

## MỤC LỤC

MỤC LỤC .....	1
DANH MỤC CÁC HÌNH .....	<b>Error! Bookmark not defined.</b>
CHƯƠNG 1: NGUYÊN LÝ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG.....	3
1.1 Các cách tiếp cận trong lập trình .....	3
1.1.1 Lập trình tuyến tính .....	3
1.1.2 Lập trình có cấu trúc.....	3
1.1.3 Lập trình hướng đối tượng.....	6
1.2 Các khái niệm cơ bản của lập trình hướng đối tượng.....	7
1.2.1 Đối tượng.....	7
1.2.2 Lớp đối tượng .....	9
1.2.3 Trừu tượng hóa .....	10
1.2.4 Bao bọc và che dấu thông tin .....	11
1.2.5 Kế thừa và mở rộng .....	12
1.2.6 Đa xạ và nạp chồng .....	13
1.2.7 Liên kết động.....	14
1.2.8 Truyền thông điệp.....	14
Chương 2: TỔNG QUAN VỀ JAVA.....	17
2.1 lịch sử phát triển của java. ....	17
2.1.2. Đặc điểm của ngôn ngữ Java. ....	17
2.1.3. Biên dịch và thông dịch chương trình java .....	19
2.2 Cấu trúc chung của một chương trình java.....	20
2.2.1 Cấu trúc chung của chương trình java .....	20
2.2.2 Ví dụ minh họa .....	21
2.3. Các dạng chương trình ứng dụng của Java.....	21
2.3.1 Chương trình ứng dụng độc lập.....	21
2.3.2 Chương trình ứng dụng nhúng .....	22
2.4 Một số thành phần cơ sở của Java .....	24
2.4.1 Các phần tử cơ sở của Java.....	24
2.4.2 Các kiểu dữ liệu trong Java .....	25
2.4.3. Khai báo biến.....	26
2.4.4 Biểu thức trong Java.....	28
2.4.5 Các phép toán trong Java.....	28
2.4.6 Câu lệnh điều khiển trong java.....	31
2.4.7 Các lệnh chuyển vị trong java .....	32
CHƯƠNG 3: GÓI, LỚP VÀ GIAO DIỆN.....	33
3.1. Gói .....	33
3.1.1 Tạo gói cho chương trình và cách sử dụng gói .....	33
3.1.2 Giới thiệu một số gói quan trọng của Java .....	34
3.2 Lớp đối tượng .....	35
3.2.1 Định nghĩa lớp .....	35
3.2.2 Các thuộc tính dữ liệu của lớp .....	36
3.2.3 Các hàm thành phần của lớp.....	36
3.2.4 Truyền tham số và gọi phương thức .....	41
3.3.Toán tử tạo lập đối tượng .....	44
3.3.1 Khai báo toán tử tạo lập.....	44
3.3.2 Toán tử this() và super() .....	46
3.3.3 Khởi khởi đầu tĩnh.....	47

3.3.4. Dọn dẹp: kết thúc và thu rác .....	48
3.4. Quan hệ kế thừa giữa các lớp .....	48
3.5 Nạp chồng .....	49
3.6 Viết chồng (viết đè) .....	50
3.7 Nạp chồng các toán tử tạo lập. ....	52
3.8 Cơ chế che bóng của các biến .....	53
3.9 Lớp nội .....	55
3.10 Xử lý ngoại lệ trong java .....	55
3.10.1 Cấu trúc phân cấp của các lớp xử lý ngoại lệ .....	56
3.10.2 Câu lệnh try, catch và finally .....	57
3.10.3 Ném ngoại lệ bằng lệnh 'throw' .....	59
3.10.4 Mệnh đề throws .....	60
3.10.5 Tự định nghĩa ngoại lệ .....	61
3.10.6 Lăn vết ngoại lệ StackTrace .....	62
3.11 Giao diện và sự mở rộng quan hệ kế thừa trong Java .....	63
3.11.1 Các bước tạo interface .....	63
3.11.2 Khai báo giao diện .....	64
3.11.3 Sử dụng giao diện .....	65
3.12. Lớp trừu tượng .....	66
3.12.1 Khai báo phương thức của lớp trừu tượng .....	66
3.12.2 Sử dụng lớp trừu tượng .....	67
CHƯƠNG 4: MẢNG VÀ CÁC LỚP CƠ SỞ TRONG GÓI JAVA.LANG .....	69
4.1. Mảng trong Java .....	69
4.1.1 Khai báo mảng .....	69
4.1.2 Tạo lập mảng .....	69
4.1.3 Truy cập các phần tử của mảng .....	70
4.1.4 Mảng nhiều chiều .....	70
4.2 Các lớp cơ sở trong gói java.lang .....	71
4.2.1 Lớp Object .....	71
4.2.2 Các lớp nguyên thủy .....	72
4.2.3 Lớp Math .....	76
4.2.4 Lớp String .....	78
4.2.5. Lớp StringBuffer .....	82

## CHƯƠNG 1: NGUYÊN LÝ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

### Mục tiêu

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:
  - Lớp (Class), Đối tượng (Object), nhận thức được sự khác biệt giữa Lớp và Đối tượng
  - Nhận thức được sự cần thiết đối với Thiết lập (Construction) và Hủy (Destruction)
  - Hiểu về tính Bền vững (Persistence).
  - Hiểu biết về tính Thừa kế (Inheritance).
  - Định nghĩa tính Đa hình (Polymorphism)
- Liệt kê những thuận lợi của phương pháp hướng Đối tượng

### 1.1 Các cách tiếp cận trong lập trình

#### 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.

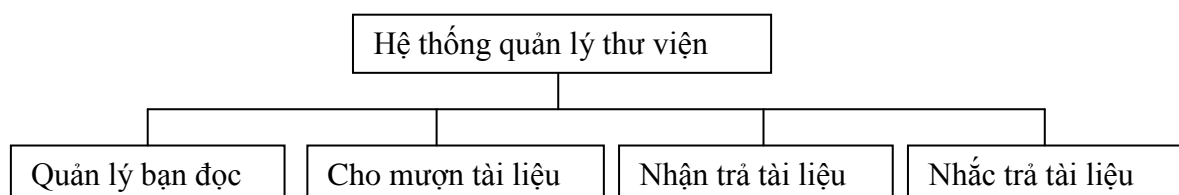
- **Dựa vào chức năng nhiệm vụ:** Khi khảo sát, phân tích một hệ thống chúng ta thường tập trung vào các nhiệm vụ mà nó cần thực hiện. Chúng ta tập trung trước hết nghiên cứu các yêu cầu của bài toán để xác định các chức năng chính của hệ thống.

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.

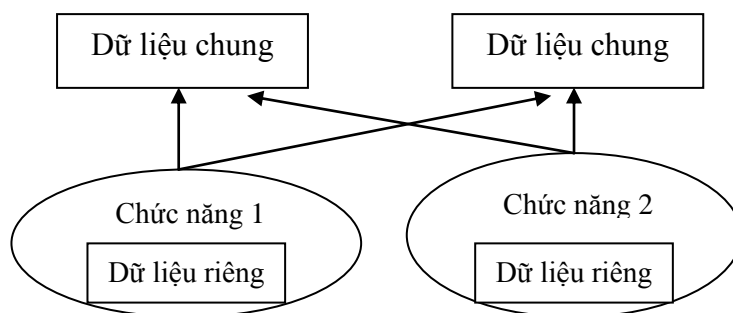
Ví dụ, hệ thống quản lý thư viện có thể phân chia từ trên xuống như sau:



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 thông 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.

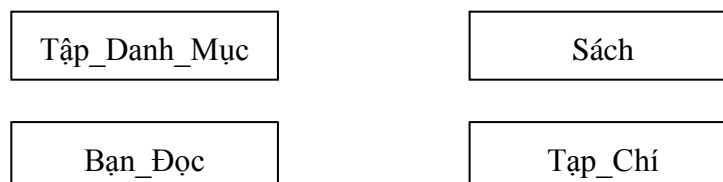
• **Khả năng tái sử dụng (Reuse) bị hạn chế và không hỗ trợ cơ chế kế thừa (Inheritance).** Để có độ thích nghi cao thì mỗi thành phần phải là tự chứa. Muốn là tự chứa hoàn toàn thì một thành phần không nên dùng các thành phần ngoại lai. Tuy nhiên, điều này lại mâu thuẫn với kinh nghiệm nói rằng các thành phần nên 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 trạng thái (sách đang mở, đóng) và có thể chứa các đối tượng nhỏ hơn (các trang sách). Phương thức của 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 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 màu 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 màu 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

Ôn 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.

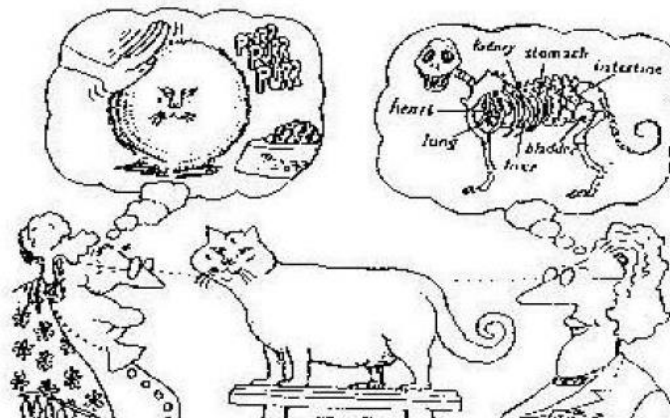
Ví dụ: lớp sinh viên có 4 thuộc tính, 2 hàm (phương thức)

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

Nam	SinhVien
Hoten	Hoten
tuoi	tuoi
quequan	quequan
lop	lop
getTen	getTen
getTuoi	getTuo...
...	

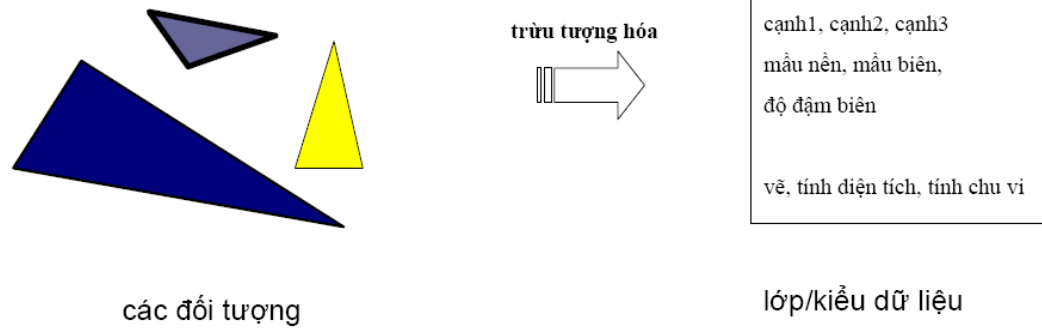
### 1.2.3 Trừu tượng hóa

- 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

- Trừu tượng hóa là sự mở rộng các khái niệm kiểu dữ liệu và cho phép định nghĩa những phép toán trừu tượng trên các kiểu dữ liệu trừu tượng.
- Để xác định lớp ta phải sử dụng khái niệm trừu 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 buộc 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 hạn 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.

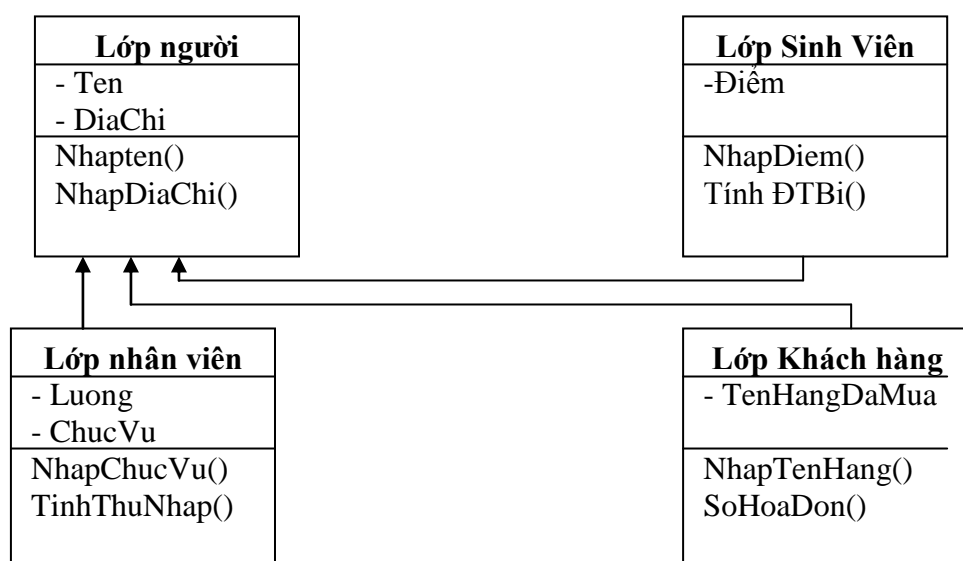
### 1.2.5 Kế thừa và mở rộng

- Nguyên lý kế thừa cho phép các đối tượng của lớp này được quyền truy cập vào một số tính chất (dữ liệu và hàm) 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 để 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 thừa 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: nhậpTen(), nhậpDiaChi(), nhậpDiem().
- \* Lớp nhân viên: - Thuộc tính: tên, địa chỉ, lương, chức vụ, và các hàm: nhậpTen(), nhậpDiaChi(), nhậpChucVu(), tínhThuNhap()
- \* Lớp khách hàng: - Thuộc tính: tên, địa chỉ, tên hàng đã mua, và các hàm: nhậpTen(), nhậpDiaChi(), nhậpTenHang(), nhậpHoaDon()



Hình 1-6: ví dụ kế thừa

\* Sự kế thừa và thiết kế hướng đối tượng.

Để phát triển cho một ứng dụng java cần:

- Chia nhỏ mã lệnh thành đơn vị nhỏ nhất. Ví dụ: Viên đạn: có các đơn vị nhỏ hơn: kích thước, tầm sát thương, số lượng, có thể cháy.
- Xem xét các tính chất chung giữa các đơn vị: Ví dụ:
- Xem xét sự khác nhau giữa các đơn vị.
- Gom các tính chất chung và lặp lại với nhau.
- Sử dụng các đối tượng để thêm vào theo yêu cầu.

### 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 thực hiện khác nhau.

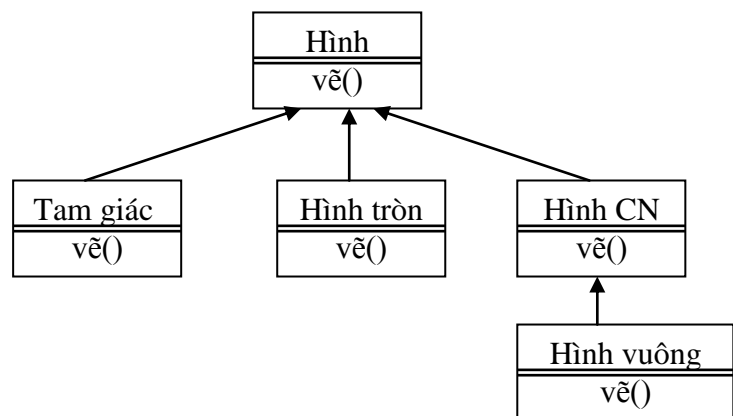
Một thông điệp (lời gọi hàm) được hiểu theo các cách khác nhau tùy theo danh sách tham số của thông điệp. Ví dụ: nhận được cùng một thông điệp “nhảy”, một con kangaroo và một con cóc nhảy theo hai kiểu khác nhau: chúng cùng có hành vi “nhảy” nhưng các hành vi này có nội dung khác nhau.



chú ý rằng kangaroo và cóc thuộc hai nhánh trong cây phả hệ động vật

**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 hàm 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, khác nhau ở danh sách tham số.

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

*float (float, float)*

*string cong (string, string)*

**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 phương thức 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

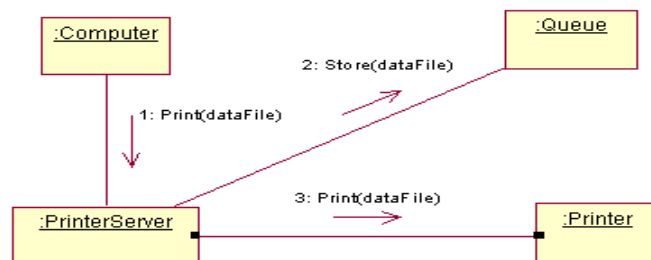
Ví dụ: khi chương trình gộp hàm vẽ cho đối tượng hình tròn thì hệ thống sẽ liên kết tới hàm vẽ được định nghĩa trong lớp hình tròn.

**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 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 được loại bỏ khi chương trình ra khỏi miền xác định 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 phân 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
- “ Phân 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).*

“ Lưu ý rằng biên dịch các chương trình hướng đối tượng cũng chậm.

Một số hệ thống “hướng đối tượng” thời kỳ đầu không có các lớp “ chỉ có các “đối tượng” và các “thông điệp” (v.d. Hypertalk)

Hiện giờ, đã có sự thống nhất rằng hướng đối tượng là: lớp - class, thừa kế - inheritance và liên kết động - dynamic binding.

*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**

- 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?



## **Chương 2: TỔNG QUAN VỀ JAVA**

### **Mục tiêu:**

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, các thành phần cơ bản trong java như:

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

### **2.1. Sơ lược về java**

#### ***2.1.1. 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

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.

#### ***2.1.2. Đặc điểm của ngôn ngữ Java.***

Ngôn ngữ Java có những đặc trưng cơ bản sau:

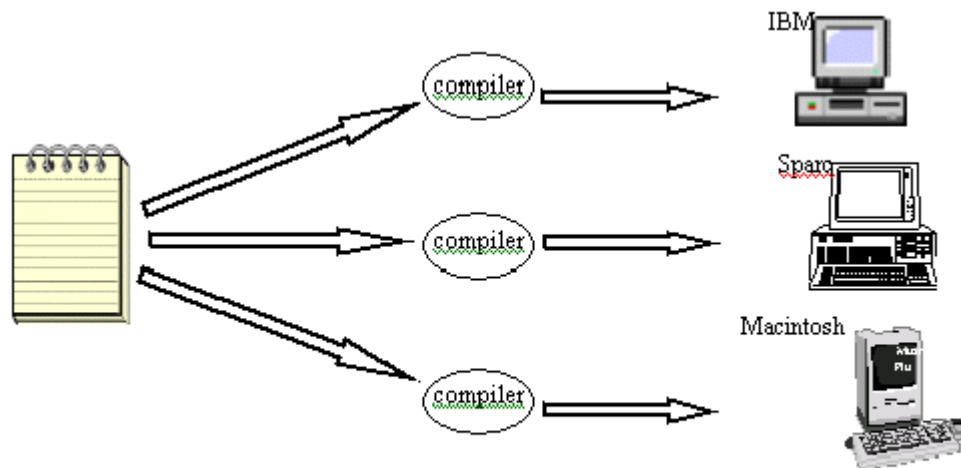
**Đơn giản:** phát triển trên một ngôn ngữ dễ học và quen thuộc với đa số người lập trình là C và Java loại bỏ các phức tạp của C và C++ như: con trỏ, đa kế thừa, định nghĩa chồng toán tử, không sử dụng lệnh “goto” cũng như file header (.h), loại bỏ cấu trúc “struct” và “union”.

**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 mã 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 bắt 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.

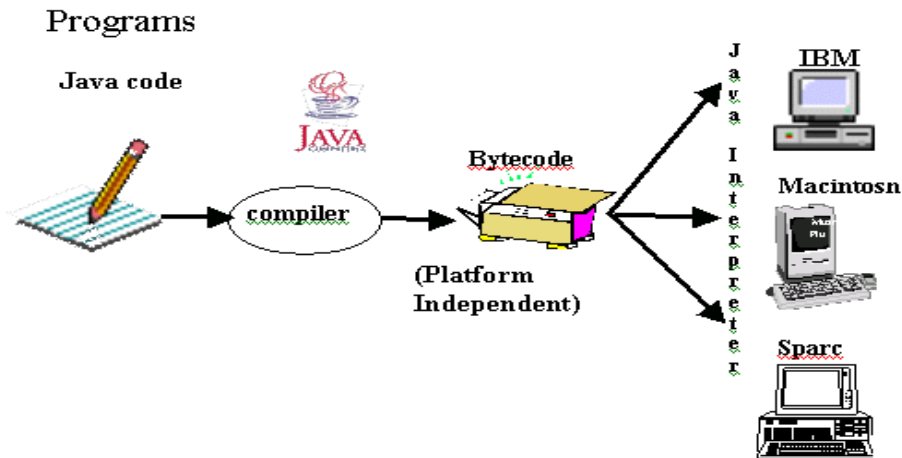
**Linh động** Java được thiết kế như một ngôn ngữ động để đáp ứng cho những môi trường mở. Các chương trình Java chứa rất nhiều thông tin thực thi nhằm kiểm soát và truy nhập đối tượng lúc chạy. Điều này cho phép khả năng liên kết động mã.

### 2.1.3. 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”

## 2.2 Cấu trúc chung của một chương trình java

### 2.2.1 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
```

### 2.2.2 Ví dụ minh họa

```
public class HelloWorldApp{  
    public static void main(String[] args){  
        System.out.println("HelloWorld");  
    }  
}
```

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

### 2.3.1 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 HelloWorldApp.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 HelloWorldApp**

- Nếu chương trình đúng ta sẽ thấy dòng chữ HelloWorld 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 HelloWorldApp**

```
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.

### 2.3.2 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 dẹp 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 tùy 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 họa `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.

## 2.4 Một số thành phần cơ sở của Java

### 2.4.1 Các phần tử cơ sở của Java

#### \* 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 \$, O, 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:

- + Định danh cho các lớp: chữ cái đầu của mỗi từ viết hoa.
- + Định danh cho các biến, phương thức, đối tượng: chữ cái đầu của mỗi từ trong định danh đều viết hoa trừ từ đầu tiên.

- **Chú thích (Comment)**

```
// Chú thích trên một dòng
/* Chú thích trên nhiều dòng
... */
/** Chú thích trong tư liệu (javadoc)
... */
```

- **Các từ khóa của java**



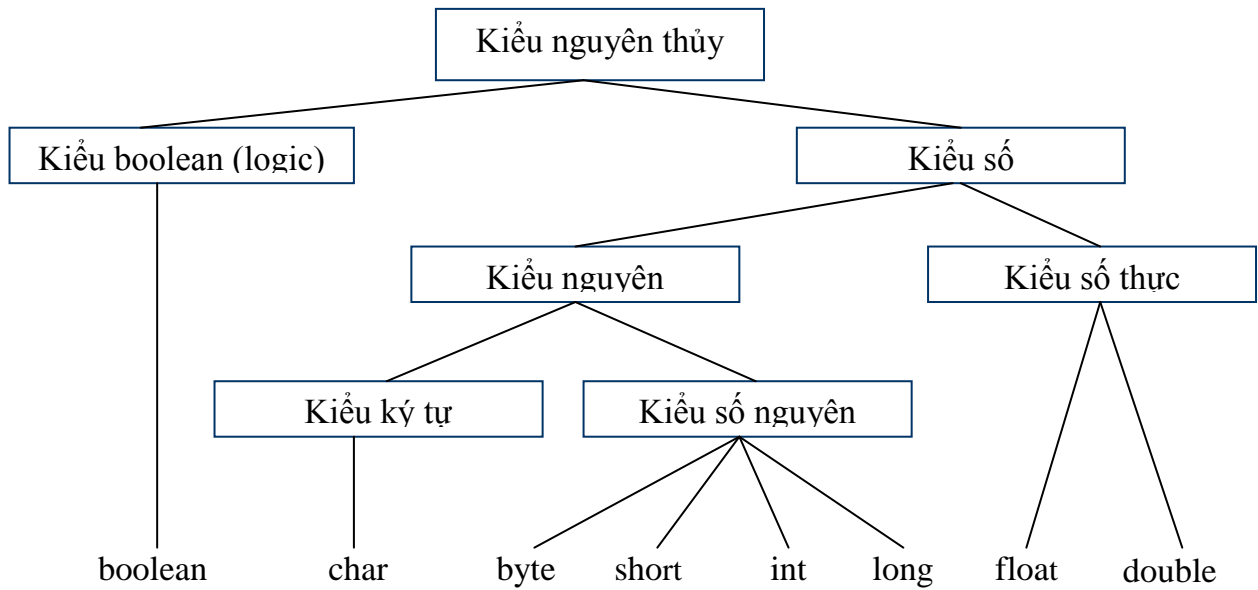
abstract	boolean	break	byte
case	catch	char	class
const	continue	default	do
double	else	extends	final
finally	float	for	goto
if	implements	import	instanceof
int	interface	long	native
new	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	try
void	volatile	while	

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

Escape Sequence	Mô tả
\n	Xuống dòng mới
\r	Chuyển con trỏ đến đầu dòng hiện hành
\t	Chuyển con trỏ đến vị trí dừng Tab kế tiếp (ký tự Tab)
\\	In dấu \
\'	In dấu nháy đơn (')
\''	In dấu nháy kép (")

#### 2.4.2 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:

Hình 2-3: Các kiểu nguyên thủy trong Java

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*, v.v

Kiểu	Kích thước (bytes)	Giá trị min	Giá trị max	Giá trị mặc định
byte	1	-256	255	0
short	2	-32768	32767	0
int	4	$-2^{31}$	$2^{31} - 1$	0
long	8	$-2^{63}$	$2^{63} - 1$	0L
float	4			0.0f
double	8			0.0d

**2.4.3. Khai báo biến.****\* Các loại biến trong Java**

- *Các biến thành phần* là các thành phần của lớp và được khởi tạo giá trị mỗi khi một đối tượng của lớp được tạo ra.
- *Các biến tham chiếu đối tượng (Object Reference)* là các biến được sử dụng để xử lý các đối tượng. Biến tham chiếu phải được khai báo và khởi tạo giá trị trước khi sử dụng.

- *Biến tĩnh (static)* cũng là các thành viên của lớp nhưng không phải đại diện cho từng đối tượng mà cho cả lớp.
- *Các biến cục bộ (local)* là những biến được khai báo trong các phương thức và trong các khối. Trong Java các biến local phải được khai báo trước khi sử dụng.

### **Khai báo biến**

[phạm vi]<Kiểu dữ liệu> <tên biến>;

hoặc khai báo 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.

<b><i>Kiểu dữ liệu</i></b>	<b><i>Giá trị mặc định</i></b>
boolean	false
char	'\u0000'
byte, short, int, long	0
float, double	+0.0F, +0.0D
Tham chiếu đối tượng	null

*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 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 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.

**2.4.4 Biểu thức trong Java**

Số ưu tiên	Tên gọi	Các phép toán	Quy 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

**2.4.5 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 2	11010111	-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

Quy tắc ép kiểu có dạng: `(<type>) <exp>`

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

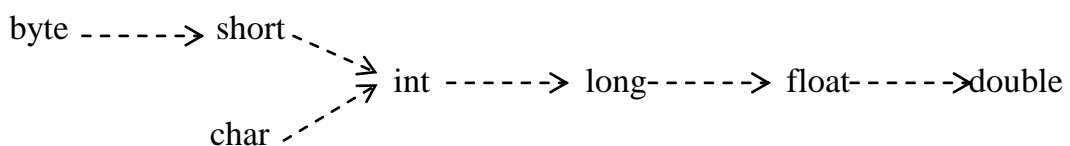
Ví dụ: `float f = (float) 100.15D;` // Chuyển số 100.15 dạng kiểu `double` sang `float`

- ✓ 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 hẹp 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 hẹp hơn được gọi là *thu hẹp kiểu*. Lưu ý là *thu hẹp kiểu có thể dẫn tới mất thông tin*.

#### 2.4.6 Câu lệnh điều khiển trong java

##### 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)

###### Câu lệnh switch

```
switch (<Biểu thức nguyên>) {
    case nhan1: <Câu lệnh 1>
    case nhan2: <Câu lệnh 2>
    . . .
    case nhann: <Câu lệnh n>
    [default: <Câu lệnh >]
}
```

##### Các câu lệnh lặp

###### \* Lệnh while

```
while (<Điều kiện kết thúc chu trình>
    <Thân chu trình>
```

###### \* Câu lệnh do-while (chu trình do-while)

```
do
    <Thân chu trình>
while (<Điều kiện kết thúc chu trình>)
```

###### \* Câu lệnh for

```
for (<Biểu thức bắt đầu>; <Điều kiện lặp>; <Biểu thức gia tăng>)
    <Thân chu trình>
```

✓ Tất cả các biến được khai báo trong <Biểu thức bắt đầu> đều là cục bộ trong khối thân của chu trình *for*.

✓ 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.

#### 2.4.7 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 bỏ 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.

##### 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 thỏa (còn *true*).

##### 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



## CHƯƠNG 3: GÓI, LỚP VÀ GIAO DIỆN

### 3.1. Gói

#### 3.1.1 Tạo gói cho chương trình và cách sử dụng gói

Gói tương tự như thư mục lưu trữ những lớp, interface và các gói con khác.

Những ưu điểm khi dùng gói (Package):

- Cho phép tổ chức các lớp vào những đơn vị nhỏ hơn
- Giúp tránh được tình trạng trùng lặp khi đặt tên.
- Cho phép bảo vệ các lớp đối tượng
- Tên gói (Package) có thể được dùng để nhận dạng chức năng của các lớp.

Tạo gói:

Sử dụng lệnh: `package <tên gói>;`

Những lưu ý khi tạo gói:

- Mã nguồn phải bắt đầu bằng lệnh ‘package’
- Mã nguồn phải nằm trong cùng thư mục mang tên của gói
- Tên gói nên bắt đầu bằng ký tự thường (lower case) để phân biệt giữa lớp đối tượng và gói
- Những lệnh khác phải viết phía dưới dòng khai báo gói là mệnh đề import, kể đến là các mệnh đề định nghĩa lớp đối tượng
- Những lớp đối tượng trong gói cần phải được biên dịch

Sử dụng gói:

Để chương trình Java có thể sử dụng những gói này, ta phải import gói vào trong mã nguồn

- Cú pháp: `import gói (importing packages):`
- Xác định tập tin cần được import trong gói Hoặc có thể import toàn bộ gói

Các bước tạo ra gói (Package)

Khai báo gói

Import những gói chuẩn cần thiết

Khai báo và định nghĩa các lớp đối tượng có trong gói

Lưu các định nghĩa trên thành tập tin .java, và biên dịch những lớp đối tượng đã được định nghĩa trong gói.

Sử dụng những gói do người dùng định nghĩa (user-defined packages):

- Mã nguồn của những chương trình này phải ở cùng thư mục của gói do người dùng định nghĩa.

- Để những chương trình Java khác sử dụng những gói này, import gói vào trong mã nguồn
- Import những lớp đối tượng cần dùng
- Import toàn bộ gói
- Tạo tham chiếu đến những thành viên của gói

### Xác lập CLASSPATH

Là danh sách các thư mục, giúp cho việc tìm kiếm các tập tin lớp đối tượng tương ứng nên xác lập CLASSPATH trong lúc thực thi (runtime), vì như vậy nó sẽ xác lập đường dẫn cho quá trình thực thi hiện hành

Gói và điều khiển truy xuất (Packages & Access Control)

	<u>public</u>	<u>protected</u>	No modifier	<u>private</u>
Same class	Yes	Yes	Yes	Yes
Same package <u>subclass</u>	Yes	Yes	Yes	No
Same package <u>non-subclass</u>	Yes	Yes	Yes	No
Different package <u>subclass</u>	Yes	Yes	No	No
Different package <u>non-subclass</u>	Yes	No	No	No

### **3.1.2 Giới thiệu một số gói quan trọng của Java**

- |  |
|--|
| + <b>java.lang:</b> Chứa các lớp quan trọng nhất của ngôn ngữ Java. Chúng bao gồm các kiểu dữ liệu cơ bản như Character, Integer,... Chúng cũng chứa các lớp làm nhiệm vụ xử lý lỗi và các lớp nhập xuất chuẩn. Một vài lớp quan trọng khác như String hay StringBuffer. |
| + <b>java.applet:</b> Đây là package nhỏ nhất chứa một mình lớp Applet. Các lớp Applet nhúng trong trang Web đều dẫn xuất từ lớp này.  |
| + <b>java.awt:</b> Package này được gọi là Abstract Window Toolkit (AWT). Chúng chứa các tài nguyên dùng để tạo giao diện đồ họa. Một số lớp bên trong là: Button, GridBagLayout, Graphics.  |
| + <b>java.io:</b> Cung cấp thư viện nhập xuất chuẩn của ngôn ngữ. Chúng cho phép tạo và quản lý dòng dữ liệu theo một vài cách.  |

+ <b>java.util:</b> Package này cung cấp một số công cụ hữu ích. Một vài lớp của package này là: Date, Hashtable, Stack, Vector và StringTokenizer.
+ <b>java.net:</b> Cung cấp khả năng giao tiếp với máy từ xa. Cho phép tạo và kết nối với Socket hoặc URL.
+ <b>java.awt.event:</b> Chứa các lớp dùng để xử lý các sự kiện trong chương trình như chuột, bàn phím.
+ <b>java.rmi:</b> Công cụ để gọi hàm từ xa. Chúng cho phép tạo đối tượng trên máy khác và sử dụng các đối tượng đó trên máy cục bộ.
+ <b>java.security:</b> Cung cấp các công cụ cần thiết để mã hóa và đảm bảo tính an toàn của dữ liệu truyền giữa máy trạm và máy chủ.
+ <b>java.sql:</b> Package này chứa Java DataBase Connectivity (JDBC), dùng để truy xuất cơ sở dữ liệu quan hệ như Oracle, SQL Server.
+ <b>java.math:</b> Package này chứa các hàm toán học như abs, sqrt, sin, cos, max, min, round, exp ...

## 3.2 Lớp đối tượng

### 3.2.1 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:

<pre>[&lt;Phạm vi hoặc kiểm soát truy nhập&gt;] <b>class</b> &lt;Tên lớp&gt;     [extends &lt;Tên lớp cha&gt;] [implements &lt;Tên giao diện&gt;]     {         &lt;Các thành phần của lớp&gt;     }</pre>
--

\* Giải thích:

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

- **abstract:** 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 cộng) Cho phép sử dụng mọi nơi trong hệ thống.
- **final:** (lớp cuối cùng) Không thể sản sinh dẫn xuất ra các lớp con
- Nếu không khai báo phạm vi là truy cập trong gói chứa lớp đó.

- *tên\_lớp\_cha*: Tên lớp được phép kế thừa
- *DS\_tên\_các\_giao\_diện*: Tên các giao diện (được viết phân cách nhau bởi 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. Những phần này sẽ được đề cập chi tiết ở các phần sau.*

Các thành phần của lớp gồm:

- Các biến thành phần : Xác định các thuộc tính(dữ liệu) của các đối tượng thuộc lớp đó
- Các toán tử tạo lập: Xác định cách cài đặt một đối tượng thuộc lớp
- Các phương thức: Xác định hành vi của các đối tượng thuộc lớp đó

### 3.2.2 Các thuộc tính dữ liệu của lớp

- Biến thành phần có thể thuộc kiểu nguyên thuỷ, kiểu tham chiếu. Có thể khai báo các biến kiểu nguyên thuỷ, kiểu mảng, kiểu String, hay kiểu lớp

\* Mẫu 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**: Biến tĩnh gắn với lớp, các đối tượng thuộc lớp có truy cập đến được.
- **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 chỉ bên trong đối tượng.
- **final**: Không thể thay đổi
- **volatile**: có thể thay đổi bất thường và thông báo cho chương trình dịch không nên thực hiện tối ưu đối với những biến như thế

### 3.2.3 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.

- 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>(<DS 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 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: Không phụ thuộc vào đối tượng cụ thể, nó tác động lên toàn thể các mẫu tạo ra từ lớp chứa nó
  - abstract: Phương thức trừu tượng, không cài đặt gì 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 cho phương thức tới hạn, nhằm ngăn các tác động của các đối tượng 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, sau đó các phương thức được thực hiện trên các đối tượng cụ thể. Để sử dụng phương thức ta dùng cấu trúc:

**<tên đối tượng>.<tên\_hàm>(<DS các tham số thực sự>)**

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

Khởi tạo giá trị cho đối tượng :

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

Khai báo đối tượng và khởi tạo giá trị luôn :

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

- Nếu phương thức là static thì gọi thông qua tên lớp

**<tên\_lớp>.<tên\_hàm\_tĩnh> (<Ds các tham số thực sự>)**

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

Kiểu của tham biến hình thức	Giá trị được truyền
Các kiểu nguyên thủy	Giá trị kiểu nguyên thủy
Kiểu lớp (class)	Giá trị tham chiếu
Kiểu mảng (array)	Giá trị tham chiếu

Ví dụ1: **Tập 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: **Tập 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 có đối tượng,

Phương thức tĩnh: không sử dụng được thuộc tính thông thường (non-static), không gọi được các phương thức thông thường.

\* **Điều khiển việc truy cập đến các thành viên của một 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.

Ví dụ

```

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 sẽ 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;// legal
    a.packageMethod();// legal
}
}

```

### 3.2.4 Truyền tham số và gọi phương thức

#### Truyền các giá trị kiểu nguyên thủy

- Bởi vì các biến hình thức là cục bộ trong định nghĩa của một hàm nên mọi thay đổi của biến hình thức không ảnh hưởng đến các tham biến hiện thời.

- Các tham biến có thể là các biểu thức và chúng phải được tính trước khi truyền vào lời gọi hàm.

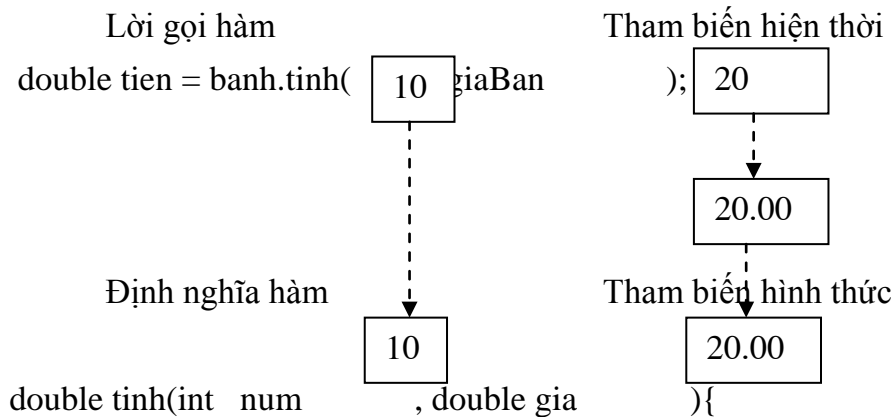
Ví dụ *Truyền các giá trị nguyên thủy*

```

class KháchHang1 { // 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    }
    }
}

```

- Cơ chế truyền tham biến đối với các giá trị nguyên thủy có thể minh họa như sau:



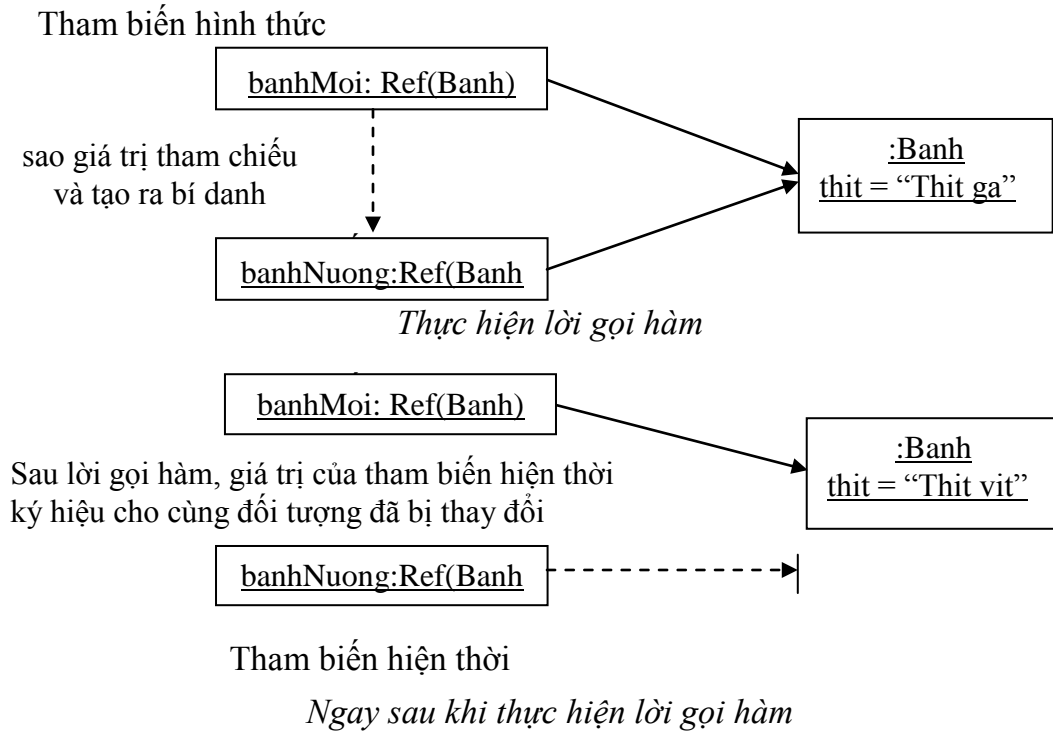
Truyền giá trị: đối với dữ liệu kiểu nguyên thủy thì giá trị của tham số (RValue) được copy lên stack, có thể truyền hằng số (vd: 10, 0.5, ...)

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

- Khi biến hiện thời tham chiếu tới đối tượng, thì giá trị tham chiếu của đối tượng sẽ được truyền cho biến hình thức.

Ví dụ *Truyền theo giá trị 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 vịt
        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
    }
}
```

### **3.3.Toán tử tạo lập đối tượng**

#### **3.3.1 Khai báo toán tử tạo lập.**

Toán tử tạo lập đối tượng được sử dụng gọi tự động một 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

\* Khai báo:

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

- Phạm\_vì có thể là: public, protected, private

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

- Danh sách các tham số : tùy 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 cấp toán tử tạo lập mặc định không tường minh.

Toán tử tạo lập mặc định tường minh

TênLớp() {

```
    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.

Toán tử tạo lập không mặc định:

Là toán tử tạo lập khi sử dụng toán tử new để tạo một đối tượng thuộc lớp thì giá trị của các biến thành phần phụ thuộc vào các tham số truyền vào.

Ví dụ: Minh họa 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!");}  
}
```

**3.3.2 Toán tử *this()* và *super()***

<p>■ Toán tử tạo lập <i>this()</i></p> <p>Toán tử tạo lập này được sử dụng để tạo ra đối tượng của lớp hiện thời</p>	<p>■ Toán tử tạo lập <i>super()</i></p> <p>Được sử dụng trong các toán tử tạo lập của lớp con để gọi tới các toán tử tạo lập thuộc lớp cha trực tiếp</p>
<pre> 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");     } } </pre>	<pre> Vdclass 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");     } } </pre>

<pre> 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                         // ...     } } </pre>	<pre> 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);    } } </pre>
--	---

### 3.3.3 Khởi khởi đầu tĩnh

Khởi khởi đầu tĩnh là một khối lệnh bên ngoài tất cả các phương thức, kể cả hàm tạo, trước khối lệnh này ta đặt từ khoá static, từ khoá này báo cho java biết đây là khối khởi đầu tĩnh, khối này chỉ được gọi 1 lần khi đối tượng đầu tiên của lớp này được tạo ra, khối khởi đầu tĩnh này cũng được java gọi tự động trước bất cứ hàm tạo nào, thông thường ta sử dụng khối khởi đầu tĩnh để khởi đầu các thuộc tính tĩnh ( static ), sau đây là một ví dụ có 1 khối khởi đầu tĩnh và một khối vô danh, để bạn thấy được sự khác nhau giữa khối khởi đầu tĩnh và khối vô danh

```

public class Untitled1 {
    public Untitled1 () {

```

```

System.out.println ( "Đây là hàm tạo" ); }
static { // đây là khối khởi đầu tĩnh
System.out.println ( "Đây là khối khởi đầu
tĩnh");
System.out.println("Khối này chỉ được gọi 1 lần khi thể hiện đầu tiên của
lớp được tạo ra"); }
{//đây là khối vô danh
System.out.println ( "Đây là khối vô danh ");
}
public static void main ( String[] args ) {
Untitled1 dt1 = new Untitled1 (); // tạo ra thể hiện thứ nhất của lớp
Untitled1 dt2 = new Untitled1 (); // tạo tiếp thể hiện thứ 2 của lớp }
}

```

khi cho chạy chương trình ta sẽ được kết quả ra như sau:

*Đây là khối khởi đầu tĩnh Khối này chỉ được gọi 1 lần khi thể hiện đầu tiên của lớp được tạo ra Đây là khối vô danh*

*Đây là hàm tạo Đây là khối vô danh Đây là hàm tạo*

Nhìn vào kết quả ra ta thấy khối khởi đầu tĩnh chỉ được java gọi thực hiện 1 lần khi đối tượng đầu tiên của lớp này được tạo, còn khối vô danh được gọi mỗi khi một đối tượng mới được tạo ra

### 3.3.4. Dọn dẹp: kết thúc và thu rác

#### **\* Phương thức finalize**

Java không có phương thức hủy bỏ. Phương thức finalize tương tự như phương thức hủy bỏ của C++, tuy nhiên nó không phải là phương thức hủy bỏ. Sở dĩ nó không phải là phương thức hủy bỏ vì khi đối tượng được hủy bỏ thì phương thức này chưa chắc đã được gọi đến. Phương thức này được gọi đến chỉ khi bộ thu rác của Java được khởi động và lúc đó đối tượng không còn được sử dụng nữa. Do vậy phương thức finalize có thể không được gọi đến.

#### **\* Cơ chế gom rác của java**

Trong java có một cơ chế thu rác tự động, nó đủ thông minh để biết đối tượng tượng nào không dùng nữa, rồi nó tự động thu hồi vùng nhớ dành cho đối tượng đó, java lại không có khái niệm hàm huỷ hay một cái gì đó tương tự.

### 3.4. Quan hệ kế thừa giữa các lớp

- *Sử dụng lại: Tồn tại nhiều loại đối tượng có các thuộc tính và hành vi tương tự hoặc liên quan đến nhau. Ví dụ; Person, Student, Manager,... Xuất hiện nhu cầu sử dụng lại các mã nguồn đã viết: Sử dụng lại thông qua copy (Tốn công, dễ nhầm, Khó sửa lỗi do tồn tại nhiều phiên bản), Sử dụng lại thông qua quan hệ has\_a (Sử dụng lớp*



cũ như là thành phần của lớp mới, Sử dụng lại cài đặt với giao diện mới, Phải viết lại giao diện, Chưa đủ mềm dẻo), Sử dụng lại thông qua cơ chế “kế thừa”(Thừa hưởng lại các thuộc tính và phương thức đã có Chi tiết hóa cho phù hợp với mục đích sử dụng mới, Thêm các thuộc tính mới, Thêm hoặc hiệu chỉnh các phương thức).

Từ một lớp cơ sở được xây dựng tốt, ta có thể dẫn xuất ra nhiều lớp mới, gọi là lớp con hay lớp kế thừa. Lớp cơ sở được gọi là lớp cha.

- Lớp con có các thuộc tính cho phép (public, protected) của lớp cha, và các đối tượng thuộc lớp con được có các hành vi cho phép (public, protected) như các đối tượng thuộc lớp cha, ngoài ra còn được bổ sung thêm các thuộc tính và các phương thức mới hoặc viết đè các hàm thành phần từ lớp cha.

```
class tên_lớp_con extends tên_lớp_cha{
    //Bổ sung các thuộc tính
    //Bổ sung các hàm thành phần
    //Ghi đè các hàm thành phần từ lớp cha
}
```

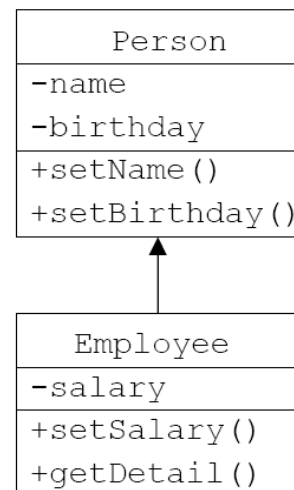
- Java chỉ hỗ trợ kế thừa đơn (tuyến tính), nghĩa là một lớp chỉ kế thừa được từ một lớp cha.

- Mọi lớp của Java đều là lớp con cháu mặc định của Object.

Ví dụ 1:

```
class Employee extends Person {
    private double salary;
    public boolean setSalary(double sal) {
        ...
        salary = sal;
        return true;
    }
}
```

```
Employee e = new Employee();
e.setName("John");
e.setSalary(3.0);
```



### 3.5 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 ít nhất một tham biến (nghĩa 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

```
class HìnhHoc{
    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')

### 3.6 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 (*throws clause* - sẽ trình bày ở chương 5).
- ✓ 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.

### 3.7 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.

```
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);
}
}

```

### 3.8 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 đề
        den1.docHoaDon(1000);                                // (10)
        den2.docHoaDon(1000);                                // (11)
        den3.docHoaDon(1000);                                // (12)
        // Truy nhập tới các biến thành phần đã bị viết đề (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!*

### 3.9 Lớp nội

Lớp nội hay lớp lồng (inner class):

Lớp lồng (lớp nội) chỉ có từ bộ JDK1.1 trở lên.

- Lớp lồng là các lớp được khai báo bên trong một lớp khác. Các lớp lồng thường được dùng như các lớp chuyển tiếp trong các giao diện. Có 2 loại lớp lồng:

+ Lớp lồng tĩnh (static): Phải truy tới các thành viên của lớp ngoài thông qua đối tượng (ít dùng)

+ Lớp lồng không tĩnh (static): truy cập được tới các thành viên của lớp ngoài không cần thông qua đối tượng (hay dùng)

### 3.10 Xử lý ngoại lệ trong java

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. 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 cấp bộ máy dùng để xử lý ngoại lệ. Có hai cách để xử lý ngoại lệ: Bắt ngoại lệ và ném ngoại lệ.

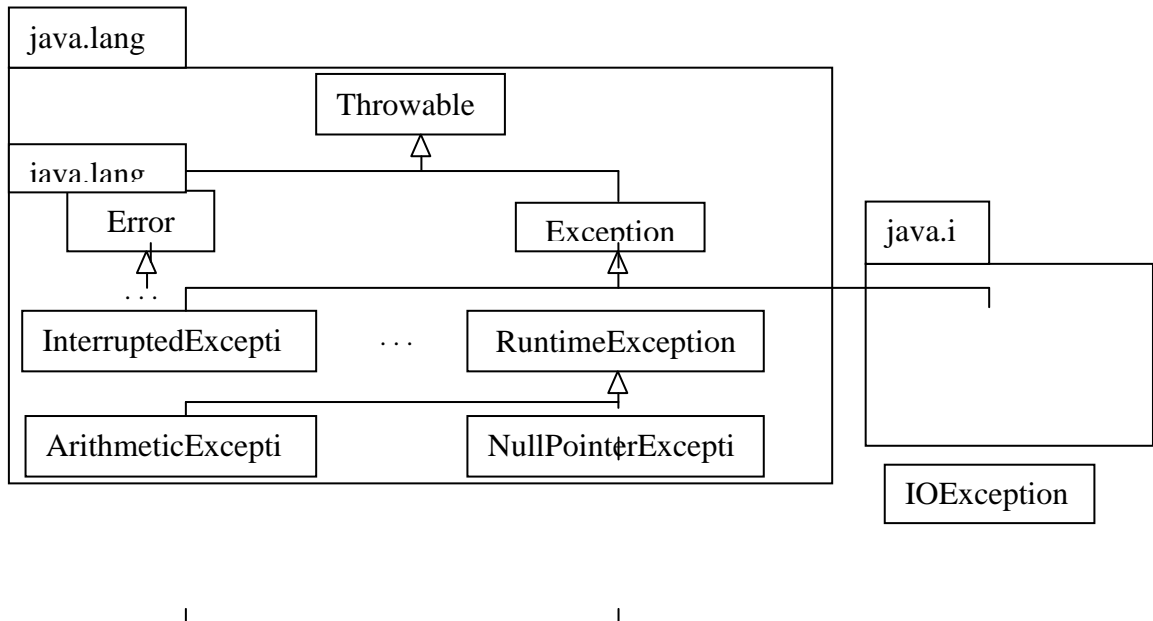
#### ***Ưu điểm của bắt ngoại lệ***

- 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

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

Ngoại lệ trong Java là các đối tượng. Tất cả các ngoại lệ đều được dẫn xuất từ lớp *Throwable*.



Ngoại lệ	Lớp cha của thứ tự phân cấp ngoại lệ
RuntimeException	Lớp cơ sở cho nhiều ngoại lệ java.lang
ArithmeticException	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
ArrayIndexOutOfBoundsException	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 đổi 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

Danh sách một số ngoại lệ

### 3.10.2 Câu lệnh *try*, *catch* và *finally*

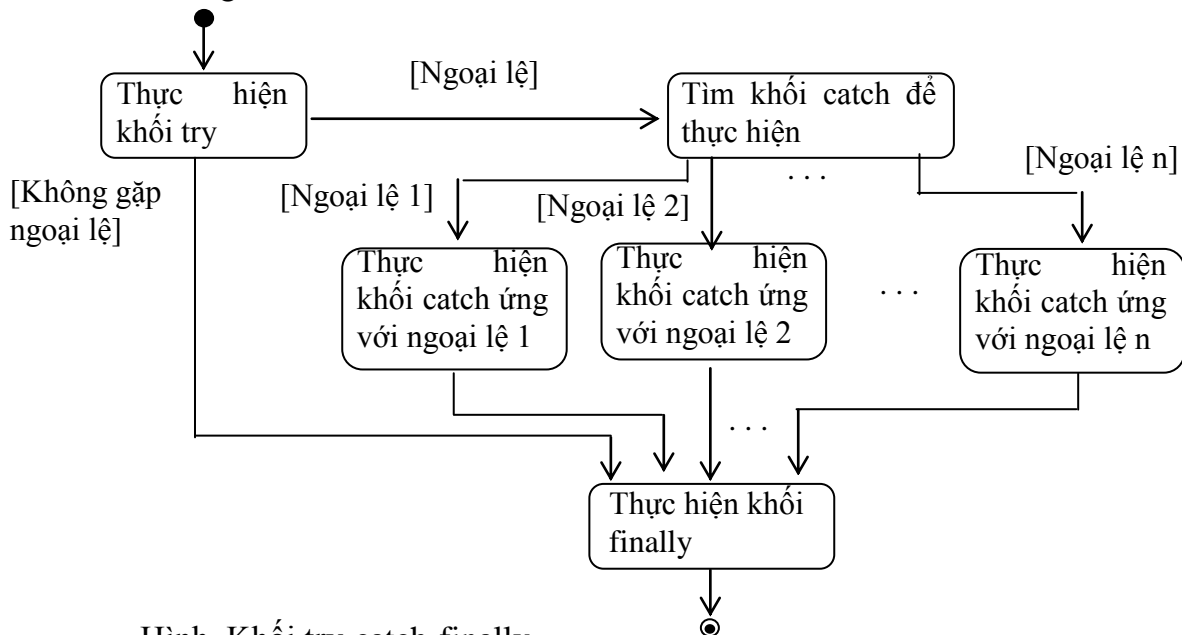
```

try { // Khởi try
    <Các câu lệnh>    }
catch (<Kiểu ngoại lệ 1> <Tham biến 1>) { // Khởi catch
    <Các câu lệnh xử lý khi xuất hiện kiểu ngoại lệ 1>    }
...
catch (<Kiểu ngoại lệ n> <Tham biến n>) { // Khởi catch
    <Các câu lệnh xử lý khi xuất hiện kiểu ngoại lệ n>
}
finally {    < Các câu lệnh phải thực hiện đến cùng> // Khởi finally }

```

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

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



Hình Khối try-catch-finally

**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. 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 tùy ý, không bắt buộc.

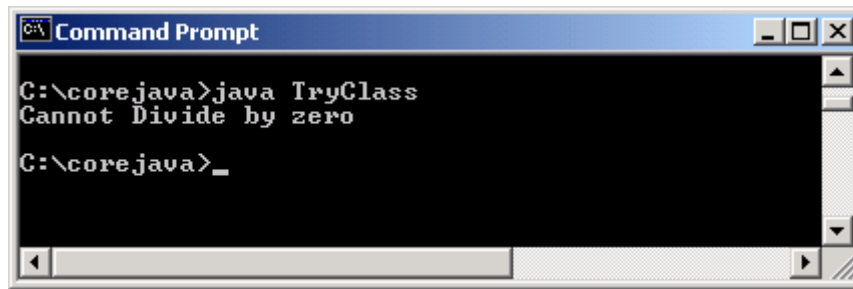
Khối *finally* có thể dùng để:

- 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.

VD 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 xuất của chương trình:

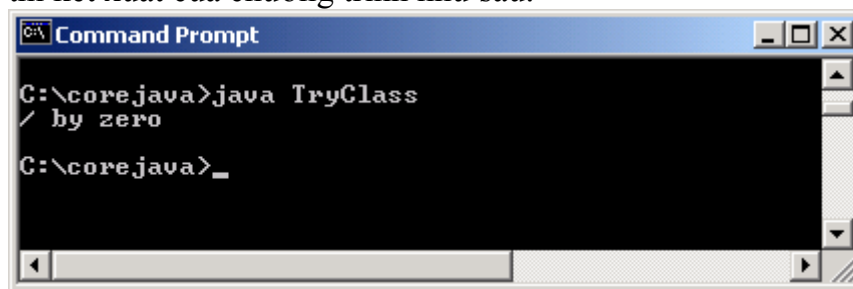


```

C:\corejava>java TryClass
Cannot Divide by zero
C:\corejava>_

```

‘a’ được sử dụng như một đối tượng của ArithmeticException 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 xuất của chương trình như sau:



```

C:\corejava>java TryClass
/ by zero
C:\corejava>_

```

### 3.10.3 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;}

```

### 3.10.4 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())

### 3.10.5 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
}
```

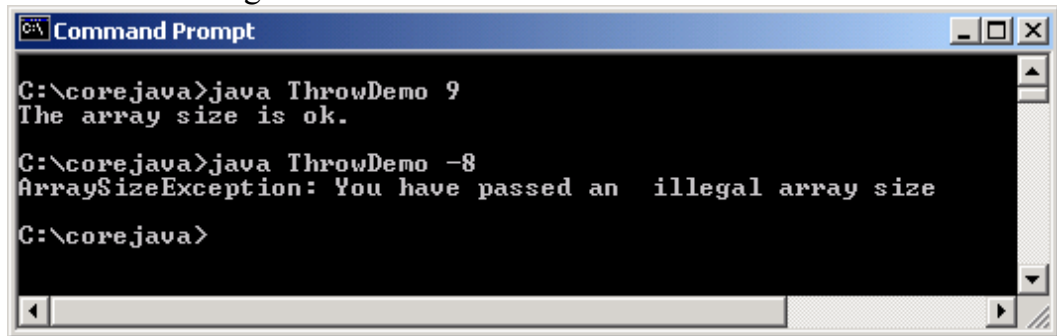
```

        {
            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 ‘try-catch’, 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



```

C:\corejava>java ThrowDemo 9
The array size is ok.

C:\corejava>java ThrowDemo -8
ArraySizeException: You have passed an illegal array size

C:\corejava>

```

### 3.10.6 Lăn vết ngoại lệ StackTrace

- Có thể sử dụng phương thức printStackTrace() để lăn 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();}
}
}
```

### 3.11 Giao diện và sự mở rộng quan hệ kế thừa trong Java

- 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 Interfaces
- Không được phép có những phương thức cụ thể (concrete methods)
- interface cần phải được hiện thực (implements).

#### 3.11.1 Các bước tạo interface

- Định nghĩa Interface
- Biên dịch Interface
- Hiện thực 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
- Nếu một lớp dẫn xuất từ một interface mà interface đó dẫn xuất từ các interface khác thì lớp đó phải định nghĩa tất cả các phương thức có trong các interface đó
- Khi định nghĩa một interface mới thì một kiểu dữ liệu tham chiếu cũng được tạo ra.

### 3.11.2 Khai báo giao diện

Cú pháp khai báo một giao tiếp như sau:

```
[public] interface <tên giao tiếp> [extends <danh sách giao tiếp>]  
{ }
```

- Tính chất: tính chất của một giao tiếp luôn là public. Nếu không khai báo tường minh thì giá trị mặc định cũng là public.
- Tên giao tiếp: tuân thủ theo quy tắc đặt tên biến của java.
- Danh sách các giao tiếp: danh sách các giao tiếp cha đã được định nghĩa để kế thừa, các
- giao tiếp cha được phân cách nhau bởi dấu phẩy. (Phần trong ngoặc vuông “[ ]” là tùy chọn).
  - Lưu ý: Một giao tiếp chỉ có thể kế thừa từ các giao tiếp khác mà không thể được kế thừa từ các lớp sẵn có.

#### Khai báo phương thức của giao tiếp

```
[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 giao tiếp luôn là public. Nếu không khai báo tường minh thì giá trị mặc định cũng là public. Đối với thuộc tính, thì chất chất luôn phải thêm là hằng (final) và tĩnh (static).
- 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 giao tiếp 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 giao tiếp đó.
- Các thuộc tính của giao tiếp 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 giao tiếp.



### 3.11.3 Sử dụng giao diện

Vì giao tiếp chỉ được khai báo dưới dạng các phương thức mẫu và các thuộc tính hằng nên việc sử dụng giao tiếp phải thông qua một lớp có cài đặt giao tiếp đó. Việc khai báo một lớp có cài đặt giao tiếp được thực hiện thông qua từ khoá implements như sau:

```
<tính chất> class <tên lớp> implements <các giao tiếp>
{
    ...
}
```

- Tính chất và tên lớp được sử dụng như trong khai báo lớp thông thường.
- Các giao tiếp: một lớp có thể cài đặt nhiều giao tiếp. Các giao tiếp được phân cách nhau bởi dấu phẩy. Khi đó, lớp phải cài đặt cụ thể tất cả các phương thức của tất cả các giao tiếp mà nó sử dụng.

Lưu ý:

- Một phương thức được khai báo trong giao tiếp phải được cài đặt cụ thể trong lớp có cài đặt giao tiếp nhưng không được phép khai báo chồng. Nghĩa là số lượng các tham số của phương thức trong giao tiếp phải được giữ nguyên khi cài đặt cụ thể trong lớp.

```
public class Shoe implements Product{
// Cài đặt phương thức được khai báo trong giao tiếp
    public float getCost(){
        return 10f;}

    // Phương thức truy nhập nhãn hiệu sản phẩm
    public String getMark(){
        return MARK;}

    // Phương thức main
    public static void main(String args[]){
        Shoe myShoe = new Shoe();
        System.out.println("This shoe is " + myShoe.getMark() +
            " having a cost of $" + myShoe.getCost());}
    }
```

Chương trình sẽ in ra dòng: "This shoe is Adidas having a cost of \$10". Hàm getMark() sẽ trả về nhãn hiệu của sản phẩm, là thuộc tính đã được khai báo trong giao tiếp. Hàm getCost() là cài đặt riêng của lớp Shoe đối với phương thức đã được khai báo trong giao tiếp Product mà nó sử dụng, cài đặt này trả về giá trị 10 đối với lớp Shoe.

### 3.12. Lớp trừu tượng

- Lớp trừu tượng là một dạng lớp đặc biệt, trong đó các phương thức chỉ được khai báo ở dạng khuôn mẫu (template) mà không được cài đặt chi tiết. Việc cài đặt chi tiết các phương thức chỉ được thực hiện ở các lớp con kế thừa lớp trừu tượng đó.
- Lớp trừu tượng được sử dụng khi muốn định nghĩa một lớp mà không thể biết và định nghĩa ngay được các thuộc tính và phương thức của nó.
- Lưu ý: Lớp trừu tượng cũng có thể kế thừa một lớp khác, nhưng lớp cha cũng phải là một lớp trừu tượng. (Khai báo kế thừa thông qua từ khoá `extends` như khai báo kế thừa thông thường).

#### 3.12.1 Khai báo phương thức của lớp trừu tượng

Tất cả các thuộc tính và phương thức của lớp trừu tượng đều phải khai báo là trừu tượng. Hơn nữa, các phương thức của lớp trừu tượng chỉ được khai báo ở dạng khuôn mẫu mà không có phần khai báo chi tiết. Cú pháp khai báo phương thức của lớp trừu tượng:

```
[public] abstract <kiểu dữ liệu trả về> <tên phương thức>
([<các tham số>]) [throws <các ngoại lệ>];
```

- Tính chất: tính chất của một thuộc tính hay phương thức của lớp trừu tượng luôn là `public`. Nếu không khai báo tường minh thì giá trị mặc định cũng là `public`.
- Kiểu dữ liệu 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:
  - Tính chất của phương thức trừu tượng không được là `private` hay `static`. Vì phương thức trừu tượng chỉ được khai báo chi tiết (nạp chồng) trong các lớp dẫn xuất (lớp kế thừa) của lớp trừu tượng. Do đó, nếu phương thức là `private` thì không thể nạp chồng, nếu phương thức là `static` thì không thể thay đổi trong lớp dẫn xuất.
- Phương thức trừu tượng chỉ được khai báo dưới dạng khuôn mẫu nên không có phần dấu móc “`{}`” mà kết thúc bằng dấu chấm phẩy “`;`”.

```
package vidu.chuong4;
abstract class Animal
{
    abstract String getName();
    abstract int getFeet();
}
```

### 3.12.2 Sử dụng lớp trừu tượng

- Lớp trừu tượng được sử dụng thông qua các lớp dẫn xuất của nó. Vì chỉ có các lớp dẫn xuất mới cài đặt cụ thể các phương thức được khai báo trong lớp trừu tượng.

```
package vidu.chuong4;
public class Bird extends Animal
{
    // Trả về tên loài chim
    public String getName()
    {
        return "Bird";
    }
    // Trả về số chân của loài chim
    public int getFeet()
    {
        return 2;
    }
}
```

- Chương trình khai báo lớp về loài mèo (Cat) cũng kế thừa từ lớp Animal trong chương trình. Lớp này cài đặt chi tiết hai phương thức đã được khai báo trong lớp Animal: phương thức getName() sẽ trả về tên loài là "Cat"; phương thức getFeet() trả về số chân của loài mèo là

```
public class Cat extends Animal
{
    // Trả về tên loài mèo
    public String getName()
    {
        return "Cat";
    }
    // Trả về số chân của loài mèo
    public int getFeet()
    {
        return 4;
    }
}
```

Chương trình này sẽ hiển thị hai dòng thông báo:

The Bird has 2 feet

The Cat has 4 feets

```
package vidu.chuong4;
public class AnimalDemo{
public static void main(String args[])
{Bird myBird = new Bird();
System.out.println("The " + myBird.getName() + " has " + myBird.getFeet()
+ " feets");
Cat myCat = new Cat();
System.out.println("The " + myCat.getName() + " has "+ myCat.getFeet() + "
feets");
```

## CHƯƠNG 4: MẢNG VÀ CÁC LỚP CƠ SỞ TRONG GÓI JAVA.LANG

### 4.1. Mảng trong Java

#### 4.1.1 Khai báo mảng

<Kiểu các phần tử>[] <Tên mảng>; hoặc  
<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

#### 4.1.2 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"`

#### 4.1.3 Truy cập các phần tử của mảng

*<Tên mảng>.length*

Phần tử thứ *i* của mảng là phần tử có chỉ số *i - 1* (*<Tên mảng>[i-1]*).

Truy cập vào phần tử của mảng: *<tên mảng>[i]*

#### 4.1.4 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.

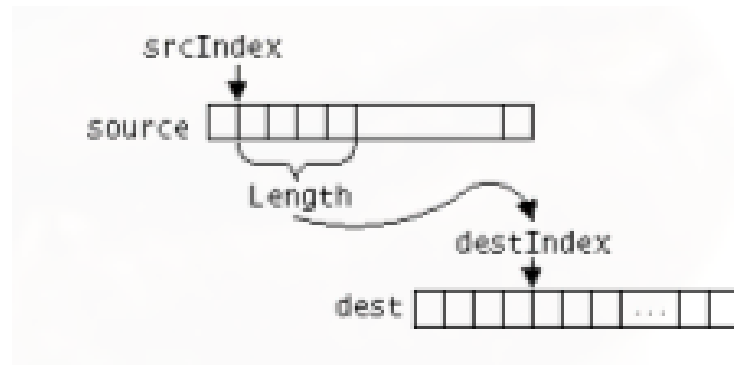
+ Chuyển từ kiểu mảng về *Object* (Mở rộng kiểu),

+ Chuyển từ *Object* sang kiểu mảng ( Thu hẹp kiểu).

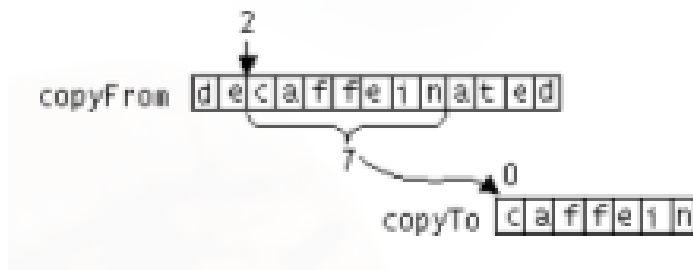
- Sao chép mảng :

Sử dụng phương thức `arrayCopy` của `System`

```
public static void arraycopy( ArrayType[] source, int srcIndex, ArrayType[] dest, int
destIndex, int length)
```



```
public class ArrayCopyDemo {
    public static void main(String[] args)
    {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```



## 4.2 Các lớp cơ sở trong gói java.lang

### 4.2.1 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 phân 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()`

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.

*Class getClass()*: Trả lại tên lớp của đối tượng hiện thời.

*boolean equals(Object obj)* : Cho lại kết quả true khi đối tượng hiện thời và obj là cùng

một đối tượng. Hàm này thường được viết đè ở các lớp con cho phù hợp với ngữ cảnh so sánh bằng nhau trong các lớp mở rộng đó.

*protected Object clone() throws CloneNotSupportedException*

Đối tượng mới được tạo ra có cùng các trạng thái như đối tượng hiện thời khi sử dụng clone(), nghĩa là tạo ra bản copy mới của đối tượng hiện thời. *String toString()* Nếu các lớp con không viết đè hàm này thì nó sẽ trả lại dạng biểu diễn văn bản (textual) của đối tượng. Hàm println() ở lớp PrintStream sẽ chuyển các đối số của nó sang dạng văn bản khi sử dụng hàm toString().

*protected void finalize() throws Throwable*: được gọi ngay trước khi đối tượng bị dọn vào “thùng rác”, nghĩa là trước khi đối tượng đó bị huỷ bỏ.

#### 4.2.2 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 định nghĩa hàm `typeValue()` tương ứng với các đối tượng nguyên thủy.

```
boolean b = boolObj.booleanValue();
```

```
int i = intObj.intValue();
```

```
float f = floatObj.floatValue();
```

```
double d = doubleObj.doubleValue(); // char c = charObj.charValue();
```

```
= charObj.toString(); // "a"
```

```
= boolObj.toString(); // "true"
```

```
= intObj.toString(); // "2002"
```

\* 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 nếu ch 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 chuỗi 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("Nhập vào " + n + " số nguyên");
        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();}
    }

```

### Lớp Void

Lớp này ký hiệu cho đối tượng của lớp Class biểu diễn cho giá trị void.

#### 4.2.3 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

		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 float hoặc double	Nhận về kiểu int nếu arg kiểu float, nhận về kiểu 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 $\arg1 \arg2$	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 số e của arg	arg là biểu thức kiểu double	double
random()	Nhận về một số giả ngẫu nhiên nằm trong khoảng [0, 1)	Không có tham số	double

#### 4.2.4 Lớp String

+ Lớp String dùng cho những chuỗi ký tự bất biến (chuỗi 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 chuỗi
String charStr = new String(characters); // charStr = "abba"
String buffStr = new String(buff);
String str = new String("abba");
```

+ Có thể sử dụng: String str = "Dung dung den toi";

**Ví dụ: Khởi tạo chuỗi trong Java (String Initialization Sample Code).**

Đây là ví dụ mô tả các phương pháp khác nhau để khởi tạo một chuỗi.

```
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','T','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 ( $0 \leq index \leq length()$ ). (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 thúc) 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 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 đầu 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 "xx"
        System.out.println("Thay And đầu tiên bằng xxx = " + str.replaceFirst("And", "xx
x"));
        //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());

    }
}

```

#### 4.2.5. Lớp *StringBuffer*

Lớp *StringBuffer* được sử dụng đối với những chuỗi ký tự động, có thể thay đổi nội dung

##### a) Tạo lập đối tượng *StringBuffer*

***StringBuffer(String str)*** Tạo ra đối tượng mới nội dung giống như chuỗi đối số *str* kích thước bằng kích thước của chuỗi *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 chuỗi *buffer* là 16.

##### b) Đọc và thay đổi các ký tự trong *StringBuffer*

***int length()*** Cho số ký tự trong chuỗi *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 chuỗi *buffer* thành *ch*. Hệ thống sẽ cho qua ngoại lệ *StringIndexOutOfBoundsException* nếu các chỉ số không tương thích.

##### c) Chuyển *StringBuffer* sang *String*

Lớp *StringBuffer* nạp chồng hàm *toString()* để chuyển chuỗi *buffer* về chuỗi của lớp cố định *String*.

**Hàm *append()*** bổ sung các ký tự vào cuối của chuỗi.

***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 đó 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 (soi gương).

***Kiểm soát cỡ của StringBuffer***

***int capacity()***: Cho lại kích thước (khả năng chứa) của xâu *buffer*.

***void ensureCapacity(int minCap):*** Đảm bảo đủ chỗ cho ít nhất *minCap* ký tự của chuỗi *buffer*.

***void setLength(int newLen):*** Đặt độ dài của chuỗi *buffer* là *newLen*.