(st, 66);*

*showpush(st, 99);*

*showpop(st);*

*showpop(st);*

*showpop(st);*

*try {*

*showpop(st);*

*} catch (EmptyStackException e) {*

*System.out.println("empty stack");*

*}*

*}*

*}*

**Kết quả:**

*stack: [ ]*

*push(42)*

*stack: [42]*

*push(66)*

*stack: [42, 66]*

*push(99)*

*stack: [42, 66, 99]*

*pop -> 99*

*stack: [42, 66]*

*pop -> 66*

*stack: [42]*

*pop -> 42*

*stack: [ ]*

*pop -> empty stack*

## 5.4. Một số bài tập về cấu trúc dữ liệu

Bài 1: Đảo ngược văn bản

Cho một dòng văn bản, đảo ngược văn bản mà không đảo ngược các từ riêng lẻ.

Ví dụ:

Input: Chúng tôi cung cấp tài liệu tốt để chuẩn bị cho các bạn sinh viên CNTT

Output: CNTT viên sinh bạn các cho bị chuẩn để tốt liệu tài cấp cung tôi Cấp

Gợi ý: Đẩy các từ riêng lẻ bắt đầu từ đầu văn bản vào một ngăn xếp. Cuối cùng, lấy tất cả các từ từ ngăn xếp và lưu trữ chúng trở lại văn bản theo thứ tự LIFO. Độ phức tạp thời gian của giải pháp này là O (n) và không gian phụ được chương trình sử dụng là O (n) cho ngăn xếp.

Bài 2: Tìm dấu ngoặc đơn trùng lặp trong một biểu thức

Đưa ra một biểu thức cân bằng có thể chứa dấu ngoặc mở và đóng, hãy kiểm tra xem biểu thức có chứa bất kỳ dấu ngoặc đơn trùng lặp nào hay không.

Ví dụ:

Input: ((x+y))+z

Output: Trùm lặp () được tìm thấy trong biểu thức con ((x + y))

Input: (x+y)

Output: Không thấy trùng lặp ()

Input: ((x+y)+((z)))

Output: Trùng lặp () được tìm thấy trong biểu thức con ((z))

Gợi ý: Chúng ta có thể sử dụng một ngăn xếp để giải quyết vấn đề này. Ý tưởng:

Nếu ký tự hiện tại trong biểu thức không phải là một dấu ngoặc đóng, thì đẩy ký tự vào ngăn xếp.

Nếu ký tự hiện tại trong biểu thức là một dấu ngoặc đóng, ta sẽ kiểm tra xem phần tử trên cùng trong ngăn xếp có phải là một dấu ngoặc mở hay không. Nếu nó đang là ngoặc đơn mở, thì biểu thức con kết thúc ở ký tự hiện tại có dạng ((exp)), nếu không, tiếp tục xuất các ký tự từ ngăn xếp cho đến khi tìm thấy kết quả phù hợp cho hiện tại

## Bài tập cuối chương

5.6. Khối lệnh thực hiện chuyện gì?

*int[] mang = new int[10];*

*int s = 0;*

*int j = 0;*

*for (int i: mang) {*

*i = ++j;*

*s += i;*

*}*

*System.out.println(“S = “ + s);*

A. Nhập dữ liệu cho mang nguyên.

B. Tính tổng mảng nguyên.

C. Nhập dữ liệu và tính tổng mảng nguyên.

D. Tất cả đều sai.

5.7. Phát biểu nào sau đây là đúng:

A. Mảng có thể lưu giữ các phần tử thuộc nhiều kiểu dữ liệu khác nhau

B. Chỉ số của mảng có thể sử dụng kiểu số thực (float, double)

C. Biểu thức array.length được sử dụng để trả về số phần tử trong mảng

D. Một phần tử của mảng không thể truyền vào trong một phương thức.

5.8. Phát biểu sau nào là SAI

A. Mảng có kích thước không đổi trong toàn bộ chương trình

B. Mảng là cấu trúc dữ liệu có khả năng lưu trữ nhiều thành phần (phần tử) dữ liệu với kiểu khác nhau

C. Mảng N phần tử được đánh chỉ số từ 0 đến N - 1

D. Chỉ số mảng bắt đầu bằng 0

5.9. Cho khai báo mảng sau. Lệnh trên in ra cái gì?

*String[][] str = {"lap", "trinh", "java"};*

*System.out.println(str[1][1]);*

A. trinh

B. java

C. Lỗi biên dịch ở dòng 1

D. Không in ra gì

# Chương VI: CÁC LUỒNG VÀO RA DỮ LIỆU CƠ BẢN

### Nội dung chính của chương

* Các luồng vào/ra
* Lớp File
* Truy cập tệp tuần tự và truy cập tệp ngẫu nhiên

### Mục tiêu cần đạt được của chương

Giúp sinh viên sẽ nắm được khái niệm stream để làm cho hoạt động I/O nhanh hơn. Gói java.io chứa tất cả các lớp cần thiết cho hoạt động input và output. Các lớp *OutputStream* và *InputStream* trong Java.

## Bài 10: Các luồng vào ra dữ liệu cơ bản (Số tiết: 03 tiết)

## 6.1. Khái niệm

### 6.1.1. Khái niệm về stream

Một stream là một dãy dữ liệu. Trong java, một stream bao gồm các byte và có 3 stream được tạo cho chúng ta một cách tự động. Tất cả các stream này được gắn với console.

1) *System.out:* output stream tiêu chuẩn

2) *System.in:* input stream tiêu chuẩn

3) *System.err:* error stream tiêu chuẩn

**Ví dụ:** in thông báo *output* và *error* tới console.

System.out.println("simple message");

System.err.println("error message");

**Ví dụ:** lấy giá trị *input* từ console.

int i=System.in.read();//tra ve ma ASCII cua ky tu dau tien

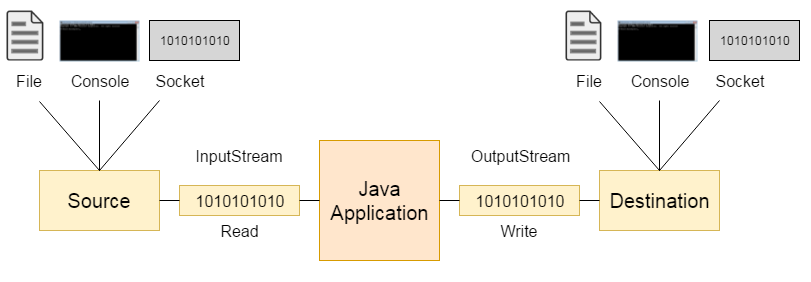
System.out.println((char)i);//in ky tu lay duoc ra man hinh

### 6.1.2. Các lớp OutputStream và InputStream

**OutputStream** Ứng dụng Java sử dụng một output stream để ghi dữ liệu đến đích, nó có thể là một tệp tin, một mảng, thiết bị ngoại vi hoặc socket.

**InputStream** Ứng dụng Java sử dụng một input stream để đọc dữ liệu từ một nguồn, nó có thể là một tệp tin, một mảng, thiết bị ngoại vi hoặc socket.

Hoạt động của của Java OutputStream và InputStream được mô tả trong hình dưới đây.



Hình 6.1 OutputStream và InputStream

## 6.2. Các luồng vào/ra

### 6.2.1. Standard Stream trong Java

**Đầu vào chuẩn (Standard Input):** Nó được sử dụng để truyền dữ liệu tới chương trình của người dùng và thường thì một bàn phím được sử dụng như là đầu vào chuẩn và được biểu diễn như *System.in*.

**Đầu ra chuẩn (Standard Output):** Nó được sử dụng để hiển thị kết quả đầu ra từ chương trình của người dùng và thường thì một màn hình máy tính được sử dụng như là đầu ra chuẩn và được biểu diễn như là *System.out*.

**Lỗi chuẩn (Standard Error):** Được sử dụng để hiển thị các lỗi trong chương trình của người dùng và thường thì một màn hình máy tính được sử dụng như là lỗi chuẩn và được biểu diễn như là *System.err*.

Ví dụ một chương trình đơn giản tạo *InputStreamReader* để đọc luồng đầu vào chuẩn tới khi người sử dụng gõ một "q".

*import java.io.\*;*

*public class ReadConsole {*

*public static void main(String args[]) throws IOException*

*{*

*InputStreamReader cin = null;*

*try {*

*cin = new InputStreamReader(System.in);*

*System.out.println("Nhap cac ky tu, 'q' de thoat.");*

*char c;*

*do {*

*c = (char) cin.read();*

*System.out.print(c);*

*} while(c != 'q');*

*}finally {*

*if (cin != null) {*

*cin.close();*

*}*

*}*

*}*

*}*

Giữ code trên trong ReadConsole.java file và thực thi và biên dịch nó như dưới đây. Chương trình này tiếp tục đọc và hiển thị kết quả tới khi chúng ta nhấn phím "q".

*$javac ReadConsole.java*

*$java ReadConsole*

*Nhap cac ky tu, 'q' de thoat.*

*1*

*1*

*e*

*e*

*q*

*q*

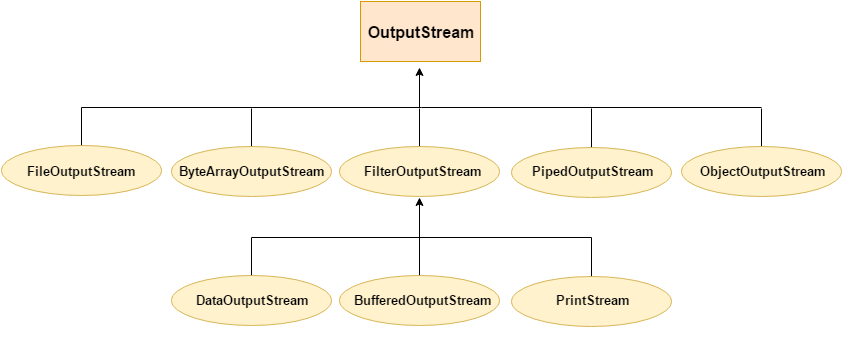
### 6.2.2. Lớp OutputStream

Lớp OutputStream là một lớp trừu tượng. Nó là super class của tất cả các lớp đại diện cho một output stream của các byte. Một output stream chấp nhận ouput các byte và gửi chúng đến một nơi có thể chứa.

Các phương thức của lớp OutputStream:

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public void write(int)throws IOException | được sử dụng để ghi một byte đến output stream hiện tại. |
| 2) public void write(byte[])throws IOException | được sử dụng để ghi một mảng các byte đến output stream hiện tại. |
| 3) public void flush()throws IOException | flush output stream hiện tại. |
| 4) public void close()throws IOException | được sử dụng để đóng output stream hiện tại. |

**OutputStream Hierarchy:**



Hình 6.2 OutputStream Hierarchy

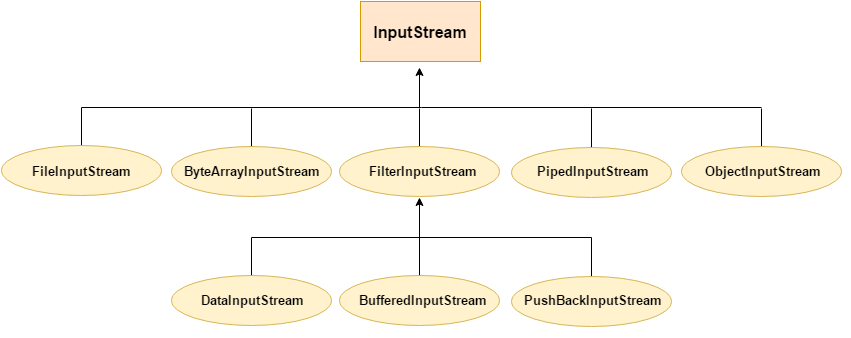
### 6.2.3. Lớp InputStream

Lớp InputStream là một lớp trừu tượng. Nó là super class của tất cả các lớp đại diện cho một input stream của các byte.

Các phương thức của lớp InputStream

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public abstract int read() throws IOException | Đọc byte kế tiếp của dữ liệu từ input stream. Nó trả về -1 khi đọc đến vị trí cuối tập tin. |
| 2) public int available() throws IOException | Trả về một ước tính về số byte có thể đọc được từ input stream hiện tại. |
| 3) public void close() throws IOException | được sử dụng để đóng input stream hiện tại. |

**InputStream Hierarchy**



Hình 6.3 InputStream Hierarchy

**Ví dụ:**

*import java.io.\*;*

*public class CopyFile {*

*public static void main(String args[]) throws IOException*

*{*

*FileInputStream in = null;*

*FileOutputStream out = null;*

*try {*

*in = new FileInputStream("input.txt");*

*out = new FileOutputStream("output.txt");*

*int c;*

*while ((c = in.read()) != -1) {*

*out.write(c);*

*}*

*}finally {*

*if (in != null) {*

*in.close();*

*}*

*if (out != null) {*

*out.close();*

*}*

*}*

*}*

*}*

Giả sử chúng ta có một file là *input.txt* có nội dung sau:

*Day la vi du ve sao chep file.*

Trong bước tiếp theo, biên dịch chương trình trên và thực thi nó, sẽ cho kết quả là tạo một file là output.txt có cùng nội dung như chúng ta có trong input.txt. Vì thế, bạn đặt code trên vào trong CopyFile.java file và làm như sau:

*$javac CopyFile.java*

*$java CopyFile*

## 6.3. Lớp File

Lớp File trong Java biểu diễn các pathname của các file và thư mục theo phương thức trừu tượng. Lớp này được sử dụng để tạo các file và thư mục, để tìm kiếm các file, xóa các file, …

Đối tượng File biểu diễn file/thư mục thực sự trên đĩa. Có các constructor sau để tạo một đối tượng File trong Java.

*File(File parent, String child);* Sẽ tạo một đối tượng File mới từ một pathname trừu tượng cha và một chuỗi pathname con.

*File(String parent, String child);* Sẽ tạo một đối tượng File mới bởi việc biến đổi URI đã cho thành một pathname trừu tượng.

*File(URI uri);* Sẽ tạo một đối tượng File mới bởi việc biến đổi URI đã cho thành một pathname trừu tượng.

Các phương thức đối với đối tượng **File:**

|  |
| --- |
| **public String getName()**  Trả về tên của file hoặc thư mục đã được biểu thị bởi pathname trừu tượng này |
| **public String getParent()**  Trả về chuỗi pathname của pathname trừu tượng cha, hoặc null nếu pathname này không là tên của một thư mục cha |
| **public File getParentFile()**  Trả về pathname trừu tượng của pathname trừu tượng cha, hoặc null nếu pathname này không là tên của một thư mục cha |
| **public String getPath()**  Biến đổi pathname trừu tượng này thành một chuỗi pathname |
| **public boolean isAbsolute()**  Kiểm tra có hay không pathname trừu tượng này là absolute. Trả về true nếu là absolute, nếu không là false |
| **public String getAbsolutePath()**  Trả về chuỗi pathname tuyệt đối của pathname trừu tượng này |
| **public boolean canRead()**  Kiểm tra có hay không ứng dụng có thể đọc file, được biểu thị bởi pathname trừu tượng này. Trả về true nếu và chỉ nếu file đã xác định bởi pathname trừu tượng này là tồn tại và có thể được đọc từ ứng dụng; nếu không là false |
| **public boolean canWrite()**  Kiểm tra có hay không ứng dụng có thể sửa đổi file, được biểu thị bởi pathname trừu tượng này. Trả về true nếu và chỉ nếu hệ thống file thực sự chứa một file được biểu thị bằng pathname trừu tượng này và ứng dụng được phép write tới file đó; nếu không là false |
| **public boolean exists()**  Kiểm tra có hay không file hoặc thư mục được biểu thị bởi pathname trừu tượng này là tồn tại. Trả về true nếu và chỉ nếu file hoặc thư mục được biểu thị bởi pathname trừu tượng này là tồn tại; nếu không là false |
| **public boolean isDirectory()**  Kiểm tra có hay không file được biểu thị bởi pathname trừu tượng này là một thư mục. Trả về true nếu và chỉ nếu file được biểu thị bởi pathname trừu tượng này là tồn tại và là một thư mục; nếu không là false |
| **public boolean isFile()**  Kiểm tra có hay không file được biểu thị bởi pathname trừu tượng này là một normal file. Một file là normal nếu nó không là một thư mục và ngoài ra, thỏa mãn các tiêu chuẩn khác phụ thuộc vào hệ thống. Bất kỳ một non-directory file được tạo bởi một ứng dụng Java được bảo đảm là một normal file. Trả về true nếu và chỉ nếu file được biểu thị bởi pathname trừu tượng này là một normal file; nếu không là false |
| **public long lastModified()**  Trả về time mà file được biểu thị bởi pathname trừu tượng này được sửa đổi lần cuối cùng. Trả về giá trị long biểu diễn thời gian mà file được sửa đổi lần cuối cùng, được tính toán bằng mili giây từ 00:00:00 GMT, 1/1/1970, hoặc 0L nếu file không tồn tại hoặc nếu một I/O error xuất hiện |
| **public long length()**  Trả về độ dài của file được biểu thị bởi pathname trừu tượng này. Giá trị trả về là không rõ ràng nếu pathname này biểu thị một thư mục |
| **public boolean createNewFile() throws IOException**  Tạo một file mới, trống với tên pathname trừu tượng này nếu và chỉ nếu một file với tên này chưa tồn tại. Trả về true nếu và chỉ nếu file với tên này không tồn tại và được tạo thành công; nếu file với tên này đã tồn tại, thì trả về false |
| **public boolean delete()**  Xóa file hoặc thư mục được biểu thị bởi pathname trừu tượng này. Nếu pathname này biểu thị một thư mục, thì thư mục phải là trống để bị xóa. Trả về true nếu và chỉ nếu file hoặc thư mục bị xóa thành công; nếu không là false |
| **public void deleteOnExit()**  Yêu cầu rằng file hoặc thư mục được biểu thị bởi pathname trừu tượng này để bị xóa khi thiết bị ảo kết thúc |
| **public String[] list()**  Trả về một mảng các chuỗi chỉ các file và thư mục trong thư mục được biểu thị bởi pathname trừu tượng này |
| **public String[] list(FilenameFilter filter)**  Trả về một mảng các chuỗi chỉ các file và thư mục trong thư mục được biểu thị bởi pathname trừu tượng này mà thỏa mãn filter đã cho |
| **public File[] listFiles()**  Trả về một mảng các pathname trừu tượng biểu thị các file trong thư mục được biểu thị bởi pathname trừu tượng này |
| **public File[] listFiles(FileFilter filter)**  Trả về một mảng các pathname trừu tượng biểu thị các file và thư mục trong thư mục được biểu thị bởi pathname trừu tượng này mà thỏa mãn filter đã cho |
| **public boolean mkdir()**  Tạo thư mục được chỉ bởi pathname trừu tượng này. Trả về true nếu và chỉ nếu thư mục được tạo; nếu không là false |
| **public boolean mkdirs()**  Tạo thư mục được chỉ bởi pathname trừu tượng này, bao gồm bất kỳ thư mục cha nào cần thiết nhưng không tồn tại. Trả về true nếu và chỉ nếu thư mục được tạo, cùng với tất cả thư mục cha cần thiết; nếu không là false |
| **public boolean renameTo(File dest)**  Đổi tên file được biểu thị bởi pathname trừu tượng này. Trả về true nếu và chỉ nếu việc đổi tên thành công; nếu không là false |
| **public boolean setLastModified(long time)**  Thiết lập thời gian chỉnh sửa cuối cùng của file hoặc thư mục được chỉ bởi pathname trừu tượng này. Trả về true nếu và chỉ nếu hoạt động này thành công; nếu không là false |
| **public boolean setReadOnly()**  Đánh dấu file hoặc thư mục được chỉ bởi pathname trừu tượng này để mà các hoạt động read-only được cho phép. Trả về true nếu hoạt động này thành công; nếu không là false |
| **public static File createTempFile(String prefix, String suffix, File directory) throws IOException**  Tạo một file trống trong thư mục đã cho, bởi sử dụng các chuỗi tiền tố và hậu tố đã cung cấp để tạo tên của nó. Trả về pathname trừu tượng biểu thị một file trống mới được tạo |
| **public static File createTempFile(String prefix, String suffix) throws IOException**  Tạo một file trống trong thư mục temporary-file, bởi sử dụng tiền tố và hậu tố đã cung cấp để tạo tên của nó. Triệu hồi phương thức này tương đương với gọi phương thức createTempFile(prefix, suffix, null). Trả về pathname trừu tượng biểu thị một file trống mới được tạo |
| **public int compareTo(File pathname)**  So sánh 2 pathname trừu tượng theo từ điển. Trả về 0 nếu tham số là cân bằng với pathname trừu tượng này, một giá trị nhỏ hơn 0 nếu pathname trừu tượng này nhỏ hơn tham số theo từ điển, hoặc một giá trị lớn hơn 0 nếu pathname trừu tượng này lớn hơn tham số theo từ điển |
| **public int compareTo(Object o)**  So sánh pathname trừu tượng này với đối tượng khác. Trả về 0 nếu tham số là cân bằng với pathname trừu tượng này, một giá trị nhỏ hơn 0 nếu pathname trừu tượng nhỏ hơn tham số theo từ điển, hoặc một giá trị lớn hơn 0 nếu pathname trừu tượng này lớn hơn tham số theo từ điển |
| **public boolean equals(Object obj)**  Kiểm tra pathname trừu tượng này có cân bằng với đối tượng đã cung cấp không. Trả về true nếu và chỉ nếu tham số là không null và là một pathname trừu tượng mà biểu thị file hoặc thư mục giống như với pathname trừu tượng này |
| **public String toString()**  Trả về chuỗi pathname của pathname trừu tượng này. Đây là chuỗi được trả về bởi phương thức getPath() |

**Ví dụ:**

*package com.tutorialspoint;*

*import java.io.File;*

*public class FileDemo {*

*public static void main(String[] args) {*

*File f = null;*

*String[] strs = {"test1.txt", "test2.txt"};*

*try{*

*// voi moi string trong mang string*

*for(String s:strs )*

*{*

*// tao file moi*

*f= new File(s);*

*// true neu file la executable*

*boolean bool = f.canExecute();*

*// tim absolute path*

*String a = f.getAbsolutePath();*

*// in absolute path*

*System.out.print(a);*

*// prints*

*System.out.println(" la executable: "+ bool);*

*}*

*}catch(Exception e){*

*// neu co bat cu I/O error nao xuat hien*

*e.printStackTrace();*

*}*

*}*

*}*

Giả sử có một file có thể thực thi là *test1.txt* và file khác là *test2.txt* là không thể thực thi trong thư mục hiện tại. Bây giờ chúng ta biên dịch và chạy chương trình trên, nó sẽ cho kết quả sau:

*test1.txt la executable: true*

*test2.txt la executable: false*

## 6.4. Truy cập tệp tuần tự và truy cập tệp ngẫu nhiên

### 6.4.1. Truy cập tệp tuần tự

**Lớp DataInputStream** trong java cho phép một ứng dụng đọc dữ liệu nguyên thủy từ luồng đầu vào một cách độc lập với máy.

Khai báo:

public class DataInputStream extends FilterInputStream implements DataInput

Các phương thức của lớp *DataInputStream:*

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| int read(byte[] b) | Nó được sử dụng để đọc mảng byte từ input stream. |
| int read(byte[] b, int off, int len) | Nó được sử dụng để đọc **len**byte dữ liệu bắt đầu từ **off**từ input stream. |
| int readInt() | Nó được sử dụng để đọc các byte đầu vào và trả về một giá trị int. |
| byte readByte() | Nó được sử dụng để đọc và trả lại một byte đầu vào. |
| char readChar() | Nó được sử dụng để đọc hai byte đầu vào và trả về một giá trị char. |
| double readDouble() | Nó được sử dụng để đọc tám byte đầu vào và trả về một giá trị double. |
| boolean readBoolean() | Nó được sử dụng để đọc một byte đầu vào và trả về true nếu byte khác zero, false nếu byte là zero. |
| int skipBytes(int x) | Nó được sử dụng để bỏ qua x các byte dữ liệu từ input stream. |
| String readUTF() | Nó được sử dụng để đọc một chuỗi đã được mã hóa bằng cách sử dụng định dạng UTF-8. |
| void readFully(byte[] b) | Nó được sử dụng để đọc byte từ input stream và lưu trữ chúng vào mảng đệm. |
| void readFully(byte[] b, int off, int len) | Nó được sử dụng để đọc **len**byte từ vị trí **off**từ input stream. |

**Ví dụ:**

*import java.io.DataInputStream;*

*import java.io.FileInputStream;*

*import java.io.IOException;*

*import java.io.InputStream;*

*public class DataInputStreamExample {*

*public static void main(String[] args) throws IOException {*

*InputStream input = null;*

*DataInputStream dis = null;*

*try {*

*input = new FileInputStream("D:\\testout.txt");*

*dis = new DataInputStream(input);*

*int count = input.available();*

*byte[] arr = new byte[count];*

*dis.read(arr);*

*for (byte bt : arr) {*

*char k = (char) bt;*

*System.out.print(k + "-");*

*}*

*} catch (IOException ex) {*

*} finally {*

*dis.close();*

*}*

*}*

*}*

Giả sử file "*testout.txt*" có nội dung: *JAVA*

Output: *J-A-V-A*

**Lớp DataOutputStream** trong java cho phép một ứng dụng ghi các kiểu dữ liệu Java nguyên thủy đến output stream một cách độc lập với máy. Ứng dụng Java thường sử dụng DataOutputStream để ghi dữ liệu mà sau này có thể được đọc bởi một DataInputStream.

**Khai báo** của lớp *DataOutputStream*

public class DataOutputStream extends FilterOutputStream implements DataOutput

**Các phương thức** của lớp *DataOutputStream*

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| int size() | Nó được sử dụng để trả về số byte đã được ghi. |
| void write(int b) | Nó được sử dụng để ghi các byte được chỉ định cho output stream. |
| void write(byte[] b, int off, int len) | Nó được sử dụng để ghi **len**byte dữ liệu bắt đầu từ **off** vào output stream. |
| void writeBoolean(boolean v) | được sử dụng để ghi Boolean đến output stream như một giá trị 1 byte. |
| void writeChar(int v) | được sử dụng để ghi char đến output stream như một giá trị 2 byte. |
| void writeChars(String s) | được sử dụng để ghi chuỗi cho output stream như là một dãy các ký tự. |
| void writeByte(int v) | được sử dụng để ghi byte đến output stream như một giá trị 1 byte. |
| void writeBytes(String s) | được sử dụng để ghi chuỗi đến output stream như là một dãy các byte. |
| void writeInt(int v) | được sử dụng để ghi int đến output stream |
| void writeShort(int v) | được sử dụng để ghi short đến output stream. |
| void writeLong(long v) | được sử dụng để ghi long đến output stream. |
| void writeUTF(String str) | được sử dụng để ghi một chuỗi đến output stream sử dụng mã hóa UTF-8 theo cách di động. |
| void flush() | được sử dụng để xả DataOutputStream. |

**Ví dụ:**

*import java.io.DataOutputStream;*

*import java.io.FileOutputStream;*

*import java.io.IOException;*

*public class DataOutputStreamExample {*

*public static void main(String[] args) throws IOException {*

*FileOutputStream file = null;*

*DataOutputStream data = null;*

*try {*

*file = new FileOutputStream("D:\\testout.txt");*

*data = new DataOutputStream(file);*

*data.writeInt(65);*

*data.flush();*

*System.out.println("Succcess...");*

*} catch (IOException ex) {*

*} finally {*

*data.close();*

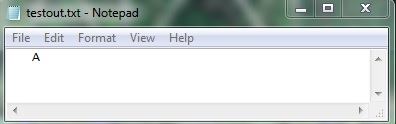
*}*

*}*

*}*

Output: Succcess...

Nội dung file *testout.txt*:



### 6.4.2. Truy cập tệp ngẫu nhiên

Sử dụng *InputStream*, *OutputStream*, *Reader* và *Writer* có một hạn chế là chỉ cho phép đọc ghi dữ liệu tuần tự, vị trí bắt đầu đọc ghi ở đầu tập tin. Vì vậy Java cung cấp *RandomAccessFile* cho phép chúng ta đọc ghi dữ liệu ngẫu nhiên ở bất kỳ vị trí nào của tập tin.

Khởi tạo *RandomAccessFile:* Để khởi tạo RandomAccessFile chúng ta có 2 *constructor* sau

*public RandomAccessFile(String name, String mode)*

*RandomAccessFile randomAccessFile = new RandomAccessFile("/Users/nguyenthanhhai/Desktop/test.txt", "r");*

*public RandomAccessFile(File file, String mode)*

*RandomAccessFile randomAccessFile = new RandomAccessFile(new File("/Users/nguyenthanhhai/Desktop/test.txt"), "rw");*

Tham số *mode* để chỉ định các quyền chúng ta có thể thao tác với tập tin. Danh sách các mode:

* r – chỉ đọc, nếu gọi write() method ở mode này thì chúng sẽ bị ném *IOException*.
* rw – đọc và ghi.
* rwd – đọc ghi đồng bộ. Tất cả các thay đổi sẽ được ghi xuống ổ đĩa đồng bộ.
* rws – đọc ghi đồng bộ. Tất cả các thay đổi kể cả siêu dữ liệu sẽ được ghi xuống ổ đĩa đồng bộ.

RamdomAccessFile cung cấp hầu hết các phương thức đọc file giúp chúng ta thao tác thuận tiện hơn. Các phương thức hỗ trợ đọc dữ liệu nguyên thuỷ: read(), readInt(), readByte(), readDouble(), readFloat(), readChar(), readLong(), …

Các phương thức đọc:

* *readLine():* đọc từng dòng trong RandomAccessFile.
* *read(byte[] b)* và *read(byte[] b, int off, int len):* đọc dữ liệu vào một mảng byte. Lưu ý: 2 phương thức trên trả về số byte đọc được từ tập tin hoặc -1 nếu đã đọc đến cuối tập tin.
* *length():* lấy độ dài của file(đơn vị byte), hay nói cách khác length() trả về số byte mà chúng ta có thể đọc được.
* *seek():* dịch chuyển con trỏ đến bất kỳ vị trí nào của file. Ví dụ mình sẽ đọc 10 byte cuối của file text.

Tương tự như phần đọc file, chúng ta cũng có cá method: write(), writeInt(), writeByte(), writeDouble(), writeFloat(), writeChar(), writeLong(), … để ghi các dữ liệu kiểu nguyên thuỷ vào file.

**Lưu ý:**

* Phải chọn mode được quyền ghi file như rw, rwd, rws.
* Dữ liệu sẽ được ghi xuống tại vị trí con trỏ đang đứng.
* Vị trí con trỏ luôn là khi vừa khởi tạo, nếu lúc này chúng ta ghi thì dữ liệu sẽ được ghi ở đầu file.
* Nếu vị trí con trỏ nằm giữa file thì dữ liệu tại đó sẽ bị ghi đè bởi dữ liệu mới.
* Sử dụng seek() để dịch chuyển vị trí con trỏ nếu không muốn dữ liệu bị ghi đè.
* *RandomAccessFile* cho phép chúng ta đọc ghi ở bất kỳ vị trí nào trong file. Khi sử dụng *write*() method chú ý kết hợp *seek*() để dịch chuyển vị trí con trỏ để tránh ghi đè dữ liệu.
* Các *mode* là quan trọng nếu ta không cấp đủ quyền thì khi thực hiện sẽ bị *IOException*.

# Tài liệu tham khảo

Tài liệu tham khảo tiếng Việt

[1] Bộ môn Công nghệ Phần mềm, Khoa Công nghệ thông tin, ĐH Công nghệ Thông tin và Truyền thông, ĐH Thái nguyên *(2018), Bài giảng Lập trình Java.*

[2] Đoàn Văn Ban (2005), *Lập trình Hướng đối tượng với Java*, NXB Khoa học kỹ thuật.

[3] Đoàn Văn Ban (1997), *Phân tích thiết kế và lập trình hướng đối tượng*, NXB Thống kê.

[4] Phương Lan (Chủ biên), *Java (các tập 1 -> 5),* NXB Lao động Xã hội.

Tài liệu tham khảo tiếng nước ngoài

[5] Herbert Schildt (2014), *Java: A beginer’s guide, Sixth Edition,* Oracle Press.

[6] Paul Deitel and Harvey Deitel (2012), *Java How to Program – 9th Edition*, Deitel & Associates, Inc.

# Các câu hỏi thường gặp

1. Tính đa hình là gì?

Câu trả lời:

Tính đa hình được mô tả ngắn gọn là “One interface, many implementations”. Nó là một đặc điểm của việc có thể gán một ý nghĩa hoặc cách sử dụng khác cho một thứ gì đó trong các ngữ cảnh khác nhau, cụ thể là cho phép một thực thể chẳng hạn như một biến, một hàm hoặc một đối tượng có nhiều hơn một dạng.

Có hai loại đa hình:

Đa hình lúc compile time: thể hiện qua method overloadding (nạp chồng phương thức)

Đa hình lúc runtime: thể hiện qua kế thừa và interface

2. Tính trừu tượng trong Java là gì?

Câu trả lời:

Tính trừu tượng đề cập đến việc xử lý các ý tưởng hơn là sự kiện. Về cơ bản, nó giải quyết việc ẩn các chi tiết và hiển thị những thứ cần thiết cho người dùng. Vì vậy, bạn có thể nó rằng tính từu tượng trong Java là quá trình ẩn các chi tiết triển khai khỏi người dùng và chỉ lộ chi tiết chức năng cho họ.

Có thể đạt được tính trừu tượng theo hai cách:

* Sử dụng lớp abstract
* Sử dụng interface

3. Sự khác biệt giữa class abstract và interface ?

Câu trả lời:

| **Abstract class** | **Interface** |
| --- | --- |
| Không hỗ trợ đa kế thừa | Một class có thể implements nhiều interface |
| Có thể định nghĩa thân của phương thức, property | Không thể định nghĩa code xử lý, chỉ có thể khai báo |
| Có thể các định phạm vi truy cập | Mọi phương thức, property đều mặc định là public |
| Một lớp trừu tượng có thể chứa các constructor | Không chứa constructor |
| Một lớp trừu tượng có thể chứa biến instance | Không thể chứa biến instance |

4. Tính kế thừa trong Java là gì?

Câu trả lời:

Kế thừa trong Java là khái niệm trong đó các thuộc tính, phương thức của một lớp có thể được kế thừa bởi một lớp khác. Nó giúp sử dụng lại mã và thiết lập mối quan hệ giữa các lớp khác nhau. Sự kế thừa được ttheer hiện giữa hai loại lớp:

* Super class (Base class - lớp cha)
* Subclass (Derived class - lớp con)

Một lớp kế thừa các thuộc tính, phương thức được gọi là subclass, lớp có các thuộc tính, phương thức được kế thừa được gọi là super class.

5. Các kiểu kế thừa trong Java?

Câu trả lời:

Java hỗ trợ 5 kiểu kế thừa:

* Đơn kế thừa: Một lớp kế thừa các thuộc tính, phương thức cảu lớp khác, tức là sẽ chỉ có một lớp cha và một lớp con
* Kế thừa thứ bậc: Khi có nhiều hơn một lớp con hay nói cách khác, nhiều lớp con cùng kế thừa một lớp cha
* Kế thừa đa cấp: Lớp con kế thừa các tính năng của lớp cha và dồng thời lớp con này lại hoạt động như một lớp cha cho lớp con khác.
* Kế thừa lai: Sự kết hợp của hai hay nhiều kiểu kế thừa.
* Đa kế thừa: Khi có một lớp con được kế thừa từ hai hay nhiều lớp cha.

Tuy nhiên, trong Java không hỗ trợ kế thừa lai và đa kế thừa mà chúng chỉ có thể được thực hiện thông qua interface

6. Constructor overloading trong Java?

Câu trả lời:

Trong Java, constructor overloading là một kỹ thuật thêm bất kỳ số lượng constructor nào vào một lớp, mỗi constructor sẽ có danh sách tham số khác nhau. Trình biên dịch sẽ sử dụng số lượng tham số và kiểu của chúng trong một danh sách để phân biệt các constructor được nạp chồng

7. Các tính chất của hướng đối tượng là gì?

Câu trả lời:

Lập trình hướng đối tượng bao gồm 4 tính chất:

* Tính đóng gói
* TÍnh kế thừa
* Tính trừu tượng
* Tính đa hình

8. Ưu điểm của lập trình hướng đối tượng là gì?

Câu trả lời:

* Giảm dư thừa mã
* Cải thiện khả năng đọc mã
* Chi phí phát triển thấp
* Chất lượng phần mềm được cải thiện
* Phát triển sản phẩm nhanh hơn
* Khả năng tái sử dụng của mã
* Dễ dàng bảo trì
* Có thể mở rộng có nghĩa là chúng tôi có thể dễ dàng thêm các tính năng mới vào tính năng hiện có.

9. Sự khác biệt giữa lập trình hướng chức năng và lập trình hướng đối tượng là gì?

Câu trả lời:

| **Lập trình hướng chức năng** | **Lập trình hướng đối tượng** |
| --- | --- |
| Dựa trên chức năng | Dựa trên các đối tượng |
| Tầm quan trọng hơn về trình tự các chức năng được thực thi | Tầm quan trọng của các trạng thái và hành vi của đối tượng |
| Không hỗ trợ đóng gói | Hỗ trợ đóng gói |
| Thực hiện theo cách tiếp cận từ trên xuống | Thực hiện theo cách tiếp cận từ dưới lên |
| Khả năng tái sử dụng mã ít hơn | Khả năng tái sử dụng mã tốt hơn |
| Phức tạp hơn để mở rộng và sửa đổi mã | Ít phức tạp hơn và dễ dàng mở rộng và sửa đổi mã |

10. Sự khác biệt giữa một lớp trừu tượng và một giao diện là gì?

Câu trả lời:

| **Lớp trừu tượng** | **Giao diện** |
| --- | --- |
| Lớp trừu tượng có thể có cả phương thức trừu tượng và không trừu tượng | Giao diện chỉ có thể có các phương thức trừu tượng. Từ Java 8, hỗ trợ các phương thức mặc định |
| Không hỗ trợ đa kế thừa | Hỗ trợ đa kế thừa |
| Sử dụng từ khóa trừu tượng | Sử dụng từ khóa giao diện |
| Sử dụng từ khóa kéo dài để kế thừa lớp trừu tượng | Sử dụng từ khóa triển khai để triển khai giao diện |
| Nó có thể mở rộng một lớp Java khác và cũng có thể triển khai giao diện | Nó chỉ có thể mở rộng giao diện khác |
| Các thành viên có thể có các công cụ sửa đổi quyền truy cập như riêng tư, được bảo vệ, v.v. | Thành viên chỉ có thể có công khai |
| Nó có thể có các biến tĩnh, không tĩnh, cuối cùng hoặc không cuối cùng. | Nó chỉ có thể có các biến tĩnh và biến |

11. Xác định khối try-catch và finally

Câu trả lời:

Một khối try-catch và finally được sử dụng để triển khai xử lý ngoại lệ. Mã có ngoại lệ ném khả năng được xác định trong khối try và các ngoại lệ được bắt trong khối bắt tương ứng.

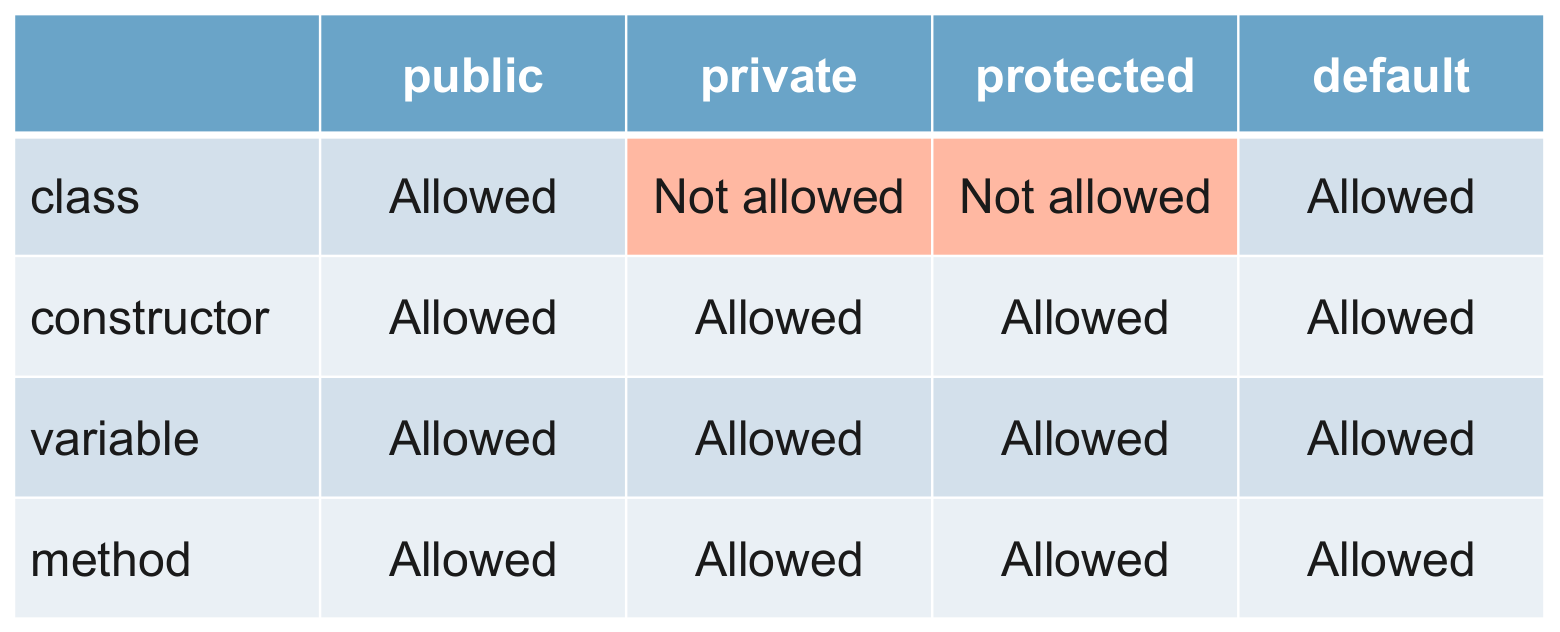
Cuối cùng khối finally đảm bảo rằng các câu lệnh trong khối này luôn thực thi bất kể có ngoại lệ xảy ra hay không.

12. Các loại phạm vi truy cập trong Java

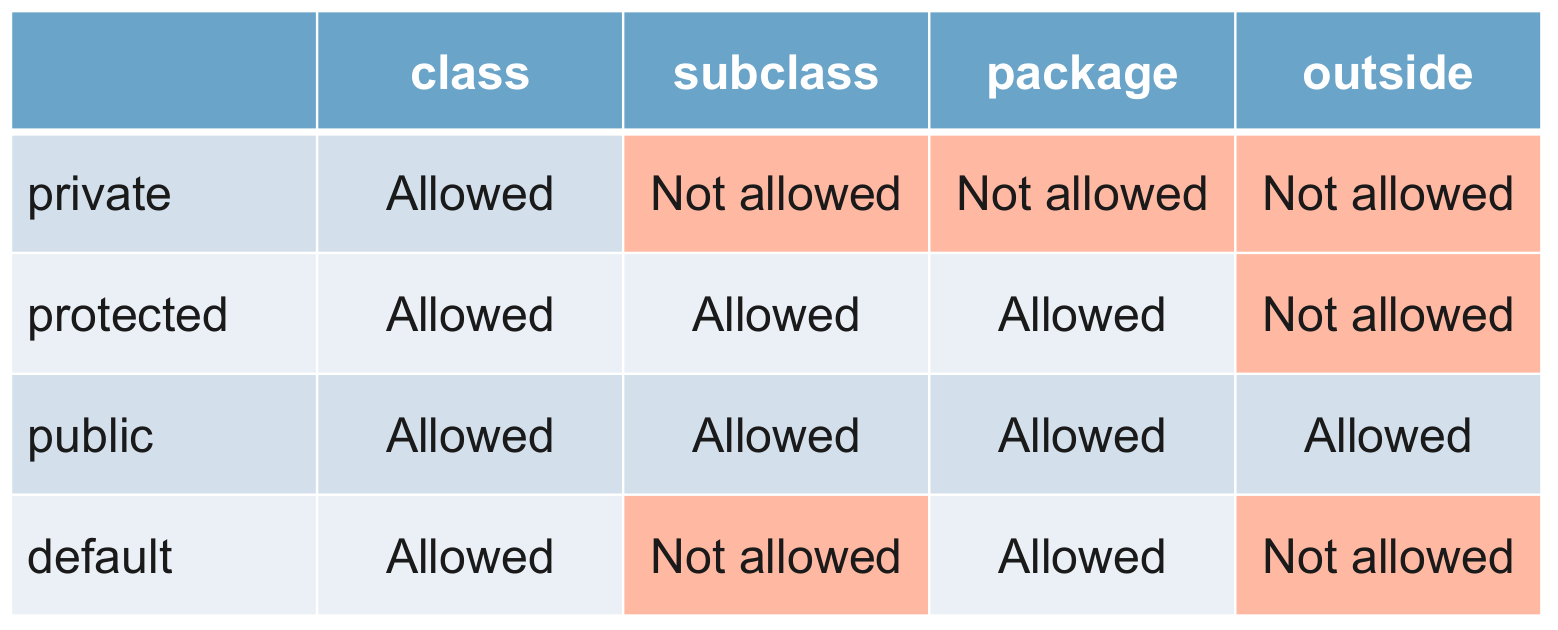
Câu trả lời:

Trong Java, phạm vi truy cập là các từ khóa đặc biệt được sử dụng để hạn chế quyền truy cập của một lớp, constructor, thành viên dữ liệu và phương thức trong một lớp khác. Có 4 loại phạm vi truy cập là: public, default, protected và private.

Đối tượng sử dụng:



Phạm vi truy cập:



13. So sánh ArrayList và Vector trong Java?

Câu trả lời:

| **ArrayList** | **Vector** |
| --- | --- |
| ArrayList không được đồng bộ hóa | Vector được đồng bộ hóa |
| ArrayList nhanh hơn do không phải đồng bộ hóa | Vector chậm hơn nhưng với luồng an toàn |
| Nếu một phần tử được chèn vào ArrayList, nó sẽ tăng kích thước Array lên 50% | Vector mặc định tăng gấp đôi kích thước mảng |
| ArrayList không xác định kích thước gia tăng | vector xác định kích thước gia tăng |
| ArrayList chỉ sử dụng Iterator để duyệt | Vector sử dụng cả Enumeration và Iterator để duyệt |

14. Giải thích phương thức main() trong chương trình Java?

Câu trả lời:

* main() trong java là điểm đầu vào cho bất kỳ chương trình Java nào. Nó luôn được viết là public static void main().
* public: là phạm vi truy cập, được sử dụng để chỉ định ai có thể truy cập vào phương thức này. public có nghĩa là phương thức này có thể được truy cập bởi bất kỳ class nào.
* static: một phương thức được khai báo là static thì phương thức đó được gọi là phương thức tĩnh,phương thức đó có thể được truy cập mà không cần tạo đối tượng. Trong trường hợp, main() không được khai báo với static thì trình biên dịch sẽ ném ra một lỗi vì hàm main() được gọi bởi JVM trước khi bất kỳ đối tượng nào được thực hiện.
* void: Là kiểu trả về của phương thức. Voi định nghĩa phương thức sẽ không trả về bất kỳ giá trị nào.
* main: Là tên của phương thức được JVM tìm kiếm như một điểm khởi đầu cho ứng dụng.
* String args[]: Là tham số truyền cho phương thức main()

15. Ý nghĩa của việc đóng gói dữ liệu?

Câu trả lời:

Đóng gói là một khái niệm trong Lập trình hướng đối tượng để kết hợp các thuộc tính và phương thức trong một đơn vị duy nhất.

Tính đóng gói giúp các lập trình viên tuân theo cách tiếp cận mô-đun để phát triển phần mềm vì mỗi đối tượng có một bộ phương thức và biến riêng và phục vụ các chức năng của nó độc lập với các đối tượng khác. Đóng gói cũng phục vụ mục đích ẩn dữ liệu.

# Bài tập thực hành

Được trình bày theo từng bài, mỗi bài sẽ tương đương với 01 buổi thực hành theo đề cương chi tiết của học phần. Bố cục mỗi bài thực hành gồm có 5 phần theo như cấu trúc dưới đây.

## Bài thực hành số 1 (Số tiết: 03 tiết)

+ Mục đích của bài thực hành

-Làm quen với môi trường thực thi và phát triển của Java JDK, IDE (Integrated Development Environment **Netbean** hoặc **Eclipse**), làm quen và sử dụng một số đối tượng đơn giản để xây dựng các chương trình java đầu tiên (HelloWorld).

+ Yêu cầu cần đạt được của bài thực hành

-Sử dụng đối tượng java.ultil.Scanner dụng cho nhập dữ liệu từ bàn phím và thực hiện các phép toán đơn giản, hiển thị dưới dạng Console(dòng lệnh) các kết quả tính toán. Nắm được cách sử dụng và hiểu ý nghĩa của hàm *main(String args[])*, làm quen với một số lỗi đơn giản và cách khắc phục.

-Cách đặt tên file chương trình chứa hàm main, hiểu ý nghĩa của <tên file>.java, quan sát kết quả biên dịch tên <tên file>.class, biên dịch lại chương trình khi có sửa đổi

**+ Bài giải mẫu (có hướng dẫn cách giải)**

Bài 1 : HelloWorld

**HelloWorld.java** (tên file)

*class HelloWorld {*

*public static void main(String[] args) {*

*System.out.println("Hello, World!");*

*}*

*}*

Bài 2 : Tính tổng của hai số được nhập từ bàn phím

*CTTinhTong.java (tên file)*

*import java.util.Scanner;*

*public class CTTinhTong {*

*public static void main(String args[]) {*

*Scanner w = new Scanner(System.in);*

*int a = 0, b = 0;*

*System.out.println("Nhap so a=");*

*a = w.nextInt();*

*System.out.println("Nhap so b=");*

*b = w.nextInt();*

*System.out.println("tong a+b=" + (a + b) + "hieu a-b=" + (a - b));*

*}*

*}*

**+ Các bài tập thực hành mức độ cơ bản**

Bài 1

1. Cài đặt JDK và cài đặt NetBean (hoặc eclipse), làm quen với các chức năng của NetBean IDE( hoặc Eclipse IDE), phân tích ưu nhược điểm của hai môi trường phát triển này.
2. Thực hiện cài đặt lại các bài giải mẫu
3. Làm quen, hiểu với một số kiểu lỗi cú pháp đơn giản và cách khắc phục

Bài 2 Viết chương trình:

1. Khai báo (import) lớp Scanner trong gói java.util
2. Sử dụng phương thức *nextLine* () của lớp Scanner để nhập vào một xâu và in xâu vừa nhập ra màn hình.

Bài 3 Viết chương trình:

1. Sử dụng lớp Scanner trong gói java.util
2. Sử dụng phương thức nextInt() để nhập vào ba số a, b, c tương ứng với độ cài của ba cạch của một tam giác. in ra thông tin chu vi (p=a+b+c) của tam giác tương ứng.

**+ Các bài thực hành mức độ nâng cao.**

Bài 1: Sử dụng phương thức nextInt() để nhập vào bốn số a, b, c, h tương ứng với độ cài của ba cạch và chiều cao tương ứng với cạnh đáy a của một tam giác. in ra thông tin diện tích (1/2\*a\*h) của tam giác tương ứng.

Bài 2 Viết chương trình thực hiện đổi gallons ra liters biết rằng 1 gallon bằng 3.7854 liter, số gallon được nhập từ bàn phím

## Bài thực hành số 2 (số tiết: 03 tiết)

+ Mục đích của bài thực hành

-Sử dụng các câu lệnh khai báo biến, quy tắc đặt tên biến, câu lệnh gán.

-Sử dụng thành thạo các kiểu dữ liệu nguyên thủy và các phép toán

-Hiểu và áp dụng thành thạo các cấu trúc điều khiển if, if else kết hợp và các phép toán quan hệ >,<,==, !=,>=, <=

+ Yêu cầu cần đạt được của bài thực hành

**+ Bài giải mẫu (có hướng dẫn cách giải)**

Bài 1 : Kiểm tra giá trị của biến *c*

*/\**

*Tên file IfDemo.java.*

*\*/*

*class IfDemo {*

*public static void main(String args[]) {*

*int a, b, c;*

*a = 2;*

*b = 3;*

*if (a < b) {*

*System.out.println("a is less than b");*

*}*

*// this won't display anything*

*if (a == b) {*

*System.out.println("you won't see this");*

*}*

*System.out.println();*

*c = a - b; // c contains -1*

*System.out.println("c contains -1");*

*if (c >= 0) {*

*System.out.println("c is non-negative");*

*}*

*if (c < 0) {*

*System.out.println("c is negative");*

*}*

*System.out.println();*

*c = b - a; // c now contains 1*

*System.out.println("c contains 1");*

*if (c >= 0) {*

*System.out.println("c is non-negative");*

*}*

*if (c < 0) {*

*System.out.println("c is negative");*

*}*

*}*

*}*

**+ Các bài tập thực hành mức độ cơ bản**

Bài 1: Viết chương trình nhập vào 3 số nguyên từ bàn phím, chương trình sẽ in ra số nguyên có giá trị lớn nhất

Bài 2: Viết chương trình nhập và 1 số nguyên từ bàn phím, chương trình sẽ in ra thông tin số vừa nhập là số ‘âm’, ‘dương’ hoặc là số ‘không’

Bài 3: Viết chương trình cho phép nhập vào 3 số thực tương ứng với độ dài của một tam giác và kiểm tra 3 số này có phải là 3 cạnh của một tam giác hay không( nếu nó thỏa mãn điều kiện tổng của hai cạnh bất kỳ đều lớn hơn cạnh thứ 3) .

Bài 4: Với n nhập từ bàn phím hay viết các chương trình tính tổng các dãy sau:

* 1. Tính tổng dãy

S=1+2-3+……(-1)n+1.n

* 1. Tính tổng dãy:

S=1!+2!+3!+…+n!

* 1. Tính tổng các số lẻ <n, nếu n chẵn: tính tổng các số chẵn <n.
  2. 

**+ Các bài thực hành mức độ nâng cao.**

Bài 1 : Viết chương trình cho phép nhập vào 3 số thực tương ứng với độ dài của một tam giác và kiểm tra 3 số này có phải là 3 cạnh của một tam giác vuông hay không ( nó là một tam giác, và bình phương của cạnh huyền bằng tổng bình phương của hai cạnh còn lại) .

Bài 2 : Dùng vòng lặp *for* viết chương trình tìm n số Fibonacci đầu tiên trong java. Số nguyên dương n được nhập từ bàn phím.

Tương tự với yêu cầu như trên nhưng dùng các cấu trúc while, do while

*Quy luật của dãy số Fibonacci: số tiếp theo bằng tổng của 2 số trước, 2 số đầu tiên của dãy số là 0, 1. Ví dụ: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...*

## Bài thực hành số 3 (số tiết: 03 tiết)

+ Mục đích của bài thực hành

-Tiếp tục về các câu lệnh điều khiển và ngoại lệ, hiểu được khái niệm lỗi thực thi (runtime)

-Ngoại lệ (Exception)

-Các kiểu ngoại lệ cơ bản (ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, NullPointerException, NumberFormatException..), cơ chế bắt và xử lý ngoại lệ, sử dụng mệnh đề throw, throws, tự định nghĩa ngoại lệ.

+ Yêu cầu cần đạt được của bài thực hành

**+ Bài giải mẫu (có hướng dẫn cách giải).**

Bài 1 : Bắt và xử lý ngoại lệ chia cho 0 với số nguyên

*class Main {*

*public static void main(String[] args) {*

*try {*

*int divideByZero = 5 / 0;*

*System.out.println("Rest of code in try block");*

*}*

*catch (ArithmeticException e) {*

*System.out.println("ArithmeticException => " + e.getMessage());*

*}*

*}*

*}*

Bài 2 : Bắt và xử lý ngoại lệ định dạng kiểu số trong java

*public static void main(String args[]) {*

*Scanner input = new Scanner(System.in);*

*Integer intNumber = null;*

*try {*

*System.out.println("Hãy nhập vào một số nguyên: ");*

*String strNumber = input.nextLine();*

*intNumber = new Integer(strNumber);*

*} catch (NumberFormatException e) {*

*intNumber=0;*

*System.out.println("Lỗi định dạng của số nguyên : "+e.);*

*}*

*System.out.println("Chuyển thành Hexa: " + Integer.toHexString(intNumber));*

*}*

*}*

**+ Các bài tập thực hành mức độ cơ bản**

Bài 1: Với các bài thực hành số 2 nâng cấp để bắt và xử lý ngoại lệ với các dữ liệu được nhập vào từ bàn phím.

Bài 2: Sử dụng cấu trúc try, và 2 khối catch xây dựng chương trình nhập vào 2 số nguyên a, b và in ra kết quả a/b. Yêu cầu bắt các ngoại lệ NumberFormatException khi nhập a và b, ArithmeticException khi thực hiện a/b.

Bài 3: Dùng mệnh đề Throw để tạo ra ngoại lệ khi nhập tuổi cho nhân viên (không âm và không lớn hơn 62 tuổi)

**+ Các bài thực hành mức độ nâng cao.**

Bài 1: Xây dựng hàm nhapTuoi() và dùng mệnh đề Throws để chuyển ngoại lệ ra ngoài hàm và thực hiện bắt và xử lý các ngoại lệ khi gọi hàm nhapTuoi().

## Bài thực hành số 4 (số tiết: 03 tiết)

+ Mục đích của bài thực hành

-Thiết kế yêu cầu bài toán theo hướng đối tượng, hiểu và cài đặt các cấu trúc lớp và khai báo các thành phần của lớp, định nghĩa hàm thành phần và cơ chế nạp chồng, viết đè trong Java,các thuộc tính kiểm soát truy cập.

- Các nội dung liên quan khác

+ Yêu cầu cần đạt được của bài thực hành

**+ Bài giải mẫu (có hướng dẫn cách giải).**

Viết chương trình xây dựng lớp Diem gồm:

-Các thuộc tính double x, y tương ứng là hai giá trị hoành độ và tung độ và các phương thức:

Các phương thức tạo lập:

*public Diem(){}*

*public Diem(double x,double y){x=x1;y=y1}*

Các phương thức :

*public void inTTDiem(){ // in giá trị x,y ra màn hình }*

*public double tinhKhoangCach(Diem d) {tính khoảng cách giữa điểm hiện thời và đối số d}*

Cài đặt lớp **Main** thực hiện tạo ra hai điểm *D1=new Diem(3,7); D2= new Diem(9,3)* và in khoảng cách của hai điểm trên.

*import java.util.Scanner;*

*public class Diem {*

*double x,y;*

*Diem() {*

*}*

*public void nhapTTDiem()*

*{Scanner input= new Scanner(System.in);*

*System.out.print("Nhap toa do x:= ");*

*x=input.nextDouble();*

*System.out.print("Nhap toa do y:= ");*

*y=input.nextDouble();*

*}*

*public void inTTDiem()*

*{System.out.println("Toa do X:"+x+"Toa do Y: "+y);*

*}*

*Diem(double x1,double y1)*

*{ x=x1;y=y1;*

*}*

*public double tinhKhoangCach(Diem d1)*

*{ return Math.sqrt((x-d1.x)\*(x-d1.x)+(y-d1.y)\*(y-d1.y));*

*}*

*}*

**+ Các bài tập thực hành mức độ cơ bản**

Bài 1: Viết chương trình:

1. Xây dựng lớp **Nguoi** gồm:

* Các thuộc tính chung : hoTen, diaChi, namSinh
* Các phương thức:

Phương thức: public void nhapTT(){// nhập thông tin cho Nguoi}

Phương thức: public void inTT(){//in thông tin là các giá trị thuộc tính cho Nguoi}, public void nhapTT(){// nhập thông tin từ bàn phím cho các thông tin của Nguoi }

2. Xây dựng lớp **Main** chứa phương thức main(): Cho phép nhập thông tin của 2,3 người và hiển thị thông tin của người vừa nhập ra màn hình.

Bài 2: Thư viện của trường đại học KHTN có nhu cầu cần quản lý việc mượn sách. Sinh viên đăng ký và tham gia mượn sách thông qua các thẻ mượn mà thư viện đã thiết kế.

-Với mỗi thẻ mượn, có các thông tin sau: số phiếu mượn , ngày mượn, hạn trả, số hiệu sách, và các thông tin riêng về mỗi sinh viên đó.

-Các thông tin riêng về mỗi sinh viên đó bao gồm: Họ tên, năm sinh, tuổi, lớp.

Hãy xây dựng các lớp: SinhVien để quản lý các thông tin riêng về mỗi sinh viên, lớp TheMuon để quản lý việc mượn sách của mỗi đọc giả.

Viết chương trình nhập và hiển thị thông tin của một thẻ mượn.

Bài 3: Để quản lý các biên lai thu tiền điện, người ta cần các thông tin như sau: Với mỗi biên lai, có các thông tin sau: thông tin về hộ sử dụng điện, chỉ số cũ, chỉ số mới, số tiền phải trả của mỗi hộ sử dụng điện

Các thông tin riêng của mỗi hộ sử dụng điện gồm: Họ tên chủ hộ, số nhà, mã số công tơ của hộ dân sử dụng điện.

1. Hãy xây dựng các lớp: **KhachHang** để lưu trữ các thông tin riêng của mỗi hộ sử dụng điện, lớp **BienLai** để quản lý việc sử dụng và thanh toán tiền điện của các hộ dân. Xây dựng các phương thức nhập, và hiển thị thông tin của mỗi hộ sử dụng điện.

2. Cài đặt chương trình thực hiện các công việc sau:

+ Nhập vào các thông tin cho 2 hộ sử dụng điện

+ Tính tiền điện phải trả cho mỗi hộ dân, biết rằng tiền phải trả được tính theo công thức sau: *Số tiền phải trả=(Số mới - số cũ) \* 850000*.

**+ Các bài thực hành mức độ nâng cao.**

Bài 1: Hãy xây dựng lớp DaGiac gồm có:

Các thuộc tính:

+ Số cạnh của đa giác

+ Mảng các số thực chứa kích thước các cạnh của đa giác

Các phương thức:

+ Nhập vào giá trị cho các cạnh của đa giác

+ Tính chu vi của đa giác

+ Hiển thị giá trị các cạnh của đa giác.

Xây dựng lớp TamGiac kế thừa từ lớp DaGiac, thực hiện ghi đè các phương thức của lớp TamGiac lên các phương thúc của lớp DaGiac.

Xây dựng hàm main() nhập kích thước của một tam giác và hiển thị chu vi, diện tích của tam giác đó.

## Bài thực hành số 5 (số tiết: 03 tiết)

+ Mục đích của bài thực hành

Tiếp phần lớp và các thành phần của lớp. Phân biệt các hàm tĩnh (static và non-static), hiểu và vận dụng các khái niệm quan trọng như toán tử tạo lập đối tượng, kế thừa giữa các lớp đối tượng, giao diện và sự mở rộng quan hệ kế thừa, phương thức trừu tượng. Thiết kế yêu cầu bài toán theo hướng đối tượng

+ Yêu cầu cần đạt được của bài thực hành

**+ Bài giải mẫu (có hướng dẫn cách giải)**

Bài 1 : Tạo 1 lớp trừu tượng (abstract) Person chứa các thông tin về người, tạo tiếp 2 lớp Students va Teachers kế thừa lớp Person, tạo lớp Execute chứa hàm main để chạy chương trình.

*import java.io.Console;*

*import java.util.Scanner;*

*abstract class Person {*

*//cai nay goi la cac property hay state-thuoc tinh cua doi tuong*

*String hoten;*

*int age;*

*String diachi;*

*int luong;*

*//cac constructor*

*Person() {*

*}*

*public Person(int age) {*

*this.age = age;*

*}*

*//cac method hay behavior-hanh vi cua doi tuong*

*public void Nhap() {*

*Console input = System.console();*

*hoten = input.readLine("Nhap ho ten:");*

*diachi = input.readLine("Nhap dia chi:");*

*}*

*public abstract void In();*

*public abstract int Tinhluong();*

*}*

*class Students extends Person {*

*int MaSV, Malop;*

*public Students(int age) {*

*super(age);*

*}*

*public Students() {*

*}*

*public void Nhap() {*

*Scanner input = new Scanner(System.in);*

*super.Nhap();*

*System.out.println("Nhap Ma SV: ");*

*MaSV = input.nextInt();*

*System.out.println("Nhap Ma Lop: ");*

*Malop = input.nextInt();*

*}*

*public void In() {*

*System.out.println(hoten);*

*System.out.println(diachi);*

*System.out.println(MaSV);*

*System.out.println(Malop);*

*}*

*public int Tinhluong() {*

*return 150000;*

*}*

*}*

*class Teachers extends Person {*

*int Makhoa;*

*public Teachers(int age) {*

*super(age);*

*}*

*public Teachers() {*

*}*

*public void Nhap() {*

*Scanner input = new Scanner(System.in);*

*super.Nhap();*

*System.out.println("Nhap Ma khoa: ");*

*Makhoa = input.nextInt();*

*}*

*public void In() {*

*System.out.println(hoten);*

*System.out.println(diachi);*

*System.out.println(Makhoa);*

*}*

*public int Tinhluong() {*

*return 500000;*

*}*

*}*

*public class Execute {*

*public static void main(String args[]) {*

*Students st = new Students();*

*st.Nhap();*

*st.In();*

*st.luong = st.Tinhluong();*

*Teachers tc = new Teachers();*

*tc.Nhap();*

*tc.In();*

*tc.luong = tc.Tinhluong();*

*}*

*}*

Bài 2 : **Shape** là lớp trừu tượng, trình triển khai của nó được cung cấp bởi lớp **Rectangle** và lớp **Circle**. Hai lớp này kế thừa lớp trừu tượng **Shape**.

*// lop truu tuong Shape*

*abstract class Shape {*

*abstract void draw();*

*}*

*//Trong tinh huong nay, trinh trien khai duoc cung cap boi ai do, vi du: nguoi su dung cuoi cung nao do*

*class Rectangle extends Shape {*

*void draw() {*

*System.out.println("Ve hinh chu nhat");*

*}*

*}*

*class Circle1 extends Shape {*

*void draw() {*

*System.out.println("Ve hinh tron");*

*}*

*}*

*//Trong tinh huong nay, phuong thuc duoc goi boi lap trinh vien hoac nguoi dung*

*class TestAbstraction1 {*

*public static void main(String args[]) {*

*Shape s = new Circle1(); //Trong tinh huong nay, doi tuong duoc cung cap thong qua phuong thuc, chang han nhu getShape()*

*s.draw();*

*}*

*}*

**+ Các bài tập thực hành mức độ cơ bản**

Bài 1 : Định nghĩa giao diện **Ivehicle** gồm có các hằng và phương thức :

*interface IVehicle {*

*// khai báo và khởi tạo hằng*

*static final String STATEID = "LA-09";*

*//phương thức trừu tượng khởi động xe*

*public void start();*

*// phương thức trừu tượng tăng tốc xe*

*public void accelerate(int speed);*

*// phương thức trừu tượng hãm phanh*

*public void brake();*

*// phương thức trừu tượng dừng xe*

*public void stop();*

*}*

Xây dựng lớp **TwoWheeler** kế thừa từ **Ivehicle** và cài đặt các phương thức và thuộc tính

*class TwoWheeler implements IVehicle {*

*// biến lưu ID của xe*

*String ID;*

*// biên lưu kiểu xe*

*String type;*

*// hàm tạo có tham số để khởi tạo các giá trị nhập vào từ người dùng*

*public TwoWheeler(String ID, String type) {*

*this.ID = ID;*

*this.type = type;*

*}*

*//ghi đè các phương thức để khởi động xe*

*@Override*

*public void start() {*

*System.out.println("Starting the " + type);*

*}*

*// ghi đè phương thức để tăng tốc xe*

*@Override*

*public void accelerate(int speed) {*

*System.out.println("Accelerating at speed:" + speed + " kmph");*

*}*

*// ghi đè phương thức để hãm phanh*

*@Override*

*public void brake() {*

*System.out.println("Applying brakes");*

*}*

*//ghe đè phương thức để dừng xe*

*@Override*

*public void stop() {*

*System.out.println("Stopping the " + type);*

*}*

*//hiển thị thông số của xe*

*public void displayDetails() {*

*System.out.println("Vehicle No.: " + STATEID + " " + ID);*

*System.out.println("Vehicle Type.: " + type);*

*}*

*}*

Xây dựng lớp ***TestVehicle***chứa hàm main khởi tạo đối tượng và test các phương thức:

*public class TestVehicle {*

*public static void main(String[] args) {*

*// xác nhận số lượng đối số dòng lệnh*

*// tạo thể hiện cho lớp TwoWheeler*

*TwoWheeler objBike = new TwoWheeler("20A 22584", "HonDa Civic");*

*// gọi các phương thức objBike.displayDetails();*

*objBike.start();*

*objBike.accelerate(200);*

*objBike.brake();*

*objBike.stop();*

*}*

*}*

Bài 2: Viết chương trình xây dựng lớp SoPhuc gồm:

Các thuộc tính riêng gồm: phanThuc, phanAo kiểu double

Các phương thức:

+ Các toán tử tạo lập : SoPhuc(), SoPhuc(float pt, float pa)

+ Phương thức nhập vào một số phức: void nhapSoPhuc()

+ Phương thức hiển thị một số phức: void inSoPhuc()

+ Phương thức cộng hai số phức : SoPhuc congSoPhuc(SoPhuc sp)

+ Phương thức nhân hai số phức: SoPhuc nhanSoPhuc(SoPhuc sp)

Cài đặt chương trình (lớp Main) nhập vào hai số phức A và B, sau đó tính số phức tổng C1=A.congSoPhuc(B), nhân C=A.nhanSoPhuc(B) rồi hiển thị kết quả ra màn hình như sau:

Tổng: (a + b.i) + (c + d.i) = (a + c) + (b + d).i

**Tích: (a + b.i)(c + d.i) = (a.c - b.d) + (b.c + a.d).i

Chia :

**+ Các bài thực hành mức độ nâng cao.**

Bài 1: Xây dựng interface Imanufacturer gồm có các phương thức:

*interface IManufacturer {*

*// phương thức trừu tượng để thêm thông tin liên hệ*

*public void addContact(String detail);*

*// phương thức trừu tượng để gọi nhà sản xuất*

*public void callManufacturer(String phone);*

*/\*\**

*\*\* phương thức để tiến hành thanh toán*

*\*/*

*public void makePayment(float amount);*

*}*

Bài 2: Xây dựng lớp **TwoWheeler1** được kế thừa từ hai giao diện **Ivehicle,** **IManufacturer** và cài đặt các phương thức :

*class TwoWheeler1 implements IVehicle, IManufacturer {*

*String ID;*

*// ID xe*

*String type; // kiểu xe*

*public TwoWheeler1(String ID, String type) {*

*this.ID = ID;*

*this.type = type;*

*}*

*@Override*

*public void start() {*

*System.out.println("Starting the " + type);*

*}*

*@Override*

*public void accelerate(int speed) {*

*System.out.println("Accelerating at speed:" + speed + " kmph");*

*}*

*@Override*

*public void brake() {*

*System.out.println("Applying brakes...");*

*}*

*@Override*

*public void stop() {*

*System.out.println("Stopping the " + type);*

*}*

*public void displayDetails() {*

*System.out.println("Vehicle No.: " + STATEID + " " + ID);*

*System.out.println("Vehicle Type.: " + type);*

*}// thực thi các phương thức của giao diện*

*@Override*

*public void addContact(String detail) {*

*System.out.println("Manufacturer: " + detail);*

*}*

*@Override*

*public void callManufacturer(String phone) {*

*System.out.println("Calling Manufacturer @: " + phone);*

*}*

*@Override*

*public void makePayment(float amount) {*

*System.out.println("Payable Amount: $" + amount);*

*}*

*}*

Tạo lớp ***TestVehicle1***và thể hiện của***TwoWheeler1.***

*public class TestVehicle1 {*

*public static void main(String[] args) {*

*// thể hiện của lớp*

*TwoWheeler1 objBike = new TwoWheeler1(args[0], args[1]);*

*objBike.displayDetails();*

*objBike.start();*

*objBike.accelerate(Integer.parseInt(args[2]));*

*objBike.brake();*

*objBike.stop();*

*objBike.addContact(args[3]);*

*objBike.callManufacturer("0989555666");*

*objBike.makePayment(50000);*

*}*

*}*

## Bài thực hành số 6 (số tiết: 03 tiết)

+ Mục đích của bài thực hành

-Sinh viên làm quen với cấu trúc mảng trong java, sử dụng cấu trúc mảng kết hợp với các đối tượng nhằm nâng cao khả năng giải quyết các bài toán trong thực tế. Thiết kế yêu cầu bài toán theo hướng đối tượng

+ Yêu cầu cần đạt được của bài thực hành

**+ Bài giải mẫu (có hướng dẫn cách giải)**

Bài 1 : Xắp xếp mảng theo phương pháp nổi bọt

*public class BubbleSort {*

*public static void main(String args[]) {*

*int nums[] = {99, -10, 100123, 18, -978,*

*5623, 463, -9, 287, 49};*

*int a, b, t;*

*int size;*

*size = 10; // number of elements to sort*

*// display original array*

*System.out.print("Original array is:");*

*for (int i = 0; i < size; i++) {*

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

*}*

*System.out.println();*

*// This is the Bubble sort.*

*for (a = 1; a < size; a++) {*

*for (b = size - 1; b >= a; b--) {*

*if (nums[b - 1] > nums[b]) { // if out of order*

*// exchange elements*

*t = nums[b - 1];*

*nums[b - 1] = nums[b];*

*nums[b] = t;*

*}*

*}*

*}*

*// display sorted array*

*System.out.print("Sorted array is:");*

*for (int i = 0; i < size; i++) {*

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

*}*

*System.out.println();*

*}*

*}*Bài 2 : Gán giá trị cho một mảng hai chiều và in ra dưới dạng bảng của ma trận 2 chiều

*class TwoD\_Arr {*

*public static void main(String args[]) {*

*int t, i;*

*int table[][] = new int[3][4];*

*for (t = 0; t < 3; ++t) {*

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

*table[t][i] = (t \* 4) + i + 1;*

*System.out.print(table[t][i] + " ");*

*}*

*System.out.println();*

*}*

*}*

*}*

**+ Các bài tập thực hành mức độ cơ bản**

Bài 1: Xây dựng lớp DaySo gồm:

Các thuộc tính riêng (private) sau đây:

* private int n; //số phần tử của dãy
* private int m[] ; //lưu trữ các giá trị của dãy

Các toán tử tạo lập: DaySo(int spt), DaySo(int m1[]), DaySo(){}

Các phương thức:

* public void inDaySo() //in dãy ra màn hình
* public void nhapDaySo()//nhập dãy số từ bàn phím
* public void inSoNguyenTo(){} //in ra các số nguyên tố thuộc dãy số
* public void inSoHoanHao()
* public void sapXepTang() //sắp xếp dãy theo chiều tăng
* public void sapXepGiam()//sắp xếp dãy theo chiều giảm
* public DaySo congDay(DaySo d1)// cộng hay dãy thành một dãy số

**+ Các bài thực hành mức độ nâng cao.**

Bài 2: Xây dựng lớp MaTran gồm:

Các thuộc tính riêng(private)

* + private int n,m;// số dòng và cột của ma trận
  + private double M[][]; // lưu trữ các phần tử của ma trận

Các phương thức tạo lập:

* + public MaTran(){}
  + public MaTran(int dong,int cot)
  + public MaTran(double M1[][])

Các phương thức:

* + public void inMaTran()// in ma trận hiện thời
  + public void nhapMaTran()// nhập ma trận từ bàn phím
  + public boolean kiemTraDoiXung()// kiểm tra tính đối xứng của mt qua đường chéo chính
  + public MaTran congMaTran(MaTran M1)
  + public MaTran nhanMaTran(MaTran M1)

## Bài thực hành số 7 (số tiết: 03 tiết)

+ Mục đích của bài thực hành

-Thiết kế yêu cầu bài toán theo hướng đối tượng, hiểu và áp dụng thành thạo cấu trúc dữ liệu mảng, lớp Object, lớp Match, lớp String

+ Yêu cầu cần đạt được của bài thực hành

**+ Bài giải mẫu (có hướng dẫn cách giải)**

Bài 1 : Viết chương trình nhập vào một mảng số nguyên có n phần tử và thực hiện các công việc sau:

-In giá trị các phần tử của mảng.

-Tìm phần tử có giá trị lớn nhất, nhỏ nhất.

-Đếm số phần tử là số chẵn.

-Sắp xếp mảng tăng dần.

**Hướng dẫn**

*import java.util.Arrays;*

*import java.util.Scanner;*

*public class Mang {*

*public static void main(String[] args) {*

*int n;*

*// int A[];*

*// Scanner scanner = new Scanner(System.in);*

*// do {*

*// System.out.println("Nhập vào số phần tử của mảng: ");*

*// n = scanner.nextInt();*

*// } while (n < 0);*

*// int A[] = new int[n];*

*// System.out.println("Nhập các phần tử cho mảng: ");*

*// for (int i = 0; i < n; i++) {*

*// System.out.print("Nhập phần tử thứ " + i + ": ");*

*// A[i] = scanner.nextInt();*

*// }*

*//khởi tạo mảng nếu không dùng nhập từ bàn phím ở đoạn trên*

*int A[] = {11, 2, 12, 9, 6, 7, 11};*

*// Xuất giá trị các phần tử của mảng*

*System.out.println("\nMảng ban đầu: ");*

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

*System.out.print(A[i] + "\t");*

*}*

*// Tìm phần tử có giá trị lớn nhất, nhỏ nhất*

*int max = A[0]; // khởi tạo phần tử lớn nhất là phần tử đầu tiên*

*int min = A[0]; // khởi tạo phần tử bé nhất là phần tử đầu tiên*

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

*if (A[i] < min) {*

*min = A[i];*

*}*

*if (A[i] > max) {*

*max = A[i];*

*}*

*}*

*System.out.print("\nPhần tử lớn nhất trong mảng là " + max);*

*System.out.print("\nPhần tử nhỏ nhất trong mảng là " + min);*

*// Đếm số phần tử là số chẵn*

*int soPhanTuChan = 0;*

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

*if (A[i] % 2 == 0) {*

*soPhanTuChan++;*

*}*

*}*

*System.out.println("\nSố phần tử chẵn có trong mảng = " + soPhanTuChan);*

*// Sắp xếp măng tăng dần*

*// Java cung cấp cho chúng ta một thư viện có sẵn*

*// để sắp xếp mảng, đó là thư viện sort của lớp Arrays*

*// mặc định thư viện này sẽ sắp xếp mảng tăng dần*

*Arrays.sort(A);*

*System.out.println("\nMảng sau khi sắp xếp: ");*

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

*System.out.print(A[i] + "\t");*

*}*

*}*

Bài 2 : Sửa bài trên theo hướng đối tượng

*import java.util.Arrays;*

*import java.util.Scanner;*

*class MyArray {*

*int n;*

*int A[];*

*//Các toán tử tạo lập*

*public MyArray() {*

*}*

*public MyArray(int n, int[] A) {*

*this.n = n;*

*this.A = A;*

*}*

*public MyArray(int n) {*

*this.n = n;*

*}*

*public MyArray(int[] A) {*

*this.A = A;*

*n = A.length;*

*}*

*public void enterValue() {*

*Scanner scanner = new Scanner(System.in);*

*do {*

*System.out.println("Nhập vào số phần tử của mảng: ");*

*n = scanner.nextInt();*

*} while (n < 0);*

*int A[] = new int[n];*

*System.out.println("Nhập các phần tử cho mảng: ");*

*for (int i = 0; i < n; i++) {*

*System.out.print("Nhập phần tử thứ " + i + ": ");*

*A[i] = scanner.nextInt();*

*}*

*}*

*public void showValue() {*

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

*System.out.print(A[i] + "\t");*

*}*

*System.out.println();*

*}*

*public int[] findMaxMinValue() {*

*int values[] = new int[2];*

*values[0] = A[0]; // khởi tạo phần tử lớn nhất là phần tử đầu tiên*

*values[1] = A[0]; // khởi tạo phần tử bé nhất là phần tử đầu tiên*

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

*if (A[i] > values[0]) {*

*values[0] = A[i];*

*}*

*if (A[i] < values[1]) {*

*values[1] = A[i];*

*}*

*}*

*return values;*

*}*

*public int countEvennumber() {*

*int count = 0;*

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

*if (A[i] % 2 == 0) {*

*count++;*

*}*

*}*

*return count;*

*}*

*public int countOddnumber() {*

*int count = 0;*

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

*if (A[i] % 2 != 0) {*

*count++;*

*}*

*}*

*return count;*

*}*

*public void sortElement() {*

*Arrays.sort(A);*

*}*

*}*

*Chương trình chứa hàm Main*

*public class MyArray1 {*

*public static void main(String args[]) {*

*int A[] = {11, 2, 12, 9, 6, 7, 11};*

*MyArray M1 = new MyArray(A);*

*System.out.println("Cac gia tri cua mang");*

*M1.showValue();*

*System.out.println("Gia Tri Max la: " + M1.findMaxMinValue()[0]);*

*System.out.println("Gia Tri Min la: " + M1.findMaxMinValue()[1]);*

*System.out.println("Gia Tri so phan tu chan la: " + M1.countEvennumber());*

*System.out.println("Gia Tri so phan tu lẻ la: " + M1.countOddnumber());*

*M1.sortElement();*

*System.out.println("Các giá trị của mảng sau khi sắp xếp ");*

*M1.showValue();*

*}*

*}*

**+ Các bài tập thực hành mức độ cơ bản**

Bài 1 Viết chương trình:

1. Xây dựng lớp SinhVien gồm:

Các thuộc tính riêng (private):

* + String maSV, hoTen, diaChi;
  + double d1, d2, d3;

Các phương thức:

* + public void nhapTTSV(), public void inTTSV(),
  + public double tinhDiemTrungBinh()

1. Xây dựng lớp DanhSachSinhVien gồm:

Các thuộc tính riêng:

* + int n;// số sinh viên
  + SinhVien DSSinhVien[];// lưu trữ thông tin của các sinh viên

Các toán tử tạo lập: DanhSachSinhVien(int ssv)

* + DanhSachSinhVien(SinhVien mSinhVien[])

Các phương thức:

* + public void nhapDanhSachSinhVien();
  + public void inDanhSachSinhVien();
  + public void inDanhSachSinhVien(String dc)//in danh sách sinh viên có địa chỉ là dc

Bài 2: Xây dựng lớp MaTran gồm có các thuộc tính riêng:

* int n,m; là các thuộc tính số dòng và số cột của ma trận
* int A[][]; lưu trữ giá trị các phần tử của ma trận

Và các phương thức sau:

* Các toán tử tạo lập(constructor)
* Nhập ma trận.
* Hiển thị ma trận.
* Tìm hàng, cột hoặc đường chéo có tổng các phần tử lớn nhất.
* Tìm ma trận chuyển vị của A
* Tìm định thức của A
* Tìm ma trận nghịch đảo của A

Xây dựng lớp Main có chứa phương thức main() sử dụng các phương thức trên.

**+ Các bài thực hành mức độ nâng cao.**

Xây dựng lớp DATHUC để biểu diễn cho 1 đa thức toán học gồm có các thuộc tính riêng (private) :

-int n ; bậc của đa thực

-int[] heso

và các phương thức :

-Các toán tử tạo lập (constructor)

-Phương thức nhập đa thức

-Phương thức in đa thức

-Phương thức cộng đa thức

public DaThuc congDaThuc(DaThuc DaThucDoiSo)

-Phương thức nhân đa thức

public DaThuc nhanDaThuc(DaThuc DaThucDoiSo)

Xây dựng lớp Main chứa hàm main và thử nghiệm kết quả cài đặt

## Bài thực hành số 8 (số tiết: 03 tiết)

+ Mục đích của bài thực hành

-Hiểu và áp dụng thành thạo cấu trúc dữ liệu xâu(String) và các phép toán trên kiểu dữ liệu xâu, lớp Object, lớp Match. Thiết kế yêu cầu bài toán theo hướng đối tượng

+ Yêu cầu cần đạt được của bài thực hành

**+ Bài giải mẫu (có hướng dẫn cách giải)**

Bài 1: Khởi tạo một chuỗi trong java

*public class StringInitialization {*

*public static void main(String a[]){*

*String objStr\_01 = "This is a string object";*

*String objStr\_02 = new String("This is also string object");*

*char[] arrChar = {'V','N','L','I','V','E','S'};*

*String objStr\_03 = new String(arrChar);*

*String objStr\_04 = objStr\_03 + " This is another String object";*

*System.out.println("objStr\_01: " + objStr\_01);*

*System.out.println("objStr\_02: " + objStr\_02);*

*System.out.println("objStr\_03: " + objStr\_03);*

*System.out.println("objStr\_04: " + objStr\_04);*

*}*

*}*

Bài 2: So sánh chuỗi trong java

*package javaandroidvn;*

*public class JavaAndroidVn {*

*public static void main(String[] args) {*

*String str1 = "Android.Vn Android.Vn";*

*String str2 = "android.vn android.vn";*

*System.out.println("So sánh phân biệt chữ hoa chữ thường: "+str1.equals(str2));*

*System.out.println("So sánh không phân biệt hoa thường: "+str1.equalsIgnoreCase(str2));*

*// So sánh thứ tự a, b, c ....*

*String str3 = "abc";*

*String str4 = "bcde";*

*System.out.println("So sánh 2 chuỗi thường!");*

*System.out.println(""+str3.compareTo(str4)); //str3 < str4 , so sánh trả về -1*

*System.out.println(""+str4.compareTo(str3)); // str4 > str3, so sánh trả về 1*

*System.out.println(""+str3.compareTo(str3)); // str3 = str3, so sánh trả về 0*

*System.out.println("So sánh 2 chuỗi hoa và thường");*

*String str5 = "ANDroid.Vn";*

*String str6 = "android.vn";*

*System.out.println(""+str5.compareToIgnoreCase(str6)); //Không phân biệt hoa và thường*

*System.out.println(""+str5.compareTo(str6)); // Phân biệt chữ hoa và chữ thường!*

*System.out.println("Chuỗi này có là tập con của chuỗi kia không?");*

*String str7 = "android.vn";*

*String str8 = "vn";*

*//Quá trình so sánh có phân biệt chữ hoa và chữ thường!*

*System.out.println("Chuỗi vn nằm ở vị trí thứ "+str7.indexOf(str8)+" của chuỗi android.vn");*

*//Khi so sánh không tìm thấy thì sẽ trả về -1*

*str8 = "TN";*

*System.out.println("Chuỗi TN nằm ở vị trí thứ "+str7.indexOf(str8)+" của chuỗi android.vn");*

*System.out.println("Chuỗi này có bắt đầu hay kết thúc bằng chuỗi kia không?");*

*str7 = "android.vn";*

*String str9 = "and";*

*String str10 = "roid.vn";*

*System.out.println("str7.startsWith(str9) = "+str7.startsWith(str9));*

*System.out.println("str7.endsWith(str10) = "+str7.endsWith(str10));*

*}*

*}*

**+ Các bài tập thực hành mức độ cơ bản**

Bài 1: Để xử lý các văn bản, người ta xây dựng lớp VanBan có thuộc tính riêng là một xâu ký tự. Hãy:

1. Xây dựng lớp VanBan có:

+Các toán tử tạo lập : VanBan(), VanBan(String st).

+Phương thức đếm số từ của xâu thuộc tính trong lớp hiện tại.

+Phương thức đếm số từ kết thúc bởi ký tự ‘G’ (không phân biệt chữ hoa/thường).

2. Xây dựng các phương thức của lớp VanBan để chuẩn hoá xâu thuộc tính trong lớp hiện tại theo các tiêu chí sau:

+ Ở đầu và cuối của xâu không có ký tự trống

+ Ở giữa xâu, tại vị trí bất kỳ không tồn tại hai ký tự trống đứng liền nhau.

Bài 2: Xây dựng lớp Xau có thuộc tính riêng là : String st và các phương thức sau :

* public void CharacterDuplicate():Tìm và in ra các ký tự xuất hiện nhiều hơn một lần trong String cho trước không phân biệt chữ hoa hay chữ thường. Nếu các ký tự trong xâu đều là duy nhất thì xuất ra “NO”. Ví dụ chuỗi “Java” thì có ký tự ‘a’ hoặc String “JaVA” cũng có kết quả tương tự.

Gợi ý: Sử dụng HashMap

* boolean StringAnagram(String st1) (st1 là xâu đối số): Kiểm tra 2 xâu hiện thời st và xâu st1 có là đảo ngược của nhau hay không. Nếu có trả về giá trị là true ngược lại trả về giá trị là false. Ví dụ “word” và “drow” là 2 chuỗi đảo ngược nhau.
* boolean CharacterUnique(char ch): Tìm ký tự chỉ xuất hiện một lần trong chuỗi, nếu ký tự ch chỉ xuất hiện một lần duy nhất trong xâu st thì trả về true, trong các trường hợp còn lại trả về false
* String ReverseString(): Trả về một xâu đảo ngược của xâu hiện thời (st) boolean Stringnotcontainsdigit() Kiểm tra một chuỗi có chứa chữ số hay không, nếu có in ra false ngược lại true.

Ví dụ

“abc”, “” => true

“1abc”, “abc1”, “123”, “a1bc”, null => false

* Gợi ý: Sử dụng java regex, đây cũng là cách ngắn gọn nhất. Hoặc có thể dùng vòng lặp và kiểm tra từng ký tự trong xâu.
* *int CountNumbeofVowelsandConsonants()*: Đếm số lượng ký tự nguyên âm và phụ âm xuất hiện trong xâu hiện thời(st). Ví dụ chuỗi “java” có 2 nguyên âm “a” và 2 phụ âm “j” và “v”.

Gợi ý: Dùng switch case để đếm số lượng ký tự nguyên âm. Số phụ âm sẽ là độ dài của chuỗi trừ cho số lượng phụ âm.

**+ Các bài thực hành mức độ nâng cao.**

Bài 1: Xây dựng lớp Xâu có thuộc tính riêng là : String st và các phương thức sau :

-int ConvertStringNumbertoIntValue():Chuyển chuỗi số nguyên sang int value. Ví dụ “5646” thành int = 5646.

Gợi ý: Dùng Integer wrapper class để chuyển đổi kiểu string sang kiểu int.

public void ReplaceCharacter(char ch1, char ch2):thay thế các ký tự ch1 bằng ch2. Ví dụ “shareprogramming.net” chuyển ‘e’ sang ‘\*’ kết qủa “shar\*programming.n\*t”.

Gợi ý: Có thể dùng String#replace() hoặc StringBuilder để lặp và thao tác.

-String ReverStringByWord(): Đảo ngược các ký tự của chuỗi cách nhau bởi dấu cách mà không dùng thư viện. Ví dụ “I am developer ” => “developer am I”. Các ký tự bên trong chỉ cách nhau đúng một dấu khoảng cách.

Gợi ý: Cần loại bỏ dấu cách ở đầu và cuối câu, thao tác cắt chuỗi theo dấu cách và dùng StringBuilder hoặc StringBuffer để nối chuỗi.

boolean CheckStringPalindrome(): Chuỗi palindrome là chuỗi sau khi đảo ngược và chuỗi ban đầu hoàn toàn giống nhau, ví dụ “aba” là một chuỗi “aba”

Gợi ý: Đảo ngược chuỗi và so sánh với chuỗi ban đầu hoặc lặp và so sánh từng cặp chữ một.

-String removeCharacterDuplicate(): Xoá các ký tự xuất hiện nhiều hơn một lần trong xâu và chỉ giữ lại ký tự đầu tiên, vi dụ bananas => bans

Gợi ý: Sử dụng HashSet hoặc ArrayList lưu các ký tự đã xuất hiện trong lúc duyệt. Nếu ký tự được duyệt chưa có tỏng HashSet hoặc ArrayList thì cộng ký tự đó vào chuỗi kết quả và thêm ký tự đó vào HashSet hoặc ArrayList. Lưu ý sử dụng StringBuilder hoặc StringBuffer để thao tác cộng chuỗi.

-String findLongestPalindromeSubstring(): Tìm substring palindrome dài nhất trong một chuỗi cho trước. Biết rằng chuỗi palindrome là chuỗi sau khi đảo ngược và chuỗi ban đầu giống nhau.

## Bài thực hành số 9 (số tiết: 03 tiết)

+ Mục đích của bài thực hành

-Sinh viên hiểu và vận dụng thành thạo các khái niệm và các lớp đối tượng liên quan đến các luồng vào ra, lớp File, các phương thức truy cập file tuần tự và file ngẫu nhiên. Thiết kế yêu cầu bài toán theo hướng đối tượng

+ Yêu cầu cần đạt được của bài thực hành

**+ Bài giải mẫu (có hướng dẫn cách giải).**

Demo các phương thức cơ bản trên lớp File

**-Tạo File**

*import java.io.File; // Import the File class*

*import java.io.IOException; // Import the IOException class to handle errors*

*public class CreateFile {*

*public static void main(String[] args) {*

*try {*

*File myObj = new File("filename.txt");*

*if (myObj.createNewFile()) {*

*System.out.println("File created: " + myObj.getName());*

*} else {*

*System.out.println("File already exists.");*

*}*

*} catch (IOException e) {*

*System.out.println("An error occurred.");*

*e.printStackTrace();*

*} }}*

**-Ghi các giá trị vào file**

*import java.io.FileWriter; // Import the FileWriter class*

*import java.io.IOException; // Import the IOException class to handle errors*

*public class WriteToFile {*

*public static void main(String[] args) {*

*try {*

*FileWriter myWriter = new FileWriter("filename.txt");*

*myWriter.write("Files in Java might be tricky, but it is fun enough!");*

*myWriter.close();*

*System.out.println("Successfully wrote to the file.");*

*} catch (IOException e) {*

*System.out.println("An error occurred.");*

*e.printStackTrace();*

*} }}*

**-Đọc các giá trị từ file**

*import java.io.File; // Import the File class*

*import java.io.FileNotFoundException; // Import this class to handle errors*

*import java.util.Scanner; // Import the Scanner class to read text files*

*public class ReadFile {*

*public static void main(String[] args) {*

*try {*

*File myObj = new File("filename.txt");*

*Scanner myReader = new Scanner(myObj);*

*while (myReader.hasNextLine()) {*

*String data = myReader.nextLine();*

*System.out.println(data);*

*}*

*myReader.close();*

*} catch (FileNotFoundException e) {*

*System.out.println("An error occurred.");*

*e.printStackTrace();*

*} }}*

**-Lấy thông tin của file**

*import java.io.File; // Import the File class*

*public class GetFileInfo {*

*public static void main(String[] args) {*

*File myObj = new File("filename.txt");*

*if (myObj.exists()) {*

*System.out.println("File name: " + myObj.getName());*

*System.out.println("Absolute path: " + myObj.getAbsolutePath());*

*System.out.println("Writeable: " + myObj.canWrite());*

*System.out.println("Readable " + myObj.canRead());*

*System.out.println("File size in bytes " + myObj.length());*

*} else {*

*System.out.println("The file does not exist.");*

*} } }*

**-Xoá một file**

*import java.io.File; // Import the File class*

*public class DeleteFile {*

*public static void main(String[] args) {*

*File myObj = new File("filename.txt");*

*if (myObj.delete()) {*

*System.out.println("Deleted the file: " + myObj.getName());*

*} else {*

*System.out.println("Failed to delete the file.");*

*} } }*

**+ Các bài tập thực hành mức độ cơ bản**

Bài 1: Viết một ứng dụng theo các mô tả như sau:

a. Tạo file nhanvien.java và thực hiện các công việc:

- Tạo một lớp có tên NhanVien bao gồm các thuộc tính: manv kiểu String, hoten kiểu String, tuoi kiểu int, luong kiểu float.- Viết các setter, getter, constructor, và toString.

- Viết phương thức main để nhập vào các thông tin cho 1 nhân viên từ bàn phím và in các thông tin của nhân viên này ra màn hình.

Lưu ý không sử dụng Scanner để nhập liệu.

b. Tạo file write\_to\_file.java, viết phương thức main cho phép nhập các thông tin cho 3 nhân viên từ bàn phím, sử dụng FileWriter để ghi các thông tin vừa nhập vào file text ‘nhanvien.txt’. Mở file ‘nhanvien.txt’ để xem đã ghi được thông tin vào file hay chưa.

c. Tạo file read\_from\_file.java, viết phương thức main, sử dụng FileReader để đọc nội dung file ‘nhanvien.txt’ và in những gì đã đọc được từ file ra màn hình.

d. Tạo file write\_object.java, tạo mảng 3 nhanvien, nhập thông tin các nhân viên, sử dụng ObjectOutputStream để ghi mảng nhân viên này vào file ‘nhanvien.bin’.

e. Tạo file read\_object.java, tạo mảng 3 nhanvien, sử dụng ObjectInputStream để đọc dữ liệu từ file ‘nhanvien.bin’ và in mảng nhân viên ra màn hình.

**+ Các bài thực hành mức độ nâng cao.**

Bài 1: Bạn hãy viết một ứng dụng Java để tìm tổng số từ (word) ở trong một tập tin. Tập tin có thể có nhiều dòng, mỗi dòng có một số

lượng từ nào đó. Dưới đây là các ví dụ về tập tin và kết quả của việc thực thi ứng dụng:

a. Ví dụ về tập tin:

Tập tin C:\Java\file1.txt chứa nội dung:

*Website chuyen ve lap trinh va dao tao ICTU*

*ICTU mang lai cho moi nguoi nhung bai hoc bo ich*

b. Ví dụ về việc chạy ứng dụng:

Xin moi nhap vao ten cua tap tin: C:\Java\file1.txt

*Tap tin cua ban co: 20 tu (words).*Bài 2: Tạo 1 ứng dụng bằng Java dùng để chuyển đổi các loại tiền tệ khác nhau. Ứng dụng gồm 2 lớp:

Lớp ExchangeRate lưu tất cả các thông tin về tỷ giá, thông tin tỷ giá có thể được đọc từ file text, dữ liệu của file text được người dùng nhập vào. Lớp ExchangeRate có phương thức readInfo() dùng để đọc thông tin từ file text này. File text có chứa nhiều dòng, mỗi dòng gồm 3 mẫu dữ liệu phân tách nhau bằng dấu chấm phẩy (;): Code1;Code2;Rate, trong đó Code1 là mã của loại tiền tệ thứ nhất, Code2 là mã của loại tiền tệ thứ 2 và Rate là tỷ giá của hai loại tiền tệ này. Ví dụ như USD;IDN;15792, USD;EUR;0.87. Thông tin đọc được từ file text được lưu vào 1 collection, ví dụ như ArrayList, Vector, ... Lớp ExchangeRate cũng có phương thức convert() gồm ba tham số: code1, code2 và số lượng tiền muốn đổi, phương thức sẽ trả về số lượng tiền sau khi được trao đổi.

Lớp thứ hai là lớp chứa phương thức Main(). Lớp này sẽ được dùng để test lớp ExchangeRate

Ứng dụng cho phép người dùng chuyển đổi tiền tệ nhiều lần tùy thích bằng cách đưa ra câu hỏi: "Bạn có muốn tiếp tục không?", câu trả lời là "yes" và "no".

Giả sử rằng bạn sẽ được cấp một file có tên "rate.txt" và có nội dung như sau:

EUR;USD;1.2

USD;IDN;15789

USD;EUR;0.83

CAD;IDN;16869

## Bài thực hành số 10 (số tiết: 03 tiết)

+ Mục đích của bài thực hành

- Thiết kế yêu cầu bài toán theo hướng đối tượng, tổng hợp các kiến thức đã học xây dựng các chương trình giải các bài toán về hướng đối tượng trên ngôn ngữ java một cách thành thạo. Khai thác được các thuộc tính và các phương thức của các lớp cơ bản trong java, sử dụng mảng, xâu, file.. và các thư viện java hỗ trợ trong các bài tập.

+ Yêu cầu cần đạt được của bài thực hành

-Sinh viên định nghĩa các lớp đối tượng theo phương pháp hướng đối tượng để giải quyết các yêu cầu của bài toán

**+ Bài giải mẫu (có hướng dẫn cách giải)**

Bài 1: Lọc các số có trong chuỗi.

*import java.util.regex.Matcher;*

*import java.util.regex.Pattern;*

*public class JavaExtractNumbersInString {*

*public static void main(String[] args) {*

*String str = "This string has numbers 35 and 21 in it.";*

*System.out.println(str);*

*// Find 1 or more numbers.*

*Pattern p = Pattern.compile("\\d+");*

*Matcher m = p.matcher(str);*

*while (m.find()) {*

*String found = str.substring(m.start(), m.end());*

*Integer i = Integer.valueOf(found);*

*System.out.println("Number found is: " + i);*

*}*

*System.out.println("\n \t -- VNLIVES.NET --");*

*}*

*}*

**Bài 2:** Đổi ngày trong java

*import java.util.Date;*

*import java.text.SimpleDateFormat;*

*import java.text.ParseException;*

*public class ConvertStringToDate {*

*public static void main(String[] args) {*

*SimpleDateFormat formatter = new SimpleDateFormat("ddMMyyy");*

*String dateString = "12042014";*

*try {*

*Date date = formatter.parse(dateString);*

*System.out.println("Date string: " + dateString);*

*System.out.println("String convert to date: " + date);*

*System.out.println("Date with formatter: " + formatter.format(date));*

*} catch (ParseException e) {*

*e.printStackTrace();*

*}*

*}*

*}*

**+ Các bài tập thực hành mức độ cơ bản**

Bài 1: Xây dựng lớp PhanSo với hai thuộc tính riêng xác định tử số và mẫu số của phân số, đồng thời có các phương thức sau:

+ Các toán tử tạo lập

+ Các phép toán cộng, trừ, nhân, chia hai phân số

+ Phép kiểm tra một phân số có phải tối giản hay không

+ Phép tìm dạng tối giản của phân số

2. Viết một chương trình ứng dụng thực hiện việc nhập vào một dãy các phân số

+ In ra màn hình dạng tối giản của các phân số đó

+ Sắp xếp các phân số theo thứ tự tăng dần của tử số của mỗi phân số

Bài 2: Để quản lý các hộ dân trong một khu phố, người ta quản lý các thông tin như sau:

* Với mỗi hộ dân, có các thuộc tính:

+ Số thành viên trong hộ (số người)

+ Số nhà của hộ dân đó (Số nhà được gắn cho mỗi hộ dân)

+ Thông tin về mỗi cá nhân trong hộ gia đình.

* Với mỗi cá nhân, người ta quản lý các thông tin như: họ và tên, tuổi, năm sinh, nghề nghiệp.

1. Hãy xây dựng các lớp: NhanSu để quản lý thông tin về mỗi cá nhân; lớp HoDan để quản lý thông tin về các hộ gia đình. Viết các phương thức để nhập, hiển thị thông tin cho mỗi cá nhân.

2. Cài đặt chương trình thực hiện các công việc sau:

+ Nhập vào một dãy gồm n hộ dân (n - nhập từ bàn phím).

+ Hiển thị ra màn hình thông tin về các hộ trong khu phố.

Bài 3: Một công ty được giao nhiệm vụ quản lý các phương tiện giao thông gồm các loại: ô tô, xe máy, xe tải.

+ Mỗi loại phương tiện giao thông cần quản lý: Số máy, hãng sản xuất, năm sản xuất, giá bán.

+ Các ô tô cần quản lý: số chỗ ngồi, kiểu động cơ

+ Xe máy cần quản lý: công suất

+ Xe tải cần quản lý: trọng tải.

1. Xây dựng các lớp XeTai, XeMay, OTo kế thừa từ lớp PTGT; đồng thời có các hàm để truy nhập, hiển thị và kiểm tra các thuộc tính của các lớp.

2. Xây dựng lớp QLPTGT cài đặt các phương thức thực hiện các chức năng sau:

+ Nhập thông tin đăng ký cho một danh sách gồm m phương tiện

+ Nhập vào số máy của một phương tiện, cho biết phương tiện đó thuộc loại gì? Và hiển thị thông tin quản lý của phương tiện đó.

**+ Các bài thực hành mức độ nâng cao.**

Bài 1: Cho hai đa thức Pn(x) và Qm(x). Hãy viết chương trình thực hiện những thao tác sau:

Tạo lập hai đa thức (nhập hệ số cho đa thức từ bàn phím hoặc file)

Tính Pn(x0) và Qm(x0)

Tìm đạo hàm cấp l ≤n của đa thức.

Tìm Pn(x) + Qm(x)

Tìm Pn(x) - Qm(x)

Tìm Pn(x) / Qm(x) và đa thức dư