



Core Java

Biên tập bởi:
Huỳnh Công Pháp

Core Java

Biên tập bởi:

Huỳnh Công Pháp

Các tác giả:

Huỳnh Công Pháp

Phiên bản trực tuyến:

<http://voer.edu.vn/c/95eeacbe>

MỤC LỤC

1. Lập trình hướng đối tượng
 2. Nhập môn Java
 3. Nền tảng của ngôn ngữ Java
 - 3.1. Cấu trúc một chương trình Java
 - 3.2. Chương trình Java đầu tiên
 - 3.3. Các lớp đối tượng trong Java
 - 3.4. Kiểu dữ liệu trong Java
 - 3.5. Các biến trong Java
 - 3.6. Phương thức trong một lớp (Method)
 - 3.7. Các toán tử
 - 3.8. Định dạng dữ liệu xuất và điều khiển luồng
 4. Các gói và giao diện
 5. AWT
 - 5.1. Giới thiệu về AWT
 - 5.2. Container (Vật chứa)
 - 5.3. Thành phần (Component)
 - 5.4. Thực đơn (Menu)
 - 5.5. Quản lý cách trình bày (Layout Manager)
 - 5.6. Xử lý các sự kiện
 6. Applets
 7. Xử lý ngoại lệ (Exception Handling)
 8. Đa luồng (Multithreading)
 9. Dòng vào / ra (I/O Streams)
 10. Cài đặt bảo mật
- Tham gia đóng góp

Lập trình hướng đối tượng

Giới thiệu về Lập trình hướng Đối tượng (Object-oriented Programming)

Lập trình hướng Đối tượng (OOP) là một phương pháp thiết kế và phát triển phần mềm. Những ngôn ngữ OOP không chỉ bao gồm cú pháp và một trình biên dịch (compiler) mà còn có một môi trường phát triển toàn diện. Môi trường này bao gồm một thư viện được thiết kế tốt, thuận lợi cho việc sử dụng các đối tượng.

Đối với một ngôn ngữ lập trình hỗ trợ OOP thì việc triển khai kỹ thuật lập trình hướng đối tượng sẽ dễ dàng hơn. Kỹ thuật lập trình hướng đối tượng cải tiến việc phát triển các hệ thống phần mềm. Kỹ thuật ấy đề cao nhân tố chức năng (functionality) và các mối quan hệ dữ liệu.

OOP là phương thức tư duy mới để giải quyết vấn đề bằng máy tính. Để đạt kết quả, lập trình viên phải nắm vấn đề thành một thực thể quen thuộc với máy tính. Cách tiếp cận hướng đối tượng cung cấp một giải pháp toàn vẹn để giải quyết vấn đề.

Hãy xem xét một tình huống cần được triển khai thành một hệ thống trên máy vi tính: việc mua bán xe hơi. Vấn đề vi tính hóa việc mua bán xe hơi bao gồm những gì?

Những yếu tố rõ ràng nhất liên quan đến việc mua bán xe hơi là:

1. Các kiểu xe hơi (model).
2. Nhân viên bán hàng.
3. Khách hàng.

Những hoạt động liên quan đến việc mua bán:

1. Nhân viên bán hàng đưa khách hàng tham quan phòng trưng bày.
2. Khách hàng chọn lựa một xe hơi.
3. Khách hàng đặt hóa đơn.
4. Khách hàng trả tiền.
5. Chiếc xe được trao cho khách hàng.

Mỗi vấn đề được chia ra thành nhiều yếu tố, được gọi là các Đối tượng (Objects) hoặc các Thực thể (Entities). Chẳng hạn như ở ví dụ trên, khách hàng, xe hơi và nhân viên bán hàng là những đối tượng hoặc thực thể.

Lập trình viên luôn luôn cố gắng tạo ra những kịch bản (scenarios) thật quen thuộc với những tình huống thực tiễn. Bước thứ nhất trong phương pháp này là làm cho máy tính liên kết với những đối tượng thế giới thực.

Tuy nhiên, máy tính chỉ là một cỗ máy. Nó chỉ thực hiện những công việc được lập trình mà thôi. Vì thế, trách nhiệm của lập trình viên là cung cấp cho máy tính những thông tin theo cách thức mà nó cũng nhận thức được cùng những thực thể như chúng ta nhận thức.

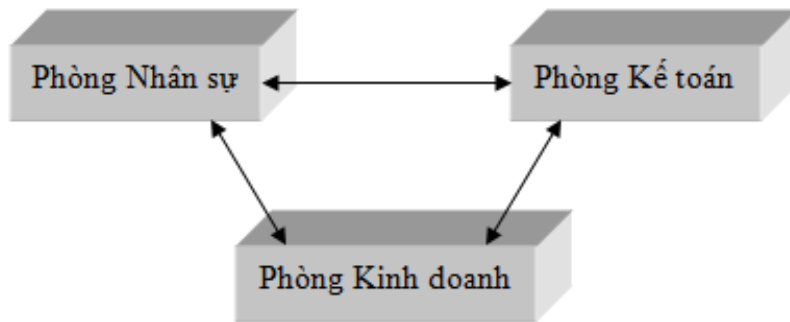
Đó chính là vai trò của kỹ thuật hướng đối tượng. Chúng ta sử dụng kỹ thuật hướng đối tượng để ánh xạ những thực thể chúng ta gặp phải trong đời sống thực thành những thực thể tương tự trong máy tính.

Phát triển phần mềm theo kỹ thuật lập trình hướng đối tượng có khả năng giảm thiểu sự lẫn lộn thường xảy ra giữa hệ thống và lĩnh vực ứng dụng.

Lập trình hướng đối tượng đề cập đến dữ liệu và thủ tục xử lý dữ liệu theo quan điểm là một đối tượng duy nhất. Lập trình hướng đối tượng xem xét dữ liệu như là một thực thể hay là một đơn vị độc lập, với bản chất riêng và những đặc tính của thực thể ấy. Bây giờ chúng ta hãy khảo sát những thuật ngữ ‘đối tượng’ (object), ‘dữ liệu’ (data) và ‘phương thức’ (method).

Có nhiều loại đối tượng khác nhau. Chúng ta có thể xem các bộ phận khác nhau trong một cơ quan là các đối tượng. Điển hình là một cơ quan có những bộ phận liên quan đến việc quản trị, đến việc kinh doanh, đến việc kế toán, đến việc tiếp thị ... Mỗi bộ phận có nhân sự riêng, các nhân sự được giao cho những trách nhiệm rõ ràng. Mỗi bộ phận cũng có những dữ liệu riêng chẳng hạn như thông tin cá nhân, bảng kiểm kê, những thể thức kinh doanh, hoặc bất kỳ dữ liệu nào liên quan đến chức năng của bộ phận đó. Rõ ràng là một cơ quan được chia thành nhiều bộ phận thì việc quản trị nhân sự và những hoạt động doanh nghiệp dễ dàng hơn. Nhân sự của cơ quan điều khiển và xử lý dữ liệu liên quan đến bộ phận của mình.

Chẳng hạn như bộ phận kế toán chịu trách nhiệm về lương bổng đối với cơ quan. Nếu một người ở đơn vị tiếp thị cần những chi tiết liên quan đến lương bổng của đơn vị mình, người ấy chỉ cần liên hệ với bộ phận kế toán. Một người có thẩm quyền trong bộ phận kế toán sẽ cung cấp thông tin cần biết, nếu như thông tin ấy có thể chia sẻ được. Một người không có thẩm quyền từ một bộ phận khác thì không thể truy cập dữ liệu, hoặc không thể thay đổi làm hư hỏng dữ liệu. Ví dụ này minh chứng rằng các đối tượng là hữu dụng trong việc phân cấp và tổ chức dữ liệu.



Minh họa cấu trúc của một cơ quan điển hình.

Khái niệm về đối tượng có thể được mở rộng đến hầu hết các lãnh vực đời sống, và hơn nữa, đến thế giới lập trình. Bất kỳ ứng dụng nào đều có thể được định nghĩa theo thuật ngữ thực thể hoặc đối tượng để tạo ra tiến trình xử lý mô phỏng theo tiến trình xử lý mà con người nghĩ ra.

Phương pháp giải quyết ‘top-down’ (từ trên xuống) cũng còn được gọi là ‘lập trình hướng cấu trúc’ (structured programming). Nó xác định những chức năng chính của một chương trình và những chức năng này được phân thành những đơn vị nhỏ hơn cho đến mức độ thấp nhất. Bằng kỹ thuật này, các chương trình được cấu trúc theo hệ thống phân cấp các module. Mỗi một module có một đầu vào riêng và một đầu ra riêng. Trong mỗi module, sự điều khiển có chiều hướng đi xuống theo cấu trúc chứ không có chiều hướng đi lên.

Phương pháp OOP cố gắng quản lý việc thừa kế phức tạp trong những vấn đề thực tế. Để làm được việc này, phương pháp OOP che giấu một vài thông tin bên trong các đối tượng. OOP tập trung trước hết trên dữ liệu. Rồi gắn kết các phương thức thao tác trên dữ liệu, việc này được xem như là phần thừa kế của việc định nghĩa dữ liệu. Bảng 1.1 minh họa sự khác biệt giữa hai phương pháp:

Một ví dụ về hai phương pháp giải quyết OOP và Structured

Phương pháp Top-Down	OOP
Chúng ta sẽ xây dựng một khách sạn.	Chúng ta sẽ xây dựng một tòa nhà 10 tầng với những dãy phòng trung bình, sang trọng, và một phòng họp lớn.
Chúng ta sẽ thiết kế các tầng lầu, các phòng và phòng họp.	Chúng ta sẽ xây dựng một khách sạn với những thành phần trên.

Trừu tượng hóa dữ liệu (Data Abstraction)

Khi một lập trình viên phải phát triển một chương trình ứng dụng thì không có nghĩa là người ấy lập tức viết mã cho ứng dụng ấy. Trước hết, người ấy phải nghiên cứu ứng dụng và xác định những thành phần tạo nên ứng dụng. Kế tiếp, người ấy phải xác định những thông tin cần thiết về mỗi thành phần.

Hãy khảo sát chương trình ứng dụng cho việc mua bán xe hơi nói trên. Chương trình phải xuất hóa đơn cho những xe hơi đã bán cho khách hàng. Để xuất một hóa đơn, chúng ta cần những thông tin chi tiết về khách hàng. Vậy bước thứ nhất là xác định những đặc tính của khách hàng.

Một vài đặc tính gắn kết với khách hàng là:

- Tên.
- Địa chỉ.
- Tuổi.
- Chiều cao.
- Màu tóc.

Từ danh sách kể trên, chúng ta xác định những đặc tính thiết yếu đối với ứng dụng. Bởi vì chúng ta đang đề cập đến những khách hàng mua xe, vì thế những chi tiết thiết yếu là:

- Tên.
- Địa chỉ.

Còn những chi tiết khác (chiều cao, màu tóc ...) là không quan trọng đối với ứng dụng. Tuy nhiên, nếu chúng ta phát triển một ứng dụng hỗ trợ cho việc điều tra tội phạm thì những thông tin chẳng hạn như màu tóc là thiết yếu.

Bên cạnh những chi tiết về khách hàng, những thông tin sau cũng cần thiết:

- Kiểu xe được bán.
- Nhân viên nào bán xe.

Bên cạnh những đặc tính của khách hàng, xe hơi và nhân viên bán hàng, chúng ta cũng cần liệt kê những hành động được thực hiện.

Công việc xuất hóa đơn đòi hỏi những hành động sau:

- Nhập tên của khách hàng.
- Nhập địa chỉ của khách hàng.
- Nhập kiểu xe.

- Nhập tên của nhân viên bán xe.
- Xuất hóa đơn với định dạng đòi hỏi.

Khung thông tin bên dưới cho thấy những thuộc tính và những hành động liên quan đến một hóa đơn:

Các thuộc tính
Tên của khách hàng
Địa chỉ của khách hàng
Kiểu xe bán
Nhân viên bán xe

Các hành động
Nhập tên
Nhập địa chỉ
Nhập kiểu xe
Nhập tên nhân viên bán xe
Xuất hóa đơn

Định nghĩa : Sự trừu tượng hóa dữ liệu là quá trình xác định và nhóm các thuộc tính và các hành động liên quan đến một thực thể cụ thể, xét trong mối tương quan với ứng dụng đang phát triển.

Tiếp theo, chúng ta muốn ứng dụng tính toán tiền hoa hồng cho nhân viên bán hàng.

Những thuộc tính liên kết với nhân viên bán hàng có tương quan với ứng dụng này là:

- Tên.
- Số lượng xe bán được.
- Tiền hoa hồng.

Những hành động đòi buộc đối với công việc này là:

- Nhập tên nhân viên bán xe.
- Nhập số lượng xe bán được.
- Tính tiền hoa hồng kiếm được.

Những thuộc tính
Tên
Số lượng xe bán được
Tiền hoa hồng

Những hành động
Nhập tên
Nhập số lượng xe bán được
Tính tiền hoa hồng

Như thế, việc trừu tượng hóa dữ liệu tra đặt ra câu hỏi ‘Đâu là những thuộc tính và những hành động cần thiết cho một vấn đề đặt ra?’

Những ưu điểm của việc Trừu tượng hóa

Những ưu điểm của việc Trừu tượng hóa là:

- Tập trung vào vấn đề.
- Xác định những đặc tính thiết yếu và những hành động cần thiết.
- Giảm thiểu những chi tiết không cần thiết.

Việc trừu tượng hóa dữ liệu là cần thiết, bởi vì không thể mô phỏng tất cả các hành động và các thuộc tính của một thực thể. Vấn đề mấu chốt là tập trung đến những hành vi cốt yếu và áp dụng chúng trong ứng dụng.

Chẳng hạn như khách hàng hoặc nhân viên bán hàng cũng có thể thực hiện những hành động sau:

- Người ấy đi lại.
- Người ấy nói chuyện.

Tuy nhiên, những hành động này không liên quan đến ứng dụng. Việc trừu tượng hóa dữ liệu sẽ loại bỏ chúng.

Lớp (Class)

Trong ứng dụng mua bán xe, chúng ta đã xác định các thuộc tính và các hành động cần có để xuất một hóa đơn cho một khách hàng.

Các hành động và các thuộc tính này là chung cho mọi khách hàng mua xe. Ví thể, chúng có thể được nhóm lại trong một thực thể đơn nhất gọi là một ‘lớp’.

Hãy khảo sát lớp có tên là ‘khách hàng’ dưới đây. Lớp này bao gồm mọi thuộc tính và hành động đòi hỏi đối với một khách hàng.

Lớp Khách hàng
Tên khách hàng
Địa chỉ khách hàng
Kiểu xe được bán
Nhân viên bán xe
Nhập tên
Nhập địa chỉ
Nhập kiểu xe được bán
Nhập tên nhân viên bán xe
Xuất hóa đơn

Định nghĩa : Một **lớp** định nghĩa một thực thể theo những thuộc tính và những hành động chung. *Hoặc* Những thuộc tính và những hành động chung của một thực thể được nhóm lại để tạo nên một đơn vị duy nhất gọi là một **lớp**. *Hoặc* Một **lớp** là một sự xác định cấp chung loại của các thực thể giống nhau.

Một lớp là một mô hình khái niệm về một thực thể. Nó mang tính cách tổng quát chứ không mang tính cách đặc thù.

Khi định nghĩa một lớp, chúng ta muốn phát biểu rằng một lớp sẽ phải có một tập hợp các thuộc tính và các hành động riêng. Chẳng hạn như một định nghĩa lớp dưới đây:

Lớp Con người
Tên
Chiều cao
Màu tóc
Viết
Nói

Lớp này định nghĩa thực thể ‘Con người’. Mọi thực thể thuộc kiểu ‘Con người’ sẽ đều có những đặc tính và những hành động như đã được định nghĩa.

Một khi một lớp đã được định nghĩa, chúng ta biết được những thuộc tính và những hành động của những thực thể ‘trông giống’ như lớp này. Vì thế, tự bản chất một lớp là một nguyên mẫu (prototype).

Một ví dụ khác về một lớp liên quan đến việc mua bán xe hơi như sau:

Lớp Nhân viên bán hàng
Tên
Số lượng xe bán được
Tiền hoa hồng
Nhập tên
Nhập số lượng xe bán được
Tính tiền hoa hồng

Lớp trên định nghĩa các thuộc tính và các hành động đặc trưng cho mọi nhân viên bán xe hơi.

Đối tượng (Object)

Một lớp là một nguyên mẫu phác họa những thuộc tính và những hành động có thể của một thực thể. Để có thể sử dụng thực thể mà lớp định nghĩa, chúng ta phải tạo một ‘đối tượng’ từ lớp đó.

Lớp là một khái niệm, còn đối tượng là một thể hiện được định nghĩa bởi lớp.

Hãy khảo sát lớp ‘Khách hàng’ được định nghĩa trên. Lớp này định nghĩa mọi thuộc tính và hành động gắn liền với một khách hàng.

Khi một người mua một xe hơi ở một cửa hàng, cửa hàng ấy có một khách hàng mới. Vào thời điểm ấy, một đối tượng giống như lớp ‘Khách hàng’ được tạo ra. Đối tượng này sẽ phải có những giá trị thực đối với các thuộc tính ‘Tên’, ‘Địa chỉ’, ‘Kiểu xe’ ...

Chẳng hạn như một khách hàng có tên là ‘Mark’, sống ở ‘London’ đã mua một xe kiểu ‘Honda Civic’ từ nhân viên bán hàng tên là ‘Tom’. Như thế, ‘Mark’ là một đối tượng của kiểu ‘Khách hàng’.

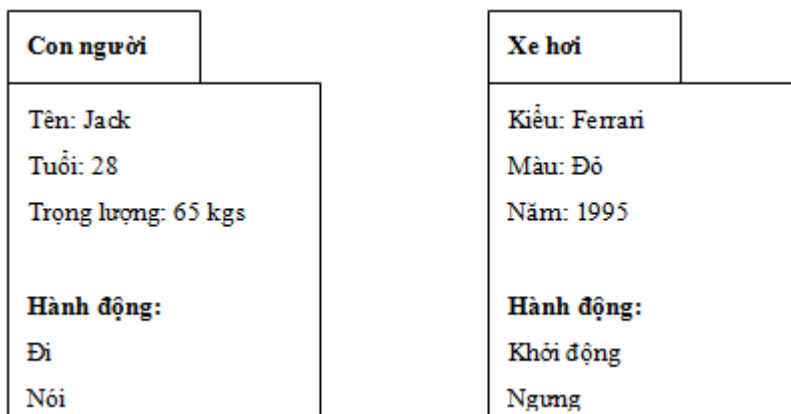
Định nghĩa : Một **đối tượng** là một trường hợp của một lớp.

Một đối tượng là một thực thể cụ thể (thông thường bạn có thể sờ chạm, xem thấy và cảm nhận).

Kể từ lúc một đối tượng hiện hữu, những thuộc tính của nó là những giá trị xác định, và những hành động được định nghĩa cho đối tượng này được thực thi.

Trong mỗi một đối tượng, các khía cạnh sau đây được xác định rõ:

- Tình trạng (state).
- Thái độ (behavior).
- Chân tính (identity).



Trình bày hai đối tượng.

Mỗi đối tượng có những đặc tính riêng mô tả đối tượng ấy là gì, hoặc hành động ra sao.

Chẳng hạn như những thuộc tính của một đối tượng ‘Con người’ sẽ là:

- Tên.
- Tuổi.
- Trọng lượng.

Những thuộc tính của một đối tượng ‘Xe hơi’ sẽ là:

- Màu sắc.
- Kiểu xe.
- Năm.

Một đối tượng cũng thực hiện một số hành động. Một xe hơi có khả năng thực hiện những hành động sau:

- Khởi động.
- Ngưng.
- Chuyển động.

Để chuyển đổi giữa các đối tượng lập trình và các đối tượng đời thực, cần phải kết hợp các thuộc tính và các hành động của một đối tượng.

Thuộc tính

Chúng ta xác định các thuộc tính và các hành động để định nghĩa một lớp.

Một khi các thuộc tính được gán cho các giá trị, chúng mô tả một đối tượng. Hãy khảo sát lớp sau:

Các thuộc tính của lớp Khách hàng
Tên của khách hàng
Địa chỉ của khách hàng
Kiểu xe được bán
Nhân viên đã bán xe

Khi thuộc tính ‘Tên’ được gán cho giá trị ‘Mark’ thì nó mô tả một đối tượng xác định được tạo từ lớp ‘Khách hàng’.

Định nghĩa : Một **thuộc tính** là một đặc tính mô tả một đối tượng.

Như thế, các thuộc tính nắm giữ các giá trị dữ liệu trong một đối tượng, chúng định nghĩa một đối tượng cụ thể.

Bởi vì một lớp là một nguyên mẫu cho nên các thuộc tính trong một lớp không thể nắm giữ các giá trị. Một thuộc tính có thể được gán một giá trị chỉ sau khi một đối tượng dựa trên lớp ấy được tạo ra.

Để có thể lưu giữ những chi tiết của một khách hàng, một thể hiện(đối tượng) của lớp ‘Khách hàng’ phải được tạo ra. Các thuộc tính của một đối tượng hiện hữu chỉ khi đối tượng ấy được tạo ra.

Mọi đối tượng của một lớp phải có cùng các thuộc tính.

Khảo sát ví dụ sau:

Các thuộc tính của lớp Con người		Đối tượng được tạo từ lớp Con người
Tên	=	Mark
Chiều cao	=	6 ft. 1 in.

Hoạt động (Operation)

Các hành động khả thi, như được định nghĩa trong một lớp, được gọi là ‘các hoạt động’.

Định nghĩa : Một **hoạt động** là một dịch vụ được yêu cầu của một đối tượng.

Các hoạt động xác định các hành động cần phải thực hiện của một đối tượng được tạo ra từ một lớp. Chẳng hạn như chúng ta không thể yêu cầu một hoạt động ‘Mua một xe hơi khác’ của một đối tượng được tạo ra từ lớp ‘Khách hàng’.

Một lớp chỉ là một nguyên mẫu. Vì thế, trong một lớp một hoạt động chỉ được định nghĩa. Còn việc áp dụng hoạt động ấy chỉ xảy ra nơi các đối tượng riêng rẽ. Chẳng hạn như hoạt động ‘Nhập Tên’ mà lớp “Khách hàng” định nghĩa có thể được thực hiện nơi một đối tượng nào đó.

Tập hợp các hoạt động được yêu cầu cho tất cả các đối tượng trong cùng một lớp là như nhau.

Phương thức (Method)

Các hoạt động định nghĩa các hành động khả thi có thể được yêu cầu của một đối tượng. Một phương thức là sự thực thi thực tế của một hoạt động.

Định nghĩa : Phương thức là sự xác định về cách thức thực thi một hoạt động được yêu cầu.

Các phương thức xác định cách thức thao tác trên các dữ liệu của một đối tượng. Bởi vì phương thức là sự thực thi thực tế một hoạt động, cho nên nó có thể được áp dụng cho một đối tượng. Một phương thức là một thuật toán, từng bước từng bước (step-by-step) xác định điều gì được thực hiện khi hoạt động ấy được yêu cầu.

Hãy khảo sát những hoạt động chung của một thực thể thuộc loại ‘Con người’: Đi, Nói. Chỉ khi một đối tượng cụ thể của loại ‘Con người’ được tạo ra thì các hành động ‘Đi’, ‘Nói’ mới được thực thi.

Thông điệp (Message)

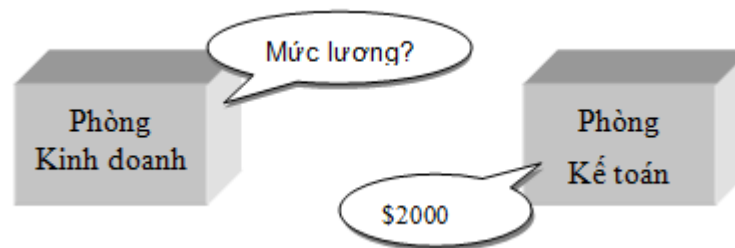
Để yêu cầu một hoạt động cụ thể nào đó được thực hiện, một thông điệp được gửi tới đối tượng nơi hoạt động này được định nghĩa.

Định nghĩa : Một **thông điệp** là một lời yêu cầu một hoạt động.

Khi một đối tượng nhận được một thông điệp, nó thực hiện một phương thức tương ứng.

Chẳng hạn, một đối tượng được tạo từ lớp ‘Khách hàng’ để nhập tên của người sử dụng. Khi đối tượng nhận được thông điệp, nó tìm và thực thi phương thức ‘Nhập tên’.

Trong trường hợp một công ty, mỗi bộ phận được coi là một đối tượng. Những thông tin được chuyển tới và được đón nhận từ mỗi bộ phận (hoặc qua thông báo liên bộ phận, hoặc qua những chỉ thị miệng) tạo nên những thông điệp giữa các đối tượng. Những chỉ thị này có thể được chuyển dịch thành những lời gọi hàm trong một chương trình.



Các đối tượng gửi thông điệp cho nhau

Trong hình 3, ‘Kinh doanh’ và ‘Kế toán’ là hai bộ phận khác nhau trong một công ty. Hai bộ phận này được coi là hai đối tượng khác nhau. Thông tin được truyền đi và được đón nhận giữa các bộ phận tạo nên các thông điệp giữa các đối tượng.

Sự kiện (Event)

Một sự kiện là một sự việc xảy ra cho một đối tượng tại một thời điểm. Để đáp ứng lại sự kiện ấy, đối tượng sẽ thực hiện một hoặc nhiều phương thức.

Nói cách khác, một sự kiện là một tác nhân mà đối tượng này gây ra cho một đối tượng khác. Chẳng hạn như click chuột trái trên một nút.

Để hiểu rõ hơn các sự kiện, hãy khảo sát ví dụ sau từ thực tế:

‘Một người sẽ thét lên khi bị thọc bằng một vật nhọn’.

‘Thọc’ là sự kiện gây ra sự phản ứng là ‘thét lên’.

Trong máy tính, một người sử dụng nhấn một nút trên bàn phím là một sự kiện chung. Sự phản hồi đối với sự kiện này là việc hiển thị ký tự tương ứng trên màn hình.

Lớp và Đối tượng

Có một sự khác biệt thực sự giữa một lớp và một đối tượng. Cần nhận thức rõ sự khác biệt này.

Một lớp định nghĩa một thực thể, trong khi đó một đối tượng là một trường hợp của thực thể ấy.

Đối tượng là một mô hình thực, trong khi lớp là một mô hình khái niệm - định nghĩa tất cả các thuộc tính và các phương thức cần thiết của một đối tượng.

Tất cả các đối tượng thuộc về cùng một lớp có cùng các thuộc tính và các phương thức.

Một lớp là một nguyên mẫu của một đối tượng. Nó xác định các hành động khả thi và các thuộc tính cần thiết cho một nhóm các đối tượng cụ thể.

Thiết lập (Construction) và Hủy (Destruction)

Thiết lập

Một lớp chỉ cung cấp những định nghĩa về các thuộc tính và các phương thức khả thi. Các thuộc tính và các phương thức có thể được truy cập chỉ khi một đối tượng dựa trên một lớp được tạo ra.

Khi một đối tượng mới được tạo, các thuộc tính của nó trở nên hiện thực và có thể được gán giá trị. Tương tự, các phương thức đã được định nghĩa cũng được áp dụng.

Định nghĩa : Thiết lập là một tiến trình hiện thực hóa một đối tượng.

Hàm **thiết lập** là một phương thức đặc biệt phải được gọi trước khi sử dụng bất kỳ phương thức nào trong một lớp. Hàm Thiết lập khởi tạo các thuộc tính, và cấp phát bộ nhớ nếu cần.

Mỗi một lớp có một hàm thiết lập.

Khảo sát lại trường hợp cửa hàng bán xe hơi. Ngay từ lúc đầu chỉ định nghĩa các lớp. Cho đến khi một khách hàng mua một xe hơi tại cửa hàng thì một đối tượng mới giống như lớp ‘Khách hàng’ mới được tạo.

Khi đối tượng này được tạo, một số khoảng trống bộ nhớ được cấp phát cho những thuộc tính của nó để lưu trữ các giá trị được gán cho các thuộc tính ấy (‘Tên’, ‘Địa chỉ’ ...). Hàm thiết lập thực hiện việc cấp phát này. Vào lúc này, mọi thuộc tính và phương thức của đối tượng sẵn sàng để sử dụng.

Tương tự như trường hợp một học sinh nhập học tại một trường học. Khi một học sinh nhập học, một vài hành động được thực hiện để nhận học sinh ấy vào trường. Đó là:

- Xếp lớp cho học sinh ấy.
- Ghi tên học sinh ấy vào danh sách.

- Xếp chỗ ngồi.

Đây là những hành động đồng loạt được thực hiện ngay lúc bắt đầu nhập học. Chúng tương tự với những hành động mà hàm thiết lập của một đối tượng thực hiện.

Hủy

Khi một đối tượng không còn cần thiết nữa thì nó sẽ bị hủy bỏ.

Sẽ lãng phí tài nguyên, chẳng hạn như bộ nhớ, nếu như tiếp tục để cho một đối tượng tồn tại một khi nó không còn cần thiết.

Định nghĩa : Hàm **Hủy** là một phương thức đặc biệt được dùng để hủy bỏ một đối tượng.

Tiến trình **Hủy** tiêu hủy một đối tượng và giải phóng khoảng trống bộ nhớ mà hàm thiết lập đã cấp phát cho nó. Hàm **Hủy** cũng triệt tiêu khả năng truy cập đến đối tượng ấy.

Một khi một đối tượng bị hủy thì các thuộc tính của nó không thể được truy cập, cũng như không một phương thức nào có thể được thực thi.

Chẳng hạn, trong trường hợp bán xe hơi, một khi nhân viên bán hàng bỏ nghề, những chi tiết của người ấy không còn liên hệ. Vì thế, đối tượng tương ứng sẽ bị hủy. Điều này giải phóng bộ nhớ đã cấp phát cho nhân viên bán hàng ấy. Khoảng trống này giờ đây có thể được tái sử dụng.

Hãy xem xét ví dụ về trường học trên đây. Khi một học sinh thôi học, tên của học sinh ấy bị loại ra khỏi danh sách, và khoảng trống được giải phóng có thể được tái cấp phát.

Các hành động đồng loạt này tương tự với công việc của hàm hủy đối với một đối tượng.

Tính Bền vững (Persistence)

Hãy khảo sát trường hợp bán xe hơi. Những chi tiết của khách hàng được lưu trữ ngay khi xe hơi đã được phân phối. Việc duy trì dữ liệu vẫn cần thiết cho đến khi dữ liệu được chỉnh sửa hoặc hủy bỏ chính thức.

Định nghĩa : **Tính Bền vững** là khả năng lưu trữ dữ liệu của một đối tượng ngay cả khi đối tượng ấy không còn tồn tại.

Cửa hàng bán xe lưu trữ chi tiết khách hàng vào một file. Những chi tiết này sẽ tồn tại trong file cho đến khi chúng bị hủy, hoặc bản thân file bị hủy.

Chúng ta đừng chạm tính bền vững mỗi ngày. Hãy xem việc sáng tác một bài thơ. Bài thơ là dữ liệu tồn tại trong tâm trí của nhà thơ. Bao lâu nhà thơ còn tồn tại thì bấy lâu bài thơ còn tồn tại. Nếu bài thơ muốn tồn tại ngay cả sau khi nhà thơ qua đời thì nó phải được viết ra giấy.

Bài thơ được viết ra giấy tạo nên sự bền vững. Bài thơ sẽ tồn tại bao lâu văn bản ấy còn được duy trì. Bài thơ ấy không còn tồn tại khi tờ giấy ấy bị xé rách, hoặc chữ nghĩa bị xóa đi.

Tính Đóng gói dữ liệu

Tiến trình trừu tượng hóa dữ liệu hỗ trợ cho việc xác định những thuộc tính và những phương thức thiết yếu.

Thông thường, các đối tượng sử dụng những thuộc tính và những phương thức không được yêu cầu bởi người sử dụng đối tượng.

Chẳng hạn như trong trường hợp lớp ‘Khách hàng’. Lớp ấy có một phương thức xuất hóa đơn. Giả sử rằng khi hóa đơn được xuất, một trong những chi tiết được in ra trên hóa đơn là ngày phân phối. Tuy nhiên chúng ta không biết thuộc tính nào qua đó chúng ta có thể xác định thông tin này.

Ngày phân phối được phát sinh bên trong đối tượng, và được hiển thị trên hóa đơn. Như thế người sử dụng không nhận thức về cách thức mà ngày phân phối được hiển thị.

Ngày phân phối có thể được xử lý theo một trong những cách sau:

- Đó là một giá trị được tính toán - Chẳng hạn, 15 ngày kể từ ngày đặt hàng.
- Đó là một giá trị cố định – Xe hơi được phân phối vào ngày mùng 2 mỗi tháng.

Đối tượng sử dụng những thuộc tính và những phương thức mang tính nội bộ. Bởi vì những thuộc tính và những phương thức có thể được che khuất khỏi tầm nhìn. Các đối tượng khác và những người sử dụng không nhận thức được các thuộc tính và / hoặc các phương thức như thế có tồn tại hay không. Tiến trình che giấu các thuộc tính, các phương thức, hoặc các chi tiết của việc thi hành được gọi là ‘đóng gói’ (encapsulation).

Định nghĩa : Đóng gói là tiến trình che giấu việc thực thi những chi tiết của một đối tượng đối với người sử dụng đối tượng ấy.

Việc đóng gói phân tích những khía cạnh có thể truy cập từ bên ngoài với những khía cạnh chỉ được sử dụng trong nội bộ của đối tượng.

Ưu điểm của việc đóng gói là có thể tạo ra bất kỳ thuộc tính hay phương thức cần thiết để đáp ứng đòi hỏi công việc khi xây dựng một lớp. Mặt khác, chỉ những thuộc tính và / hoặc những phương thức có thể được truy cập từ bên ngoài lớp thì mới nhìn thấy.

Một ví dụ khác về việc đóng gói là lớp ‘Nhân viên bán hàng’ đã được định nghĩa ở trên. Khi phương thức tính tiền hoa hồng được thực thi, người sử dụng không biết chi tiết của việc tính toán. Tất cả những gì họ biết chỉ là tổng số tiền hoa hồng mà họ phải trả cho nhân viên bán hàng.

Một trường hợp về đóng gói mà chúng ta gặp trong đời sống hằng ngày là việc giao dịch kinh doanh ở một cửa hàng. Khách hàng yêu cầu sản phẩm X. Họ được trao cho sản phẩm X, và họ phải trả tiền cho sản phẩm ấy. Sau khi khách hàng yêu cầu sản phẩm, người bán hàng thực hiện những hành động sau:

- Kiểm tra mặt hàng trên kệ hàng.
- Giảm số lượng mặt hàng trong bảng kiểm kê sau khi bán.

Tuy nhiên, khách hàng không được biết những chi tiết này.

Tính thừa kế

Hãy khảo sát các lớp sau:

Lớp Sinh viên	Lớp Nhân viên	Lớp Khách hàng
Tên	Tên	Tên
Địa chỉ	Địa chỉ	Địa chỉ
Điểm môn 1	Lương	Kiểu xe đã bán
Điểm môn 2	Chức vụ	Nhập tên
Nhập tên	Nhập tên	Nhập địa chỉ
Nhập địa chỉ	Nhập địa chỉ	Nhập kiểu xe
Nhập điểm	Nhập chức vụ	Xuất hóa đơn
Tính tổng điểm	Tính lương	

Trong tất cả ba lớp, chúng ta thấy có một vài thuộc tính và hoạt động chung. Chúng ta muốn nhóm những thuộc tính và những hoạt động ấy lại, và định nghĩa chúng trong một lớp ‘Người’.

Lớp Người

Tên
Địa chỉ
Nhập tên
Nhập địa chỉ

Ba lớp ‘Sinh viên’, ‘Nhân viên’ và ‘Khách hàng’ có những thành phần giống lớp ‘Người’. Nói cách khác, ba lớp ấy có tất cả các thuộc tính và các phương thức của lớp ‘Người’, ngoài ra chúng còn có những thuộc tính và những phương thức riêng.

Chúng ta cần phải định nghĩa lớp ‘Người’ và sử dụng nó trong khi định nghĩa các lớp ‘Sinh viên’, ‘Nhân viên’ và ‘Khách hàng’.

Chúng ta xây dựng một lớp ‘Người’ với những thuộc tính và những hoạt động như đã trình bày ở hình trên. Kế tiếp, chúng ta xây dựng lớp ‘Khách hàng’ bao gồm lớp ‘Người’ cộng với những thuộc tính và những phương thức riêng.

Chúng ta có thể định nghĩa các lớp ‘Sinh viên’ và ‘Nhân viên’ theo cùng cách thức trên. Như thế, cả ba lớp ‘Khách hàng’, ‘Sinh viên’ và ‘Nhân viên’ đều chia sẻ những thuộc tính và những phương thức mà lớp ‘Người’ cung cấp.

Lớp Sinh viên	Lớp Nhân viên	Lớp Khách hàng
Điểm môn 1	LươngChức vụ	Kiểu xe bán được
Điểm môn 2	Chức vụ	Nhập kiểu xe
Nhập điểm	Nhập chức vụ	Xuất hóa đơn
Tính tổng tiền	Tính lương	

Theo ngôn ngữ hướng đối tượng, lớp ‘Khách hàng’ được gọi là thừa kế lớp ‘Người’.

Định nghĩa: Tính thừa kế cho phép một lớp chia sẻ các thuộc tính và các phương thức được định nghĩa trong một hoặc nhiều lớp khác.

Có hai khái niệm quan trọng khác liên kết với tính thừa kế. Lớp ‘Khách hàng’ là lớp ‘Người’ cộng thêm cái khác. Như thế, lớp ‘Khách hàng’ có tất cả các thuộc tính và các phương thức được định nghĩa trong lớp ‘Người’ cộng với các thuộc tính và các hoạt động của riêng nó.

Trong ví dụ này, lớp ‘Khách hàng’ được gọi là ‘lớp con’ (subclass).

Định nghĩa: Lớp thừa hưởng từ một lớp khác được gọi là Subclass.

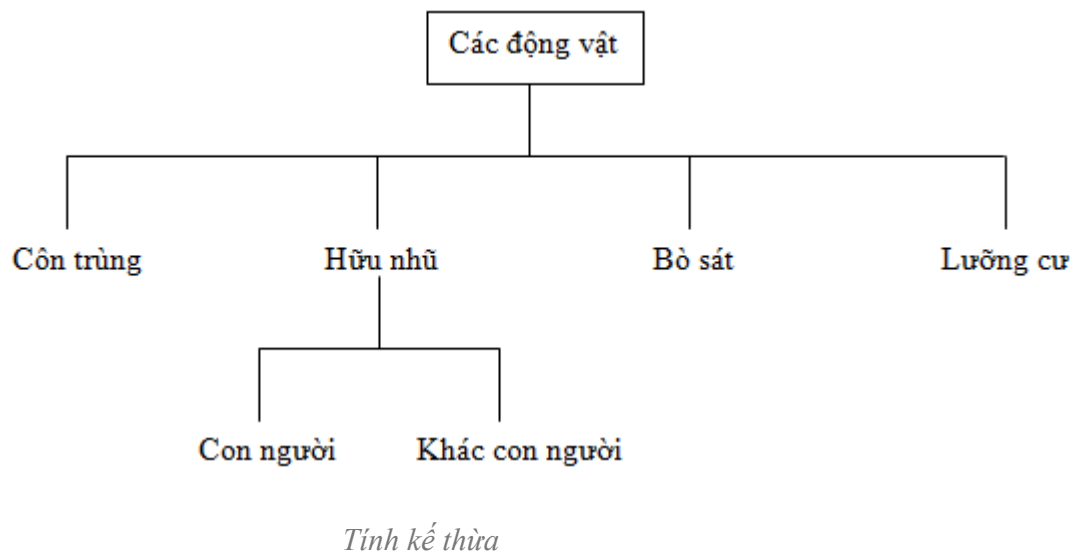
Trong ví dụ trên, lớp ‘Người’ được coi là ‘lớp trên’ (superclass).

Định nghĩa: Một **Superclass** là một lớp mà các đặc tính của nó được một lớp khác thừa hưởng.

Hãy xem xét ví dụ về lớp ‘Các động vật’ ở hình 4. ‘Các động vật’ là lớp trên cùng mà các lớp khác kế thừa. Chúng ta có một dãy các lớp trung gian – ‘Côn trùng’, ‘Hữu nhũ’, ‘Bò sát’, ‘Lưỡng cư’ - mà dãy các lớp dưới kế thừa.

Các lớp ‘Côn trùng’, ‘Hữu nhũ’, ‘Bò sát’, ‘Lưỡng cư’ là những lớp con của lớp trên ‘Các động vật’. Như thế, những lớp này có tất cả những thuộc tính và các hoạt động của lớp ‘Các động vật’, cộng thêm những thuộc tính và những phương thức của riêng chúng.

Lớp ‘Hữu nhũ’ là lớp mà các lớp ‘Con người’ và ‘Khác con người’ thừa kế. Như thế, các lớp ‘Con người’ và ‘Khác con người’ là các lớp con của lớp trên ‘Hữu nhũ’.

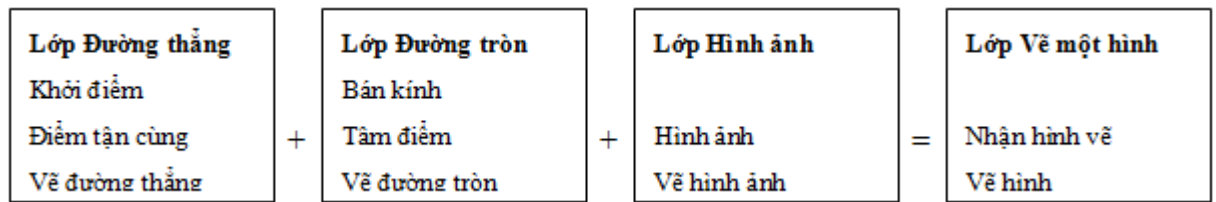


Tính Đa Thừa kế

Trong tất cả các ví dụ trên, một lớp thừa kế chỉ từ một lớp. Ngay cả trong ví dụ thừa kế về các loại phương tiện di chuyển, mỗi lớp con chỉ có một lớp cha. Trường hợp như thế gọi là ‘thừa kế đơn’ (single inheritance).

Trong ‘đa thừa kế’, một lớp con thừa kế từ hai hay nhiều lớp cha.

Hãy khảo sát ví dụ sau:



Trong hình trên, chúng ta đã xây dựng một lớp ‘Vẽ một hình’, lớp này thừa hưởng ba lớp: ‘Đường thẳng’, ‘Đường tròn’, ‘Hình ảnh’. Như thế lớp ‘Vẽ một hình’ kết hợp chức năng của ba lớp trên thêm vào chức năng được định nghĩa bên trong nó.

Lớp ‘Vẽ một hình’ là một ví dụ về tính đa thừa kế.

Có thể sử dụng tính đa thừa kế để xây dựng một lớp mới, lớp này dẫn xuất chức năng của nó từ một vài lớp khác. Như thế, xét theo góc cạnh của người sử dụng lớp mới này, chỉ cần một lớp mà cung cấp tất cả các chức năng. Như vậy, họ không cần phải sử dụng nhiều đối tượng khác nhau.

Sự thuận lợi quan trọng nhất của tính thừa kế là nó thúc đẩy việc tái sử dụng mã chương trình.

Trong ví dụ trên, chúng ta có ba lớp ‘Đường thẳng’, ‘Đường tròn’ và ‘Hình ảnh’. Giả thiết rằng ba người khác nhau xây dựng ba lớp này riêng biệt. Bây giờ, người sử dụng cần xây dựng một lớp để vẽ đường thẳng, vẽ đường tròn cũng như hiển thị hình ảnh. Vì thế họ tìm kiếm xem có lớp nào đáp ứng một hoặc tất cả các yêu cầu đó. Nếu có những lớp cung cấp chức năng thỏa yêu cầu thì người sử dụng sẽ thừa kế những lớp đó để tạo một lớp mới.

Giờ đây người sử dụng chỉ còn phải viết mã chương trình cho những đặc tính chưa có sau tiến trình thừa kế. Người sử dụng có thể sử dụng chính ba lớp trên. Tuy nhiên, sự thừa kế cung cấp một bó những chức năng hỗn độn trong một lớp.

Tính Đa hình

Trong một chương trình có cấu trúc (a structured program), một phương thức chỉ ứng dụng cho một đối tượng. Chẳng hạn xét toán tử ‘Cộng’. Toán tử này chỉ tính tổng của hai số nguyên. Khi truyền hai giá trị 2 và 3 thì nó hiển thị 5. Chúng ta không thể có một loại toán tử ‘Cộng’ để tính tổng của hai giá trị văn bản (text) ‘Hello!’ và ‘How are you?’ để có được chuỗi văn bản kết quả ‘Hello! How are you?’

Trong hệ thống hướng đối tượng thì tình huống mô tả trên là có thể.

Định nghĩa : Tính đa hình cho phép một phương thức có các cách thể hiện khác nhau trên nhiều loại đối tượng khác nhau.

Với tính đa hình, nếu cùng một phương thức ứng dụng cho các đối tượng thuộc các lớp khác nhau thì nó đưa đến những kết quả khác nhau. Bản chất của sự việc chính là phương thức này bao gồm cùng một số lượng các tham số.

Tính đa hình là một trong những đặc tính quan trọng nhất của hệ thống hướng đối tượng.

Một ví dụ khác là phương thức hiển thị. Tùy thuộc vào đối tượng tác động, phương thức ấy có thể hiển thị một chuỗi, hoặc vẽ một đường thẳng, hoặc hiển thị một hình ảnh. Hãy khảo sát hình sau:

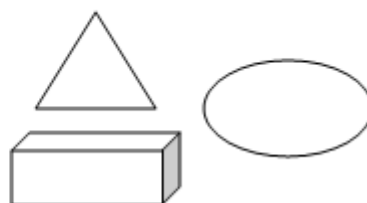
Lớp: Hình thể

Các lớp con

Các phương thức:

Vẽ

Di chuyển



Lớp “Hình thể” và các lớp con

Hình trên cho thấy rằng ‘Vẽ’ là một phương thức được chia sẻ giữa các lớp con của lớp ‘Hình thể’. Tuy nhiên, phương thức Vẽ được ứng dụng cho hình hộp sẽ khác với hình êlip.

Tính đa hình hỗ trợ tính đóng gói.

Xét trên mức độ người sử dụng, họ chỉ cần một phương thức ‘Vẽ’ của lớp ‘Hình thể’. Còn cách thức mà phương thức ‘Vẽ’ được thực thi cho các trường hợp khác nhau thì họ không cần biết.

Những ưu điểm của Phương pháp hướng đối tượng

Lập trình hướng đối tượng đòi hỏi một sự chuyển hướng quan trọng trong tư duy của các lập trình viên. Phương pháp này làm cho tốc độ phát triển các chương trình mới nhanh hơn, và nếu được sử dụng đúng đắn phương pháp này sẽ cải tiến việc duy trì (maintenance), việc tái sử dụng (reusability) và việc đánh giá (revision) phần mềm.

Những điểm ưu điểm của phương pháp hướng đối tượng là:

- Phương pháp này tiến hành tiến trình phân tích, thiết kế và phát triển một vấn đề trong khuôn khổ những khái niệm và thuật ngữ thuộc lĩnh vực ứng dụng. Vì

thể, có một sự tương hợp cao nhất giữa việc phát triển ứng dụng và vấn đề thực tế.

- Chẳng hạn như trong trường hợp bán xe hơi, ở mọi giai đoạn của việc phân tích, thiết kế và phát triển ứng dụng, luôn luôn có tiếng nói của khách hàng, của nhân viên bán hàng ...
- Phương pháp này hỗ trợ việc chia sẻ bên trong một ứng dụng.
- Phương pháp này hỗ trợ việc tái sử dụng các đối tượng khi các ứng dụng mới được phát triển. Đây là ưu điểm rất quan trọng xét trong khía cạnh giảm thiểu chi phí về lâu về dài.
- Chẳng hạn như hành vi của khách hàng một khi được mô hình hóa trong một ứng dụng thì có thể được sử dụng lại cho những ứng dụng liên hệ có bao gồm mô hình khách hàng.
- Phương pháp này giảm thiểu các lỗi và những vấn đề liên quan đến việc bảo trì ứng dụng do khả năng tái sử dụng các đối tượng.
- Phương pháp này tăng tốc tiến trình thiết kế và phát triển, một lần nữa đây là kết quả của việc tái sử dụng các đối tượng.

Tóm tắt bài học

- Lập trình hướng Đối tượng là một cách tư duy mới để giải quyết vấn đề bằng máy vi tính. Thay vì nỗ lực đưa vấn đề vào trong khuôn khổ quen thuộc với máy vi tính, phương pháp hướng đối tượng tìm kiếm một giải pháp toàn vẹn cho một vấn đề.
- Sự trừu tượng hóa dữ liệu là tiến trình xác định và nhóm các thuộc tính và các phương thức liên quan đến một thực thể cụ thể, trong tương quan với một ứng dụng.
- Một lớp định nghĩa một thực thể theo những thuộc tính và những phương thức chung.
- Một đối tượng là một trường hợp của một lớp.
- Một lớp định nghĩa một thực thể, còn đối tượng là thực thể hiện thực.
- Tiến trình hiện thực hóa một đối tượng được gọi là Thiết lập (Construction).
- Tiến trình hủy bỏ một đối tượng được gọi là Hủy (Destruction).
- Tính bền vững là khả năng lưu trữ dữ liệu của một đối tượng vượt quá thời gian tồn tại của đối tượng đó.
- Việc đóng gói là tiến trình che giấu việc thực thi chi tiết của một đối tượng đối với người sử dụng đối tượng ấy.
- Tính thừa kế là cơ chế cho phép một lớp chia sẻ các thuộc tính và các phương thức được định nghĩa trong một hoặc nhiều lớp khác.
- Tính đa hình là một thuộc tính cho phép một phương thức có các tác động khác nhau trên nhiều đối tượng khác nhau.
- Phương pháp hướng đối tượng đưa ra tiến trình phân tích, thiết kế và phát triển ứng dụng trong khuôn khổ các khái niệm và các thuật ngữ thuộc lĩnh vực ứng dụng.

Kiểm tra sự tiến bộ

1. Sự trừu tượng hóa dữ liệu đồng nghĩa với sự che giấu dữ liệu. **(Đúng/Sai)**
2. Định nghĩa sự Trừu tượng hóa dữ liệu.
3. Việc đóng gói dữ liệu che giấu những chi tiết thực thi đối với những đối tượng khác. **(Đúng/Sai)**
4. Tính đa hình cho phép chúng ta tạo những đối tượng khác nhau với cùng một tên. **(Đúng/Sai)**
5. Mỗi một đối tượng có một _____, _____ và _____ được xác định.
6. Tất cả các đối tượng của một lớp đều có cùng một tập hợp các thuộc tính. **(Đúng/Sai)**
7. Một đối tượng định nghĩa một thực thể, trong khi một lớp là thực thể một cụ thể. **(Đúng/Sai)**
8. Định nghĩa tính Đa hình.

Bài tập

Thiết kế các thành phần và các hành động khi một khách hàng thực hiện một giao dịch ATM (Automatic Teller Machine).

Liệt kê những thuộc tính và những phương thức cần có để vẽ một hình đa giác.

Nhập môn Java

Giới thiệu Java

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

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

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

Java là gì

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

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

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

Các đặc trưng của Java

- Đơn giản

- Hướng đối tượng
- Độc lập phần cứng và hệ điều hành
- Mạnh
- Bảo mật
- Phân tán
- Đa luồng
- Động

Đơn giản

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

Hướng đối tượng

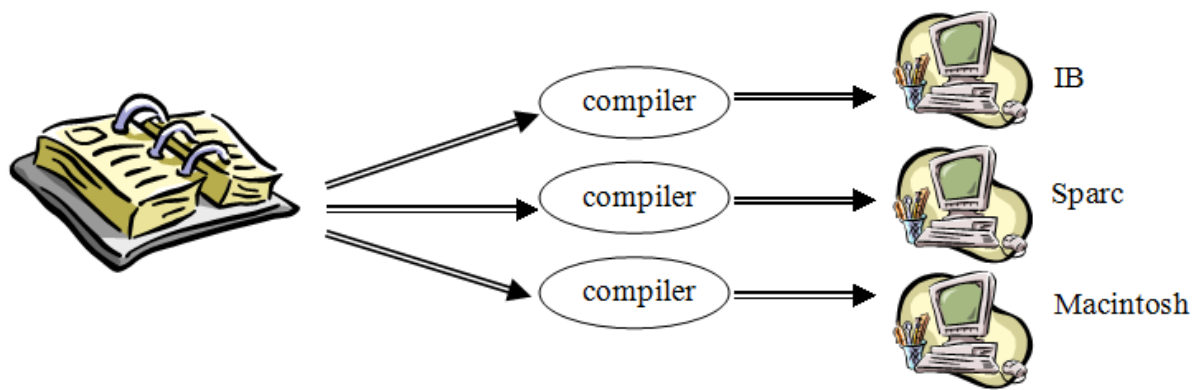
Java được thiết kế xoay quanh mô hình hướng đối tượng. Vì vậy trong Java, tiêu điểm là dữ liệu và các phương pháp thao tác lên dữ liệu đó. Dữ liệu và các phương pháp mô tả trạng thái và cách ứng xử của một đối tượng trong Java.

Độc lập phần cứng và hệ điều hành

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

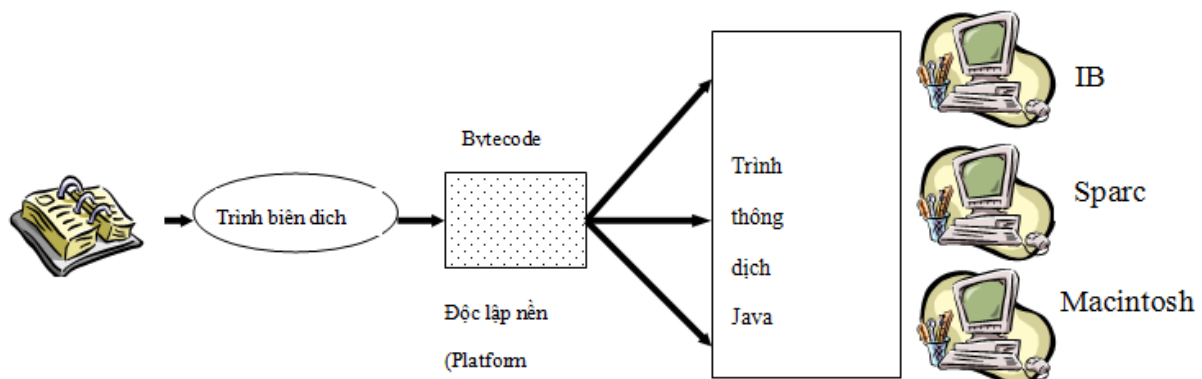
Ở mức mã nguồn, người lập trình cần mô tả kiểu cho mỗi biến. Kiểu dữ liệu trong Java nhất quán cho tất cả các hệ điều hành và phần cứng khác nhau. Java có riêng một thư viện các lớp cơ sở. Vì vậy chương trình Java được viết trên một máy có thể dịch và chạy trơn tru trên các loại máy khác mà không cần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

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



Cách biên dịch truyền thống

Đối với các chương trình viết bằng C, C++ hoặc một ngôn ngữ nào khác, trình biên dịch sẽ chuyển tập lệnh thành mã máy (machine code), hay lệnh của bộ vi xử lý. Những lệnh này phụ thuộc vào CPU hiện tại trên máy bạn. Nên khi muốn chạy trên loại CPU khác, chúng ta phải biên dịch lại chương trình. **Hình 1** thể hiện quá trình để thực thi chương trình viết bằng C++ trên các loại máy khác nhau. **Hình 2** thể hiện quá trình thực thi chương trình viết bằng Java chạy trên nhiều hệ máy khác nhau.



Dịch chương trình Java

Môi trường phát triển của Java được chia làm hai phần: Trình biên dịch và trình thông dịch. Không như C hay C++, trình biên dịch của Java chuyển mã nguồn thành dạng bytecode độc lập với phần cứng mà có thể chạy trên bất kỳ CPU nào.

Nhưng để thực thi chương trình dưới dạng bytecode, tại mỗi máy cần phải có trình thông dịch của Java hay còn gọi là máy ảo Java. Máy ảo Java chuyển bytecode thành mã lệnh mà CPU thực thi được.

Mạnh mẽ

Java là ngôn ngữ yêu cầu chặt chẽ về kiểu dữ liệu. Phải khai báo kiểu dữ liệu tường minh khi viết chương trình. Java kiểm tra lúc biên dịch và cả trong thời gian thông dịch vì vậy Java loại bỏ một số loại lỗi lập trình nhất định.

Java không sử dụng con trỏ và các phép toán con trỏ. Java kiểm tra tất cả các truy nhập đến mảng, chuỗi khi thực thi để đảm bảo rằng các truy nhập đó không ra ngoài giới hạn kích thước. Java kiểm tra sự chuyển đổi kiểu dữ liệu từ dạng này sang dạng khác lúc thực thi.

Trong các môi trường lập trình truyền thống, lập trình viên phải tự mình cấp phát bộ nhớ. Trước khi chương trình kết thúc thì phải tự giải phóng bộ nhớ đã cấp. Vấn đề nảy sinh khi lập trình viên quên giải phóng bộ nhớ đã xin cấp trước đó. Trong chương trình Java, lập trình viên không phải bận tâm đến việc cấp phát bộ nhớ. Quá trình cấp phát, giải phóng 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

Viruses là nguyên nhân gây ra sự lo lắng trong việc sử dụng máy tính. Trước khi có Java, các lập trình viên phải quét virus các tệp trước khi tải về hay thực hiện chúng. Thông thường việc này cũng không loại trừ hoàn toàn virus. Ngoài ra chương trình khi thực thi có khả năng tìm kiếm và đọc các thông tin nhạy cảm trên máy của người sử dụng mà người sử dụng không hề hay biết.

Java cung cấp một môi trường quản lý thực thi chương trình. Nó cho rằng không có một đoạn mã nào là an toàn cả. Và vì vậy Java không chỉ là ngôn ngữ lập trình thuần túy mà còn cung cấp nhiều mức để kiểm soát tính an toàn khi thực thi chương trình.

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

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

Lớp thứ ba được đảm bảo bởi trình thông dịch. Chúng kiểm tra xem bytecode có đảm bảo các qui tắc an toàn trước khi thực thi.

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

Phân tán

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

Đa luồng

Chương trình Java đa luồng(Multithreading) để thực thi các công việc đồng thời. Chúng cũng cung cấp giải pháp đồng bộ giữa các luồng. Đặc tính hỗ trợ đa này cho phép xây dựng các ứng dụng trên mạng chạy hiệu quả.

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.

Các kiểu chương trình Java

Chúng ta có thể xây dựng các loại chương trình Java như sau:

Applets

Applet là chương trình được tạo ra để sử dụng trên Internet thông qua các trình duyệt hỗ trợ Java như IE hay Netscape. Bạn có thể dùng Java để xây dựng Applet. Applet được nhúng bên trong trang Web. Khi trang Web hiển thị trong trình duyệt, Applet sẽ được tải về và thực thi tại trình duyệt.

Ứng dụng thực thi qua dòng lệnh

Các chương trình này chạy từ dấu nhắc lệnh và không sử dụng giao diện đồ họa. Các thông tin nhập xuất được thể hiện tại dấu nhắc lệnh.

Ứng dụng đồ họa

Đây là các chương trình Java chạy độc lập cho phép người dùng tương tác qua giao diện đồ họa.

Servlet

Java thích hợp để phát triển ứng dụng nhiều lớp. Applet là chương trình đồ họa chạy trên trình duyệt tại máy trạm. Ở các ứng dụng Web, máy trạm gửi yêu cầu tới máy chủ. Máy chủ xử lý và gửi kết quả trở lại máy trạm. Các Java API chạy trên máy chủ chịu trách nhiệm xử lý tại máy chủ và trả lời các yêu cầu của máy trạm. Các Java API chạy trên máy chủ này mở rộng khả năng của các ứng dụng Java API chuẩn. Các ứng dụng trên máy chủ này được gọi là các Servlet. hoặc Applet tại máy chủ. Xử lý Form của HTML là cách sử dụng đơn giản nhất của Servlet. Chúng còn có thể được dùng để xử lý dữ liệu, thực thi các giao dịch và thường được thực thi thông qua máy chủ Web.

Ứng dụng cơ sở dữ liệu

Các ứng dụng này sử dụng JDBC API để kết nối tới cơ sở dữ liệu. Chúng có thể là Applet hay ứng dụng, nhưng Applet bị giới hạn bởi tính bảo mật.

Máy ảo Java (JVM-Java Virtual Machine)

Máy ảo Java là trái tim của ngôn ngữ Java. Môi trường Java bao gồm năm phần tử sau:

- Ngôn ngữ
- Định nghĩa Bytecode
- Các thư viện lớp Java/Sun
- Máy ảo Java (JVM)
- Cấu trúc của file .class

Các phần tử tạo cho Java thành công là

- Định nghĩa Bytecode
- Cấu trúc của file .class
- Máy ảo Java (JVM)

Khả năng cơ động của file .class cho phép các chương trình Java viết một lần nhưng chạy ở bất kỳ đâu. Khả năng này có được nhờ sự giúp đỡ của máy ảo Java.

Máy ảo Java là gì ?

Máy ảo là một phần mềm dựa trên cơ sở máy tính ảo. Nó có tập hợp các lệnh logic để xác định các hoạt động của máy tính. Người ta có thể xem nó như một hệ điều hành thu nhỏ. Nó thiết lập các lớp trừu tượng cho: Phần cứng bên dưới, hệ điều hành, mã đã biên dịch.

Trình biên dịch chuyển mã nguồn thành tập các lệnh của máy ảo mà không phụ thuộc vào phần cứng cụ thể. Trình thông dịch trên mỗi máy sẽ chuyển tập lệnh này thành chương trình thực thi. Máy ảo tạo ra một môi trường bên trong để thực thi các lệnh bằng cách:

- Nạp các file .class
- Quản lý bộ nhớ
- Dọn “rác”

Việc không nhất quán của phần cứng làm cho máy ảo phải sử dụng ngăn xếp để lưu trữ các thông tin sau:

- Các “Frame” chứa các trạng thái của các phương thức.
- Các toán hạng của mã bytecode.
- Các tham số truyền cho phương thức.
- Các biến cục bộ.

Khi JVM thực thi mã, một thanh ghi cục bộ có tên “**Program Counter**” được sử dụng. Thanh ghi này trỏ tới lệnh đang thực hiện. Khi cần thiết, có thể thay đổi nội dung thanh ghi để đổi hướng thực thi của chương trình. Trong trường hợp thông thường thì từng lệnh một nối tiếp nhau sẽ được thực thi.

Một khái niệm thông dụng khác trong Java là trình biên dịch “**Just In Time-JIT**”. Các trình duyệt thông dụng như Netscape hay IE đều có JIT bên trong để tăng tốc độ thực thi chương trình Java. Mục đích chính của JIT là chuyển tập lệnh bytecode thành mã máy cụ thể cho từng loại CPU. Các lệnh này sẽ được lưu trữ và sử dụng mỗi khi gọi đến.

Quản lý bộ nhớ và dọn rác

Trong C, C++ hay Pascal người lập trình sử dụng phương pháp trực tiếp để cấp phát và thu hồi bộ nhớ ở vùng “Heap”. Heap là vùng bộ nhớ lớn được phân chia cho tất cả các luồng.

Để quản lý Heap, bộ nhớ được theo dõi qua các danh sách sau:

- Danh sách các vùng nhớ chưa sử dụng.
- Danh sách các vùng đã cấp.

Khi có một yêu cầu về cấp phát bộ nhớ, hệ thống xem xét trong danh sách chưa cấp phát để lấy ra khối bộ nhớ đầu tiên có kích cỡ sát nhất với lượng bộ nhớ cần thiết. Kỹ thuật cấp phát này giảm tối thiểu việc phân mảnh của heap.

“**Coalescing**” là kỹ thuật khác cũng giảm thiểu việc phân mảnh của heap bằng cách gom lại các vùng nhớ chưa dùng liền nhau thành một khối lớn hơn. Còn kỹ thuật sắp xếp lại các phần đã dùng để tạo vùng nhớ chưa sử dụng lớn hơn gọi là “**Compaction**”.

Java sử dụng hai heap riêng biệt cho cấp phát vùng nhớ tĩnh và vùng nhớ động. Một heap (heap tĩnh) chứa các định nghĩa về lớp, các hằng và danh sách các phương thức. Heap còn lại (heap động) được chia làm hai phần được cấp phát theo hai chiều ngược nhau. Một bên chứa đối tượng còn một bên chứa con trỏ trỏ đến đối tượng đó.

“**Handle**” là cấu trúc bao gồm hai con trỏ. Một trỏ đến bảng phương thức của đối tượng, con trỏ thứ hai trỏ đến chính đối tượng đó. Chú ý rằng khi “compaction” cần cập nhật lại giá trị con trỏ của cấu trúc “handle”.

Thuật toán dọn rác có thể áp dụng cho các đối tượng đặt trong heap động. Khi có yêu cầu về bộ nhớ, trình quản lý heap trước tiên kiểm tra danh sách bộ nhớ chưa cấp phát. Nếu không tìm thấy khối bộ nhớ nào phù hợp (về kích cỡ) thì trình dọn rác sẽ được kích hoạt khi hệ thống rỗi. Nhưng khi đòi hỏi bộ nhớ cấp bách thì trình dọn rác sẽ được kích hoạt ngay.

Trình dọn rác gọi phương thức finalize của đối tượng trước khi dọn dẹp đối tượng. Hàm này sẽ dọn dẹp các tài nguyên bên ngoài như các file đang mở. Công việc này không được trình dọn rác thực thi.

Quá trình kiểm tra file .class

Việc kiểm tra được áp dụng cho tất cả các file .class sắp được nạp lên bộ nhớ để đảm bảo tính an toàn. Trình “Class Loader” sẽ kiểm tra tất cả các file .class không thuộc hệ điều hành với mục đích giám sát sự tuân thủ các nghi thức để phát hiện các file .class có nguy cơ gây hư hỏng đến bộ nhớ, hệ thống file cục bộ, mạng hoặc hệ điều hành. Quá trình kiểm tra sẽ xem xét tổng thể tính nguyên vẹn của một lớp.

File .class bao gồm ba phần logic là:

- Bytecode
- Thông tin về Class như phương thức, giao diện và các giá trị hằng số được tập hợp trong quá trình biên dịch.
- Các thuộc tính về lớp.

Các thông tin của file .class được xem xét riêng rẽ trong các bảng sau:

- Bảng Field chứa các thuộc tính
- Bảng Method chứa các hàm của class
- Bảng Interface và các hằng số.

Quá trình kiểm tra file .class được thực hiện ở bốn mức:

- Mức đầu tiên thực hiện việc kiểm tra cú pháp để đảm bảo tính cấu trúc và tính toàn vẹn cú pháp của file .class được nạp.
- Mức thứ hai sẽ xem xét file .class để đảm bảo các file này không vi phạm các nguyên tắc về sự nhất quán ngữ nghĩa.
- Mức thứ ba sẽ kiểm tra bytecode. Trong bước này sẽ kiểm tra số thông số truyền vào phương thức, khả năng truy xuất sai chỉ số của mảng, chuỗi, biểu thức.
- Mức thứ tư sẽ kiểm tra trong thời gian thực thi để giám sát các việc còn lại mà ba bước trên chưa làm. Ví dụ như liên kết tới các lớp khác trong khi thực thi, hay kiểm tra quyền truy xuất. Nếu mọi điều thỏa mãn, lớp sẽ được khởi tạo.

Bộ công cụ phát triển JDK (Java Development Kit)

Sun Microsystem đưa ra ngôn ngữ lập trình Java qua sản phẩm có tên là Java Development Kit (JDK). Ba phiên bản chính là:

Java 1.0 - Sử dụng lần đầu vào năm 1995

Java 1.1 – Đưa ra năm 1997 với nhiều ưu điểm hơn phiên bản trước.

Java 2 – Phiên bản mới nhất

JDK bao gồm Java Plug-In, chúng cho phép chạy trực tiếp Java Applet hay JavaBean bằng cách dùng JRE thay cho sử dụng môi trường thực thi mặc định của trình duyệt.

JDK chứa các công cụ sau:

Trình biên dịch, 'javac'

Cú pháp:

```
javac [options] sourcecodename.java
```

Trình thông dịch, 'java'

Cú pháp:

```
java [options] classname
```

Trình dịch ngược, 'javap'

javap dịch ngược bytecode và in ra thông tin về các thuộc tính (các trường), các phương thức của một lớp.

Cú pháp:

```
javap [options] classname
```

Công cụ sinh tài liệu, 'javadoc'

Tiện ích này cho phép ta tạo ra tệp HTML dựa trên các lời giải thích trong mã chương trình (phần nằm trong cặp dấu `/* */`).

Cú pháp:

```
javadoc [options] sourcecodename.java
```

Chương trình tìm lỗi - Debug, 'jdb'

Cú pháp:

```
jdb [options] sourcecodename.java
```

hay

```
jdb -host -password [options] sourcecodename.java
```

Chương trình xem Applet , 'appletviewer'

Cú pháp:

```
appletviewer [options] url
```

Java Core API

Nhân Java API được thay thế bởi phiên bản JFC 1.1.

Một số package thông dụng được:

java.lang

Chứa các lớp quan trọng nhất của ngôn ngữ Java. Chúng bao gồm các kiểu dữ liệu cơ bản như ký tự, số nguyên,... Chúng cũng chứa các lớp làm nhiệm vụ xử lý lỗi và các lớp vào ra chuẩn. Một vài lớp quan trọng khác như String hay StringBuffer.

java.applet

Đây là package nhỏ nhất chứa một mình lớp Applet. Các Applet nhúng trong trang Web hay chạy trong appletviewer đều thừa kế từ lớp này.

java.awt

Package này được gọi là Abstract Window Toolkit (AWT). Chúng chứa các lớp dùng để tạo giao diện đồ họa. Một số lớp bên trong là: Button, GridBagLayout, Graphics.

java.io

Cung cấp thư viện vào ra chuẩn. Chúng cho phép tạo và quản lý dòng dữ liệu theo nhiều cách.

java.util

Package này cung cấp một số công cụ hữu ích. Một vài lớp của package này là: Date, Hashtable, Stack, Vector và StringTokenizer.

java.net

Cung cấp khả năng giao tiếp với máy từ xa. Cho phép tạo và kết nối tới Socket hoặc URL.

java.awt.event

Chứa các lớp, giao diện dùng để xử lý các sự kiện trong chương trình như chuột, bàn phím.

java.rmi

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.

java.security

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.

java.sql

Package này chứa Java DataBase Connectivity (JDBC), dùng để truy xuất cơ sở dữ liệu quan hệ như Oracle, SQL Server,....

Các đặc trưng mới của Java 2

Các phiên bản trước của Java chỉ thích hợp để viết các ứng dụng nhỏ trên Web hơn là xây dựng các ứng dụng chạy trên mạng để đảm nhiệm toàn bộ các công việc của của một công ty hoặc hệ thống phân tán. Java 2 đáp ứng yêu cầu này. Một vài đặc trưng của chúng là:

- **Swing** Đây là một tập các lớp và giao diện mới dùng để tạo giao diện ứng dụng đồ họa "Look and Feel".
- **Kéo và thả** Đây là khả năng di chuyển thông tin giữa các ứng dụng hay các phần khác nhau của chương trình.
- **Java 2D API** Chứa tập hợp các lớp hỗ trợ cho ảnh và đồ họa hai chiều.
- **Java sound** Tập hợp hoàn toàn mới các lớp đặc trưng về âm thanh của Java.
- **RMI** (Remote Method Invocation) cho phép các ứng dụng gọi các phương thức của đối tượng tại máy khác và cho phép giao tiếp với chúng.

Tóm tắt

- Java là ngôn ngữ biên dịch và thông dịch
- Các đặc trưng của Java
 - Đơn giản
 - Hướng đối tượng
 - Độc lập phần cứng
 - Mạnh
 - Bảo mật
 - Phân tán
 - Đa luồng
 - Động
- Máy ảo Java
- Java Development Kit
- Các đặc trưng mới của Java 2

Kiểm tra sự tiến bộ

1. Bạn có thể viết các chương trình dạng thủ tục bằng Java. (**Đúng/Sai**)
2. Java là ngôn ngữ có kiểu dữ liệu chặt chẽ. (**Đúng/Sai**)
3.là chương trình Java chạy độc lập, và sử dụng giao diện đồ họa để người sử dụng nhập dữ liệu.
4.sử dụng JDBC API để kết nối cơ sở dữ liệu.
5.hiểu một dòng các lệnh máy tính trừu tượng.
6. Coalescing và Compaction là gì?
7. Lệnh được sử dụng để dịch file mã nguồn .java.
8. Lớp phải là lớp cha của các applet, applet là chương có thể nhúng vào trang Web hay chạy bằng Java AppletViewer.

Bài tập

Cài đặt Java 2

Gõ các lệnh sau tại dấu nhắc và liệt kê các tham số khác nhau của chúng:

- javac
- java

Nền tảng của ngôn ngữ Java

Cấu trúc một chương trình Java

Phần đầu của một chương trình Java xác định thông tin môi trường. Để làm được việc này, chương trình được chia thành các lớp hoặc các gói riêng biệt. Những gói này sẽ được chỉ dẫn trong chương trình. Thông tin này được chỉ ra với sự trợ giúp của lệnh nhập “import”. Mỗi chương trình có thể có nhiều hơn một lệnh nhập. Dưới đây là một ví dụ về lệnh nhập:

```
import java. awt.*;
```

Lệnh này nhập gói ‘awt’. Gói này dùng để tạo các đối tượng GUI. Ở đây java là tên của thư mục chứa gói ‘awt’. Ký hiệu “*” chỉ tất cả các lớp thuộc gói này.

Trong java, tất cả các mã, bao gồm các biến và cách khai báo nên được thực hiện trong phạm vi một lớp. Bởi vậy, từng khai báo lớp được tiến hành sau lệnh nhập. Một chương trình đơn giản có thể chỉ có một vài lớp. Những lớp này có thể mở rộng thành các lớp khác. Mỗi lệnh đều được kết thúc bởi dấu chấm phẩy “;”. Chương trình còn có thể bao gồm các ghi chú, chỉ dẫn. Khi dịch, chương trình dịch sẽ tự loại bỏ các ghi chú này.

Dạng cơ bản của một lớp được xác định như sau :

```
class classname { /* Đây là dòng ghi chú*/ int num1,num2;  
// Khai báo biến với các dấu phẩy giữa các biến Show() {  
// Method body statement (s); // Kết thúc bởi dấu chấm  
phẩy } }
```

“Token” là đơn vị riêng lẻ, nhỏ nhất, có ý nghĩa đối với trình biên dịch của một chương trình Java. Một chương trình java là tập hợp của các “token”

Các “token” được chia thành năm loại:

- Định danh (identifiers): Dùng để thể hiện tên biến, phương thức, hoặc các lớp. Chương trình biên dịch sẽ xác định các tên này là duy nhất trong chương trình. Khi khai báo định danh cần lưu ý các điểm sau đây:
 - Mỗi định danh được bắt đầu bằng một chữ cái, một ký tự gạch dưới (_) hay dấu \$. Các ký tự tiếp theo có thể là chữ cái, chữ số, dấu \$ hoặc một ký tự được gạch dưới.

- Mỗi định danh chỉ được chứa hai ký tự đặc biệt này là dấu gạch dưới (`_`) và dấu `$`. Ngoài ra không được phép sử dụng bất kỳ ký tự đặc biệt nào khác.
- Các định danh không được chứa dấu cách “ ” (space).
- Từ khoá/từ dự phòng (Keyword/Reserve Words): Một số định danh đã được Java xác định trước. Người lập trình không được phép sử dụng chúng như một định danh. Ví dụ ‘class’, ‘import’ là những từ khoá.
- Ký tự phân cách (separator): Thông báo cho trình biên dịch việc phân nhóm các phần tử của chương trình. Một vài ký tự phân cách của java được chỉ ra dưới đây: { } ; ,
- Nguyên dạng (literals): Là các giá trị không đổi trong chương trình. Nguyên dạng có thể là các số, chuỗi, các ký tự hoặc các giá trị Boolean. Ví dụ 21, ‘A’, 31.2, “This is a sentence” là những nguyên dạng.
- Các toán tử: Các quá trình đánh giá, tính toán được thực hiện trên dữ liệu hoặc các đối tượng. Java có một tập lớn các toán tử. Chúng ta sẽ thảo luận chi tiết ở chương này.

Chương trình Java đầu tiên

Chúng ta hãy bắt đầu từ chương trình Java cổ điển nhất với một ứng dụng đơn giản. Chương trình sau đây cho phép hiển thị một thông điệp:

```
// This is a simple program called "First.java" class
First { public static void main(String args[]) {
System.out.println("My first program in Java"); } }
```

Tên file đóng vai trò rất quan trọng trong Java. Chương trình biên dịch Java chấp nhận phần mở rộng **.java**. Trong Java, mã lệnh phải nằm trong các lớp. Bởi vậy tên lớp và tên file phải trùng nhau. Java phân biệt chữ hoa và chữ thường (case-sensitive). Ví dụ tên file ‘First’ và ‘first’ là hai file khác nhau.

Để biên dịch mã nguồn, ta sử dụng trình biên dịch **java**. Trình biên dịch xác định tên của file nguồn tại dòng lệnh như mô tả dưới đây:

C:\jdk1.2.1\bin>javac First.java

Trình dịch java tạo ra file First.class chứa các mã “bytecodes”. Những mã này chưa thể thực thi được. Để chương trình thực thi được ta cần dùng trình thông dịch “**java interpreter**”

Lệnh được thực hiện như sau:

C:\jdk1.2.1\bin>java First

Kết quả sẽ hiển thị trên màn hình như sau:

My first program in Java

Phân tích chương trình đầu tiên

```
// This is a simple program called " First.java "
```

Ký hiệu “// ” dùng để thuyết minh dòng lệnh. Trình biên dịch sẽ bỏ qua dòng thuyết minh này. Java còn hỗ trợ thuyết minh nhiều dòng. Loại thuyết minh này có thể bắt đầu với /* và kết thúc với */

```
/*This is a comment that extends to two lines*/ / *This is
a multi line comment */
```

Dòng kế tiếp khai báo lớp có tên ‘First’. Để tạo một lớp thêm ta bắt đầu với từ khoá ‘class’, kế đến là tên lớp (và cũng chính là tên file).

```
class First
```

Tên lớp nói chung nên bắt đầu bằng chữ in hoa.

Từ khoá ‘class’ khai báo định nghĩa lớp. ‘First’ là tên của lớp. Một định nghĩa lớp nằm trọn vẹn nằm giữa hai ngoặc móc mở ({) và đóng (}). Các ngoặc này đánh dấu bắt đầu và kết thúc một khối lệnh.

```
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. Chúng ta sẽ tìm hiểu từng từ trong lệnh 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. Trong trường hợp này, phương thức “**main**” được khai báo ‘public’, bởi vậy JVM có thể truy xuất phương thức này.

Từ khoá ‘**static**’ cho phép **main** được gọi tới mà không cần tạo ra một thể hiện (instance) của lớp. Nhưng trong trường hợp này, bản copy của phương thức **main** được phép tồn tại trên bộ nhớ, thậm chí không có một thể hiện của lớp đó được tạo ra. Điều này rất quan trọng vì JVM trước tiên gọi phương thức **main** để thực thi chương trình. Vì lý do này phương thức **main** cần phải là tĩnh (static). 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.

Phương thức ‘**main()**’ sẽ thực hiện một số tác vụ nào đó, nó là điểm mốc mà từ đó tất cả các ứng dụng Java được khởi động.

‘String args[]’ là tham số dùng trong phương thức ‘**main**’. Các biến số trong dấu ngoặc đơn nhận từng thông tin được chuyển vào ‘**main**’. Những biến này là các tham số của phương thức. Thậm chí ngay khi không có một thông tin nào được chuyển vào ‘**main**’, phương thức vẫn được thực hiện với các dữ liệu rỗng – không có gì trong dấu ngoặc đơn.

‘args[]’ là một mảng kiểu “String”. Các đối số (arguments) từ các dòng lệnh được lưu vào mảng. Mã nằm giữa dấu ngoặc móc ({ }) của ‘**main**’ được gọi là ‘**method block**’. Các lệnh được thực thi trong ‘**main**’ cần được viết trong khối này.

```
System.out.println(" My first program in Java ");
```

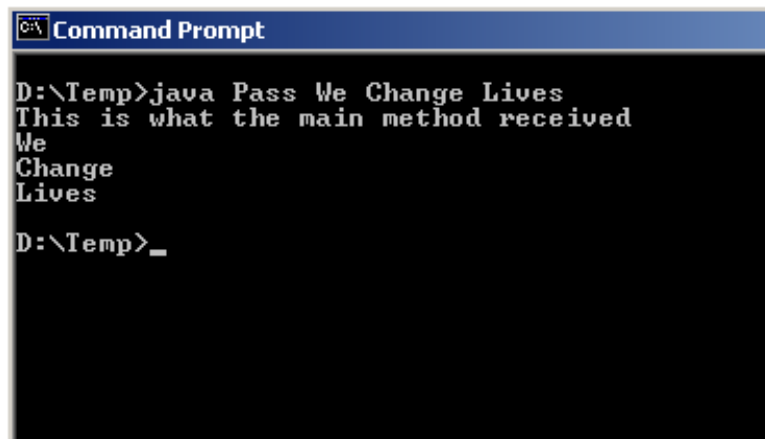
Dòng lệnh này hiển thị chuỗi “My first program in Java” trên màn hình. Phát biểu ‘**println()**’ tạo ra một cổng xuất (output). Phương thức này cho phép hiển thị chuỗi được truyền vào ra ‘System.out’. Ở đây ‘System’ là một lớp đã định trước, nó cho phép truy nhập vào hệ thống và ‘out’ là một chuỗi xuất được kết nối với đầu nhắc (console).

Truyền đối số trong dòng lệnh

Các mã sau đây cho ta thấy các tham số (argument) của các dòng lệnh được tiếp nhận như thế nào trong phương thức ‘**main**’.

```
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]); } }
```

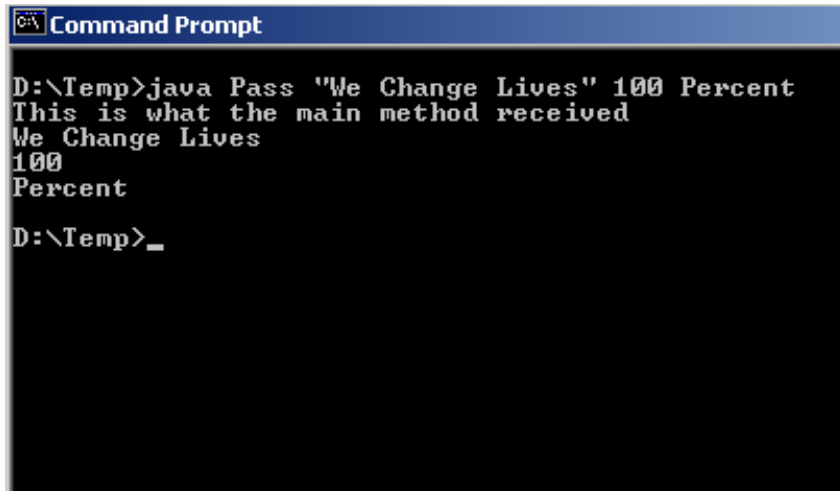
Hình vẽ dưới đây thể hiện sự thực thi của chương trình.



```
C:\> Command Prompt  
D:\Temp>java Pass We Change Lives  
This is what the main method received  
We  
Change  
Lives  
D:\Temp>_
```

Truyền tham số từ dòng lệnh

Khi gặp một dấu trắng (space), có thể hiểu một chuỗi được kết thúc. Nhưng thông thường một chuỗi được kết thúc khi gặp dấu nhảy kíp. Hình vẽ dưới đây sẽ mô tả điều này.



```
C:\> Command Prompt

D:\Temp>java Pass "We Change Lives" 100 Percent
This is what the main method received
We Change Lives
100
Percent
D:\Temp>_
```

Truyền tham số là chuỗi

Các lớp đối tượng trong Java

Trong ngôn ngữ Java, lớp là một đơn vị mẫu có chứa các số liệu và các mã liên quan đến một thực thể nào đó. Chúng hình thành nền tảng của toàn bộ ngôn ngữ Java. Dữ liệu hoặc mã nguồn được viết ra luôn đặt bên trong một lớp. Khi xác định một lớp, bạn thực chất xác định một kiểu dữ liệu. Loại dữ liệu mới này được sử dụng để xác định các biến mà ta thường gọi là “đối tượng”. Đối tượng là các thể hiện (instance) của lớp. Tất cả các đối tượng đều thuộc về một lớp có chung đặc tính và hành vi. Mỗi lớp xác định một thực thể, trong khi đó mỗi đối tượng là một thể hiện thực sự.

Bạn còn có thể định nghĩa một lớp bên trong một lớp khác. Đây là lớp xếp lồng nhau, các thể hiện (instance) của lớp này tồn tại bên trong thể hiện của một lớp che phủ chúng. Nó chi phối việc truy nhập đến các thành phần của thể hiện bao phủ chúng.

Khai báo lớp

Khi bạn khai báo một lớp, bạn cần xác định dữ liệu và các phương thức của lớp đó.

Cú pháp:

```
class classname { var_datatype variablename; :  
met_datatype methodname(parameter_list) : }
```

Trong đó:

class - Từ khoá xác định lớp

classname - Tên của lớp

var_datatype - kiểu dữ liệu của biến

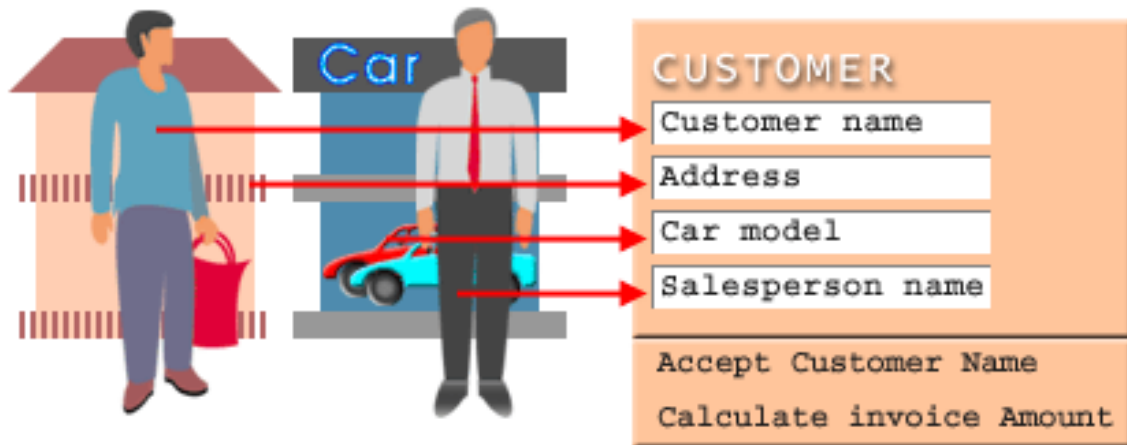
variablename - Tên của biến

met_datatype - Kiểu dữ liệu trả về của phương thức

methodname - Tên của phương thức

parameter_list – Các tham số được của phương thức

Hình 1 mô tả bằng hình ảnh lớp “Khách hàng”. Những đặc điểm của lớp xác định các khoản mục dữ liệu được lưu cất, và các hành vi xác định các phương thức được tính đến. Đối tượng của lớp này sẽ lưu lại các chi tiết cá nhân của khách hàng.



Lớp Khách hàng

Trong lớp “Khách hàng”, các khoản mục dữ liệu bao gồm:

- Tên khách hàng
- Địa chỉ
- Kiểu xe
- Tên người bán hàng

Các phương thức gồm:

- Nhập các chi tiết của khách hàng
- In các hoá đơn

Các lớp lồng nhau (nested classes)

Việc định nghĩa một lớp bên trong một lớp khác được gọi là lớp lồng (nesting). Lớp lồng chỉ nằm trong phạm vi lớp bao quanh nó. Có hai loại lớp lồng:

- Lớp kiểu tĩnh (static)

Lớp kiểu tĩnh được định nghĩa với từ khoá **static**. Lớp tĩnh có thể truy nhập vào các thành viên của lớp phủ nó thông qua một đối tượng. Do vậy lớp tĩnh thường ít được sử dụng.

- Lớp kiểu không tĩnh (non static)

Lớp bên trong (inner) quan trọng nhất của các lớp lồng. Đó là các lớp non-static. Định nghĩa lớp bên trong chỉ có thể xác định được trong phạm vi lớp ngoài cùng. Lớp bên trong có thể truy nhập tất cả các thành viên của lớp bao nó, song không thể ngược lại. Đoạn chương trình sau mô tả lớp được tạo lập ra sao và sử dụng như thế nào:

```
class Outer { //Outer class constructor class Inner {  
//Inner class constructor } }
```

Cú pháp sau đây cho phép truy nhập vào lớp bên trong

```
Outer.Inner obj=new Outer().new Inner();
```

Kiểu dữ liệu trong Java

Các ứng dụng luôn xử lý dữ liệu ở đầu vào và xuất dữ liệu kết quả ở đầu ra. Đầu vào, đầu ra, và kết quả của các quá trình tính toán đều liên quan đến dữ liệu. Trong môi trường tính toán, dữ liệu được phân lớp theo các tiêu chí khác nhau phụ thuộc vào bản chất của nó. Ở mỗi tiêu chí, dữ liệu có một tính chất xác định và có một kiểu thể hiện riêng biệt.

Java cung cấp một vài kiểu dữ liệu. Chúng được hỗ trợ trên tất cả các nền. Ví dụ, dữ liệu loại int (integer) của Java được thể hiện bằng 4 bytes trong bộ nhớ của tất cả các loại máy bất luận ở đâu chạy chương trình Java. Bởi vậy các chương trình Java không cần phải thay đổi khi chạy trên các nền khác nhau.

Trong Java kiểu dữ liệu được chia thành hai loại:

- Các kiểu dữ liệu nguyên thủy (primitive)
- Các kiểu dữ liệu tham chiếu (reference)

Dữ liệu kiểu nguyên thủy

Java cung cấp tám kiểu dữ liệu nguyên thủy

Kiểu dữ liệu nguyên thủy

Kiểu dữ liệu	Độ dài theo số bit	Phạm vi biểu diễn giá trị	Mô tả
byte	8	-128 đến 127	Số liệu kiểu byte là một loại điển hình dùng để lưu trữ một giá trị bằng một byte. Chúng được sử dụng rộng rãi khi xử lý một file văn bản
char	16	'\u0000' to '\uffff'	Kiểu Char sử dụng để lưu tên hoặc các dữ liệu ký tự. Ví dụ tên người lao động
boolean	1	"True" hoặc "False"	Dữ liệu boolean dùng để lưu các giá trị "Đúng" hoặc "sai" Ví dụ : Người lao động có đáp ứng được yêu cầu của công ty hay không ?

short	16	-32768 đến 32767	Kiểu short dùng để lưu các số có giá trị nhỏ dưới 32767. Ví dụ số lượng người lao động.
Int	32	-2,147,483,648 đến +2,147,483,648	Kiểu int dùng để lưu một số có giá trị lớn đến 2,147,483,648. Ví dụ tổng lương mà công ty phải trả cho nhân viên.
long	64	-9,223,372,036'854,775,808 đến +9,223,372,036'854,775,808	Kiểu long được sử dụng để lưu một số có giá trị rất lớn đến 9,223,372,036'854,775,808 . Ví dụ dân số của một nước
float	32	-3.40292347E+38 đến +3.40292347E+38	Kiểu float dùng để lưu các số thập phân đến 3.40292347E+38 Ví dụ : giá thành sản phẩm
double	64	-1,79769313486231570E+308 đến +1,79769313486231570E+308	Kiểu double dùng để lưu các số thập phân có giá trị lớn đến 1,79769313486231570E+308 Ví dụ giá trị tín dụng của ngân hàng nhà nước.

Kiểu dữ liệu tham chiếu (reference)

Trong Java có 3 kiểu dữ liệu tham chiếu

Kiểu dữ liệu tham chiếu

Kiểu dữ liệu	Mô tả
Mảng (Array)	Tập hợp các dữ liệu cùng kiểu. Ví dụ : tên sinh viên
Lớp (Class)	Tập hợp các biến và các phương thức. Ví dụ : lớp “Sinh viên” chứa toàn bộ các chi tiết của một sinh viên và các phương thức thực thi trên các chi tiết đó.
Giao diện (Interface)	Là một lớp trừu tượng được tạo ra cho phép cài đặt đa thừa kế trong Java.

Ép kiểu (Type casting)

Có thể bạn sẽ gặp tình huống khi cộng một biến có dạng **integer** với một biến có dạng **float**. Để xử lý tình huống này, Java sử dụng tính năng ép kiểu (type casting) của các phần mềm trước đó C, C++. Lúc này một kiểu dữ liệu sẽ chuyển đổi sang kiểu khác. Khi sử dụng tính chất này, bạn cần thận trọng vì khi điều chỉnh dữ liệu có thể bị sai giá trị.

Đoạn mã sau đây thực hiện phép cộng một giá trị dấu phẩy động (float) với một giá trị nguyên (integer).

```
float c=34.896751f; Int b = (int)c +10;
```

Đầu tiên giá trị dấu phẩy động **c** được đổi thành giá trị nguyên 34. Sau đó nó được cộng với 10 và kết quả là giá trị 44 được lưu vào **b**.

Sự nở rộng (widening) – quá trình làm tròn số theo hướng nở rộng không làm mất thông tin về độ lớn của mỗi giá trị. Biến đổi theo hướng nở rộng chuyển một giá trị sang một dạng khác có độ rộng phù hợp hơn so với nguyên bản. Biến đổi theo hướng lại thu nhỏ lại (narrowwing) làm mất thông tin về độ lớn của giá trị được chuyển đổi. Chúng không được thực hiện khi thực hiện phép gán. Ở ví dụ trên giá trị thập phân sau dấu phẩy sẽ bị mất.

Các biến trong Java

Các ứng dụng sử dụng các biến để lưu trữ các dữ liệu cần thiết hoặc các dữ liệu được tạo ra trong quá trình thực thi chương trình. Các biến được xác định bởi một tên biến và có một phạm vi tác động. Phạm vi tác động của biến được xác định một cách rõ ràng trong chương trình. Mỗi biến được khai báo trong một khối chương trình chỉ có tác động trong phạm vi khối đó, không có ý nghĩa và không được phép truy nhập từ bên ngoài khối.

Việc khai báo một biến bao gồm 3 thành phần: kiểu biến, tên của nó và giá trị ban đầu được gán cho biến (không bắt buộc). Để khai báo nhiều biến ta sử dụng dấu phẩy để phân cách các biến, Khi khai báo biến, luôn nhớ rằng Java phân biệt chữ thường và chữ in hoa (case -sensitive).

Cú pháp:

```
Datatype indentifier [=value] [, indentifier[=value]... ];
```

Để khai báo một biến nguyên (int) có tên là **counter** dùng để lưu giá trị ban đầu là 1, ta có thể thực hiện phát biểu sau đây:

```
int counter = 1;
```

Java có những yêu cầu hạn chế đặt tên biến mà bạn có thể gán giá trị vào. Những hạn chế này cũng giống các hạn chế khi đặt tên cho các định danh mà ta đã thảo luận ở các phần trước của chương này.

Khai báo mảng

Mảng được dùng để lưu trữ các khoản mục (items) cùng kiểu dữ liệu liên kế nhau trong bộ nhớ. Mỗi lần ta khai báo kích thước của một mảng, nó sẽ không thể thay đổi. Dữ liệu trên mảng có thể là kiểu dữ liệu nguyên thủy hoặc đối tượng. Cũng như các biến, ta có thể gán các giá trị vào mảng tại các phần tử được tạo ra trong mảng. Nếu không, Java sẽ gán giá trị mặc định vào tất cả các phần tử của mảng, giá trị mặc định phụ thuộc vào kiểu dữ liệu. Ví dụ : nếu kiểu dữ liệu là nguyên (int) thì giá trị mặc định ban đầu sẽ là 0.

Mảng có thể được khai báo bằng ba cách :

Khai báo mảng

Cách khai báo	Mô tả	Cú pháp	Ví dụ
---------------	-------	---------	-------

Chỉ đơn thuần khai báo	Chỉ đơn thuần khai báo mảng	Datatype identifier[]	char ch[]; khai báo mảng ký tự có tên ch
Khai báo và tạo mảng	Khai báo và cấp phát bộ nhớ cho các phần tử mảng sử dụng toán tử “new”	Datatype identifier[]=new datatype [size]	char ch[] = newchar [10]; Khai báo một mảng ch và lưu trữ 10 ký tự
Khai báo,kiến tạo và khởi tạo	Khai báo mảng,cấp phát bộ nhớ cho nó và gán các giá trị ban đầu cho các phần tử của mảng	Datatype identifier[]= {value1,value2...valueN };	char ch []= {‘A’,’B’,’C’,’D’ };khai báo mảng ch và lưu 4 chữ cái kiểu ký tự

Để xác định tên và số phần tử của mảng ta cần xem xét các phần tử mảng.Số phần tử bắt đầu với 0 cho phần tử đầu,1 cho phần tử thứ hai và cứ tiếp như vậy.

Phương thức trong một lớp (Method)

Phương thức xác định giao diện cho phần lớn các lớp. Trong khi đó Java cho phép bạn định nghĩa các lớp mà không cần phương thức. Bạn cần định nghĩa phương thức truy cập dữ liệu mà bạn đã lưu trong một lớp.

Phương thức được định nghĩa như một hành động hoặc một tác vụ thật sự của đối tượng. Nó còn được định nghĩa như một hành vi mà trên đó các thao tác cần thiết được thực thi.

Cú pháp :

```
access_specifier modifier datatype  
method_name(parameter_list) { //body of method }
```

Trong đó:

access_specifier: Chỉ định truy cập vào phương thức.

modifier: Cho phép bạn đặt thuộc tính cho phương thức.

datatype: Kiểu dữ liệu mà phương thức trả về. Nếu không có một giá trị nào được trả về, kiểu dữ liệu có thể là void.

method_name: Tên của phương thức

parameter_list: Chứa tên của tham số được sử dụng trong phương thức và kiểu dữ liệu. Dấu phẩy được dùng để phân cách các tham số.

Khai báo phương thức trong một lớp :

Đoạn mã sau đây định nghĩa lớp **Temp** chứa một giá trị nguyên (**int**). Lớp này chứa hai phương thức là: **show()** và **main()**. Cả hai phương thức đều có khả năng truy cập bên ngoài lớp khi chúng được khai báo như **public**. Nếu chúng không trả về một giá trị nào, kiểu dữ liệu trả về được định nghĩa như kiểu **void**.

Phương thức **show()** hiển thị một giá trị của biến **x**. Ở phương thức **main()**, hai thí dụ của đối tượng thuộc lớp **Temp** được khai báo. Đối tượng thứ nhất gồm giá trị mặc định của biến **x**. Nó được hiển thị ngay khi gọi phương thức **show()** lần đầu tiên. Giá trị của **x** được thay đổi dùng cho đối tượng thứ hai. Nó được hiển thị khi ta gọi phương thức **show()** lần thứ hai.

```
class Temp { static int x=10;//variable public static void
show();//method { System.out.println(x);} public static
void main(String args[]) { Temp t = new Temp();// object 1
t.show();//method call Temp t1=new Temp();// object 2
t1.x=20; t1.show(); } }
```

Các chỉ định truy xuất của phương thức

Các chỉ định truy xuất dùng để giới hạn khả năng truy nhập vào một phương thức. Java cung cấp các chỉ định truy xuất sau đây:

- **Công cộng (Public):** Phương thức có chỉ định truy xuất public có thể được nhìn thấy từ mọi gói hoặc mọi lớp.
- **Bảo vệ (Protected):** Các lớp mở rộng từ lớp hiện hành trong cùng một gói, hoặc tại các gói khác nhau có thể truy cập các phương thức loại này.
- **Riêng tư (Private):** Phương thức riêng tư chỉ có thể được truy cập nhờ phương thức công cộng trong cùng một lớp.

Các bổ nghĩa phương thức

Các bổ nghĩa phương thức cho phép ta thiết lập các thuộc tính của phương thức. Java cung cấp các bổ nghĩa sau:

- **Tĩnh (static):** phương thức có thể được gọi mà không cần đến đối tượng. Nó chỉ được sử dụng đối với các dữ liệu và các phương thức tĩnh khác.
- **Trừu tượng (abstract):** Ngụ ý rằng phương thức không có một mã (code) và nó sẽ được bổ sung ở các lớp con (subclass). Loại phương thức này được sử dụng trong các lớp kế thừa.
- **Kết thúc (final):** Phương thức không thể được thừa kế hoặc ghi đè (Overridden).
- **Tự nhiên (native):** Chỉ ra rằng phần thân của phương thức được viết trên các ngôn ngữ khác Java ví dụ C, hoặc C++.
- **Đồng bộ (synchronized):** Sử dụng với phương thức trong quá trình thực thi threads. Nó cho phép chỉ một thread được truy cập vào khối mã tại một thời điểm.
- **Linh hoạt (volatile):** Được sử dụng với các biến để thông báo rằng giá trị của biến có thể được thay đổi vài lần khi thực thi chương trình và giá trị của nó không được đặt vào thanh ghi.

Bảng dưới đây chỉ ra nơi mà các bổ nghĩa được sử dụng:

Sử dụng các bổ nghĩa

Bổ nghĩa	Phương thức	Biến	Lớp
public	Yes	Yes	Yes
private	Yes	Yes	Yes (Nested class)
protected	Yes	Yes	Yes (Nested class)
abstrac	Yes	No	Yes
final	Yes	Yes	Yes
native	Yes	No	No
volatile	No	Yes	No

Nạp chồng (overloading) và Ghi đè (overriding) phương thức

Những phương thức được nạp chồng (**overload**) là những phương thức trong cùng một lớp, có cùng một tên song có danh sách các tham số khác nhau. Sử dụng việc nạp chồng phương thức để thực thi các phương thức giống nhau đối với các kiểu dữ liệu khác nhau. Ví dụ phương thức **swap()** có thể bị nạp chồng (**overload**) bởi các tham số của kiểu dữ liệu khác như **integer**, **double** và **float**

Phương thức được ghi đè (**overriden**) là phương thức có mặt ở lớp cha (**superclasses**) cũng như ở các lớp kế thừa. Phương thức này cho phép một lớp tổng quát chỉ định các phương thức sẽ là phương thức chung trong các lớp con. Ví dụ lớp xác định phương thức tổng quát `area()`. Phương thức này có thể được hiện thực trong một lớp con để tìm diện tích một hình cụ thể như hình chữ nhật, hình vuông ...

Phương thức nạp chồng là một hình thức đa hình (**polymorphism**) trong quá trình biên dịch (**compile**). Còn phương thức ghi đè là một hình thức đa hình trong quá trình thực thi (**runtime**).

Đoạn chương trình sau mô tả nạp chồng phương thức được thực hiện như thế nào

```
//defined once
protected void performTask(double salary){
.... System.out.println("Salary is : " + salary); .... }
//overloaded-defined the second time with different
parameters
protected void performTask(double salary, int
bonus){ ... System.out.println("Total Salary is:" +
salary+bonus); ... }
```

Phương thức khởi tạo (**Constructor**) của lớp có thể bị nạp chồng (**overload**)

Phương thức ghi đè (**Overriden**) được định nghĩa lại ở các lớp con. Đoạn mã sau đây mô tả phương thức ghi đè.

Ở đây ta dùng từ khoá “**this**” biểu thị đối tượng hiện hành, trong khi đó ‘**super**’ được sử dụng để chỉ đối tượng lớp cha.

Phương thức ghi đè không phải là phương thức tĩnh (**static**). Nó là loại non-static.

Các đoạn mã sau đây mô tả việc thực thi ghi đè phương thức trong Java.

```
class SupperClass // Tạo lớp cơ bản { int a; SupperClass()  
// constuctor { } SupperClass(int b) //overloaded  
constructor { a=b; } public void message() {  
System.out.println("In the super class" ); } } class  
SubClass Extends SupperClass { // derriving a class int a;  
SubClass(int a) { //subclass constructor this.a; } public  
void message(){ // overiding the base class message()  
System.out.println("In the sub class"); } }
```

Bây giờ chúng ta sẽ tạo ra một đối tượng lớp cha và gán một lớp nhỏ tham chiếu đến nó như sau:

```
SuperClassss spObj=new SubClass(22);
```

Câu lệnh ‘spObj.message()’ thuộc phương thức lớp của SubClass. Ở đây kiểu đối tượng được gán cho ‘spObj’ sẽ chỉ được xác định khi chương trình thực thi. Điều này được biết dưới khái niệm ‘liên kết động’ (**dynamic binding**).

Phương thức khởi tạo lớp

Phương thức khởi tạo lớp là một loại phương thức đặc biệt rất khác với các phương thức thông thường. Nó không có kiểu trả về. Nó có tên trùng với tên của lớp. Hàm khởi tạo lớp thực thi như một phương thức hoặc một chức năng bình thường song nó không trả về bất cứ một giá trị nào. Nói chung chúng được dùng để khởi tạo các biến thành viên của một lớp và nó được gọi mỗi khi bạn tạo ra đối tượng của lớp đó.

Phương thức khởi tạo lớp có hai loại:

- Tường minh (**explicit**): Bạn có thể lập trình những phương thức khởi tạo lớp khi định nghĩa lớp. Khi tạo một đối tượng của một lớp, những giá trị mà bạn truyền vào phải khớp với những tham số của phương thức khởi tạo (số lượng, thứ tự và kiểu dữ liệu của các tham số)

- Ngầm định (**Implicit**): Khi bạn không định nghĩa một phương thức khởi tạo cho một lớp, JVM cung cấp một phương thức khởi tạo ngầm định.

Bạn có thể định nghĩa nhiều phương thức khởi tạo cho một lớp. Giống như các phương thức khác, phương thức khởi tạo lớp có thể bị nạp chồng (**overload**)

Một phương thức khởi tạo:

Đoạn mã sau đây định nghĩa một phương thức khởi tạo tường minh (**explicit**) cho một lớp Employee. Phương thức khởi tạo bao gồm tên và tuổi. Chúng được coi như các tham số và gán các giá trị của chúng vào các biến của lớp. Chú ý rằng từ khoá '**this**' được sử dụng để tham chiếu đến đối tượng hiện hành của lớp.

```
class Employee { String name; int age; Employee (String varname, int varage) { this.name = varname; this.age = varage; } public static void main (String arg[]) { Employee e = new Employee ("Allen", 30); } }
```

Phương thức khởi tạo của lớp dẫn xuất

Phương thức khởi tạo của một lớp dẫn xuất có tên trùng với tên của lớp dẫn xuất đó. Câu lệnh dùng để gọi phương thức khởi tạo của cha phải là câu lệnh đầu tiên trên phương thức khởi tạo của lớp con đó. Lý do là lớp cha hình thành trước khi có các lớp con.

Các toán tử

Một chương trình thực tế bao hàm việc tạo ra các biến. Các toán tử kết hợp các giá trị đơn hoặc các biểu thức con thành những biểu thức mới, phức tạp hơn và có thể trả về giá trị. Điều này liên quan đến việc thực hiện các phép toán logic, số học, quan hệ và so sánh trên các biểu thức.

Java cung cấp nhiều dạng toán tử sau:

- Toán tử số học
- Toán tử bit
- Toán tử quan hệ
- Toán tử logic
- Toán tử điều kiện
- Toán tử gán

Các toán tử số học

Các toán hạng của các toán tử số học phải ở dạng số. Các toán hạng kiểu boolean không sử dụng được, song các toán hạng ký tự cho phép sử dụng loại toán tử này. Một vài kiểu toán tử được liệt kê trong bảng dưới đây.

Các toán tử số học

Toán tử	Mô tả
+	Cộng. Trả về giá trị tổng hai toán hạng. Ví dụ $5+3$ trả về kết quả là 8
-	Trừ. Trả về giá trị khác nhau giữa hai toán hạng hoặc giá trị phủ định của toán hạng. Ví dụ $5-3$ kết quả là 2 và -10 trả về giá trị âm của 10
*	Nhân. Trả về giá trị là tích hai toán hạng. Ví dụ $5*3$ kết quả là 15
/	Chia. Trả về giá trị là thương của phép chia. Ví dụ $6/3$ kết quả là 2
%	Phép lấy modulo. Giá trị trả về là phần dư của phép chia. Ví dụ $10\%3$ giá trị trả về là 1
++	Tăng dần. Tăng giá trị của biến lên 1. Ví dụ $a++$ tương đương với $a = a+1$
--	Giảm dần. Giảm giá trị của biến 1 đơn vị. Ví dụ $a--$ tương đương với $a = a-1$
+=	Cộng và gán giá trị. Cộng các giá trị của toán hạng bên trái vào toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ $c+=a$ tương đương $c=c+a$

-=	Trừ và gán giá trị Trừ các giá trị của toán hạng bên trái vào toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ <code>c-= a</code> tương đương với <code>c=c-a</code>
*=	Nhân và gán Nhân các giá trị của toán hạng bên trái với toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ <code>c *= a</code> tương đương với <code>c=c*a</code>
/=	Chia và gán Chia giá trị của toán hạng bên trái cho toán hạng bên phải và gán giá trị trả về vào toán hạng bên trái. Ví dụ <code>c /= a</code> tương đương với <code>c=c/a</code>
%=	Lấy số dư và gán Chia giá trị của toán hạng bên trái cho toán hạng bên phải và gán giá trị số dư vào toán hạng bên trái. Ví dụ <code>c%=a</code> tương đương với <code>c=c%a</code>

Chương trình sau mô tả việc sử dụng toán tử số học

```
class ArithmeticOp { public static void main(String
args[]){ int p=5,q=12,r=20,s; s=p+q;
System.out.println("p+q is "+s); s=p%q;
System.out.println("p%q is "+s); s*=r;
System.out.println("s*=r is "+s);
System.out.println("Value of p before operation is "+p);
p++; System.out.println("Value of p after operation is
"+p); double x=25.75,y=14.25,z; z=x-y; System
.out.println("x-y is " +z); z-=2.50;
System.out.println("z-=2.50 is "+z);
System.out.println("Value of z before operation is"+z); z-
-; System.out.println("Value of z after operation is"+z);
z=x/y; System.out.println("x/y is" +z); } }
```

Đầu ra của chương trình là

```
p+q is 17 p%q is 5 s*=r is 100 Value of p before operation
is 9.0 Value of z after operation is 8.0 x/y is
1.8070175438596429
```

Toán tử Bit

Các toán tử dạng Bit cho phép ta thao tác trên từng Bit riêng biệt trong các kiểu dữ liệu nguyên thủy. Toán tử Bit dựa trên cơ sở đại số Boolean. Nó thực hiện phép tính trên hai bit có vị trí tương ứng trên hai toán hạng để tạo ra một kết quả mới. Một vài dạng toán tử kiểu này được liệt kê dưới đây

Các toán tử Bit

Toán tử	Mô tả
~	Phủ định bit (NOT)Trả về giá trị âm của một số. Ví dụ $a=10$ thì $\sim a=-10$
&	Toán tử ANDTrả về giá trị là 1 nếu các toán hạng là 1 và 0 trong các trường hợp khác. Ví dụ nếu $a=1$ và $b=0$ thì $a\&b$ trả về giá trị 0
	Toán tử ORTrả về giá trị là 1 nếu một trong các toán hạng là 1 và 0 trong các trường hợp khác. Ví dụ nếu $a=1$ và $b=0$ thì $a b$ trả về giá trị 1
^	Exclusive ORTrả về giá trị là 1 nếu chỉ một trong các toán hạng là 1 và trả về 0 trong các trường hợp khác. Ví dụ nếu $a=1$ và $b=1$ thì $a\wedge b$ trả về giá trị 0
>>	Dịch sang phảiChuyển toàn bộ các bit của một số sang phải một vị trí, giữ nguyên dấu của số âm. Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bit cần dịch. Ví dụ $x=31$ tức là 00011111 vậy $x>>2$ sẽ là 00000111.
<<	Dịch sang tráiChuyển toàn bộ các bit của một số sang trái một vị trí, giữ nguyên dấu của số âm. Toán hạng bên trái là số bị dịch còn số bên phải chỉ số vị trí mà các bit cần dịch.

Các toán tử quan hệ

Các toán tử quan hệ kiểm tra mối quan hệ giữa hai toán hạng. Kết quả của một biểu thức có dùng các toán tử quan hệ là những giá trị Boolean (logic “đúng” hoặc “sai”). Các toán tử quan hệ được sử dụng trong các cấu trúc điều khiển.

Các toán tử quan hệ

Toán tử	Mô tả
==	So sánh bằngToán tử này kiểm tra sự tương đương của hai toán hạngVí dụ if (a==b) trả về giá trị “True” nếu giá trị của a và b như nhau
!=	So sánh khácKiểm tra sự khác nhau của hai toán hạngVí dụ if(a!=b) Trả về giá trị “true” nếu a khác b
>	Lớn hơnKiểm tra giá trị của toán hạng bên phải lớn hơn toán hạng bên trái hay khôngVí dụ if(a>b) . Trả về giá trị “true” nếu a lớn hơn b, ngược lại (nhỏ hơn hoặc bằng), trả về ‘False’

<	Nhỏ hơn Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn toán hạng bên trái hay không Ví dụ <code>if(a<b)</code> . Trả về giá trị “true” nếu a nhỏ hơn b , ngược lại (lớn hơn hoặc bằng trả về ‘False’)
>=	Lớn hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có lớn hơn hoặc bằng toán hạng bên trái hay không Ví dụ <code>if(a>=b)</code> . Trả về giá trị “true” nếu a lớn hơn hoặc bằng b , ngược lại (nhỏ hơn trả về ‘False’)
<=	Nhỏ hơn hoặc bằng Kiểm tra giá trị của toán hạng bên phải có nhỏ hơn hoặc bằng toán hạng bên trái hay không Ví dụ <code>if(a<=b)</code> . Trả về giá trị “true” nếu a nhỏ hơn hoặc bằng b , ngược lại (lớn hơn trả về ‘false’)

Đoạn chương trình sau đây mô tả việc sử dụng các toán tử quan hệ

```
class RelationalOp { public static void main (String
args[]){ float a= 10.0f; double b=10.0; if (a== b)
System.out.println("a and b are equal"); else
System.out.println("a and b are not equal"); } }
```

Kết quả chương trình sẽ hiển thị

a and b are not equal

Trong chương trình trên cả a và b là những số có dấu phẩy động, dạng dữ liệu có khác nhau, a là kiểu float còn b là kiểu double. Tuy vậy chúng không phải là cùng một kiểu. Bởi vậy khi kiểm tra giá trị của các toán hạng, kiểu dữ liệu cần phải được kiểm tra.

Các toán tử logic

Các toán tử logic làm việc với các toán hạng Boolean. Một vài toán tử kiểu này được chỉ ra dưới đây

Các toán tử Logic

Toán tử	Mô tả
&	Và (AND)Trả về một giá trị “Đúng” (True) nếu chỉ khi cả hai toán tử có giá trị “True”Ví dụ: <code>if(science>90) AND (math>75)</code> thì gán “Y” cho biến “được nhận học bổng”
I	Hoặc (OR)Trả về giá trị “True” nếu một giá trị là True hoặc cả hai đều là TrueVí dụ Nếu <code>age_category</code> is ‘Senior_citizen’ OR <code>special_category</code> is ‘handicapped’ thì giảm giá tua lữ hành. Giá cũng sẽ được giảm nếu cả hai điều kiện đều được thỏa mãn

^	XORTrả về giá trị True nếu chỉ một trong các giá trị là True, các trường hợp còn lại cho giá trị False (sai)
!	Toán hạng đơn tử NOT. Chuyển giá trị từ True sang False và ngược lại. Ví dụ: Quá trình thực thi các dòng lệnh tiếp tục cho đến khi kết thúc chương trình.

Các toán tử điều kiện

Toán tử điều kiện là một loại toán tử đặc biệt vì nó gồm ba thành phần cấu thành biểu thức điều kiện

Cú pháp :

biểu thức 1?biểu thức 2: biểu thức 3;

biểu thức 1 Biểu thức logic. Trả trả về giá trị True hoặc False

biểu thức 2 Giá trị trả về nếu biểu thức 1 xác định là True

biểu thức 3 Giá trị trả về nếu biểu thức 1 xác định là False

Câu lệnh sau đây kiểm tra có những người đi làm bằng vé tháng có tuổi lớn hơn 65 không và gán một tiêu chuẩn cho họ. Nếu những người này có tuổi là 55, tiêu chuẩn gán là “**Regular**”

```
CommuterCategory=(CommuterAge>65)?"Senior Citizen":
"Regular"
```

Toán tử gán

Toán tử gán (=) dùng để gán một giá trị vào một biến. Bạn có thể gán nhiều giá trị đến nhiều biến cùng một lúc.

Ví dụ đoạn lệnh sau gán một giá trị cho biến **num**. Thì giá trị trong biến **num** được gán cho nhiều biến trên một dòng lệnh đơn.

```
int num = 20000; int p,q,r,s; p=q=r=s=num;
```

Dòng lệnh cuối cùng được thực hiện từ phải qua trái. Đầu tiên giá trị ở biến num được gán cho ‘s’, sau đó giá trị của ‘s’ được gán cho ‘r’ và cứ tiếp như vậy.

Thứ tự ưu tiên của các toán tử

Các biểu thức được viết ra nói chung gồm nhiều toán tử. Thứ tự ưu tiên quyết định trật tự thực hiện các toán tử trên các biểu thức. Bảng dưới đây liệt kê thứ tự thực hiện các toán tử trong Java

Thứ tự ưu tiên

Thứ tự	Toán tử
1.	Các toán tử đơn như +, -, ++, --
2.	Các toán tử số học và các toán tử dịch như *, /, +, -, <<, >>
3.	Các toán tử quan hệ như >, <, >=, <=, =, !=
4.	Các toán tử logic và Bit như &&, , &, , ^
5.	Các toán tử gán như =, *=, /=, +=, -=

Thay đổi thứ tự ưu tiên

Để thay đổi thứ tự ưu tiên trên một biểu thức, bạn có thể sử dụng dấu ngoặc đơn (). Từng phần của biểu thức được giới hạn trong ngoặc đơn được thực hiện trước tiên. Nếu bạn sử dụng nhiều ngoặc đơn lồng nhau thì toán tử nằm trong ngoặc đơn phía trong sẽ thực thi trước, sau đó đến các vòng phía ngoài. Nhưng trong phạm vi một ngoặc đơn thì quy tắc thứ tự ưu tiên vẫn giữ nguyên tác dụng.

Định dạng dữ liệu xuất và điều khiển luồng

Định dạng dữ liệu xuất dùng chuỗi thoát (Escape sequence)

Nhiều khi dữ liệu xuất được hiển thị trên màn hình, chúng cần phải được định dạng. Việc định dạng này cần sự trợ giúp của chuỗi thoát (Escape sequences) do Java cung cấp

Chúng ta hãy xem ví dụ dưới đây

```
System.out.println("Happy\tBirthday");
```

Cho ta dữ liệu xuất như sau : **Happy Birthday**

Bảng dưới đây liệt kê một số chuỗi thoát và công dụng của chúng

Các chuỗi thoát

Chuỗi thoát	Mô tả
\n	Đưa con trỏ đến dòng kế tiếp (Bắt đầu một dòng mới)
\r	Đưa con trỏ về đầu dòng (Giống ký tự carriage return)
\t	Đưa con trỏ đến vị trí Tab-Stop (Như vị trí Tab của ký tự)
\\	In vạch chéo ngược (backslash)
\'	In dấu nháy đơn (')
\"	In dấu nháy kép (")

Điều khiển luồng

Tất cả các môi trường phát triển ứng dụng đều cung cấp một quy trình ra quyết định (decision-making) được gọi là điều khiển luồng, nó điều khiển việc thực thi ứng dụng. Điều khiển luồng cho phép người phát triển phần mềm kiểm tra sự tồn tại của một điều kiện nào đó và ra quyết định phù hợp với điều kiện đó.

Vòng lặp là một cấu trúc chương trình giúp bạn có thể dùng để thực hiện việc lặp lại các hành động khi thực thi chương trình mà không cần viết lại các đoạn chương trình nhiều lần.

Điều khiển rẽ nhánh

- Mệnh đề if-else

- Mệnh đề switch-case

Vòng lặp (Loops)

- Vòng lặp while
- Vòng lặp do-while
- Vòng lặp for

Câu lệnh if-else

Câu lệnh if-else kiểm tra kết quả của một điều kiện và thực thi một thao tác phù hợp trên cơ sở kết quả đó. Dạng của câu lệnh if-else rất đơn giản

Cú pháp

```
If (condition) { action 1 statements; } else { action 2 statements; }
```

Condition: Biểu thức Boolean như toán tử so sánh. Biểu thức này trả về giá trị True hoặc False

action 1: Các dòng lệnh được thực thi khi giá trị trả về là True

else: Từ khoá xác định các câu lệnh tiếp sau được thực hiện nếu điều kiện trả về giá trị False

action 2: Các câu lệnh được thực thi nếu điều kiện trả về giá trị False

Đoạn chương trình sau kiểm tra xem các số là chẵn hay lẻ và hiển thị thông báo phù hợp

```
class CheckNumber { public static void main(String args[])
{ int num =10; if(num %2 == 0) System.out.println (num+
"is an even number"); else System.out.println (num +"is an
odd number"); }}
```

Ở đoạn chương trình trên num được gán giá trị nguyên là 10. Trong câu lệnh **if-else** điều kiện **num %2** trả về giá trị 0 và điều kiện thực hiện là True. Thông báo “10 is an even number” được in ra. Lưu ý rằng cho đến giờ chỉ có một câu lệnh tác động được viết trong đoạn “if” và “else”, bởi vậy không cần phải được đưa vào dấu ngoặc móc.

Hình vẽ dưới đây mô tả cách dùng **if-else**

Tên		Tom	Else-if	John	else	Henry
Điều kiện	if	Giám đốc		> 3 năm kinh nghiệm		
Tăng lương		✓		✓		✗

If - else

Câu lệnh switch-case

Phát biểu switch-case có thể được sử dụng tại câu lệnh if-else. Nó được sử dụng trong tình huống một biểu thức cho ra nhiều kết quả. Việc sử dụng câu lệnh switch-case cho phép việc lập trình dễ dàng và đơn giản hơn.

Cú pháp

```
switch (expression) { case 'value':action 1 statement;
break; case 'value':action 2 statement; break; : : case
'valueN': actionN statement; break; default:
default_action statement; }
```

expression - Biến chứa một giá trị xác định

value1,value 2,...valueN: Các giá trị hằng số phù hợp với giá trị trên biến expression .

action1,action2...actionN: Các phát biểu được thực thi khi một trường hợp tương ứng có giá trị True

break: Từ khoá được sử dụng để bỏ qua tất cả các câu lệnh sau đó và giành quyền điều khiển cho cấu trúc bên ngoài switch

default: Từ khóa tùy chọn được sử dụng để chỉ rõ các câu lệnh nào được thực hiện chỉ khi tất cả các trường hợp nhận giá trị False

default - action: Các câu lệnh được thực hiện chỉ khi tất cả các trường hợp nhận giá trị False

Đoạn chương trình sau xác định giá trị trong một biến nguyên và hiển thị ngày trong tuần được thể hiện dưới dạng chuỗi. Để kiểm tra các giá trị nằm trong khoảng 0 đến 6

chương trình sẽ thông báo lỗi nếu nằm ngoài phạm vi trên.

```
class SwitchDemo { public static void main(String args[])
{ int day =4; switch(day) { case 0 :
System.out.println("Sunday"); break; case 1 :
```

```

System.out.println("Monday"); break; case 2 :
System.out.println("Tuesday"); break; case 3 :
System.out.println("Wednesday"); break; case 4 :
System.out.println("Thursday"); break; case
5: System.out.println("Friday"); break; case 6
: System.out.println("Saturday"); break; default:
System.out.println("Invalid day of week"); } } }

```

Nếu giá trị của biến day là 4 ,chương trình sẽ hiển thị **Thursday**,và cứ tiếp như vậy .

Vòng lặp While

Vòng lặp **while** được sử dụng khi vòng lặp được thực hiện mãi cho đến khi điều kiện thực thi vẫn là True. Số lượng lần lặp không được xác định trước song nó sẽ phụ thuộc vào từng điều kiện.

Cú pháp

```

while(condition) { action statement; : : }

```

condition: Biểu thức Boolean, nó trả về giá trị True hoặc False. Vòng lặp sẽ tiếp tục cho đến khi nào giá trị True được trả về.

action statement: Các câu lệnh được thực hiện nếu condition nhận giá trị True

Đoạn chương trình sau tính giai thừa của số 5. Giai thừa được tính như tích $5*4*3*2*1$.

```

class WhileDemo { public static void main(String args[]) {
int a = 5,fact = 1; while (a >= 1) { fact *=a; a--; }
System.out.println("The Factorial of 5 is "+fact); } }

```

Ở ví dụ trên, vòng lặp được thực thi cho đến khi điều kiện $a \geq 1$ là **True**. Biến **a** được khai báo bên ngoài vòng lặp và được gán giá trị là 5. Cuối mỗi vòng lặp, giá trị của **a** giảm đi 1. Sau năm vòng giá trị của **a** bằng 0. Điều kiện trả về giá trị False và vòng lặp kết thúc. Kết quả sẽ được hiển thị “ **The factorial of 5 is 120**”

Đoạn chương trình sau hiển thị tổng của 5 số chẵn đầu tiên

```

class ForDemo { public static void main(String args[]) {
int i=1,sum=0; for (i=1;i<=10;i+=2) sum+=i;
System.out.println ("sum of first five old numbers is
"+sum); } }

```

Ở ví dụ trên, *i* và *sum* là hai biến được gán các giá trị đầu là 1 và 0 tương ứng. Điều kiện được kiểm tra và khi nó còn nhận giá trị *True*, câu lệnh tác động trong vòng lặp được thực hiện. Tiếp theo giá trị của *i* được tăng lên 2 để tạo ra số chẵn tiếp theo. Một lần nữa, điều kiện lại được kiểm tra và câu lệnh tác động lại được thực hiện. Sau năm vòng, *i* tăng lên 11, điều kiện trả về giá trị *False* và vòng lặp kết thúc. Thông báo: **Sum of first five odd numbers is 25** được hiển thị.

Tóm tắt bài học

- Lệnh **import** được sử dụng trong chương trình để truy cập các gói thư viện Java.
- Chương trình Java chứa một tập các gói. Chương trình có thể chứa các dòng giải thích. Trình biên dịch sẽ bỏ qua các dòng giải thích này.
- “Token” là thành phần nhỏ nhất của chương trình. Có năm loại “token”
 - Định danh (identifiers)
 - Từ khóa (keywords)
 - Ký tự phân cách (separators)
 - Nguyên dạng (Literals)
 - Các toán tử
- Java có các kiểu dữ liệu xây dựng sẵn gọi là kiểu dữ liệu nguyên thủy. Các biến được khai báo với một dữ liệu xác định. Hãy thận trọng khi khai báo tên biến để loại trừ khả năng nhầm lẫn.
- Java cung cấp các chỉ định truy xuất sau đây :
 - Công cộng (public)
 - Bảo vệ (protected)
 - Riêng tư (private)
- Java cung cấp các bổ nghĩa (modifiers) sau đây:
 - Tĩnh (static)
 - Trừu tượng (abstract)
 - final
- Khởi tạo hàm có hai kiểu :
 - Tường minh (Explicit)
 - Ngầm định (Implicit)
- Java cung cấp nhiều dạng toán tử:
 - Các toán tử số học
 - Các toán tử bit
 - Các toán tử quan hệ
 - Các toán tử logic
 - Toán tử điều kiện
 - Toán tử gán
- Ứng dụng Java có một lớp chứa phương thức *main*. Các tham số có thể được truyền vào phương thức *main* nhờ các tham số lệnh (command line parameters). Java cung cấp những cấu trúc sau:

- if-else
- switch
- for
- while
- do while

Kiểm tra sự tiến bộ

1. Trong Java, kiểu dữ liệu dạng byte nằm trong giới hạn từ.....đến.....
2. Hãy chỉ các danh định hợp lệ:
 1. Tel_num
 2. Empl
 3. 8678
 4. batch.no
3. Cho biết kết quả thu được khi thực hiện đoạn chương trình sau?


```
class me
{
    public static void main(String srgs[ ]) {
        int sales=820; int profit=200;
        System.out.println((sale+profit)/10*5);
    }
}
```
4.là sự cài đặt của các hành động của đối tượng
5. Phương thức **public** có thể truy cập phương thức **private** trong cùng một lớp. **(Đúng/Sai)**
6. '**static**' hàm ý rằng phương thức không có mã và được bổ sung trong các lớp con **(Đúng/Sai)**
7. Khi bạn không định nghĩa một hàm khởi tạo cho một lớp, JVM sẽ cung cấp một hàm mặc định hoặc một hàm khởi tạo ẩn (implicit). **(Đúng/Sai)**
8. Vòng lặp while thực thi ít nhất một lần thậm chí nếu điều kiện được xác định là False **(Đúng/Sai)**

Bài tập

Hãy viết một đoạn chương trình để in ra dòng chữ ” Welcome to the world of Java”

Hãy viết hai phương thức khởi tạo tường minh cho một lớp dùng để tính diện tích hình chữ nhật. Khi một giá trị được truyền vào phương thức khởi tạo, nó cho rằng độ dài và chiều rộng bằng nhau và bằng giá trị truyền vào. Lúc đó, nó sẽ tính diện tích tương ứng. Khi hai giá trị được truyền vào, nó sẽ tính diện tích hình chữ nhật.

Viết một chương trình thực hiện những việc sau đây:

- Khai báo và gán giá trị đầu cho các biến m và n là 100 và 200 tương ứng.
- Theo các điều kiện: nếu m bằng 0, hiển thị kết quả tương ứng.
- Nếu m lớn hơn n , hiển thị kết quả tương ứng.
- Kiểm tra giá trị n là chẵn hay lẻ.

Viết một chương trình hiển thị tổng các bội số của 7 nằm giữa 1 và 100.

Viết chương trình để cộng bảy số hạng của dãy sau:

$$1!+2!+3!+\dots\dots\dots$$

Các gói và giao diện

Giới thiệu

Gói và giao diện là hai thành phần chính của chương trình Java. Các gói được lưu trữ theo kiểu phân cấp, và được nhập (import) một cách tường minh vào những lớp mới được định nghĩa. Các giao diện có thể được sử dụng để chỉ định một tập các phương thức. Các phương thức này có thể được hiện thực bởi một hay nhiều lớp.

Một tập tin nguồn Java có thể chứa một hoặc tất cả bốn phần sau đây:

- Một câu lệnh khai báo gói (package).
- Những câu lệnh nhập thêm các gói hoặc các lớp khác vào chương trình (import).
- Một khai báo lớp công cộng (public)
- Một số các lớp dạng riêng tư (private) của gói.

Một tập tin nguồn Java sẽ có khai báo lớp public đơn. Tất cả những phát biểu khác tùy chọn. Chương trình nên được viết theo thứ tự: đặt tên gói (package), lệnh nhập các gói (import), và định nghĩa lớp (class).

Các giao diện

Giao diện là một trong những khái niệm quan trọng nhất của ngôn ngữ Java. Nó cho phép một lớp có nhiều lớp cha (superclass). Các chương trình Java có thể thừa kế chỉ một lớp tại một thời điểm, nhưng có thể hiện thực hàng loạt giao diện. Giao diện được sử dụng để thay thế một lớp trừu tượng, không có một sự kế thừa mã thực thi nào. Giao diện tương tự như các lớp trừu tượng. Sự khác nhau ở chỗ một lớp trừu tượng có thể có những hành vi cụ thể, nhưng một giao diện thì không thể có một phương thức cụ thể nào có hành vi của riêng mình. Các giao diện cần được hiện thực. Một lớp trừu tượng có thể được thừa kế, nhưng không thể tạo ra được thể hiện (đối tượng).

Các bước để tạo một giao diện được liệt kê ở dưới đây:

- Định nghĩa giao diện: Một giao diện được định nghĩa như sau:

```
//Giao diện với các phương thức public interface myinterface { public void add(int x,int y); public void volume(int x,int y,int z); } //Giao diện để định nghĩa các hằng số public interface myconstants { public static final double price=1450.00; public static final int counter=5; }
```

- Chương trình trên được dịch như sau:

javac myinterface.java

- Một giao diện được hiện thực với từ khoá “**implement**”. Trong trường hợp trên, giao diện cho phép ứng dụng mối quan hệ “**is a**” .
`class demo implements myinterface`
- Nếu nhiều hơn một giao diện được thực thi, các tên sẽ được ngăn cách với nhau bởi một dấu phẩy. Điều này được trình bày như sau: `class Demo`
`implements MyCalc, Mycount`

Hãy ghi nhớ các lưu ý sau trong khi tạo một giao diện:

- Tất cả các phương thức trong các giao diện này phải là kiểu **public**.
- Các phương thức được định nghĩa trong một lớp mà lớp này hiện thực giao diện.

Hiện thực giao diện

Các giao diện không thể thừa kế (**extends**) các lớp, nhưng chúng có thể thừa kế các giao diện khác. Nếu khi bạn hiện thực một giao diện mà thừa kế các giao diện khác, bạn định nghĩa đè (**override**) các phương thức trong giao diện mới giao diện đã thừa kế. Trong ví dụ trên, các phương thức chỉ được khai báo, mà không được định nghĩa. Các phương thức phải được định nghĩa trong một lớp mà lớp đó hiện thực giao diện này. Nói một cách khác, bạn cần chỉ ra hành vi của phương thức. Tất cả các phương thức trong các giao diện phải là kiểu **public**. Bạn không được sử dụng các bộ ngữ (**modifiers**) chuẩn khác như **protected**, **private**,...khi khai báo các phương thức trong giao diện.

Đoạn mã chương trình dưới đây biểu diễn một giao diện được cài đặt như thế nào:

```
import java.io.*; class Demo implements myinterface {
public void add(int x,int y) { System.out.println("
" +(x+y)); //Giả sử phương thức add được khai báo trong
giao diện } public void volume(int x,int y,int z) {
System.out.println(" " +(x*y*z)); //Giả sử phương thức
volume được khai báo trong giao diện } public static void
main(String args[]) { Demo d=new Demo(); d.add(10,20);
d.volume(10,10,10); } }
```

Khi bạn định nghĩa một giao diện mới, có nghĩa là bạn đang định nghĩa một kiểu dữ liệu tham chiếu mới. Bạn có thể sử dụng các tên giao diện ở bất cứ nơi đâu như bất kỳ kiểu dữ liệu khác. Chỉ có một thể hiện (instance) của lớp mà lớp đó thực thi giao diện có thể được gán cho một biến tham chiếu. Kiểu của biến tham chiếu đó là tên của giao diện.

Các gói

Gói được coi như các thư mục, đó là nơi bạn tổ chức các lớp và các giao diện của bạn. Các chương trình Java được tổ chức như những tập của các gói. Mỗi gói gồm có nhiều lớp, và/hoặc các giao diện được coi như là các thành viên của nó. Đó là một phương án thuận lợi để lưu trữ các nhóm của những lớp có liên quan với nhau dưới một cái tên cụ thể. Khi bạn đang làm việc với một chương trình ứng dụng, bạn tạo ra một số lớp. Các lớp đó cần được tổ chức một cách hợp lý. Điều đó trở nên dễ dàng khi ta tổ chức các tập tin lớp thành các gói khác nhau. Hãy tưởng tượng rằng mỗi gói giống như một thư mục con. Tất cả các điều mà bạn cần làm là đặt các lớp và các giao diện có liên quan với nhau vào các thư mục riêng, với một cái tên phản ánh được mục đích của các lớp.

Nói tóm lại, các gói có ích cho các mục đích sau:

- Chúng cho phép bạn tổ chức các lớp thành các đơn vị nhỏ hơn (như là các thư mục), và làm cho việc xác định vị trí trở nên dễ dàng và sử dụng các tập tin của lớp một cách phù hợp.
- Giúp đỡ để tránh cho việc đặt tên bị xung đột (trùng lặp tên). Khi bạn làm việc với một số các lớp bạn sẽ cảm thấy khó để quyết định đặt tên cho các lớp và các phương thức. Đôi lúc bạn muốn sử dụng tên giống nhau mà tên đó liên quan đến lớp khác. Các gói giấu các lớp để tránh việc đặt tên bị xung đột.
- Các gói cho phép bạn bảo vệ các lớp, dữ liệu và phương thức ở mức rộng hơn trên một nền tảng class-to-class.
- Các tên của gói có thể được sử dụng để nhận dạng các lớp.

Các gói cũng có thể chứa các gói khác.

Để tạo ra một lớp là thành viên của gói, bạn cần bắt đầu mã nguồn của bạn với một khai báo gói, như sau:

```
package mypackage;
```

Hãy ghi nhớ các điểm sau trong khi tạo gói:

- Đoạn mã phải bắt đầu với một phát biểu “package”. Điều này nói lên rằng lớp được định nghĩa trong tập tin là một phần của gói xác định.
- Mã nguồn phải nằm trong cùng một thư mục, mà thư mục đó lại là tên gói của bạn.
- Quy ước rằng, các tên gói sẽ bắt đầu bằng một chữ thường để phân biệt giữa lớp và gói.
- Các phát biểu khác có thể xuất hiện sau khai báo gói là các câu lệnh nhập, sau chúng bạn có thể bắt đầu định nghĩa lớp của bạn.
- Tương tự tất cả các tập tin khác, mỗi lớp trong một gói cần được biên dịch.

- Để cho chương trình Java của bạn có khả năng sử dụng các gói đó, hãy nhập (import) chúng vào mã nguồn của bạn.
- Sự khai báo sau đây là hợp lệ và không hợp lệ :

Hợp lệ :

```
package mypackage; import java.io.*;
```

Không hợp lệ :

```
import java.io.*; package mypackage;
```

Bạn có các tùy chọn sau trong khi nhập vào một gói:

- Bạn có thể nhập vào một tập tin cụ thể từ gói: `import java.mypackage.calculate`
- Bạn có thể nhập (import) toàn bộ gói: `import java.mypackage.*;`

Máy ảo Java (JVM) sẽ quản lý các thành phần nằm trong các gói đã được nhập vào (import).

Bạn đã sẵn sàng làm việc với một lệnh nhập `import -java.io.*`. Bản thân Java đã được cài đặt sẵn một tập các gói, bảng dưới đây đề cập đến một vài gói có sẵn của Java:

Các gói trong Java

Gói	Mô tả
java.lang	Không cần phải khai báo nhập. Gói này luôn được nhập cho bạn.
java.io	Bao gồm các lớp để trợ giúp cho bạn tất cả các thao tác vào ra.
java.applet	Bao gồm các lớp để bạn cần thực thi một applet trong trình duyệt.
java.awt	Các thành phần để xây dựng giao diện đồ họa (GUI).
java.util	Cung cấp nhiều lớp và nhiều giao diện tiện ích khác nhau, như là các cấu trúc dữ liệu, lịch, ngày tháng, v.v..
java.net	Cung cấp các lớp và các giao diện cho việc lập trình mạng TCP/IP.

Bên cạnh đó, Java còn cung cấp thêm nhiều gói để phát triển ứng dụng và applet. Nếu bạn không khai báo các gói trong đoạn mã của bạn, thì các lớp và các giao diện của bạn sau khi kết thúc sẽ nằm trong một gói mặc định mà không có tên. Thông thường, gói mặc định này chỉ có ý nghĩa cho các ứng dụng nhỏ hoặc các ứng dụng tạm thời. Khi bạn bắt đầu việc phát triển cho một ứng dụng lớn, bạn có khuynh hướng phát triển một số

các lớp. Bạn cần tổ chức các lớp đó trong các thư mục khác nhau để dễ dàng truy cập. Để làm được điều này, bạn phải đặt chúng vào các gói.

Ý nghĩa lớn nhất của gói là bạn có khả năng sử dụng các tên lớp giống nhau, nhưng bạn phải đặt chúng vào các gói khác nhau.

Tạo một gói

Gói là một phương thức hữu dụng để nhóm các lớp mà tránh được các tên trùng nhau. Các lớp với những tên giống nhau có thể đặt vào các gói khác nhau. Các lớp được định nghĩa bởi người sử dụng cũng có thể được nhóm lại trong các gói.

Các bước sau đây cho phép tạo nên một gói do người dùng định nghĩa:

- Khai báo gói bằng cách sử dụng cú pháp thích hợp. Đoạn mã phải bắt đầu với khai báo gói. Điều này chỉ ra rằng lớp được định nghĩa trong tập tin là một phần của gói xác định. `package mypackage;`
- Sử dụng phát biểu `import` để nhập các gói chuẩn theo yêu cầu. `import java.util.*;`
- Khai báo và định nghĩa các lớp sẽ nằm trong gói đó. Tất cả các thành phần của gói sẽ là `public`, để có thể được truy cập từ bên ngoài. Máy ảo Java (JVM) quản lý tất cả các phần tử nằm trong gói đó.

```
package mypackage; //khai báo gói
import java.util.*; public class Calculate
//định nghĩa một lớp { int var; Calculate(int n) {
... var = n; //các phương thức //... public class
Display //định nghĩa một lớp { ...//Các phương thức }
} }
```
- Lưu các định nghĩa trên trong một tập tin với phần mở rộng `.java`, và dịch các lớp được định nghĩa trong gói. Việc dịch có thể thực hiện với tham số `“-d”`. Chức năng này tạo một thư mục trùng với tên gói, và đặt tập tin `.class` vào thư mục được chỉ rõ.

`javac -d d:\temp Calculate.java`

Nếu khai báo gói không có trong chương trình, lớp hoặc giao diện đó sẽ nằm trong gói mặc định mà không có tên. Nói chung, gói mặc định này thì chỉ có nghĩa cho các ứng dụng nhỏ hoặc tạm thời.

Hãy ghi nhớ các điểm sau đây khi bạn khai thác các gói do người dùng định nghĩa trong các chương trình khác:

- Mã nguồn của các chương trình đó phải tồn tại trong cùng một thư mục với gói được định nghĩa bởi người sử dụng.

- Để cho các chương trình Java khác sử dụng được các gói đó, hãy khai báo chúng vào đoạn mã nguồn.
- Để nhập một lớp ta dùng: `import java.mypackage.Calculate;`
- Để nhập toàn bộ một gói, ta làm như sau: `import java.mypackage.*;`
- Tạo một tham chiếu đến các thành phần của gói. Ta dùng đoạn mã đơn giản sau: `import java.io.*; import mypackage.Calculate;`
`class PackageDemo{ public static void main(String args[]){ Calculate calc = new Calculate(); } }`

Nếu phát biểu import cho gói đó không được sử dụng, thì khi sử dụng lớp đó phải chỉ ra lớp đó ở gói nào. Cú pháp như sau: `mypackage.Calculate calc = new mypackage.Calculate();`

Thiết lập đường dẫn cho lớp (classpath)

Chương trình dịch và chương trình thông dịch tìm kiếm các lớp trong thư mục hiện hành, và tập tin nén (zip) chứa các lớp của JDK. Điều này có nghĩa các tập tin nén chứa các lớp của JDK và thư mục hiện hành chứa mã nguồn tự động được đặt vào **classpath**. Tuy nhiên, trong một vài trường hợp, bạn cần phải tự thiết lập classpath.

Classpath là một danh sách các thư mục, danh sách này trợ giúp để tìm kiếm các tập tin .class tương ứng. Thông thường, ta không nên thiết lập môi trường classpath lâu dài. Nó chỉ thích hợp khi thiết lập CLASSPATH để chạy chương trình, chỉ thiết lập đường dẫn cho việc thực thi hiện thời.

javac -classpath c:\temp Packagedemo.java

Thứ tự của các mục trong classpath rất quan trọng. Khi bạn thực thi đoạn mã của bạn, máy ảo Java sẽ tìm kiếm các mục trong classpath theo thứ tự các thư mục trong classpath, cho đến khi nó tìm thấy lớp cần tìm.

Ví dụ của một nhóm

```
package mypackage; public class calculate { public double
volume(double height, double width, double depth) { return
(height*width*depth); } public int add(int x, int y) {
return (x+y); } public int divide(int x, int y) { return
(x/y); } }
```

Để sử dụng gói này, bạn cần phải:

- Nhập lớp được sử dụng.
- Nhập toàn bộ gói.
- Sử dụng các thành phần của gói.

Bạn cần dịch tập tin này. Nó có thể được dịch với tùy chọn `-d`, nhờ đó `javac` nó tạo một thư mục với tên của gói và đặt tập tin `.class` vào thư mục này.

`javac -d c:\temp calculate.java`

Chương trình biên dịch tạo một thư mục được gọi là “mypackage” trong thư mục temp, và lưu trữ tập tin `calculate.class` vào thư mục này.

Biểu diễn cách sử dụng một gói

```
import java.io.*; import mypackage.calculate; class
PackageDemo{ public static void main(String args[]){
calculate calc = new calculate(); int sum =
calc.add(10,20); double vol =
calc.volume(10.3f,13.2f,32.32f); int div =
calc.divide(20,4); System.out.println("The addition is:
"+sum); System.out.println("The Volume is: "+vol);
System.out.println("The division is: "+sum); } }
```

Nếu bạn sử dụng một lớp từ một gói khác, mà không sử dụng khai báo `import` cho gói đó, thì khi đó, bạn cần phải sử dụng tên lớp với tên gói. `mypackage.calculate`
`calc = new mypackage.calculate();`

Gói và điều khiển truy xuất

Các gói chứa các lớp và các gói con. Các lớp chứa dữ liệu và đoạn mã. Java cung cấp nhiều mức độ truy cập thông qua các lớp, các gói và các chỉ định truy cập. Bảng sau đây sẽ tóm tắt quyền truy cập các thành phần của lớp:

Truy cập đến các thành phần của lớp

	public	protected	No modifier	private
Cùng lớp	Yes	Yes	Yes	Yes
Cùng gói- lớp thừa kế (Subclass)	Yes	Yes	Yes	No
Cùng gói-không thừa kế (non-Subclass)	Yes	Yes	Yes	No
Khác gói-lớp thừa kế (subclass)	Yes	Yes	No	No
Khác gói-không thừa kế (non-Subclass)	Yes	No	No	No

Gói java.lang

Mặc định, mỗi chương trình java đều nhập gói java.lang. Vì thế, không cần lệnh nhập gói java.lang này trong chương trình.

Lớp bao bọc (wrapper class)

Các kiểu dữ liệu nguyên thủy thì không phải là các đối tượng. Vì thế, chúng không thể tạo ra hay truy cập bằng phương thức. Để tạo và thao tác kiểu dữ liệu nguyên thủy, ta sử dụng “wrapper class” tương ứng với. Bảng sau liệt kê các lớp trình bao bọc (wrapper). Các phương thức của mỗi lớp này có trong phần phụ lục.

Các lớp trình bao bọc cho các
kiểu dữ liệu nguyên thủy.

Kiểu dữ liệu	Lớp trình bao bọc
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Một vài phương thức của lớp Wrapper

```
Boolean wrapBool = new Boolean("false"); Integer num1 =  
new Integer("31"); Integer num2 = new Integer("3"); Int  
sum = num1.intValue()*num2.intValue(); //intValue() là một  
hàm của lớp trình bao bọc Integer.
```

Chương trình sau đây minh họa cách sử dụng lớp wrapper cho kiểu dữ liệu int

```
class CmdArg { public static void main(String args[]) {  
int sum = 0; for(int i = 0;i<args.length;i++) sum+=  
Integer.parseInt(args[i]); System.out.println("Tổng là:  
"+sum); } }
```

Vòng lặp for được sử dụng để tìm tổng của các số được truyền vào từ dòng lệnh. Các số đó được lưu trữ trong mảng String args[]. Thuộc tính “length” xác định số các phần tử trong mảng args[]. Mảng args[] là kiểu String. Vì thế, các phần tử phải được đổi sang kiểu dữ liệu int trước khi cộng chúng. Quá trình chuyển đổi được thực hiện với sự giúp đỡ của lớp trình bao bọc “Integer”. Phương thức “parseInt()” trong lớp “Integer” thực hiện quá trình chuyển đổi của kiểu dữ liệu chuỗi sang kiểu dữ liệu số nguyên.

Tất cả các lớp trình bao bọc, ngoại trừ lớp “**Character**” có một phương thức tĩnh “**valueOf()**” nhận một chuỗi, và trả về một giá trị số nguyên được. Các lớp bao bọc của byte, int, long, và short cung cấp các hằng số **MIN_VALUE** và **MAX_VALUE**. Các lớp bao bọc của double và long cũng cung cấp các hằng **POSITIVE_INFINITY** và **NEGATIVE_INFINITY**.

Lớp String (lớp chuỗi)

Chuỗi là một dãy các ký tự. Lớp String cung cấp các phương thức để thao tác với các chuỗi. Nó cung cấp các phương thức khởi tạo (constructor) khác nhau:

```
String str1 = new String( ); //str1 chứa một chuỗi rỗng.  
String str2 = new String("Hello World"); //str2 chứa  
"Hello World" char ch[] = {'A','B','C','D','E'}; String  
str3 = new String(ch); //str3 chứa "ABCDE" String str4 =  
new String(ch,0,2); //str4 chứa "AB" vì 0- tính từ ký tự  
bắt đầu, 2- là số lượng ký tự kể từ ký tự bắt đầu.
```

Toán tử “+” được sử dụng để cộng chuỗi khác vào chuỗi đang tồn tại. Toán tử “+” này được gọi như là “nối chuỗi”. Ở đây, nối chuỗi được thực hiện thông qua lớp “**StringBuffer**”. Chúng ta sẽ thảo luận về lớp này trong phần sau. Phương thức “**concat()**” của lớp String cũng có thể thực hiện việc nối chuỗi. Không giống như toán tử “+”, phương thức này không thường xuyên nối hai chuỗi tại vị trí cuối cùng của chuỗi đầu tiên. Thay vào đó, phương thức này trả về một chuỗi mới, chuỗi mới đó sẽ chứa giá trị của cả hai. Điều này có thể được gán cho chuỗi đang tồn tại. Ví dụ:

```
String strFirst, strSecond, strFinal; StrFirst =  
"Charlie"; StrSecond = "Chaplin"; //...bằng cách sử dụng  
phương thức concat( ) để gán với một chuỗi đang tồn tại.  
StrFinal = strFirst.concat(strSecond);
```

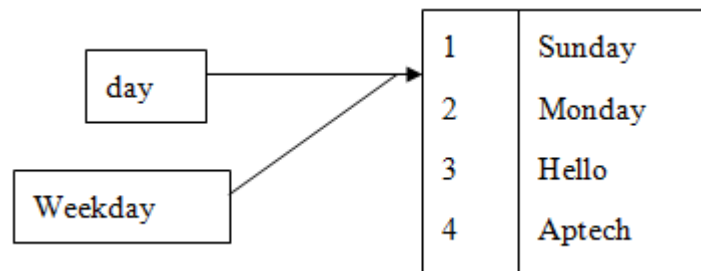
Phương thức **concat()** chỉ làm việc với hai chuỗi tại một thời điểm.

Chuỗi mặc định (String pool)

Một chương trình Java có thể chứa nhiều chuỗi. “**String Pool**” đại diện cho tất cả các chữ được tạo trong chương trình. Mỗi khi một chuỗi được tạo, String Pool tìm kiếm trong nó, nếu tìm thấy nếu chuỗi đã tồn tại thì không tạo thể hiện mà chỉ gán thể tìm thấy cho chuỗi mới. Việc này tiết kiệm rất nhiều không gian bộ nhớ. Ví dụ:

```
String day = "Monday"; String weekday = "Monday";
```

Ở đây, một thể hiện cho biến “day”, biến đó có giá trị là “Monday”, được tạo trong String Pool. Khi chuỗi bằng chữ “weekday” được tạo, có giá trị giống như của biến “day”, một thể hiện đang tồn tại được gán đến biến “weekday”. Vì cả hai biến “day” và “weekday” cũng đều nhằm chỉ vào chuỗi giống hệt nhau trong String Pool. Hình ảnh sau minh hoạ khái niệm của “**String Pool**”.



Khái niệm của String Pool

Các phương thức của lớp String

Trong phần này, chúng ta sẽ xem xét các phương thức của lớp String.

- `charAt()`

Phương thức này trả về một ký tự tại một vị trí trong chuỗi.

```
String name = new String("Java Language"); char ch =  
name.charAt(5);
```

Biến “ch” chứa giá trị “L”, từ đó vị trí các số bắt đầu từ 0.

- `startsWith()`

Phương thức này trả về giá trị kiểu logic (Boolean), phụ thuộc vào chuỗi có bắt đầu với một chuỗi con cụ thể nào đó không.

```
String strname = "Java Language"; boolean flag =  
strname.startsWith("Java");
```


Biến “flag” chứa giá trị true.

- `endsWith()`

Phương thức này trả về một giá trị kiểu logic (boolean), phụ thuộc vào chuỗi kết thúc bằng một chuỗi con nào đó không

```
String strname = "Java Language"; boolean flag =  
strname.endsWith("Java");
```

Biến “flag” chứa giá trị false.

- `copyValueOf()`

Phương thức này trả về một chuỗi được rút ra từ một mảng ký tự được truyền như một đối số. Phương thức này cũng lấy hai tham số nguyên. Tham số đầu tiên chỉ định vị trí từ nơi các ký tự phải được rút ra, và tham số thứ hai chỉ định số ký tự được rút ra từ mảng.

```
char name[] = {'L','a','n','g','u','a','g','e'}; String  
subname = String.copyValueOf(name,5,2);
```

Bây giờ biến “subname” chứa chuỗi “ag”.

- `toCharArray()`

Phương thức này chuyển chuỗi thành một mảng ký tự.

```
String text = new String("Hello World"); char textArray[]  
= text.toCharArray();
```

- `indexOf()`

Phương thức này trả về thứ tự của một ký tự nào đó, hoặc một chuỗi trong phạm vi một chuỗi. Các câu lệnh sau biểu diễn các cách khác nhau của việc sử dụng hàm.

```
String day = new String("Sunday"); int index1 =  
day.indexOf('\n'); //chứa 2 int index2 =  
day.indexOf('z',2); //chứa -1 nếu "z" không tìm thấy tại  
vị trí 2. int index3 = day.indexOf("Sun"); //chứa mục 0
```

- `toUpperCase()`

Phương thức này trả về chữ hoa của chuỗi.

```
String lower = new String("good morning");
System.out.println("Uppercase: "+lower.toUpperCase( ));
    • toLowerCase()
```

Phương thức này trả về chữ thường của chuỗi.

```
String upper = new String("APTECH");
System.out.println("Lowercase: "+upper.toLowerCase( ));
    • trim()
```

Phương thức này cắt bỏ khoảng trắng hai đầu chuỗi. Hãy thử đoạn mã sau để thấy sự khác nhau trước và sau khi cắt bỏ khoảng trắng.

```
String space = new String(" Spaces ");
System.out.println(space);
System.out.println(space.trim()); //Sau khi cắt bỏ khoảng
trắng
    • equals()
```

Phương thức này so sánh nội dung của hai đối tượng chuỗi.

```
String name1 = "Aptech", name2 = "APTECH"; boolean flag =
name1.equals(name2);
```

Biến “flag” chứa giá trị false.

Lớp StringBuffer

Lớp StringBuffer cung cấp các phương thức khác nhau để thao tác một đối tượng dạng chuỗi. Các đối tượng của lớp này rất mềm dẻo, đó là các ký tự và các chuỗi có thể được chèn vào giữa đối tượng StringBuffer, hoặc nối thêm dữ liệu vào tại vị trí cuối. Lớp này cung cấp nhiều phương thức khởi tạo. Chương trình sau minh họa cách sử dụng các phương thức khởi tạo khác nhau để tạo ra các đối tượng của lớp này.

```
class StringBufferCons { public static void main(String
args[]) { StringBuffer s1 = new StringBuffer();
StringBuffer s2 = new StringBuffer(20); StringBuffer s3 =
new StringBuffer("StringBuffer"); System.out.println("s3 =
"+ s3); System.out.println(s2.length()); //chứa 0
System.out.println(s3.length()); //chứa 12
System.out.println(s1.capacity()); //chứa 16
```

```
System.out.println(s2.capacity()); //chứa 20
System.out.println(s3.capacity()); //chứa 28 } }
```

“**length()**” và “**capacity()**” của StringBuffer là hai phương thức hoàn toàn khác nhau. Phương thức “**length()**” đề cập đến số các ký tự mà đối tượng thực chứa, trong khi “**capacity()**” trả về tổng dung lượng của một đối tượng (mặc định là 16) và số ký tự trong đối tượng StringBuffer.

Dung lượng của StringBuffer có thể thay đổi với phương thức “**ensureCapacity()**”. Đối số int đã được truyền đến phương thức này, và dung lượng mới được tính toán như sau:

$$\text{NewCapacity} = \text{OldCapacity} * 2 + 2$$

Trước khi dung lượng của StringBuffer được đặt lại, điều kiện sau sẽ được kiểm tra:

- Nếu dung lượng(NewCapacity) mới lớn hơn đối số được truyền cho phương thức “ensureCapacity()”, thì dung lượng mới (NewCapacity) được đặt.
- Nếu dung lượng mới nhỏ hơn đối số được truyền cho phương thức “ensureCapacity()”, thì dung lượng được đặt bằng giá trị tham số truyền vào.

Chương trình dưới đây sẽ minh họa dung lượng được tính toán và được đặt như thế nào.

```
class test{ public static void main(String args[]){
StringBuffer s1 = new StringBuffer(5);
System.out.println("Dung lượng của bộ nhớ đệm =
"+s1.capacity()); //chứa 5 s1.ensureCapacity(8);
System.out.println("Dung lượng của bộ nhớ đệm =
"+s1.capacity()); //chứa 12 s1.ensureCapacity(30);
System.out.println("Dung lượng của bộ nhớ đệm =
"+s1.capacity()); //chứa 30 } }
```

Trong đoạn mã trên, dung lượng ban đầu của s1 là 5. Câu lệnh

```
s1.ensureCapacity(8);
```

Thiết lập dung lượng của s1 đến 12 $= (5 * 2 + 2)$ bởi vì dung lượng truyền vào là 8 nhỏ hơn dung lượng được tính toán là 12 .

```
s1.ensureCapacity(30);
```

Thiết lập dung lượng của “s1” đến 30 bởi vì dung lượng truyền vào là 30 thì lớn hơn dung lượng được tính toán $(12 * 2 + 2)$.

Các phương thức lớp StringBuffer

Trong phần này, chúng ta sẽ xem xét các phương thức của lớp StringBuffer với một chương trình.

- `append()`

Phương thức này nối thêm một chuỗi hoặc một mảng ký tự vào cuối cùng của đối tượng StringBuffer.

```
StringBuffer s1 = new StringBuffer("Good");  
s1.append("evening");
```

Giá trị trong s1 bây giờ là “goodevening”.

- `insert()`

Phương thức này có hai tham số. Tham số đầu tiên là vị trí chèn. Tham số thứ hai có thể là một chuỗi, một ký tự (char), một giá trị nguyên (int), hay một giá trị số thực (float) được chèn vào. Vị trí chèn sẽ lớn hơn hay bằng 0, và nhỏ hơn hay bằng chiều dài của đối tượng StringBuffer. Bất kỳ đối số nào, trừ ký tự hoặc chuỗi, được chuyển sang chuỗi và sau đó mới được chèn vào.

```
StringBuffer str = new StringBuffer("Java sion");  
str.insert(1, 'b');
```

Biến “str” chứa chuỗi “Jbava sion”.

- `charAt()`

Phương thức này trả về một giá trị ký tự trong đối tượng StringBuffer tại vị trí được chỉ định.

```
StringBuffer str = new StringBuffer("James Gosling"); char  
letter = str.charAt(6); //chứa "G"
```

- `setCharAt()`

Phương thức này được sử dụng để thay thế ký tự trong một StringBuffer bằng một ký tự khác tại một vị trí được chỉ định.

```
StringBuffer name = new StringBuffer("Jawa");  
name.setCharAt(2, 'v');
```

Biến “name” chứa “Java”.

- `setLength()`

Phương thức này thiết lập chiều dài của đối tượng `StringBuffer`. Nếu chiều dài được chỉ định nhỏ hơn chiều dài dữ liệu hiện tại của nó, thì các ký tự thừa sẽ bị cắt bớt. Nếu chiều dài chỉ định nhiều hơn chiều dài dữ liệu thì các ký tự null được thêm vào phần cuối của `StringBuffer`

```
StringBuffer str = new StringBuffer(10);  
str.setLength(str.length() + 10);
```

- `getChars()`

Phương thức này được sử dụng để trích ra các ký tự từ đối tượng `StringBuffer`, và sao chép chúng vào một mảng. Phương thức `getChars()` có bốn tham số sau:

Chỉ số đầu: vị trí bắt đầu, từ nơi mà ký tự được lấy ra.

Chỉ số kết thúc: vị trí kết thúc

Mảng: Mảng đích, nơi mà các ký tự được sao chép.

Vị trí bắt đầu trong mảng đích: Các ký tự được sao chép vào mảng đích từ vị trí này.

```
StringBuffer str = new StringBuffer("Leopard"); char ch[]  
= new char[10]; str.getChars(3, 6, ch, 0);
```

Bây giờ biến “ch” chứa “par”

- `reverse()`

Phương thức này đảo ngược nội dung của một đối tượng `StringBuffer`, và trả về một đối tượng `StringBuffer` khác.

```
StringBuffer str = new StringBuffer("devil"); StringBuffer  
strrev = str.reverse();
```

Biến “strrev” chứa “lived”.

Lớp `java.lang.Math`

Lớp này chứa các phương thức tĩnh (static) để thực hiện các thao tác toán học. Chúng được mô tả như sau:

Cú pháp là `Math.<tên hàm>`

- `abs()`

Phương thức này trả về giá trị tuyệt đối của một số. Đối số được truyền đến nó có thể là kiểu `int`, `float`, `double`, hoặc `long`. Kiểu dữ liệu `byte` và `short` được chuyển thành kiểu `int` nếu chúng được truyền tới như là một đối số. Ví dụ:

```
int num = -1; Math.abs(num) //trả về 1.
```

- `ceil()`

Phương thức này tìm thấy số nguyên nhỏ nhất lớn hơn hoặc bằng đối số được truyền vào.

- `floor()`

Phương thức này trả về số nguyên lớn nhất nhỏ hơn hoặc bằng đối số được truyền vào.

```
System.out.println(Math.ceil(8.02)); //trả về 9.0
System.out.println(Math.ceil(-1.3)); //trả về -1.0
System.out.println(Math.ceil(100)); //trả về 100.0
System.out.println(Math.floor(-5.6)); //trả về -6.0
System.out.println(Math.floor(201.1)); //trả về 201
System.out.println(Math.floor(100)); //trả về 100
```

- `max()`

Phương thức này tìm giá trị lớn nhất trong hai giá trị được truyền vào. Các đối số được truyền vào có thể là kiểu `int`, `long`, `double`, và `float`.

- `min()`

Phương thức này tìm giá trị nhỏ nhất trong hai giá trị được truyền vào. Các đối số được truyền vào có thể là kiểu `int`, `long`, `double` và `float`.

- `round()`

Phương thức này làm tròn đối số có dấu phẩy động. Ví dụ, câu lệnh `Math.round(34.5)` trả về 35.

- `random()`

Phương thức này trả về một số ngẫu nhiên kiểu `double` giữa 0.0 và 1.0.

- `sqrt()`

Phương thức này trả về căn bậc hai của một số. Ví dụ, câu lệnh `Math.sqrt(144)` trả về 12.0.

- `sin()`

Phương thức này trả về sine của một số, nếu góc được truyền đến bằng radian. Ví dụ: **`Math.sin(Math.PI/2)`** trả về 1.0, giá trị của $\sin 45^0$.

$\text{PI}/2$ radian = 90 độ. Giá trị của “PI” được định nghĩa trong lớp `Math` (`Math.PI`).

- `cos()`

Phương thức này trả về cosine của một góc tính bằng radian.

- `tan()`

Phương thức này trả về tan của một góc tính bằng radian.

Lớp Runtime (Thời gian thực hiện chương trình)

Lớp `Runtime` chứa thông tin về môi trường thực thi. Lớp này được sử dụng cho việc quản lý bộ nhớ, và việc thực thi của các quá trình xử lý khác. Mỗi chương trình Java có một thể hiện của lớp này, để cho phép ứng dụng giao tiếp với môi trường. Nó không thể được khởi tạo, một ứng dụng không thể tạo ra một thể hiện của thuộc lớp này. Tuy nhiên, chúng ta có thể tham chiếu thể hiện trong lúc thực hiện chương trình từ việc dùng phương thức `getRuntime()`.

Bây giờ, chúng ta biết rằng việc thu gom các dữ liệu không thích hợp trong Java là một tiến trình tự động, và chạy một cách định kỳ. Để kích hoạt một cách thủ công bộ thu thập dữ liệu không còn được sử dụng ta gọi phương thức `gc()` trên đối tượng `Runtime` hiện thời. Để xem chi tiết việc cấp phát bộ nhớ, sử dụng các phương thức `totalMemory()` và `freeMemory()`.

```
Runtime r = Runtime.getRuntime(); ... .. long freemem =  
r.freeMemory(); long totalmem = r.totalMemory(); r.gc();
```

Bảng sau trình bày một vài phương thức của lớp này:

Lớp Runtime	
Phương thức	Ý nghĩa

exit(int)	Dừng việc thực thi, và trả về giá trị của chương trình cho hệ điều hành. Nếu thoát bình thường thì trả về 0; giá trị khác 0 cho thoát không bình thường.
freeMemory()	Trả về kích thước bộ nhớ chưa sử dụng tính bằng byte
getRuntime()	Trả về thể hiện Runtime
gc()	Gọi bộ phận thu thập rác.
totalMemory()	Trả về kích thước bộ nhớ tính bằng byte.
exec(String)	Chạy chương trình ở môi trường bên ngoài

```
class RuntimeDemo { public static void main(String args[])
{ Runtime r = Runtime.getRuntime(); Process p = null; try
{ p = r.exec("calc.exe"); } catch(Exception e) {
System.out.println("Error executing calculator"); } } }
```

Bạn có thể tham chiếu đến Runtime hiện hành thông qua phương thức `Runtime.getRuntime()`.

Sau đó, bạn có thể chạy chương trình `calc.exe` và tham chiếu đến `calc.exe` trong đối tượng `Process`.

Lớp System

Lớp `System` cung cấp các tiện ích như là, dòng vào, dòng ra chuẩn và dòng lỗi. Nó cũng cung cấp phương thức để truy cập các thuộc tính liên quan đến hệ thống Runtime của Java, và các thuộc tính môi trường khác nhau như là, phiên bản (version), đường dẫn, hay các dịch vụ, v.v.. Các trường của lớp này là **in**, **out**, và **err**, các trường này tiêu biểu cho dòng vào, ra và lỗi chuẩn tương ứng.

Bảng sau mô tả các phương thức của lớp này:

Lớp System

Phương thức	Mục đích
exit(int)	Dừng việc thực thi, và trả về giá trị của đoạn mã. 0 cho biết có thể thoát ra một cách bình thường.
gc()	Gọi bộ phận thu thập rác.
getProperties()	Trả về thuộc tính của hệ thống thời gian chạy Java.
setProperties()	Thiết lập các thuộc tính hệ thống hiện hành.

currentTimeMillis()	Trả về thời gian hiện tại bằng mili giây (ms), được tính từ lúc 0 giờ ngày 01 tháng 01 năm 1970.
arrayCopy(Object, int, Object, int, int)	Sao chép mảng.

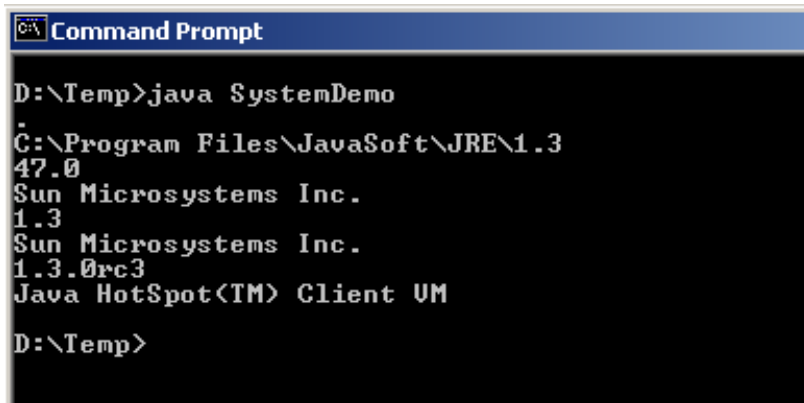
Lớp System không thể tạo thể hiện (instance) được.

Đoạn mã trong chương trình sau đọc và hiển thị một vài các thuộc tính môi trường Java.

```
class SystemDemo { public static void main(String args[])
{
System.out.println(System.getProperty("java.class.path"));
System.out.println(System.getProperty("java.home"));
System.out.println(System.getProperty("java.class.version"));
System.out.println(System.getProperty("java.specification.vendor"));
System.out.println(System.getProperty("java.specification.version"));
System.out.println(System.getProperty("java.vendor"));
System.out.println(System.getProperty("java.vendor.url"));
System.out.println(System.getProperty("java.version"));
System.out.println(System.getProperty("java.vm.name")); }
}
```

Mỗi thuộc tính cần in ra cần được cung cấp như một tham số (dạng chuỗi) đến phương thức System.getProperty(). Phương thức này sẽ trả về thông tin tương ứng và phương thức System.out.println() in ra màn hình.

Kết quả chương trình trên như sau:



```

C:\>Command Prompt
D:\Temp>java SystemDemo
.
C:\Program Files\JavaSoft\JRE\1.3
47.0
Sun Microsystems Inc.
1.3
Sun Microsystems Inc.
1.3.0rc3
Java HotSpot(TM) Client VM
D:\Temp>

```

Ví dụ về lớp System

Lớp Class

Các thể hiện của lớp này chứa trạng thái thời gian thực hiện của một đối tượng trong ứng dụng Java đang chạy. Điều này cho phép chúng ta truy cập thông tin về đối tượng trong thời gian chạy.

Chúng ta có thể lấy một đối tượng của lớp này, hoặc một thể hiện bằng một trong ba cách sau:

Sử dụng phương thức `getClass()` của đối tượng.

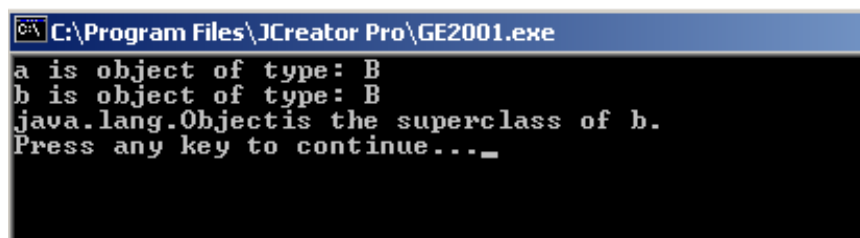
- Sử dụng phương thức tĩnh `forName()` của lớp để lấy một thể hiện của lớp thông qua tên của lớp đó.
- Sử dụng một đối tượng `ClassLoader` để nạp một lớp mới.

Lớp Class không có phương thức xây dựng (constructor).

Các chương trình sau minh họa cách sử dụng phương thức của một lớp để truy cập thông tin của lớp đó:

```
interface A { final int id = 1; final String name =  
"Diana"; } class B implements A { int deptno; } class  
ClassDemo { public static void main(String args[]) { A a =  
new B(); B b = new B(); Class x; x = a.getClass();  
System.out.println("a is object of type: "+x.getName());  
x= b.getClass(); System.out.println("b is object of type:  
"+x.getName()); x=x.getSuperclass();  
System.out.println(x.getName()+ "is the superclass of  
b."); } }
```

Kết quả chạy chương trình được mô tả như hình dưới đây:



```
C:\Program Files\JCreator Pro\GE2001.exe  
a is object of type: B  
b is object of type: B  
java.lang.Object is the superclass of b.  
Press any key to continue..._
```

Quá trình xuất ra các kết quả của lớp Class.

Lớp Object

Lớp Object là một lớp cha của tất cả các lớp. Dù là một lớp do người dùng định nghĩa không thừa kế lại bất kỳ một lớp nào khác, theo mặc định nó thừa kế lớp Object.

Một vài các phương thức của lớp Object được biểu diễn bên dưới:

Lớp Object	
Phương thức	Mục đích
<code>equals(Object)</code>	So sánh đối tượng hiện tại với đối tượng khác.
<code>finalize()</code>	Phương thức cuối cùng. Thông thường bị định nghĩa đè ở lớp con.
<code>notify()</code>	Thông báo cho Thread (luồng) mà hiện thời trong trạng thái đang chờ.
<code>notifyAll()</code>	Thông báo tất cả các Thread (luồng) hiện hành trong trạng thái chờ.
<code>toString()</code>	Trả về một chuỗi đại diện cho đối tượng.
<code>wait()</code>	Đưa Thread (luồng) vào trạng thái chờ.

Trong chương trình sau, chúng ta không khai báo bất kỳ lớp hoặc gói nào. Bây giờ, chúng ta có thể tạo bằng cách sử dụng phương thức `equals()`. Bởi vì, theo mặc định lớp `ObjectDemo` mở rộng lớp `Object`.

```
Class ObjectDemo { public static void main(String args[])
{ if (args[0].equals("Aptech")) System.out.println("Yes,
Aptech is the right choice!"); } }
```

Gói java.util

Gói `Java.util` cung cấp một số lớp tiện ích Java, thường xuyên trong tất cả các loại chương trình ứng dụng. Nó bao gồm một số lớp sau:

- `Hashtable`
- `Random`
- `Vector`
- `StringTokenizer`

Lớp Hashtable (bảng băm)

Lớp `Hashtable` mở rộng lớp trừu tượng `Dictionary`, lớp này cũng được định nghĩa trong gói `java.util`. *Hashtable* được sử dụng để ánh xạ khoá (key) đến giá trị (value). Ví dụ, nó

có thể được sử dụng để ánh xạ các tên đến tuổi, những người lập trình đến những dự án, chức danh công việc đến lương, và cứ như vậy.

Hashtable mở rộng kích thước khi các phần tử được thêm vào. Khi một *Hashtable* mới, bạn có thể chỉ định dung lượng ban đầu và yếu tố nạp (load factor). Điều này sẽ làm cho *hashtable* tăng kích thước lên, bất cứ lúc nào việc thêm vào một phần tử mới làm vượt qua giới hạn hiện hành của *Hashtable*. Giới hạn của *Hashtable* là dung lượng nhân lên bởi các yếu tố được nạp. Ví dụ: một bảng băm với dung lượng 100, và một yếu tố nạp là 0.75 sẽ có một giới hạn là 75 phần tử. Các phương thức xây dựng cho bảng băm được biểu diễn trong bảng sau:

Các phương thức xây dựng Hashtable

Constructor	Purpose
Hashtable(int)	Xây dựng một bảng mới với dung lượng ban đầu được chỉ định.
Hashtable(int, float)	Xây dựng một lớp mới với dung lượng ban đầu được chỉ định và yếu tố nạp.
Hashtable()	Xây dựng một lớp mới bằng cách sử dụng giá trị mặc định cho dung lượng ban đầu và yếu tố nạp.

```
Hashtable hash1 = new Hashtable(500, 0.80);
```

Trong trường hợp này, Bảng băm “hash1” sẽ lưu trữ 500 phần tử. Khi bảng băm lưu trữ vừa đầy 80% (một yếu tố nạp vào của 0.80), kích thước tối đa của nó sẽ được tăng lên.

Mỗi phần tử trong một hashtable bao gồm một khoá và một giá trị. Các phần tử được thêm vào bảng băm bằng cách sử dụng phương thức put(), và được truy lục bằng cách sử dụng phương thức get(). Các phần tử có thể được xoá từ một bảng băm với phương thức remove(). Các phương thức contains() và containsKey() có thể được sử dụng để tra cứu một giá trị hoặc một khoá trong bảng băm. Một vài phương thức của Hashtable được tóm tắt trong bảng sau:

Các phương thức lớp Hashtable

Phương thức	Mục đích
clear()	Xoá tất cả các phần tử từ bảng băm.
clone()	Tạo một bản sao của Hashtable.
contains(Object)	Trả về True nếu bảng băm chứa các đối tượng được chỉ định.
containsKey(Object)	Trả về True nếu bảng băm chứa khoá được chỉ định.
elements()	Trả về một tập hợp phần tử của bảng băm.

get(Object key)	Trả về đối tượng có khoá được chỉ định.
isEmpty()	Trả về true nếu bảng băm rỗng.
keys()	Trả về tập hợp các khoá trong bảng băm.
put(Object, Object)	Thêm một phần tử mới vào bảng băm (Object, Object) là khoá và giá trị.
rehash()	Thay đổi bảng băm thành một bảng băm lớn hơn.
remove(Object key)	Xoá một đối tượng được cho bởi khoá được chỉ định.
size()	Trả về số phần tử trong bảng băm.
toString()	Trả về đại diện chuỗi được định dạng cho bảng băm.

Chương trình sau sử dụng lớp Hashtable. Trong chương trình này, tên của các tập ảnh là các khoá, và các năm là các giá trị.

“**contains**” được sử dụng để tra cứu phần tử nguyên 1969, để thấy có danh sách chứa bất kỳ các tập ảnh từ 1969.

“**containsKey**” được sử dụng để tìm kiếm cho khoá “Animals”, để tìm tập ảnh đó trong bảng băm.

Phương thức “get()” được sử dụng để tìm tập ảnh “Wish You Were Here” có trong bảng băm không. Phương thức get() trả về phần tử cùng với khoá (tên và năm).

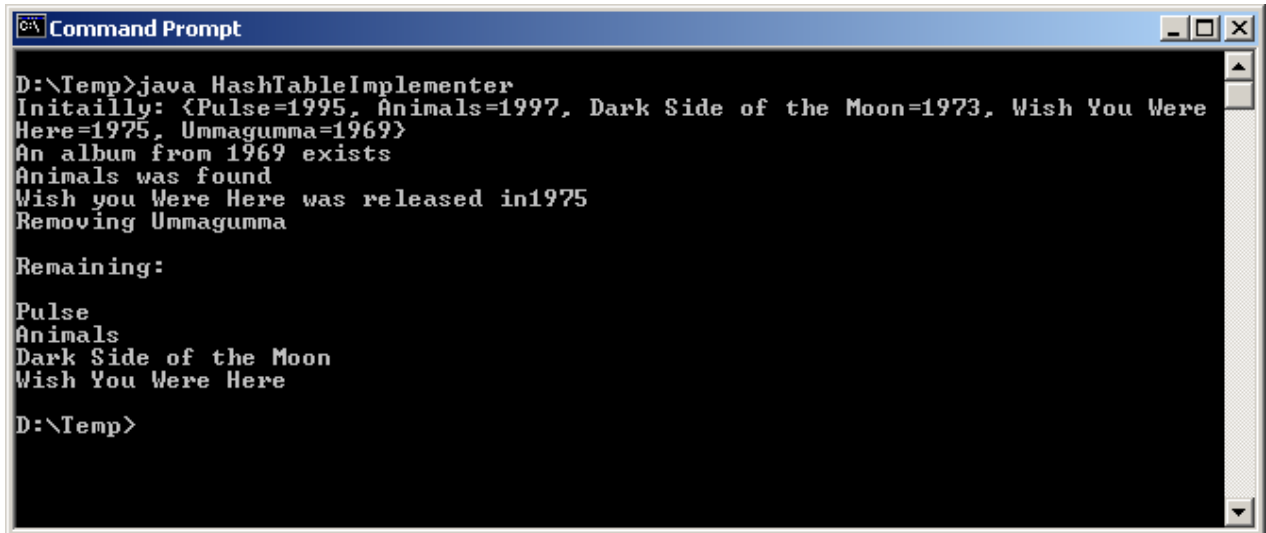
```
import java.util.*; public class HashTableImplementer {
public static void main(String args[]) { //tạo một bảng
băm mới Hashtable ht = new Hashtable(); //thêm các tập ảnh
tốt nhất của Pink Floyd ht.put("Pulse", new
Integer(1995)); ht.put("Dark Side of the Moon", new
Integer(1973)); ht.put("Wish You Were Here", new
Integer(1975)); ht.put("Animals", new Integer(1997));
ht.put("Ummagumma", new Integer(1969)); //Hiển thị bảng
băm System.out.println("Initailly: "+ht.toString());
//kiểm tra cho bất kỳ tập ảnh nào từ 1969
if(ht.contains(new Integer(1969))) System.out.println("An
album from 1969 exists"); //kiểm tra cho tập ảnh các con
thú if(ht.containsKey("Animals"))
System.out.println("Animals was found"); //Tìm ra Integer
year = (Integer)ht.get("Wish You Were Here");
System.out.println("Wish you Were Here was released
```

```

in"+year.toString()); //Xoá một tập ảnh
System.out.println("Removing Ummagumma\r\n");
ht.remove("Ummagumma"); //Duyệt qua tất cả các khoá trong
bảng. System.out.println("Remaining:\r\n");
for(Enumeration enum = ht.keys(); enum.hasMoreElements();)
System.out.println((String)enum.nextElement()); } }

```

Kết quả sẽ chạy chương trình như sau:



```

C:\> Command Prompt
D:\Temp>java HashTableImplementer
Initailly: {Pulse=1995, Animals=1997, Dark Side of the Moon=1973, Wish You Were
Here=1975, Ummagumma=1969}
An album from 1969 exists
Animals was found
Wish you Were Here was released in1975
Removing Ummagumma

Remaining:

Pulse
Animals
Dark Side of the Moon
Wish You Were Here
D:\Temp>

```

Kết quả của HashTableImplementer

Lớp random

Lớp này là một bộ tạo số giả ngẫu nhiên (pseudo-random). Có hai phương thức xây dựng được định nghĩa. Một trong những phương thức xây dựng này lấy giá trị khởi đầu (seed) như một tham số. Phương thức xây dựng khác không có tham số, và sử dụng thời gian hiện tại như một giá trị khởi đầu. Việc xây dựng một bộ tạo số ngẫu nhiên với một giá trị khởi đầu là một ý tưởng hay, trừ khi bạn muốn bộ tạo số ngẫu nhiên luôn tạo ra một tập các giá trị giống nhau. Mặt khác, thỉnh thoảng nó rất hữu ích để tạo ra trình tự giống nhau của các số random. Điều này có ý nghĩa trong việc gỡ rối chương trình. Một khi bộ tạo số ngẫu nhiên được tạo ra, bạn có thể sử dụng bất kỳ các phương thức sau đây để cập một giá trị từ nó:

- nextDouble()
- nextFloat()
- nextGaussian()
- nextInt()
- nextLong()

Các phương thức xây dựng và các phương thức của lớp Random được tóm tắt trong bảng sau:

Các phương thức lớp Random

Phương thức	Mục đích
random()	tạo ra một bộ tạo số ngẫu nhiên mới
random(long)	Tạo ra một bộ tạo số ngẫu nhiên mới dựa trên giá trị khởi tạo được chỉ định.
nextDouble()	Trả về một giá trị kiểu double kế tiếp giữa 0.0 đến 1.0 từ bộ tạo số ngẫu nhiên.
nextFloat()	Trả về một giá trị kiểu float kế tiếp giữa 0.0F và 1.0F từ bộ tạo số ngẫu nhiên.
nextGaussian()	Trả về giá trị kiểu double được phân phối Gaussian kế tiếp từ bộ tạo số ngẫu nhiên. Tạo ra các giá trị Gaussian sẽ có một giá trị trung bình của 0, và một độ lệch tiêu chuẩn của 1.0.
nextInt()	Trả về giá trị kiểu Integer kế tiếp từ một bộ tạo số ngẫu nhiên.
nextLong()	Trả về giá trị kiểu long kế tiếp từ một bộ tạo số ngẫu nhiên.
setSeed(long)	Thiết lập giá trị khởi tạo từ bộ tạo số ngẫu nhiên.

Lớp Vector

Một trong các vấn đề với một mảng là chúng ta phải biết nó lớn như thế nào khi chúng ta tạo nó. Trong thực tế có nhiều trường hợp không thể xác định kích thước của mảng trước khi tạo nó.

Lớp Vector của Java giải quyết vấn đề này. Nó cung cấp một dạng mảng với kích thước ban đầu, mảng này có thể tăng thêm khi nhiều phần tử được thêm vào. Một lớp Vector lưu trữ các mục là kiểu Object, nó có thể dùng để lưu trữ các thể hiện của bất kỳ lớp Java nào. Một lớp Vector có thể lưu trữ các phần tử khác nhau, các phần tử khác nhau này là thể hiện của các lớp khác nhau.

Tại bất kỳ thời điểm, một lớp Vector có dung lượng để lưu trữ một số lượng nào đó các phần tử. Khi một lớp Vector dùng hết dung lượng của nó, thì dung lượng của nó được gia tăng bởi một số lượng riêng cho Vector đó. Lớp Vector cung cấp ba phương thức xây dựng khác nhau mà có thể chúng ta chỉ định dung lượng khởi tạo, và tăng số lượng của một Vector, khi nó được tạo ra. Các phương thức xây dựng này được tóm tắt trong bảng sau:

Các phương thức xây dựng của lớp Vector

Phương thức xây dựng	Mục đích
Vector(int)	Tạo ra một lớp Vector mới với dung lượng ban đầu được chỉ định.
Vector(int, int)	Tạo ra một lớp Vector mới với dung lượng ban đầu được chỉ định, và lượng tăng.
Vector()	Tạo ra một lớp Vector mới với dung lượng khởi tạo mặc định, và lượng tăng mặc định.

Một phần được thêm vào một lớp Vector bằng cách sử dụng phương thức `addElement()`. Tương tự, một phần tử có thể được thay thế bằng cách sử dụng phương thức `setElementAt()`. Một lớp Vector có thể tìm kiếm bằng cách sử dụng phương thức `contains()`, phương thức này đơn giản chỉ tìm sự xuất hiện của một đối tượng trong Vector. Phương thức `elements()` trả về một tập hợp các đối tượng được lưu trữ trong lớp Vector. Các phương thức này và các phương thức thành viên khác của lớp Vector được tóm tắt trong bảng dưới đây:

Các phương thức lớp Vector

Phương thức	Mục đích
<code>addElement(Object)</code>	Thêm phần tử được chỉ định vào lớp Vector.
<code>capacity()</code>	Trả về dung lượng hiện thời của lớp Vector.
<code>clone()</code>	sao chép lớp vector, nhưng không phải là các phần tử của nó.
<code>contains(Object)</code>	Trả về True nếu lớp Vector chứa đối tượng được chỉ định.
<code>copyInto(Object [])</code>	Sao chép các phần tử của lớp Vector vào mảng được chỉ định.
<code>elementAt(int)</code>	Lấy phần tử vị trí được chỉ định.
<code>elements()</code>	Trả về một bảng liệt kê của các phần tử trong lớp Vector.
<code>ensureCapacity(int)</code>	Đảm bảo rằng lớp Vector có thể lưu trữ ít nhất dung lượng tối thiểu được chỉ định.
<code>firstElement()</code>	Trả về phần tử đầu tiên trong lớp Vector.
<code>indexOf(Object)</code>	Tìm kiếm lớp Vector, và trả về chỉ mục đầu tiên tìm thấy đối tượng.

<code>indexOf(Object, int)</code>	Tìm kiếm lớp Vector bắt đầu từ vị trí chỉ định, trả về vị trí đầu tiên tìm thấy.
<code>insertElementAt(Object, int)</code>	Chèn đối tượng được chỉ định tại vị trí được chỉ định.
<code>isEmpty()</code>	Trả về True nếu lớp Vector không có phần tử.
<code>lastElement()</code>	Trả về phần tử cuối cùng trong lớp Vector.
<code>lastIndexOf(Object)</code>	Tìm kiếm lớp Vector, và trả về chỉ mục của đối tượng tìm thấy cuối cùng.
<code>lastIndexOf(Object, int)</code>	Tìm kiếm lớp Vector bắt đầu tại số chỉ mục được chỉ định, và trả về chỉ mục của phần tử cuối cùng tìm thấy.
<code>removeAllElements()</code>	Xoá tất cả các phần tử từ lớp Vector.
<code>removeElement(Object)</code>	Xoá đối tượng được chỉ định từ lớp Vector.
<code>removeElementAt(int)</code>	Xoá đối tượng tại chỉ mục được chỉ định.
<code>setElementAt(Object, int)</code>	Thay thế đối tượng tại chỉ mục được chỉ định với đối tượng được chỉ định.
<code>setSize(int)</code>	Thiết lập kích thước của lớp Vector thành kích thước mới được chỉ định.
<code>Size()</code>	Trả về số của các phần tử hiện thời trong lớp Vector.
<code>toString()</code>	Trả về một chuỗi chứa nội dung của lớp Vector.
<code>trimToSize()</code>	Định lại kích thước của lớp Vector để di chuyển dung lượng thừa trong nó.

Chương trình sau tạo ra một lớp Vector vect. Nó chứa 6 phần tử: “Numbers In Words”, “One”, “Two”, “Three”, “Four”, “Five”. Phương thức `removeElement()` được sử dụng để xoá các phần tử từ vect.

```
import java.util.*; public class VectorImplementation {
public static void main(String args[]) { Vector vect = new
Vector(); vect.addElement("One"); vect.addElement("Two");
vect.addElement("Three"); vect.addElement("Four");
vect.addElement("Five"); vect.insertElementAt("Numbers In
Words",0); vect.insertElementAt("Four",4);
System.out.println("Size: "+vect.size());
System.out.println("Vector "); for(int i = 0;
i<vect.size(); i++) {
```

```

System.out.println(vect.elementAt(i)+" , "); }
vect.removeElement("Five"); System.out.println("");
System.out.println("Size: "+vect.size());
System.out.println("Vector "); for(int i =
0;i<vect.size();i++) { System.out.print(vect.elementAt(i)+
" , "); } } }

```

Quá trình hiển thị kết quả sẽ được mô tả như hình dưới.

```

D:\Temp>java VectorImplementation
Size: 7
Vector
Numbers In Words ,
One ,
Two ,
Three ,
Four ,
Four ,
Five ,

Size: 6
Vector
Numbers In Words , One , Two , Three , Four , Four ,
D:\Temp>_

```

Kết quả của chương trình minh họa lớp Vector.

Lớp StringTokenizer

Một lớp StringTokenizer có thể sử dụng để tách một chuỗi thành các phần tử (token) nhỏ hơn. Ví dụ, mỗi từ trong một câu có thể coi như là một token. Tuy nhiên, lớp StringTokenizer đã đi xa hơn việc phân tách các từ trong câu. Để tách ra các thành token ta có thể tùy biến chỉ ra một tập dấu phân cách các token khi khởi tạo đối tượng StringTokenizer. Nếu ta không chỉ ra tập dấu phân cách thì mặc định là dấu trắng (space, tab, ...). Ta cũng có thể sử dụng tập các toán tử toán học (+, *, /, và -) trong khi phân tích một biểu thức.

Bảng sau tóm tắt 3 phương thức xây dựng của lớp StringTokenizer:

Các phương thức xây dựng của lớp StringTokenizer.

Phương thức xây dựng	Ý nghĩa
StringTokenizer(String)	Tạo ra một đối tượng StringTokenizer mới dựa trên chuỗi được chỉ định.

<code>StringTokenizer(String, String)</code>	Tạo ra một đối tượng <code>StringTokenizer</code> mới dựa trên (<code>String</code> , <code>String</code>) chuỗi được chỉ định và một tập các dấu phân cách.
<code>StringTokenizer(String, String, boolean)</code>	Tạo ra một đối tượng <code>StringTokenizer</code> dựa trên chuỗi được chỉ định, một tập các dấu phân cách, và một cờ hiệu cho biết nếu các dấu phân cách sẽ được trả về như các token hay không.

Các phương thức xây dựng ở trên được sử dụng trong các ví dụ sau:

```
StringTokenizer st1 = new StringTokenizer("A Stream of words");
StringTokenizer st2 = new StringTokenizer("4*3/2-1+4", "+-*/", true);
StringTokenizer st3 = new StringTokenizer("aaa,bbbb,ccc", ",", true);
```

Trong câu lệnh đầu tiên, `StringTokenizer` của “st1” sẽ được xây dựng bằng cách sử dụng các chuỗi được cung cấp và dấu phân cách mặc định. Dấu phân cách mặc định là khoảng trắng, tab, các ký tự xuống dòng. Các dấu phân cách này thì chỉ sử dụng khi phân tách văn bản, như với “st1”.

Câu lệnh thứ hai trong ví dụ trên xây dựng một đối tượng `StringTokenizer` cho các biểu thức toán học bằng cách sử dụng các ký hiệu `*`, `+`, `/`, và `-`.

Câu lệnh thứ 3, `StringTokenizer` của “st3” sử dụng dấu phẩy như một dấu phân cách.

Lớp `StringTokenizer` cài đặt giao diện `Enumeration`. Vì thế, nó bao gồm các phương thức `hasMoreElements()` và `nextElement()`. Các phương thức có thể sử dụng của lớp `StringTokenizer` được tóm tắt trong bảng sau:

Các phương thức lớp `StringTokenizer`

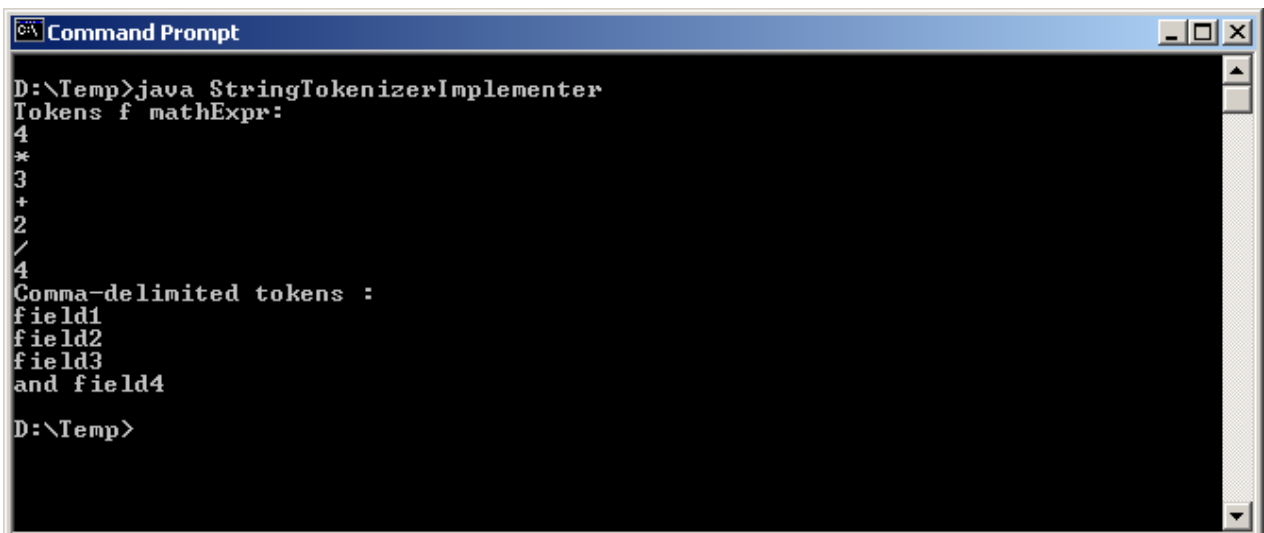
Phương thức	Mục đích
<code>countTokens()</code>	Trả về số các token còn lại.
<code>hasMoreElements()</code>	Trả về <code>True</code> nếu còn có token đang được đánh dấu trong chuỗi. Nó thì giống hệt như <code>hasMoreTokens</code> .
<code>hasMoreTokens()</code>	Trả về <code>True</code> nếu còn có token đang được đánh dấu trong chuỗi. Nó giống hệt như <code>hasMoreElements</code> .
<code>nextElement()</code>	Trả về token kế tiếp trong chuỗi. Nó thì giống như <code>nextToken</code> .
<code>nextToken()</code>	Trả về Token kế tiếp trong chuỗi. Nó thì giống như <code>nextElement</code> .

nextToken(String)	Thay đổi bộ dấu phân cách bằng chuỗi được chỉ định, và sau đó trả về token kế tiếp trong chuỗi.
-------------------	---

Hãy xem xét chương trình đã cho ở bên dưới. Trong ví dụ này, hai đối tượng StringTokenizer đã được tạo ra. Đầu tiên, “st1” được sử dụng để phân tách một biểu thức toán học. Thứ hai, “st2” phân tách một dòng của các trường được phân cách bởi dấu phẩy. Cả hai tokenizer, phương thức hasMoreTokens() và nextToken() được sử dụng để duyệt qua tập các token, và hiển thị các token.

```
import java.util.*; public class
StringTokenizerImplementer { public static void
main(String args[]) { // đặt một biểu thức toán học và tạo
một tokenizer cho chuỗi đó. String mathExpr = "4*3+2/4";
StringTokenizer st1 = new StringTokenizer(mathExpr, "+/-",
true); //trong khi vẫn còn các token, hiển thị
System.out.println("Tokens of mathExpr: ");
while(st1.hasMoreTokens())
System.out.println(st1.nextToken()); //tạo một chuỗi của
các trường được phân cách bởi dấu phẩy và tạo //một
tokenizer cho chuỗi. String commas =
"field1,field2,field3,and field4"; StringTokenizer st2 =
new StringTokenizer(commas,",",false); //trong khi vẫn còn
token, hiển thị. System.out.println("Comma-delimited
tokens : "); while (st2.hasMoreTokens())
System.out.println(st2.nextToken()); } }
```

Kết quả chạy chương trình được mô tả như hình dưới.



```

C:\> Command Prompt

D:\Temp>java StringTokenizerImplementer
Tokens of mathExpr:
4
*
3
+
2
/
4
Comma-delimited tokens :
field1
field2
field3
and field4
D:\Temp>
```

Tóm tắt bài học

- Khi không có sự thi hành để thừa kế, một giao diện được sử dụng thay cho một lớp trừu tượng.
- Một gói là một thư mục để tổ chức các giao diện và các lớp của bạn.
- CLASSPATH là một danh sách của các thư mục mà JVM tìm kiếm các tập tin lớp.
- Lớp java.lang.Math cung cấp các phương thức để thực hiện các hàm toán học.
- Các kiểu dữ liệu nguyên thủy có thể được xử lý truy cập thông qua các lớp trình bao bọc (Wrapper) của chúng.
- Các lớp String được sử dụng để tạo và xử lý chuỗi, các chuỗi có thể được gán, có thể được so sánh và được nối vào nhau.
- String Pool đại diện cho tất cả các chuỗi đã được tạo ra trong một chương trình.
- Lớp StringBuffer cung cấp các phương thức khác nhau để xử lý chuỗi. Các đối tượng của lớp này thì linh động hơn. Đó là, các ký tự hoặc các chuỗi có thể được chèn vào giữa đối tượng StringBuffer, hoặc được nối vào vị trí cuối cùng của chuỗi.
- Lớp Runtime đóng gói môi trường thời gian chạy.
- Lớp System cung cấp các tiện ích như là, xuất, nhập chuẩn, và các luồng lỗi.
- java.util chứa các lớp sau:
 - Hashtable
 - Random
 - Vector
 - StringTokenizer
- Lớp Hashtable có thể được sử dụng để tạo một mảng của các khoá và các giá trị. Nó cho phép các phần tử được tra cứu bởi khoá hoặc giá trị.
- Lớp Random là một bộ tạo số giả ngẫu nhiên mà có thể trả về các giá trị kiểu integer, dấu phẩy động (floating-point), hoặc phân bố Gaussian.
- Lớp Vector có thể sử dụng để lưu trữ bất kỳ các đối tượng nào. Nó có thể lưu trữ các đối tượng của nhiều lớp khác nhau.
- Lớp StringTokenizer cung cấp một cơ chế mềm dẻo cho việc phân tách các chuỗi.

Kiểm tra sự tiến bộ

-luôn là lệnh đầu tiên trước các lệnh: import, class trong chương trình Java.
- Một giao diện có thể chứa nhiều các phương thức. (**Đúng/Sai**)
- Trong khi tạo gói, thì mã nguồn phải nằm trong thư mục có tên như tên gói. (**Đúng/Sai**)

-là một danh sách của các thư mục, mà JVM sẽ tìm kiếm các tập tin lớp.
- Lớp bao bọc (wrapper) cho các kiểu dữ liệu double và long cung cấp hai hằng số làvà.....
-phương thức được sử dụng để thay thế một ký tự trong lớp StringBuffer bằng một ký tự khác tại vị trí được chỉ định.
-được sử dụng để ánh xạ các khoá với các giá trị.
- Phương thức..... của lớp StringTokenizer trả về số token còn lại.

Bài tập

Tạo một giao diện và sử dụng nó trong một chương trình của Java để hiển thị bình phương và lũy thừa 3 của một số.

Tạo một gói và viết một hàm, hàm đó trả về giai thừa của một đối số được truyền vào trong một chương trình.

Viết một chương trình bằng cách sử dụng các hàm của lớp Math để hiển thị bình phương của các số lớn nhất và nhỏ nhất của một tập các số được nhập vào bởi người sử dụng tại dòng lệnh.

Hãy tạo ra sổ ghi nhớ của chính bạn, nơi mà những con số được nhập vào như sau:

Joy	34543
Jack	56765
Tina	34567

Chương trình phải làm như sau:

- Kiểm tra xem số 3443 có tồn tại trong sổ ghi nhớ của bạn hay không.
- Kiểm tra xem mẫu tin của Jack có hiện hữu trong sổ ghi nhớ của bạn hay không.
- Hiển thị số điện thoại của Tina.
- Xoá số điện thoại của Joy.
- Hiển thị các mẫu tin còn lại.

Viết một chương trình mà nhập vào một số điện thoại tại dòng lệnh, như một chuỗi có dạng (091) 022-6758080. Chương trình sẽ hiển thị mã quốc gia (091), mã vùng (022), và số điện thoại (6758080) (Sử dụng lớp StringTokenizer).

AWT

Giới thiệu về AWT

Các ứng dụng phần mềm hiện nay rất thân thiện vì được trình bày nhiều màn hình giao diện đồ họa đẹp mắt. Các ngôn ngữ lập trình hiện nay cung cấp các đối tượng đồ họa, chúng có thể được điều khiển bởi người lập trình, hay bởi người sử dụng. Một trong số những kết quả quan trọng nhất chính là các ngôn ngữ hiện nay được dựa trên Giao diện người dùng đồ họa (Graphical User Interface - GUI). Trong chương này, ta sẽ thảo luận về Java hỗ trợ tính năng GUI cùng các sự thi hành của chúng.

GUI cung cấp chức năng nhập liệu theo cách thân thiện với người dùng. GUI đa dạng từ ứng dụng đến ứng dụng và có thể chứa nhiều điều khiển như hộp văn bản, nhãn, hộp danh sách hay các điều khiển khác. Các ngôn ngữ lập trình khác nhau cung cấp nhiều cách khác nhau để tạo GUI. Các ngôn ngữ như VB hay VC++ có thể cung cấp chức năng kéo và thả trong khi đó phần mềm giống như C++ yêu cầu người lập trình phải viết toàn bộ mã để xây dựng GUI.

Một phần tử (element) GUI được thiết lập bằng cách sử dụng thủ tục sau:

- Tạo đối tượng
- Xác định sự xuất hiện ban đầu của đối tượng
- Chỉ ra nó nằm ở đâu
- Thêm phần tử vào giao diện trên màn hình

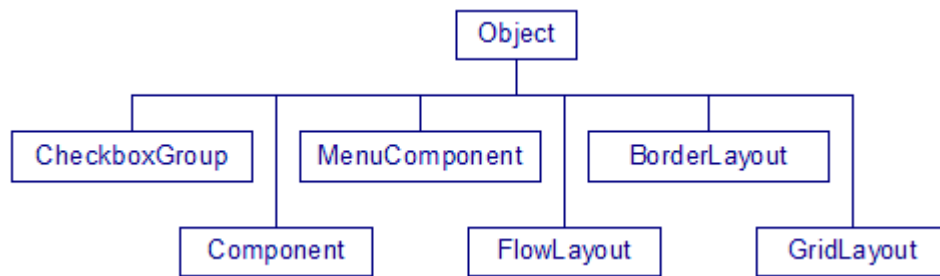
Một thành phần (component) GUI là một đối tượng trực quan. Người dùng tương tác với đối tượng này thông qua con trỏ chuột hay bàn phím. Các thành phần như là button, label v.v... có thể được nhìn thấy trên màn hình. Bất kỳ cái gì chung cho tất cả các thành phần GUI đều được tìm thấy trong lớp Component. Để tạo các đối tượng GUI chúng ta cần nhập gói **java.awt**.

AWT là viết tắt của Abstract Windowing Toolkit. AWT là một bộ các lớp trong Java cho phép chúng ta tạo GUI và chấp nhận các nhập liệu của người dùng thông qua bàn phím và chuột. AWT cung cấp các thành phần khác nhau để tạo GUI hiệu quả và lôi cuốn người sử dụng. Các thành phần này có thể là:

- Vật chứa (**Container**)
- Thành phần (**Component**)
- Trình quản lý cách trình bày (**Layout manager**)
- Đồ họa (**Graphic**) và các tính năng vẽ (**draw**)
- Phong chữ (**Font**)

- Sự kiện (**Event**)

Gói AWT chứa các lớp, giao diện và các gói khác. Hình sau đây mô tả một phần nhỏ của hệ thống phân cấp lớp AWT.



Hệ thống cây phân cấp lớp AWT

Container (Vật chứa)

Container là vùng mà bạn có thể đặt các thành phần giao diện của bạn vào đó. Bất cứ vật gì mà kế thừa từ lớp Container sẽ là vật chứa. **Applet** là một vật chứa, applet được dẫn xuất từ Panel, lớp Panel lại được dẫn xuất từ lớp Container.

Một vật chứa có thể chứa nhiều phần tử, các phần tử này có thể được vẽ hay được tô màu tùy thích. Bạn hãy xem vật chứa như một cửa sổ. Như khung (**frame**), pane, latch, hook, và các thành phần có kích thước nhỏ hơn khác.

Gói java.awt chứa một lớp gọi là **Container**. Lớp này trực tiếp hay gián tiếp phái sinh ra hai vật chứa được sử dụng phổ biến nhất là Frame và Panel.

Frame và Panel là các vật chứa thường được sử dụng. Frame là cửa sổ độc lập nhưng ngược lại Panel là vùng nằm trong cửa sổ khác. Panel không có các đường biên, chúng được trình bày trong một cửa sổ do trình duyệt hay appletviewer cung cấp. **Appletviewer** là một công cụ được JDK hỗ trợ để xem các applet. **Frame** là lớp con của Window. Chúng được trình bày trong một cửa sổ độc lập, cửa sổ này có chứa các đường biên xung quanh.

Frame

Frame không phụ thuộc vào applet và trình duyệt. **Frame** có thể hoạt động như một vật chứa hay như một thành phần (**component**). Bạn có thể sử dụng một trong những **constructor** sau để tạo một **frame**:

- **Frame()**: Tạo một frame nhưng không hiển thị (invisible)
- **Frame(String title)**: Tạo một frame không hiển thị, có tiêu đề.

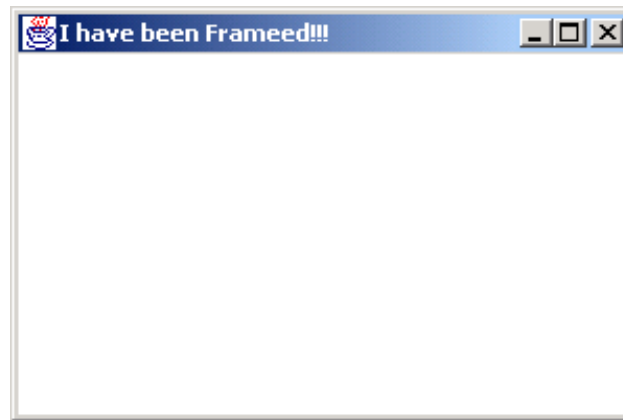
Chương trình dưới đây sẽ minh họa cách tạo một Frame.

```
import java.awt.*; class FrameDemo extends Frame { public
FrameDemo(String title) { super(title); } public static
void main(String args[]) { FrameDemo f=new FrameDemo("I
have been Framed!!!"); f.setSize(300,200);
f.setVisible(true); } }
```

Lớp được định nghĩa **Framedemo** là một lớp con của lớp **Frame**. Lớp FrameDemo này có một phương thức khởi tạo, trong phương thức khởi tạo này ta cho gọi phương thức **super()**. Nó sẽ gọi phương thức khởi tạo của lớp cha (trong trường hợp này là **Frame**). Mục đích của **super()** là gọi phương thức khởi tạo của lớp cha. Nó sẽ tạo một đối tượng của lớp con, lớp con này sẽ tạo **Frame**. Tuy nhiên, Frame vẫn không nhìn thấy được

và không có kích thước. Để làm được điều này, ta sử dụng hai phương thức nằm trong phương thức main: **setSize()** và **setVisible()**.

Kết xuất của chương trình giống như **hình 1**



Frame

Panel

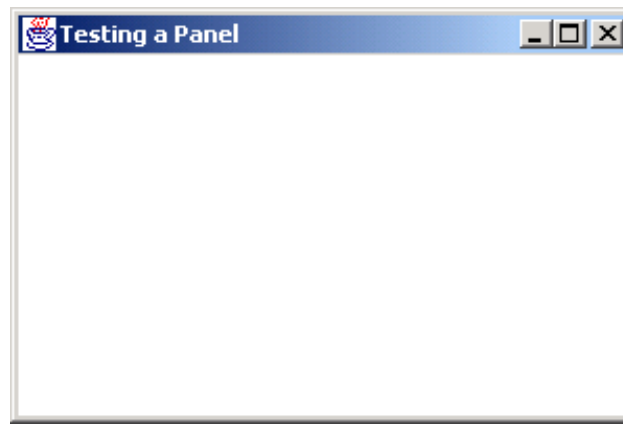
Panel được sử dụng để nhóm một số các thành phần lại với nhau. Cách đơn giản nhất để tạo một panel là sử dụng phương thức khởi tạo của nó, hàm **Panel()**.

Chương trình sau sẽ chỉ ra cách tạo một panel:

```
import java.awt.*; class Paneltest extends Panel { public
static void main(String args[]) { Paneltest p=new
Paneltest(); Frame f=new Frame("Testing a Panel");
f.add(p); f.setSize(300,200); f.setVisible(true); } public
Paneltest() { } }
```

Panel không thể được nhìn thấy trực tiếp. Do đó, chúng ta cần thêm **panel** đến một **frame**. Vì vậy ta cần tạo một frame mới và thêm **Panel** mới được tạo này vào đó. Tuy nhiên, **frame** sẽ không nhìn thấy được, và không có kích thước. Chúng ta sử dụng hai phương thức trong phương thức main – **setSize()** và **setVisible()** để thiết lập kích thước và hiển thị **frame**.

Kết xuất của chương trình:



Panel

Dialog

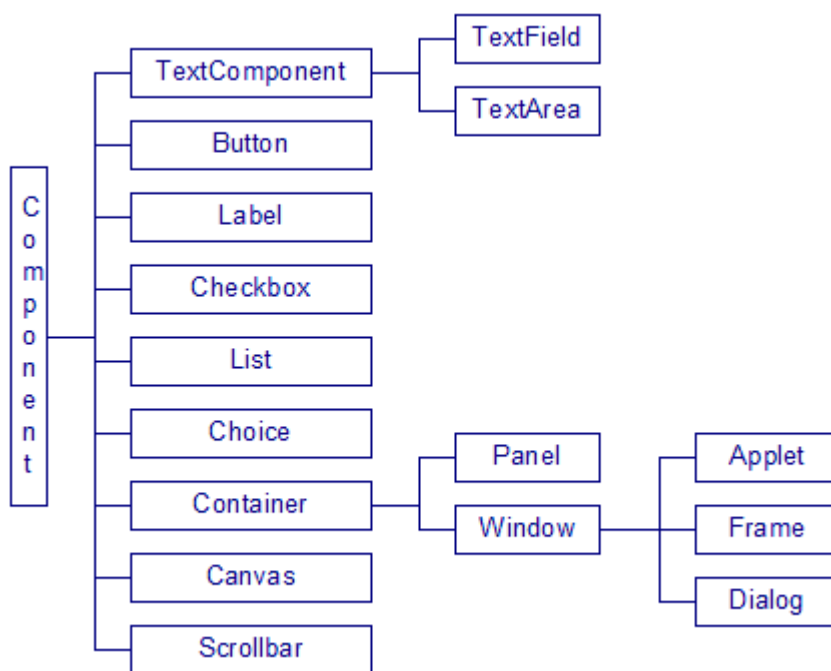
Lớp '**Dialog**' tương tự như lớp **Frame**, nghĩa là **Dialog** là lớp con của lớp Window. Đối tượng dialog được tạo như sau:

```
Frame myframe=new Frame("My frame"); // calling frame
String title = "Title"; boolean modal = true; // whether
modal or not Dialog dlg=new Dialog(myframe, title, modal);
```

Tham số '**modal**' chỉ ra rằng dialog sẽ ngăn chặn bất kỳ tương tác nào xảy đến với các cửa sổ được mở khác, trong khi **dialog** đang được hiển thị trên màn hình. Kiểu hộp thoại này ngăn chặn người dùng tương tác với các cửa sổ khác (của cùng ứng dụng) trên màn hình, cho tới khi dialog được đóng lại.

Thành phần (Component)

Một component có thể được đặt trên giao diện người dùng, có thể được thay đổi kích thước hay làm cho nhìn thấy, ẩn. Ví dụ được dùng phổ biến nhất là Textfield, Label, Checkbox, Textarea v.v... Và các thành phần cao cấp khác như Scrollbar, Scrollpane và Dialog. Tuy nhiên chúng không được sử dụng thường xuyên.



Các lớp thành phần

Bây giờ chúng ta hãy xét một số thành phần thường được sử dụng.

Nhãn (Label)

Lớp này được sử dụng để trình bày một String. Nó không thể được sửa đổi. Đây là một chuỗi chỉ đọc. Sử dụng một trong những constructor sau đây để tạo một label:

- **Label()** Tạo một Label trống.
- **Label(String labeltext)** Tạo một Label với nội dung được cho.
- **Label(String labeltext, int alignment)** Tạo một Label với một chế độ canh lề (alignment) , canh lề có thể là Label.LEFT, Label.RIGHT hay Label.CENTER.

Các phương thức được sử dụng phổ biến của label được trình bày ở bảng bên dưới:

Các phương thức của Label

Phương thức	Chức năng
-------------	-----------

setFont(Font f)	Thay đổi phông chữ của Label
setText(String s)	Thiết lập nhãn cho Label
getText()	Lấy nội dung hiện tại của nhãn

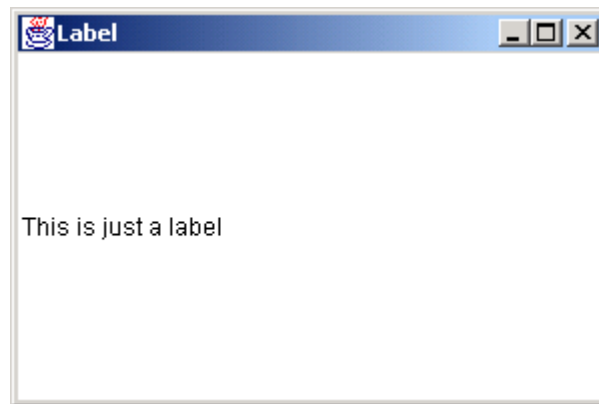
Cách sử dụng của Label

```
import java.awt.*; class Labeltest extends Frame { Label label1=new Label("This is just a label"); public Labeltest(String title) { super(title); add(label1); } public static void main(String args[]) { Labeltest f=new Labeltest("Label"); f.setSize(300,200); f.show(); } }
```

`label1=new Label("This is just a label");` ; Tạo đối tượng Label

`add(label1);` Label sẽ hiển thị chỉ khi nó được thêm vào container. Ở đây, Frame là container mà thành phần Label được thêm vào. Việc này được thực hiện bằng cách sử dụng phương thức **add()**.

Kết xuất của chương trình được chỉ ra ở **hình 2**



Label

Ô văn bản (TextField)

Một **TextField** là một vùng chỉ chứa một dòng văn bản, trong đó văn bản có thể được hiển thị hay được nhập vào bởi người dùng. Trong Java, một trong những **constructor** sau có thể được sử dụng để tạo một **TextField**:

- **TextField()**: Tạo một textfield mới.
- **TextField(int columns)**: Tạo một **textfield** mới với số cột được cho trước.
- **TextField(Strings)**: Tạo một **textfield** mới với chuỗi văn bản được cho trước.
- **TextField(Strings, intcolumns)**: Tạo một **textfield** mới với nội dung và số cột được cho trước.

Các phương thức thường được sử dụng của đối tượng **TextField** được tóm tắt trong bảng sau:

Các phương thức của TextField

Phương thức	Chức năng
setEchoChar(char)	Đặt các kí tự được hiện ra thay thế ký tự nhập vào.
setText(String s)	Gán nội dung cho TextField.
getText()	Lấy nội dung của TextField.
setEditable(boolean)	Xác định TextField có soạn thảo được hay không. Nó chỉ được soạn thảo khi giá trị tham số truyền vào là True.
isEditable()	Xác định xem trường có đang trong mode soạn thảo hay không. Giá trị trả về kiểu Boolean.

Cách sử dụng TextField

```
import java.awt.*; class TextFieldTest extends Frame { TextField tf1=new
TextField(30); public TextFieldTest(String title) { super(title); setLayout(new
FlowLayout()); add(tf1); } public static void main(String args[]) { TextFieldTest f=new
TextFieldTest("TextField"); f.setSize(300,200); f.show(); } }
```

Trong chương trình này, chúng ta sử dụng phương thức `setLayout()` để thay đổi cách trình bày của các thành phần trên vật chứa. Layout manager có chức năng sắp xếp các thành phần trong một vật chứa.

Kết xuất của chương trình được chỉ ra ở hình bên dưới:



TextField

Vùng văn bản (TextArea)

Một Textarea được sử dụng khi văn bản nhập vào có trên hai hay nhiều dòng. Textarea có một scrollbar. TextArea là một trường văn bản có thể được soạn thảo với nhiều dòng.

Để tạo một Textarea, làm theo các bước sau:

1. Tạo một đối tượng.
2. Chỉ ra số dòng, số cột đối tượng này cần có.
3. Bố trí phần tử này trên màn hình.

Trong Java, bạn có thể sử dụng các constructor sau để tạo TextArea:

- **TextArea():** Tạo một TextArea mới.
- **TextArea(int rows, int cols):** Tạo một TextArea mới với số lượng cột và dòng được cho trước.
- **TextArea(String text):** Tạo một TextArea mới với nội dung được cho trước.
- **TextArea(String text, int rows, int cols):** Tạo một TextArea mới với nội dung, số dòng và số cột được cho trước.

Các phương thức thường được sử dụng nhiều nhất của TextArea:

Các phương thức của TextArea

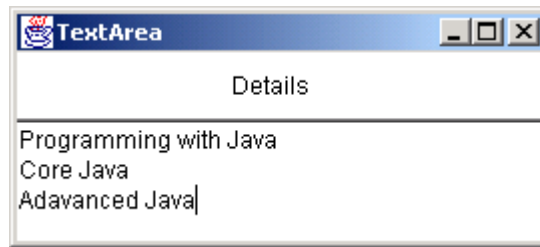
Phương thức	Chức năng
setText(String)	Gán nội dung cho TextArea.
getText()	Trả về nội dung của TextArea.
setEditable(boolean)	Xác định xem TextArea có thể được soạn thảo hay không. TextArea có thể được soạn thảo khi giá trị này là True.
isEditable()	Xác định xem TextArea có đang trong chế độ soạn thảo được không. Trả về giá trị là kiểu Boolean.
insertText(String, int)	Chèn chuỗi được vào vị trí được cho trước.
replaceText(String, int, int)	Thay thế văn bản nằm giữa vị trí int, int cho trước.

Cách sử dụng TextArea

```
import java.awt.*; class TextAreatest extends Frame { Label lbl=new Label("Details");
TextArea ta1=new TextArea(); public TextAreatest(String title) { super(title);
setLayout(new FlowLayout()); add(lbl); add(ta1); } public static void main(String
```

```
args[]) { TextArea t=new TextArea("TextArea"); t.setSize(300,200); t.show(); }
}
```

Kết xuất của chương trình được chỉ ra ở hình bên dưới:



TextArea

Button (nút ấn)

Nút ấn hay còn gọi là nút lệnh là một phần không thể thiếu của bất kỳ GUI nào. Sử dụng button là cách dễ nhất để nhận các tác động của người dùng.

Để tạo một button, bạn làm theo các bước sau:

1. Tạo phần tử Button với một nhãn chỉ ra mục đích của Button.
2. Bố trí phần tử này trên màn hình.
3. Hiển thị phần tử trên màn hình.

Sử dụng một trong hai constructor sau để tạo các button trong Java:

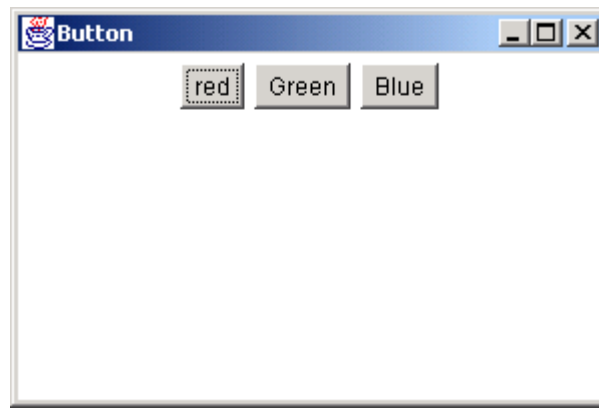
- Button()
- Button(String text)

Sử dụng setLabel() và getLabel() để thiết lập và lấy giá trị nhãn của button.

Chương trình minh họa tạo ra 3 button

```
import java.awt.*; class Buttontest extends Frame { Button b1 = new Button("red");
Button b2 = new Button("Green"); Button b3 = new Button("Blue"); public
Buttontest(String title) { super(title); setLayout(new FlowLayout()); add(b1); add(b2);
add(b3); } public static void main(String args[]) { Buttontest t= new
Buttontest("Button"); t.setSize(300,200); t.show(); } }
```

Kết xuất của chương trình được chỉ ra ở **hình 5**



Button

Checkbox và RadioButton

Checkbox được sử dụng khi người dùng tiến hành chọn một hay nhiều tùy chọn. Người dùng phải click trên các checkbox để chọn hay bỏ chọn chúng. Một radiobutton cũng tương tự như một checkbox. Nó được sử dụng như một option button để xác định các chọn lựa. Bạn chỉ có thể chọn một option trong nhóm các nút radiobutton, ngược lại bạn có thể chọn nhiều hơn một checkbox tại một thời điểm.

Làm theo các bước sau để tạo các checkbox hay radiobutton:

1. Tạo phần tử.
2. Xác định trạng thái khởi đầu của phần tử (chọn hay không chọn).
3. Bố trí các phần tử trên màn hình.
4. Hiển thị các phần tử trên màn hình.

Thành phần checkbox có thể sử dụng một lớp phụ được gọi là CheckboxGroup để tạo ra các radiobutton.

Sử dụng các constructor sau để tạo các checkbox trong Java:

- **Checkbox():** Tạo một checkbox trống.
- **Checkbox(String text):** Tạo một checkbox với nhãn được cho.

Để tạo các radiobutton, đầu tiên chúng ta tạo đối tượng CheckboxGroup như sau:

```
CheckboxGroup cg=new CheckboxGroup();
```

Sau đó chúng ta tạo các đối tượng, như sau:

```
Checkbox male=new Checkbox("male", cg, true);  
Checkbox female=new Checkbox("female", cg, false);
```

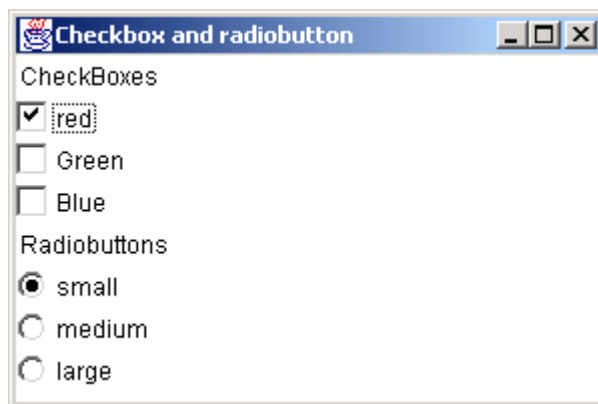
Chúng ta sử dụng các phương thức **setState()** và **getState()** để thiết lập và nhận về trạng thái của checkbox.

Cách sử dụng các checkbox và các radiobutton

```
import java.awt.*; class Checkboxtest extends Frame { Label l1=new
Label("CheckBoxes"); Checkbox b1=new Checkbox("red",true); Checkbox b2=new
Checkbox("Green",false); Checkbox b3=new Checkbox("Blue",false); Label l2=new
Label("Radiobuttons"); CheckboxGroup cb=new CheckboxGroup(); Checkbox
b4=new Checkbox("small",cb,true); Checkbox b5=new Checkbox("medium",cb,false);
Checkbox b6=new Checkbox("large",cb,false); public Checkboxtest(String title) {
super(title); setLayout(new GridLayout(8,1)); add(l1); add(b1); add(b2); add(b3);
add(l2); add(b4); add(b5); add(b6); } public static void main(String args[]) {
Checkboxtest t=new Checkboxtest("Checkbox and radiobutton"); t.setSize(300,200);
t.show(); } }
```

Đầu tiên chúng ta tạo một đối tượng Frame, đối tượng này hoạt động như một container sẽ chứa thành phần checkbox mà ta đã tạo. Sau đó ta tạo 6 checkbox, 02 checkbox được đánh dấu chọn. Để làm được điều này, ta đưa giá trị true như một tham số cho hàm constructor Checkbox, ngoài ra còn có một tham số String là nhãn của checkbox. Để hiển thị các điều khiển này theo dạng lưới, ta phải thiết lập cách trình bày về dạng GridLayout có 8 dòng và 1 cột. Cuối cùng, ta tạo một biểu hiện cho lớp Checkboxtest và thiết lập kích thước cho Frame. Để hiển thị nó, ta cho gọi phương thức show().

Kết xuất được chỉ ra ở hình bên dưới:



Checkbox

Danh sách chọn lựa (Choice List)

Thỉnh thoảng, rất cần thiết để trình bày một danh sách các chọn lựa đến người dùng trên một GUI. Người dùng có thể click vào một hay nhiều mục từ danh sách. Một danh sách chọn lựa được tạo bằng cách sử dụng một số các chuỗi (String) hay các giá trị văn bản.

Để tạo các danh sách chọn lựa, hãy làm theo các bước được cho sau đây:

1. Tạo danh sách các phần tử.
2. Thêm các mục (có kiểu là String) vào danh sách, mỗi lần chỉ thêm được một mục.
3. Bố trí danh sách trên màn hình.
4. Hiển thị danh sách trên màn hình.

Java hỗ trợ lớp Choice cho phép chúng ta tạo các danh sách chứa nhiều mục. Khi danh sách vừa được tạo ra, nó sẽ rỗng.

```
Choice colors=new Choice();
```

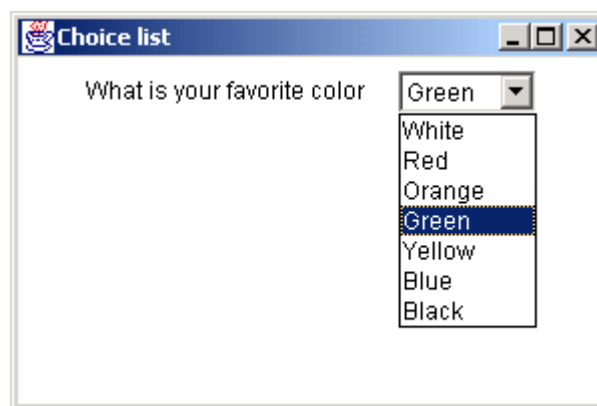
Mỗi thời điểm chỉ thêm được một item bằng cách sử dụng phương thức addItem như được chỉ ra bên dưới:

```
colors.addItem("Red"); colors.addItem("Green");
```

Cách tạo một danh sách chọn lựa

```
import java.awt.*; class Choicetest extends Frame { Label l1=new Label("What is your  
favorite color"); Choice colors=new Choice(); public Choicetest(String title) {  
super(title); setLayout(new FlowLayout()); add(l1); colors.addItem("White");  
colors.addItem("Red"); colors.addItem("Orange"); colors.addItem("Green");  
colors.addItem("Yellow"); colors.addItem("Blue"); colors.addItem("Black");  
add(colors); } public static void main(String args[]) { Choicetest t=new  
Choicetest("Choice list"); t.setSize(300,200); t.show(); } }
```

Kết xuất được chỉ ra ở hình bên dưới:



Danh sách chọn lựa

Thực đơn (Menu)

Ngôn ngữ Java có một tập hợp các lớp đối tượng để tạo các menu. Có hai loại menu – pull down và pop-up. Menu làm cho ứng dụng ta xây dựng dễ sử dụng hơn. Ta chỉ có đặt duy nhất một thanh menubar trong một frame. Menubar là một thanh nằm ngang được đặt tại đỉnh của frame. Nó liệt kê các mục chọn khác nhau hay còn gọi là menu. Một menu độc lập có thể chứa các mục chọn con, các mục con này được gọi là Menu Item. Java cung cấp các Checkbox MenuItem, chúng có thể được bật hay mở, phụ thuộc vào trạng thái. Chương trình sau sẽ minh họa cách sử dụng của menubar, menu, menuItem, và CheckboxMenuItem.

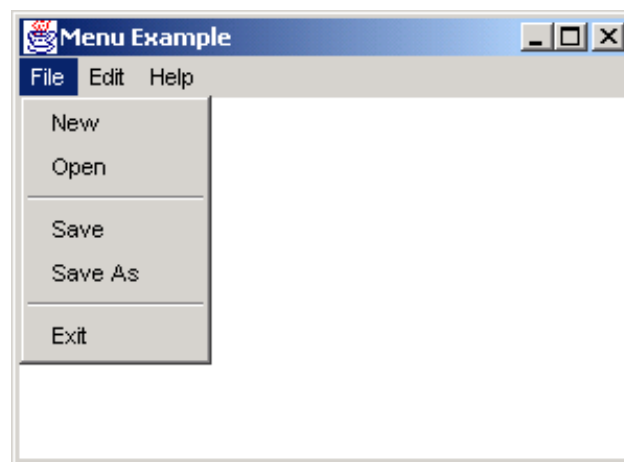
```
import java.awt.*; import java.awt.event.*; class MyFrame
extends Frame implements ActionListener, MouseListener {
MenuItem exitItem; PopupMenu optionsMenu; Frame frame;
public MyFrame() { setTitle("Menu Example");
setSize(300,200); MenuBar mbar=new MenuBar();
setMenuBar(mbar); Menu fileMenu=new Menu("File");
mbar.add(fileMenu); fileMenu.addActionListener(this);
MenuItem newItem=new MenuItem("New");
fileMenu.add(newItem); MenuItem openItem=new
MenuItem("Open"); fileMenu.add(openItem);
fileMenu.addSeparator(); MenuItem saveItem=new
MenuItem("Save"); fileMenu.add(saveItem); MenuItem
saveAsItem=new MenuItem("Save As");
fileMenu.add(saveAsItem); fileMenu.addSeparator();
exitItem=new MenuItem("Exit"); fileMenu.add(exitItem);
saveAsItem.addActionListener(this); Menu editMenu=new
Menu("Edit"); mbar.add(editMenu);
editMenu.addActionListener(this); MenuItem cutItem=new
MenuItem("Cut"); editMenu.add(cutItem); MenuItem
copyItem=new MenuItem("Copy"); editMenu.add(copyItem);
MenuItem pasteItem=new MenuItem("Paste");
editMenu.add(pasteItem); editMenu.addSeparator(); Menu
helpMenu=new Menu("Help"); mbar.add(helpMenu);
helpMenu.addActionListener(this); MenuItem contentItem=new
MenuItem("Content"); helpMenu.add(contentItem); MenuItem
indexItem=new MenuItem("Index"); helpMenu.add(indexItem);
Menu findMenu=new Menu("Find"); helpMenu.add(findMenu);
addMouseListener(this); MenuItem nameItem=new
MenuItem("Search by Name"); findMenu.add(nameItem);
```

```

MenuItem cacheItem=new MenuItem("Search from cache");
findMenu.add(cacheItem); optionsMenu=new
PopupMenu("Options"); editMenu.add(optionsMenu);
optionsMenu.addActionListener(this); MenuItem readItem=new
MenuItem("Read Only"); optionsMenu.add(readItem);
optionsMenu.addSeparator(); Menu formatMenu=new
Menu("Format text"); optionsMenu.add(formatMenu);
this.add(optionsMenu); formatMenu.addActionListener(this);
CheckboxMenuItem insertItem=new
CheckboxMenuItem("Insert",true);
formatMenu.add(insertItem); CheckboxMenuItem
overtimeItem=new CheckboxMenuItem("Overtime",false);
formatMenu.add(overtimeItem); } public void
actionPerformed(ActionEvent ae) { if
(ae.getActionCommand().equals("Exit")) { System.exit(0); }
} public void mouseEntered(MouseEvent m){} public void
mouseExited(MouseEvent m){} public void
mouseClicked(MouseEvent m) {
optionsMenu.show(this,m.getX(),m.getY()); } public void
mouseReleased(MouseEvent m){} public void
mousePressed(MouseEvent m){} public static void
main(String[] args) { MyFrame frame=new MyFrame();
frame.show(); } }

```

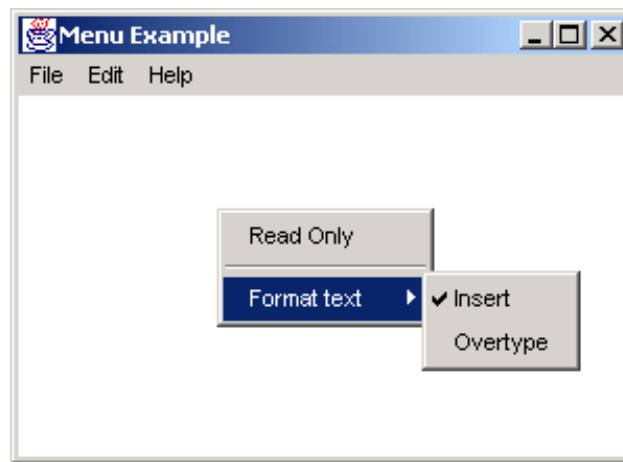
Khi bạn thực thi chương trình trên, một màn hình với các trình đơn File, Edit và Help được hiển thị. Khi bạn click vào mục File, bạn sẽ thấy kết xuất sau đây:



Pull-down Menu

Một menu có thể chứa các menu con. Khi bạn click vào trình đơn Help, 3 mục con có tên là Content, Index và Find sẽ xuất hiện. Trong trình đơn Find, có 2 mục con là Search

by name và Search from Cache. Mặt khác một pop-up menu sẽ hiện ra nếu bạn nhấn chuột phải trên màn hình:



Pop-up menu

Các mục chọn được trình bày trên pop-up menu là Read-Only và Format text. Mục 'Format text' có 2 mục con là Insert và Overtyping. Những mục chọn con này thuộc kiểu `CheckboxMenuItem`. Khi bạn click vào mục chọn, nó sẽ được đánh dấu và bạn có thể thấy dấu chọn tương ứng trên mục được chọn đó. Ngôn ngữ Java cung cấp các lớp khác nhau. Những lớp này được sử dụng để tạo thành Menubar, Menu, MenuItem và `CheckboxMenuItem` trong chương trình.

Tóm tắt

- GUI giúp chúng ta tạo giao diện hình ảnh cho một ứng dụng. Mặt khác nó cũng giúp ta phát triển các ứng dụng người dùng nhiều hiệu quả hơn.
- Thành phần GUI là một đối tượng trực quan. Người dùng có thể sử dụng chuột hay bàn phím để tương tác với đối tượng này.
- Các thành phần GUI như các button, label, checkbox và radio button mà được sử dụng trong ứng dụng hay applet thì có thể được thấy trên màn hình. Bất cứ thao tác nào mà liên quan tới tất cả các thành phần GUI đều được tìm thấy trong lớp `Component`. Ta cần sử dụng các lớp tồn tại trong gói `java.awt` để tạo các thành phần GUI này.
- Hệ thống GUI xử lý tất cả các tương tác của người dùng với sự hỗ trợ của mô hình hướng sự kiện (event-driven). Một sự kiện được kích hoạt khi người sử dụng tạo một hành động như là di chuyển chuột, nhấn phím, nhả phím v.v....
- Các kiểu trình bày khác nhau:
 - `FlowLayout`
 - `BorderLayout`
 - `CardLayout`
 - `GridLayout`
 - `GridBagLayout`

- Phương thức 'setLayout()' được sử dụng để đặt layout.
- FlowLayout là Layout Manager mặc định cho các applet và các panel. Các thành phần được sắp xếp từ góc trái trên đến góc phải bên dưới của màn hình.
- BorderLayout sắp xếp các thành phần trong 'North', 'South', 'East', 'West' và 'Center' của một container.
- GridLayout đặt các thành phần trong các dòng và các cột. Tất cả các thành phần đều có cùng kích thước.
- CardLayout đặt các thành phần trên đỉnh của các thành phần khác. Nó tạo một stack của một số thành phần, thường thường là các panel.
- GridBagLayout bố trí các thành phần một cách chính xác hơn các layout manager khác. Nó tương tự như grid layout. Sự khác nhau duy nhất ở đây là thành phần không cần có cùng kích thước và có thể được đặt trong bất kỳ dòng hay cột nào.
- Trong mô hình xử lý sự kiện, ứng dụng cho phép bạn đăng ký các handler được gọi là các listener cho các đối tượng.
- Một Event Listener lắng nghe một sự kiện cụ thể nào đó mà một đối tượng thiết lập. Nó sẽ gọi lần lượt các phương thức xử lý sự kiện. Lớp layout manager cung cấp một phương tiện để điều khiển cách trình bày vật lý của các thành phần GUI.
- Có hai kiểu menu – pull-down và pop-up.

Kiểm tra sự tiến bộ

1. Mỗi thành phần GUI là một đối tượng? (**Đúng/Sai**)
2. Vật chứa là gì?
3. TextArea là một hộp văn bản chỉ chứa được một dòng, trong đó có thể hiện thị văn bản hoặc có thể nhập vào bởi người sử dụng? (**Đúng/Sai**)
4. CheckBox sử dụng thêm lớp radioButtonGroup để tạo ra radiobuttons? (**Đúng/Sai**)
5. FlowLayout là cách trình bày mặc định của Applet và Panel? (**Đúng/Sai**)
6. BorderLayout là cách trình bày mặc định của Window, Frame và Dialog? (**Đúng/Sai**)
7. ActionEvent là gì?
8. Chỉ có thể đặt được duy nhất một thanh menu trên Frame? (**Đúng/Sai**)

Bài tập

Viết chương trình Java để nhập thông tin chi tiết về người sử dụng như hình sau:

Exercise

Name

Address

Sex ☒ Male ☐ Female Hobbies ☒ Reading ☐ Sports ☐ Travelling

Age group

Sửa bài tập 1 để có giao diện như sau:

Exercise

Name

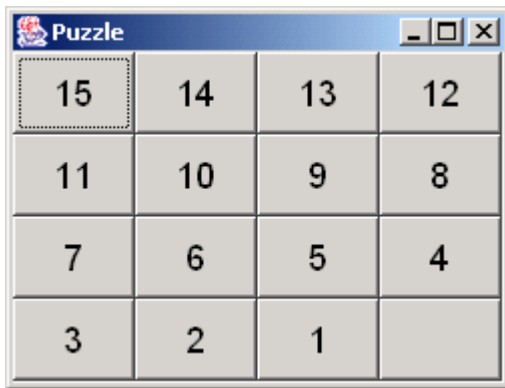
Address

Sex ☒ Male ☐ Female

Hobbies ☒ Reading ☐ Sports ☐ Travelling

Age group

Viết giao diện chương trình trò chơi puzzle như sau:



Sửa bài 3 để khi người sử dụng click vào nút cùng hàng hoặc cùng cột với nút không có số thì đổi chỗ nút vừa click và nút không số cho nhau. Khi người sử dụng ấn nút không cùng hàng hay không cùng cột thì hiển thị thông báo "Không hợp lệ" còn khi người sử dụng đã sắp xếp các nút theo thứ tự từ 1 đến 15 thì hiện ra thông báo chúc mừng.

Viết chương trình có menu pop-up như hình dưới đây:



Quản lý cách trình bày (Layout Manager)

Layout manager điều khiển cách trình bày vật lý của các phần tử GUI như là button, textbox, option button v.v... Một layout manager tự động bố trí các thành phần này trong container.

Các kiểu trình bày khác nhau:

- Flow layout
- Border layout
- Card layout
- Grid layout
- GridBag Layout

Tất cả các thành phần mà chúng ta vừa tạo sử dụng layout manager mặc định. Cho ví dụ, '**FlowLayout**' là cách trình bày mặc định của một applet. Layout manager này sẽ tự động sắp xếp các thành phần. Tất cả các thành phần được đặt trong một container, và được sắp xếp nhờ layout manager tương ứng. Layout manager được thiết lập bằng phương thức '**setLayout()**'.

Bây giờ chúng ta sẽ tìm hiểu chi tiết các cách trình bày và cách bố trí các thành phần của ta vào những vị trí mong muốn.

FlowLayout manager

'**FlowLayout**' là layout manager mặc định cho **Applet** và **Panel**. Các thành phần được sắp xếp từ góc trái trên đến góc phải dưới của màn hình. Khi một số thành phần được tạo, chúng được sắp xếp theo hàng, từ trái sang phải. Các **constructor** của **FlowLayout**:

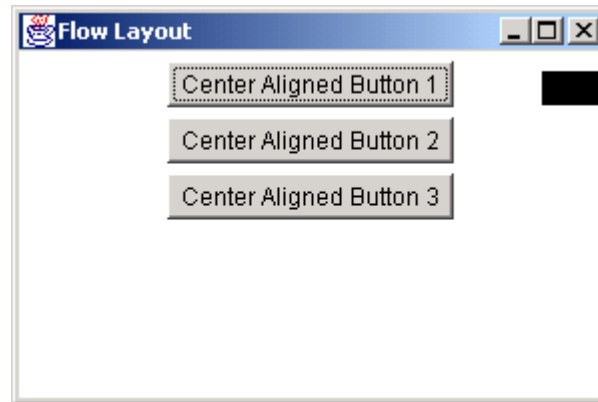
```
FlowLayout mylayout = new FlowLayout() // constructor
//constructor with alignment specified FlowLayout
exLayout=new FlowLayout(FlowLayout.RIGHT);
setLayout(exLayout); //setting the layout to Flowlayout
```

Các điều khiển có thể được canh về bên trái, bên phải hay ở giữa. Để canh các điều khiển về bên phải, bạn sử dụng cú pháp sau:

```
setLayout(new FlowLayout(FlowLayout.RIGHT));
Chương trình minh họa về Flowlayout Manager
import java.awt.*; class Fltest extends Frame { Button b1=new Button("Center
Aligned Button 1"); Button b2=new Button("Center Aligned Button 2"); Button
b3=new Button("Center Aligned Button 3"); public Fltest(String title) { super(title);
```

```
setLayout(new FlowLayout(FlowLayout.CENTER)); add(b1); add(b2); add(b3); }
public static void main(String args[]) { Fltest t=new Fltest("Flow Layout");
t.setSize(300,200); t.show(); } }
```

Kết xuất của chương trình chỉ ra ở [\[link\]](#).



Flowlayout

BorderLayout Manager

‘**BorderLayout**’ là layout manager mặc định cho ‘**Window**’, ‘**Frame**’ và ‘**Dialog**’. Layout này sắp xếp tối đa 5 thành phần trong một container. Những thành phần này có thể được đặt ở các hướng ‘North’, ‘South’, ‘East’, ‘West’ và ‘Center’ của container.

- **NORTH** – Đặt ở đỉnh của container.
- **EAST** – Đặt phía bên phải của container.
- **SOUTH** – Đặt ở phía dưới của container.
- **WEST** – Đặt phía bên trái của container.
- **CENTER** – Đặt ở giữa của container.

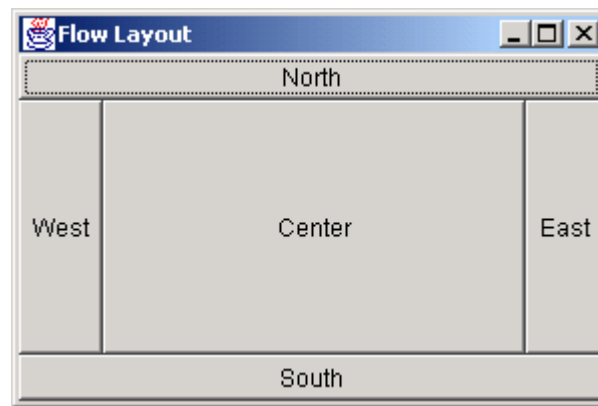
Để thêm một thành phần vào vùng ‘North’, bạn sử dụng cú pháp sau:

```
Button b1=new Button("North Button"); // khai báo thành
phần setLayout(new BorderLayout()); // thiết lập layout
add(b1,BorderLayout.NORTH); // thêm thành phần vào layout
```

Các thành phần vẫn giữ nguyên vị trí tương đối của chúng kể cả khi container bị thay đổi kích thước. Các thành phần được đặt trong vùng ‘North’, ‘South’ được dàn nằm ngang trong khi đó các thành phần đặt trong vùng ‘East’ và ‘West’ lại được dàn thẳng đứng. Các thành phần được đặt trong vùng ‘center’ sẽ được dàn đều vào những khu vực nằm giữa của container.

```
add(b2, BorderLayout.CENTER); // thêm thành phần vào vùng  
'center'
```

Khi tất cả các thành phần được đặt vào các vùng tương ứng, lúc đó Frame sẽ giống như sau:



BorderLayout

BorderLayout có thể chứa nhiều hơn 5 thành phần. Để thực hiện điều này, chúng ta có thể sử dụng các Panel với các layout khác nhau để chứa các thành phần, và sau đó đặt các panel này vào trong BorderLayout.

CardLayout Manager

CardLayout có thể lưu trữ một ngăn xếp (stack) các giao diện. Mỗi giao diện giống như một bảng (card). Bảng thường là đối tượng Panel. Một thành phần độc lập như button sẽ điều khiển cách trình bày các bảng ở lớp trên cùng.

Đầu tiên, chúng ta bố trí tập hợp các thành phần được yêu cầu trên các panel tương ứng. Mỗi panel sẽ được bố trí vào các layout khác nhau. Ví dụ:

```
panelTwo.setLayout(new GridLayout(2,1));
```

Panel chính sẽ chứa những panel này. Chúng ta thiết lập layout của panel chính là CardLayout như sau:

```
CardLayout card=new CardLayout();  
panelMain.setLayout(card);
```

Bước kế tiếp là thêm các panel khác vào panel chính:

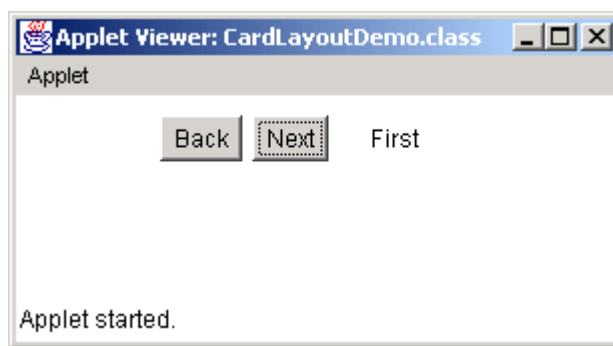
```
panelMain.add("Red Panel", panelOne); panelMain.add("Blue  
Panel", panelTwo);
```

Phương thức '**add()**' sử dụng hai tham số. Tham số đầu tiên là một String làm nhãn của panel và tham số thứ hai là tên đối tượng Panel.

Chương trình minh họa Cardlayout

```
import java.awt.*; import java.applet.*; /*<applet code="CardLayoutDemo.class"
width="300" height="100"></applet>*/ public class CardLayoutDemo extends Applet
{ Button back,next; Label lbl1,lbl2,lbl3,lbl4; TextField other1; Panel
p1,first,second,third,fourth; CardLayout c1; public void init() { back=new
Button("Back"); next=new Button("Next"); add(back); add(next); c1=new
CardLayout(); p1=new Panel(); p1.setLayout(c1); // Set panel layout to CardLayout
lbl1=new Label("First"); lbl2=new Label("Second"); lbl3=new Label("Third");
lbl4=new Label("Fourth"); //First panel first=new Panel(); first.add(lbl1); //Second
panel second=new Panel(); second.add(lbl2); //Third panel third=new Panel();
third.add(lbl3); //Fourth panel fourth=new Panel(); fourth.add(lbl4); //Add panels to the
card deck panel p1.add("1",first); p1.add("2",second); p1.add("3",third);
p1.add("4",fourth); add(p1); } }
```

Kết xuất của chương trình như sau:



CardLayout

Trong hình bên trên, các panel được thêm vào panel chính như là các thẻ riêng biệt. Vì thế chỉ có thẻ đầu tiên mới được thấy trên màn hình. Nhưng người dùng có thể điều hướng sang các panel khác sử dụng các phương thức của CardLayout.

GridLayout Manager

'GridLayout' trợ giúp việc chia container vào trong ô lưới. Các thành phần được đặt trong các ô giao của dòng và cột. Mỗi lưới nên chứa ít nhất một thành phần. Một lưới được sử dụng khi tất cả các thành phần có cùng kích thước.

GridLayout được tạo tạo như sau:

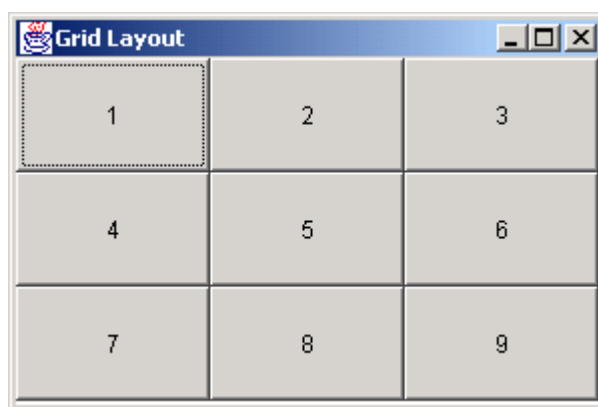
```
Gridlayout g1=new GridLayout(4,3);
```

4 là số dòng và 3 là số cột.

Chương trình minh họa cách trình bày lưới

```
import java.awt.*; class Glttest extends Frame { Button btn[]; String str[]={"1", "2",  
"3", "4", "5", "6", "7", "8", "9"}; public Glttest(String title) { super(title);  
setLayout(new GridLayout(3,3)); btn=new Button[str.length]; for (int i=0;  
i<str.length;i++) { btn[i]=new Button(str[i]); add(btn[i]); } } public static void  
main(String args[]) { Glttest t=new Glttest("Grid Layout"); t.setSize(300,200); t.show();  
} }
```

Kết xuất chương trình như sau:



GridLayout

GridBagLayout Manager

‘GridBagLayout’ là cách trình bày hiệu quả và phức tạp hơn bất cứ cách trình bày nào khác. Layout này đặt các thành phần vào vị trí chính xác. Với layout này, các thành phần không cần có cùng kích thước. Nó tương tự như GridLayout manager, khi các thành phần được sắp xếp trong lưới theo dòng và cột. Tuy nhiên, thứ tự đặt các thành phần không theo nguyên tắc từ trái sang phải và từ trên xuống dưới.

```
GridBagLayout gb=new GridBagLayout()  
ContainerName.setLayout (gb) ;
```

Để sử dụng layout này, bạn cần cung cấp thông tin về kích thước và vị trí của mỗi thành phần. Lớp ‘GridBagLayoutConstraints’ chứa tất cả các thông tin mà lớp GridLayout cần để bố trí và định kích thước mỗi thành phần. Bảng sau liệt kê danh sách các biến thành viên của lớp GridBagConstraints:

Các biến thành viên của lớp GridBagConstraints

Các biến thành viên	Mục đích
weightx, weighty	Chỉ ra sự phân phối của khoảng trống trong GridBagLayout. Giá trị mặc định cho các biến này là 0.
gridwidth, gridheight	Chỉ ra số lượng các ô (cell) chiều ngang và chiều dọc trong vùng hiển thị của một thành phần.
ipadx, ipady	Chỉ ra lượng làm thay đổi chiều cao và chiều rộng tối thiểu của thành phần. Nó sẽ thêm 2*ipadx vào chiều rộng tối thiểu và 2*ipady vào chiều cao tối thiểu của thành phần. Giá trị mặc định cho cả hai là 0.
anchor	Chỉ ra cách sắp xếp các thành phần trong cell. Mặc định sẽ đặt vào giữa cell. Các thành viên dữ liệu tĩnh (static) sau đây có thể được sử dụng: <ul style="list-style-type: none"> GridBagConstraints.NORTH GridBagConstraints.EAST GridBagConstraints.WEST GridBagConstraints.SOUTH GridBagConstraints.NORTHEAST GridBagConstraints.SOUTHEAST
gridx, gridy	Chỉ ra vị trí cell sẽ đặt thành phần. Khi thiết lập giá trị của gridx là 'GridbagConstraints.RELATIVE' thì thành phần được thêm sẽ nằm ở vị trí bên phải của thành phần cuối cùng.
fill	Chỉ ra cách mà một thành phần được bố trí vào cell thế nào nếu như cell lớn hơn thành phần. Mặc định là kích thước thành phần không thay đổi.

Bảng sau đây cung cấp một danh sách các biến dữ liệu tĩnh là các giá trị cho biến fill:

Các biến thành viên dữ liệu tĩnh của biến fill

Giá trị	Mô tả
GridBagConstraints.NONE	Mặc định, không làm thay đổi kích thước của thành phần.
GridBagConstraints.HORIZONTAL	Tăng chiều rộng của thành phần theo chiều ngang (HORIZONTAL) để làm cho thành phần khớp với chiều ngang.
GridBagConstraints.VERTICAL	Tăng chiều cao của thành phần theo chiều đứng (VERTICAL) để làm cho thành phần khớp với chiều dọc.

GridBagConstraints.BOTH	Tăng chiều rộng, chiều cao của thành phần theo cả chiều ngang và chiều dọc.
insets	Xác định khoảng cách top, bottom, left và right giữa các thành phần. Mặc định là 0.

Sử dụng phương thức ‘setConstraints()’ để thiết lập các hằng số cho mỗi thành phần. Cho ví dụ:

```
gblay.setConstraints(lbl, gbc);
```

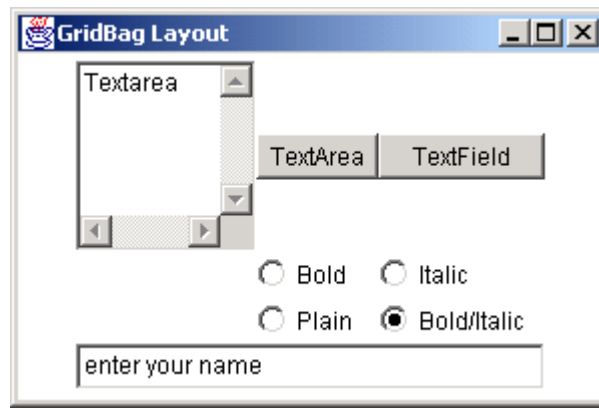
‘gblay’ là đối tượng của lớp **GridBagLayout**, lbl là thành phần ‘Label’ và ‘gbc’ là đối tượng của lớp **GridBagConstraints**.

Chương trình minh họa của GridBagLayout và GridBagConstraints

```
import java.awt.*; class Gbltest extends Frame { TextArea ta; TextField tf; Button
b1,b2; Checkgroup cbg; Checkbox cb1,cb2,cb3,cb4; GridBagLayout gb;
GridBagConstraints gbc; public Gbltest(String title) { super(title); gb=new
GridBagLayout(); setLayout(gb); gbc=new GridBagConstraints(); ta=new
TextArea("Textarea",5,10); tf=new TextField("enter your name"); b1=new
Button("TextArea"); b2=new Button("TextField"); cbg=new Checkgroup();
cb1=new Checkbox("Bold", cbg,false); cb2=new Checkbox("Italic", cbg,false);
cb3=new Checkbox("Plain", cbg,false); cb4=new Checkbox("Bold/Italic", cbg,true);
gbc.fill=GridBagConstraints.BOTH; addComponent(ta,0,0,4,1);
gbc.fill=GridBagConstraints.HORIZONTAL; addComponent(b1,0,1,1,1);
gbc.fill=GridBagConstraints.HORIZONTAL; addComponent(b2,0,2,1,1);
gbc.fill=GridBagConstraints.HORIZONTAL; addComponent(cb1,2,1,1,1);
gbc.fill=GridBagConstraints.HORIZONTAL; addComponent(cb2,2,2,1,1);
gbc.fill=GridBagConstraints.HORIZONTAL; addComponent(cb3,3,1,1,1);
gbc.fill=GridBagConstraints.HORIZONTAL; addComponent(cb4,3,2,1,1);
gbc.fill=GridBagConstraints.HORIZONTAL; addComponent(tf,4,0,1,3); } public void
addComponent(Component c, int row, int col, int nrow, int ncol) { gbc.gridx=col;
gbc.gridy=row; gbc.gridwidth=ncol; gbc.gridheight=ncol; gb.setConstraints(c,gbc);
add(c); } public static void main(String args[]) { Gbltest t=new Gbltest("GridBag
Layout"); t.setSize(300,200); t.show(); } }
```

Khi một container bị thay đổi kích thước và khi khoảng trống phụ tồn tại, các thành phần có chiều rộng lớn hơn sẽ chiếm giữ nhiều khoảng trống hơn là các thành phần có giá trị về chiều rộng nhỏ hơn.

Kết xuất của chương trình được chỉ ra ở [\[link\]](#)



GridBagLayout

Giải thích đoạn mã trên:

```
gbc.fill=GridBagConstraints.BOTH;
```

Thành viên fill của lớp GridBagConstraints chỉ ra thành phần có thể được mở rộng theo hướng nằm ngang và thẳng đứng. Cú pháp sau mô tả thành phần chỉ được mở rộng theo hướng nằm ngang:

```
gbc.fill=GridBagConstraints.HORIZONTAL;
```

Cú pháp sau sẽ thêm vào thành phần TextArea với số dòng và số cột cần chiếm:

```
addComponent (ta, 0, 2, 4, 1);
```

0 – Khởi đầu từ dòng thứ 0

2 – Khởi đầu từ dòng thứ 2

4 – ta chiếm giữ 4 dòng

1 – ta chiếm 1 cột

Sử dụng cú pháp sau để bố trí các thành phần vào trong dòng và cột nào đó:

```
gbc.gridx=col; gbc.gridy=row;
```

Ở đây (gridx,gridy) là cột và dòng nơi mà thành phần có thể được đặt vào.

Sử dụng cú pháp sau để chỉ ra số lượng các cột và dòng mà các thành phần có thể chiếm giữ:

```
gbc.gridwidth=ncol; gbc.gridheight=nrow;
```

Ở đây, gridwidth xác định số lượng các cột mà một thành phần chiếm giữ và gridheight xác định số lượng các dòng mà một thành phần chiếm giữ.

Khi một container bị thay đổi kích thước và khi khoảng trống phụ tồn tại, các thành phần có chiều rộng lớn hơn sẽ chiếm giữ nhiều khoảng trống hơn là các thành phần có giá trị về chiều rộng nhỏ hơn.

Xử lý các sự kiện

Các hệ thống GUI xử lý các tương tác người dùng với sự trợ giúp của mô hình hướng sự kiện (event-driven). Tương tác của người dùng có thể là di chuyển chuột, nhấn phím, nhả phím v.v... Tất cả các thao tác này thiết lập một sự kiện của một loại nào đó.

Việc xử lý những sự kiện này phụ thuộc vào ứng dụng. Abstract Windowing Toolkit (AWT) xử lý một vài sự kiện. Môi trường mà các ứng dụng này được thi hành ví dụ như trình duyệt cũng có thể xử lý các sự kiện khác. Người lập trình cũng cần phải xử lý những sự kiện nhất định và cần phải viết hàm xử lý các sự kiện đó.

Ứng dụng cần đăng ký một hàm xử lý sự kiện với một đối tượng. Hàm xử lý sự kiện này sẽ được gọi bất cứ khi nào sự kiện tương ứng phát sinh. JDK1.2 làm việc theo mô hình xử lý sự kiện này.

Trong quy trình này, ứng dụng cho phép bạn đăng ký các phương thức (handler), hay gọi là listener với các đối tượng. Những handler này tự động được gọi khi một sự kiện thích hợp phát sinh.

Một Event Listener lắng nghe một sự kiện nào đó mà một đối tượng đã thiết lập. Mỗi event listener cung cấp các phương thức xử lý những sự kiện này. Lớp thi hành listener cần phải định nghĩa những phương thức này. Để sử dụng mô hình này, bạn làm theo các bước sau:

- Cài đặt giao diện listener thích hợp. Cấu trúc như sau:

```
public class MyApp extends Frame implements ActionListener
```

- Xác định tất cả các thành phần tạo ra sự kiện. Các thành phần có thể là các button, label, menu item, hay window.

Cho ví dụ, để đăng ký một thành phần với listener, ta có thể sử dụng:

```
exitbtn.addActionListener(This);
```

- Xác định tất cả các sự kiện được xử lý. Các sự kiện có thể là một 'ActionEvent' nếu một button được click hay một 'MouseEvent' nếu như chuột được kéo đi.
- Thi hành các phương thức của listener và viết hàm xử lý sự kiện tương ứng với các phương thức.

Các sự kiện khác nhau và mô tả về chúng

Lớp sự kiện	Mô tả
ActionEvent	Phát sinh khi một button được nhấn, một item trong danh sách chọn lựa được nhấn đúp (double-click) hay một menu được chọn.

AdjustmentEvent	Phát sinh khi một thanh scrollbar được sử dụng.
ComponentEvent	Phát sinh khi một thành phần được thay đổi kích thước, được di chuyển, bị ẩn hay làm cho hoạt động được.
FocusEvent	Phát sinh khi một thành phần mất hay nhận focus từ bàn phím.
ItemEvent	Phát sinh khi một mục menu được chọn hay bỏ chọn; hay khi một checkbox hay một item trong danh sách được click.
WindowEvent	Phát sinh khi một cửa sổ được kích hoạt, được đóng, được mở hay thoát.
TextEvent	Phát sinh khi giá trị trong thành phần textfield hay textarea bị thay đổi.
MouseEvent	Phát sinh khi chuột di chuyển, được click, được kéo hay thả ra.
KeyEvent	Phát sinh khi bàn phím ấn, nhả.

Các giao diện cần được cài đặt để xử lý một trong số những sự kiện này là:

- ActionListener
- AdjustmentListener
- ComponentListener
- FocusListener
- ItemListener
- WindowListener
- TextListener
- MouseListener
- MouseMotionListener
- KeyListener

Các giao diện định nghĩa một số phương thức để xử lý mỗi sự kiện. Những phương thức này sẽ được nạp chồng trong lớp mà cài đặt những giao diện này.

Chương trình sau đây sử dụng một ActionListener để xử lý các sự kiện liên quan với một button. ActionEvent có hai phương thức:

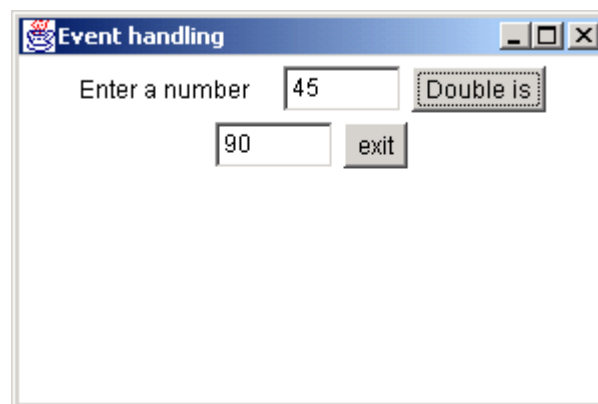
- **getSource()**: Để trả về nguồn của sự kiện.
- **toString()**: Để trả về chuỗi tương đương với sự kiện.

Chương trình sau sẽ trình bày cách tính gấp đôi của một số được nhập vào. Chương trình này được thực hiện bằng cách kết hợp các phương thức của lớp, nghĩa là các phương thức xử lý sự kiện và giao diện. Việc click trên một button sẽ làm khởi động

ActionEvent và gọi phương thức actionPerformed(). Nó sẽ kiểm tra button được click với sự trợ giúp của hàm getSource và trả về kết quả thích hợp.

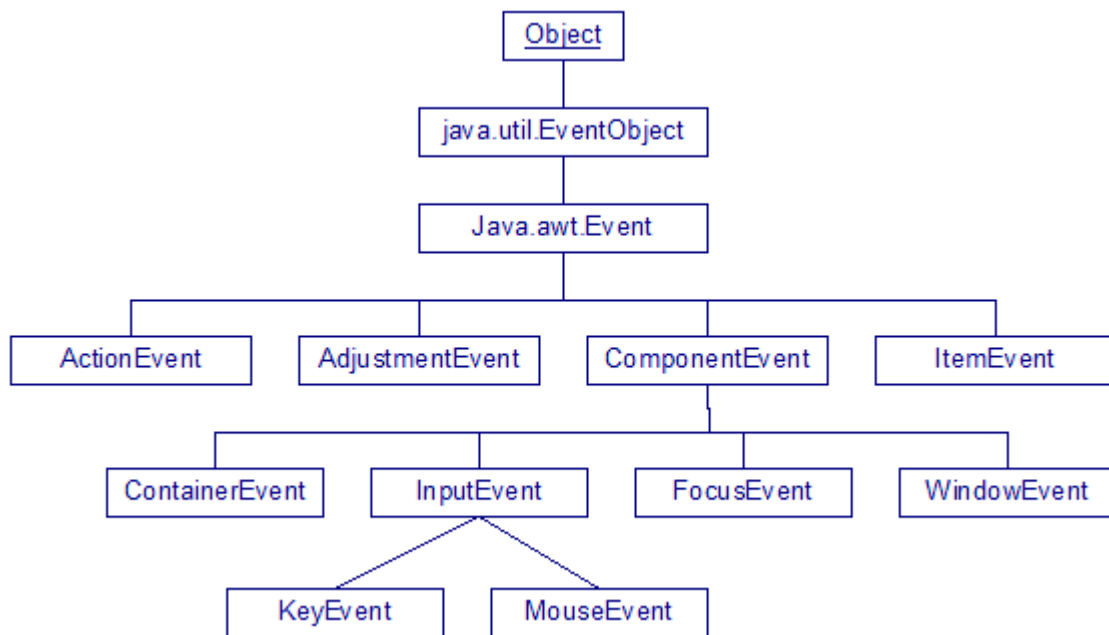
```
import java.awt.*; import java.awt.event.*; class evttest
extends Frame implements ActionListener { Label lab=new
Label("Enter a number"); TextField tf1=new TextField(5);
TextField tf2=new TextField(5); Button btnResult=new
Button("Double is"); Button ext=new Button("exit"); public
evttest(String title) { super(title); setLayout(new
FlowLayout()); btnResult.addActionListener(this);
ext.addActionListener(this); add(lab); add(tf1);
add(btnResult); add(tf2); add(ext); } public void
actionPerformed(ActionEvent ae) { if
(ae.getSource()==btnResult) { int
num=Integer.parseInt(tf1.getText())*2;
tf2.setText(String.valueOf(num)); } if
(ae.getSource()==ext) { System.exit(0); } } public static
void main(String args[]) { evttest t=new evttest("Event
handling"); t.setSize(300,200); t.show(); } }
```

Kết xuất của chương trình được chỉ ra ở hình bên dưới:



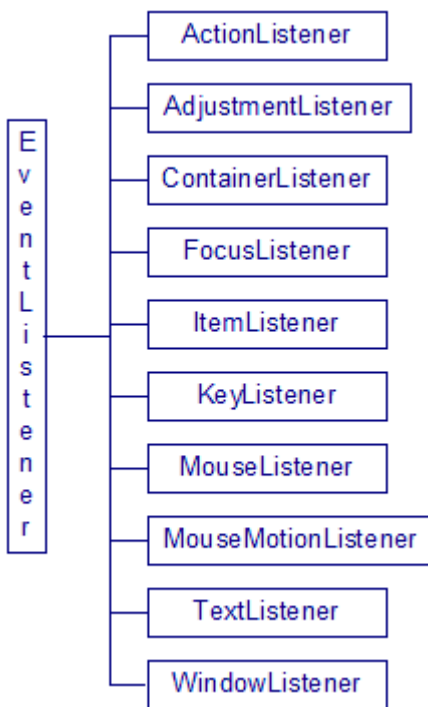
Xử lý sự kiện

[\[link\]](#) chỉ ra một phần của cây phân cấp các lớp của gói Event



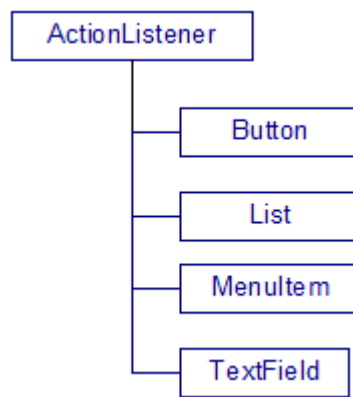
Gói Event

[\[link\]](#) chỉ ra thứ tự phân cấp các giao diện của các Event Listener

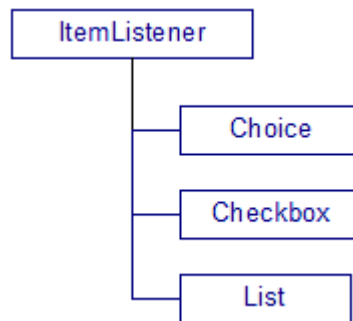


Event Listener

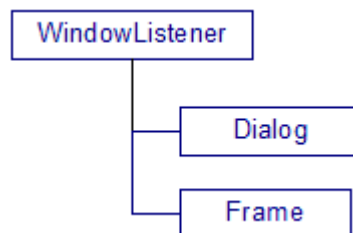
[\[link\]](#) là danh sách các Listener được sử dụng cho các thành phần chỉ ra



Action Listener

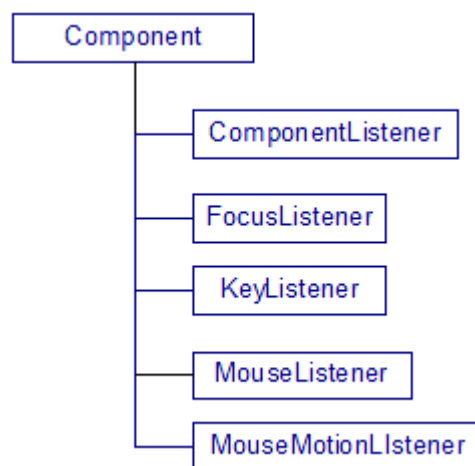


Item Listener



Window Listener

Các Listener cho lớp Component được chỉ ra trong [\[link\]](#)



Các Component

Applets

Java Applet

Applet là một chương trình Java có thể chạy trong trình duyệt web. Tất cả các applet đều là các lớp con của lớp 'Applet'.

Lớp Applet thuộc package 'java.applet'. Lớp Applet bao gồm nhiều phương thức để điều khiển quá trình thực thi của applet. Để tạo applet, bạn cần import hai gói sau:

- java.applet
- java.awt

Cấu trúc của một Applet

Một Applet định nghĩa cấu trúc của nó từ 4 sự kiện xảy ra trong suốt quá trình thực thi. Đối với mỗi sự kiện, một phương thức được gọi một cách tự động. Các phương thức này được minh họa trong [\[link\]](#)

Điều quan trọng là không phải lúc nào applet cũng bắt đầu từ ban đầu. Mà nó bắt đầu từ vị trí tiếp theo của quá trình thực thi trước đó.

Ngoài những phương thức cơ bản này, còn có những phương thức 'paint()' và 'repaint()'. Phương thức paint() dùng để hiển thị một đường thẳng (line), text, hoặc một hình ảnh trên nền applet. Đối số của phương thức này là đối tượng của lớp Graphics. Lớp này thuộc gói java.awt. Câu lệnh sau được dùng để import lớp Graphics:

```
import java.awt.Graphics;
```

Các phương thức của một Applet

Phương thức	Chức năng
init()	Được gọi trong quá trình khởi tạo applet. Trong quá trình khởi tạo, nó sẽ tạo đối tượng để cung cấp cho applet. Phương thức này được dùng để tải các hình ảnh đồ họa, khởi tạo các biến và tạo các đối tượng.
start()	Được gọi khi một applet bắt đầu thực thi. Một khi quá trình khởi tạo hoàn tất, thì applet được khởi động. Phương thức này được dùng để khởi động lại applet sau khi nó đã ngừng trước đó

stop()	Được gọi khi ngừng thực thi một applet. Một applet bị ngừng trước khi nó bị huỷ.
destroy()	Được dùng để huỷ một applet. Khi một applet bị huỷ, thì bộ nhớ, thời gian thực thi của vi xử lý, không gian đĩa được trả về cho hệ thống.

Phương thức '**repaint()**' được dùng khi cửa sổ cần cập nhật lại. Phương thức này chỉ cần một thông số. Tham số này là đối tượng của lớp Graphics.

Applet sử dụng phương thức '**showStatus()**' để hiển thị thông tin trên thanh trạng thái. Phương thức có tham số thuộc kiểu dữ liệu String. Để lấy các thông tin của applet, user có thể override phương thức '**getAppletInfo()**' của lớp Applet. Phương thức này trả về 1 đối tượng kiểu String.

Các phương thức của applet **init()**, **start()**, **stop()**, **destroy()**, và **paint()** được thừa kế từ một applet. Nhưng mặc định những phương thức này không thực thi một thao tác nào cả.

Đây là ví dụ đơn giản của applet. Câu lệnh sau tạo một lớp có tên là 'Applet1', lớp này sẽ kế thừa tất cả các phương thức và biến của lớp '**applet**'.

```
public class Applet1 extends Applet
```

Phương thức **init()** và **paint()** thường được dùng để thực hiện một số hàm để khởi tạo và vẽ applet. Phương thức '**g.drawString()**' chỉ ra vị trí mà đoạn văn bản được vẽ ở đâu trên màn hình.

```
Chương trình hiển thị một chuỗi ở dòng 70 và cột 80
import java.awt.*; import java.applet.*; public class Applet1 extends Applet { int num;
public void init() { num = 6; } public void paint (Graphics g) { g.drawString ("Hello to
Applet. Chapter " + num, 70, 80); showStatus (getAppletInfo()); //Hiển thị một chuỗi
được trả về từ hàm getAppletInfo() trên //thanh trạng thái } public String
getAppletInfo() //user overrides { return "Created by Aptech"; } }
```

Sử dụng cú pháp sau để dịch một Applet:

```
javac Applet1.java
```

Để thực thi một applet, ta cần tạo một file HTML. File HTML này sử dụng thẻ applet. Thẻ applet này lấy tham số đầu tiên là đường dẫn của file applet.

Thẻ applet có hai thuộc tính sau:

- Width

- Height

Để truyền tham số vào applet, sử dụng param, sau đó là thuộc tính value. Sau đây là ví dụ của thẻ applet:

```
<applet code=Applet1 width=300 height=200> </applet>
```

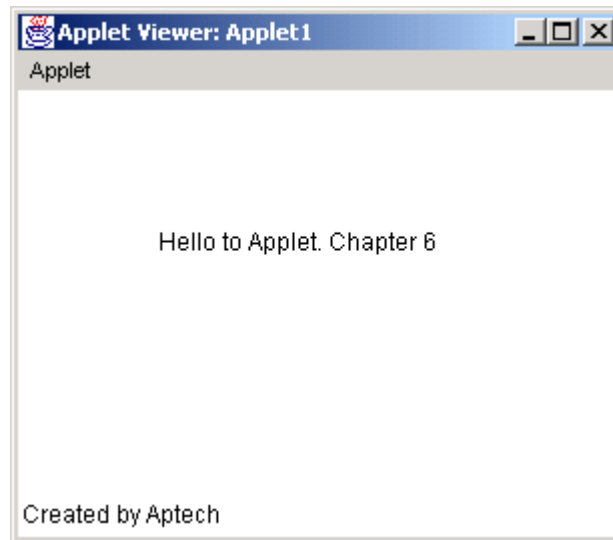
Lúc này, ta có thể thực thi applet này bằng cách dùng trình xem applet. Đây là công cụ của JDK. Để chạy file HTML trong trình xem applet, ta gõ câu lệnh sau:

```
appletviewer abc.html // 'abc.html' là tên của file HTML
```

Một tùy chọn khác của applet là ta thêm thẻ applet như là một dòng chú thích trong đoạn code. Lúc đó, applet được dịch, và thực thi bằng cách sử dụng lệnh sau:

```
appletviewer Applet1.java
```

Sau đây là kết quả của chương trình trên:



Applet

Sự khác nhau giữa Application và Applet

Sau đây là sự khác nhau giữa application và applet:

- Để thực thi các application chúng ta dùng trình thông dịch java, trong khi đó applet có thể chạy được trên các trình duyệt (có hỗ trợ Java) hay sử dụng công cụ appletViewer, công cụ này đi kèm với JDK.
- Quá trình thực thi của application bắt đầu từ phương thức 'main()'. Tuy nhiên applet thì không làm như vậy.

- Các application sử dụng 'System.out.println()' để hiển thị kết quả ra màn hình trong khi đó applet sử dụng phương thức 'drawString()' để xuất ra màn hình.

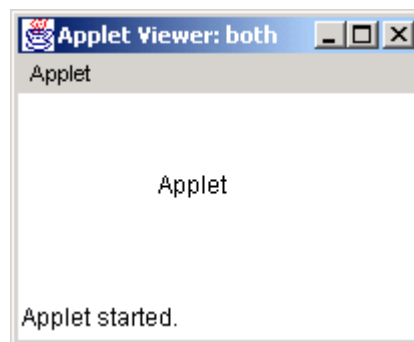
Một điều đáng lưu ý là một chương trình Java đơn lẻ thì có thể vừa là application vừa là applet. Chức năng của applet được bỏ qua khi nó được thực thi như là một application và ngược lại.

```
import java.applet.Applet; import java.awt.*; /* <applet
code = "both" width = 200 height = 100> </applet> */
public class both extends Applet { Button btn; public void
init() { btn = new Button ("Click"); } public void paint
(Graphics g) { g.drawString ("Applet", 70, 50); } public
static void main (String args[]) { both app = new both();
app.init(); System.out.println("Application Main"); } }
```

Sau khi biên dịch chương trình, nó có thể được thực thi như là một applet bằng cách sử dụng cú pháp sau:

```
appletviewer both.java
```

Kết quả như sau:

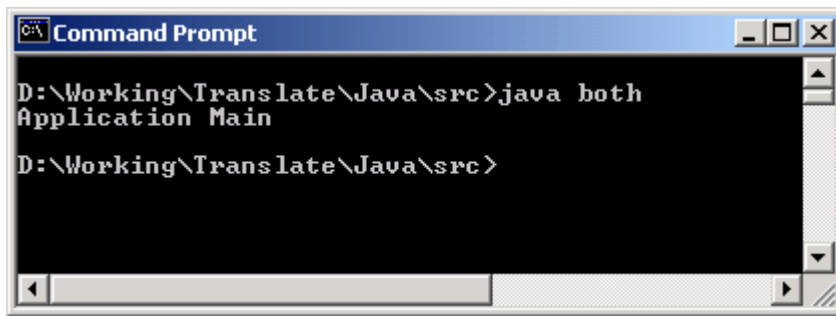


Applet

Nếu chạy chương trình trên như một application, thì sử dụng cú pháp sau:

```
java both
```

Kết quả là:



Application

Khi applet chạy trên trình duyệt web, đặc điểm này thực sự hữu ích khi bạn muốn tải applet trong một frame mới. Ví dụ: trong applet được tạo để chat, một số website sử dụng một cửa sổ chat riêng biệt để chat. Bạn cũng có thể kết hợp các đặc điểm của frame và applet vào trong một chương trình.

Những giới hạn bảo mật trên applet

Có một số hạn chế mà applet không thể làm được. Bởi vì các applet của Java có thể phá hỏng toàn bộ hệ thống của người sử dụng. Các lập trình viên Java có thể viết các applet để xóa file, lấy các thông tin các nhân của hệ thống...

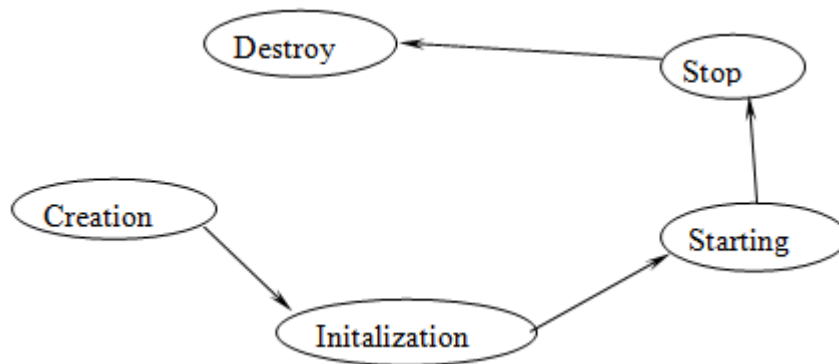
Vì thế, các applet của java không thể làm các việc sau:

- Không thể đọc hoặc ghi file trên hệ thống file của người sử dụng.
- Không thể giao tiếp với các site internet, nhưng chỉ có thể với các trang web có applet mà thôi.
- Không thể chạy bất cứ chương trình gì trên hệ thống của người đọc.
- Không thể tải bất cứ chương trình được lưu trữ trong hệ thống của người sử dụng.

Những giới hạn trên chỉ đúng khi các applet được chạy trên trình duyệt Netscape Navigator hoặc Microsoft Internet Explorer.

Chu trình sống của một Applet

Chu trình sống của một Applet được mô tả ở sơ đồ dưới đây:



Chu trình sống của một applet

Trước tiên, applet được tạo.

Bước kế tiếp là khởi tạo. Điều này xảy ra khi một applet được nạp. Quá trình này bao gồm việc tạo các đối tượng mà applet cần. Phương thức `init()` được định nghĩa để cung cấp các hành vi để khởi tạo.

Một khi applet được khởi tạo, applet sẽ được khởi động. Applet có thể khởi động ngay cả khi nó đã được ngừng trước đó. Ví dụ, nếu trình duyệt nhảy đến một liên kết nào đó ở trang khác, lúc đó applet sẽ bị ngừng, và được khởi động trở lại khi người sử dụng quay về trang đó.

Sự khác nhau giữa quá trình khởi tạo và quá trình khởi động là một applet có thể khởi động nhiều lần, nhưng quá trình khởi tạo thì chỉ xảy ra một lần.

Phương thức '`start()`' được override để cung cấp các thao tác khởi động cho applet.

Phương thức '`stop()`' chỉ được gọi khi user không còn ở trang đó nữa, hoặc trang đó đã được thu nhỏ lại ở dưới thanh taskbar.

Kế tiếp là phương thức '`destroy()`'. Phương thức này giúp applet dọn dẹp trước khi nó được giải phóng khỏi vùng nhớ, hoặc trước khi duyệt kết thúc. Phương thức này được dùng để huỷ những luồng (thread) hay quá trình đang chạy.

Phương thức '`destroy()`' khác với phương thức `finalize()` là phương thức `destroy()` chỉ dùng cho applet, trong khi `finalize()` là cách tổng quát để dọn dẹp applet.

Phương thức `paint()` cũng là một phương thức quan trọng khác. Phương thức này cho phép ta hiển thị một cái gì đó trên màn hình. Có thể là text, đường thẳng, màu nền, hoặc hình ảnh. Phương thức này xảy ra nhiều lần trong suốt quá trình applet tồn tại. Phương thức này thực thi một lần sau khi applet được khởi tạo. Nó sẽ lặp đi lặp lại khi di chuyển từ cửa sổ trình duyệt sang cửa sổ khác. Nó cũng xảy ra khi cửa sổ trình duyệt thay đổi vị trí của nó trên màn hình.

Phương thức 'paint()' có một tham số. Tham số này là đối tượng của lớp Graphics. Lớp Graphics thuộc lớp java.awt, chúng ta phải import trong đoạn code của applet. Chúng ta có thể sử dụng đoạn mã sau:

```
import java.awt.Graphics;
```

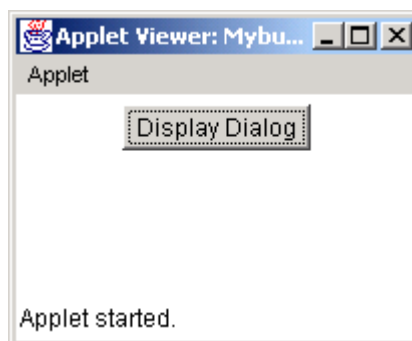
Truyền tham số cho Applet

Trong chương trình sau, chúng ta sẽ truyền tham số cho applet. Thành phần nút 'bNext' có tên được truyền như là một tham số. Phương thức 'init()' sẽ kiểm tra tham số có tên là 'mybutton'. Sau đó, nó tạo một nút với chuỗi đó như là tên của nút. Nếu không có tham số truyền vào, nút đó có tên mặc định là 'Default'.

Bây giờ chúng ta định nghĩa thẻ <PARAM> trong đoạn mã HTML như sau:

```
/* <applet code="Mybutton1" width="100" height="100">
<PARAM NAME="mybutton" value="Display Dialog"> </applet>
*/ import java.awt.*; import java.applet.*; /* <applet
code="Mybutton1" width="200" height="100"> <PARAM
NAME="mybutton" value="Display Dialog"> </applet> */
public class Mybutton1 extends Applet { Button bNext;
public void init() { /*getParameter returns the value of
the specified parameter in the form of a String object*/
String str = getParameter("mybutton"); //when no parameter
is passed if (str==null) str = new String ("Default");
//when parameter is passed bNext = new Button(str); add
(bNext); } }
```

Sau đây là kết quả của chương trình trên:



truyền tham số cho applet

Bây giờ chúng ta sẽ sử dụng lớp Graphics để vẽ các hình chẳng hạn như: đường thẳng, hình oval, và hình chữ nhật. Chúng ta sẽ học lớp Font trong các phần sau. Lớp này có thể dùng để hiển thị văn bản bằng bất cứ font nào.

Lớp Graphics

Java cung cấp gói AWT cho phép ta vẽ các hình đồ họa. Lớp Graphics bao gồm tập hợp rất nhiều phương thức. Nhưng phương thức này được sử dụng để vẽ bất cứ hình nào trong các hình sau:

- Oval
- Rectangle
- Square
- Circle
- Lines
- Text

Bạn có thể vẽ những hình này bằng bất cứ màu nào. Frame, Applet và Canvas là các môi trường để hiển thị đồ họa.

Để vẽ bất cứ hình ảnh nào chúng ta cần phải có nền đồ họa (Graphical Background). Để có được một nền đồ họa, chúng ta gọi phương thức 'getGraphics()' hay bất cứ phương thức nào trong các phương thức sau đây:

- repaint()

Được gọi khi cần vẽ lại những đối tượng đã vẽ.

- update(Graphics g)

Được gọi một cách tự động bởi phương thức 'repaint()'.

Phương thức này sẽ xóa những đối tượng đã vẽ, và truyền nó cho đối tượng của lớp Graphics để gọi phương thức 'paint()';

- paint(Graphics g)

Được gọi bởi phương thức update().

Đối tượng được truyền cho phương thức này được dùng để vẽ. Phương thức này dùng để vẽ các hình ảnh đồ họa khác nhau.

Việc gọi phương thức `paint()` lặp đi lặp lại thông qua phương thức `repaint()` sẽ xoá đi các hình đã vẽ trước đó. Để vẽ các hình mới mà vẫn giữ lại những hình đã vẽ trước đó, chúng ta cần override lại phương thức `update()`.

```
public void update (Graphics g) { paint (g); }
```

Ở đây, phương thức `update()` sẽ không xoá những đối tượng đã vẽ, nhưng chỉ gọi phương thức `paint()`. Để làm được điều này, nó truyền đối tượng của lớp `Graphics` hoặc `GraphicsContext` cho phương thức `paint()`. Ở đây, đối tượng của lớp `Graphics` là 'g'.

Vẽ các chuỗi, các ký tự và các byte

Chương trình sau minh hoạ các vẽ các chuỗi, ký tự và các byte.

Để vẽ hoặc in một chuỗi, lớp `Graphics` cung cấp phương thức '**`drawString()`**'. Cú pháp như sau:

```
drawString (String str, int xCoor, int yCoor);
```

Ba tham số là:

- Chuỗi cần vẽ.
- Toạ độ X trên frame, nơi chuỗi cần được vẽ.
- Toạ độ Y trên frame, nơi chuỗi cần được vẽ.

Để vẽ hoặc xuất các ký tự trên frame, lớp `Graphics` cung cấp phương thức '**`drawChars()`**'. Cú pháp như sau:

```
drawChars (char array[], int offset, int length, int  
xCoor, int yCoor);
```

Chú thích các tham số:

- Mảng các ký tự.
- Vị trí bắt đầu, nơi các ký tự được vẽ.
- Số các ký tự cần được vẽ.
- Toạ độ X, nơi các ký tự cần được vẽ.
- Toạ độ Y, nơi các ký tự cần được vẽ.

Lớp `Graphics` cung cấp phương thức '**`drawBytes()`**' để vẽ hoặc in các byte ra frame. Cú pháp của phương thức này như sau:

```
drawBytes (byte array[], int offset, int length, int  
xCoor, int yCoor);
```


5 tham số của phương thức trên là:

- Mảng các byte.
- Vị trí offset hay vị trí bắt đầu.
- Số byte cần vẽ.
- Toạ độ X.
- Toạ độ Y.

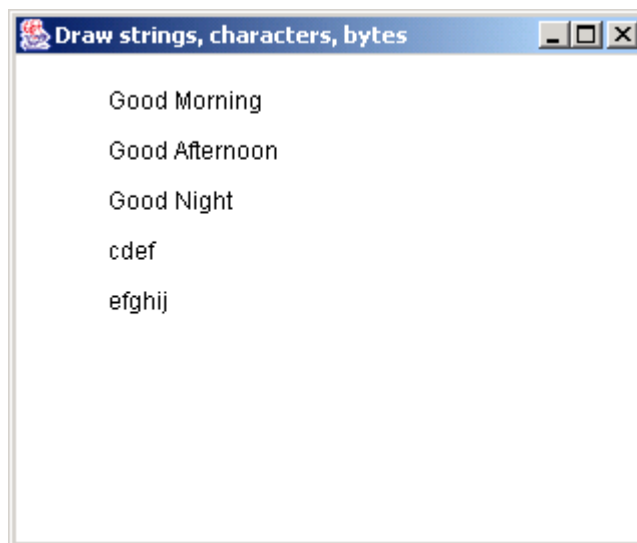
Đối với một ký tự hoặc một mảng các byte, chúng ta có thể in một phần của mảng mà thôi. Ở đây, toạ độ x và y là toạ độ tính theo dòng.

Chương trình minh hoạ cách vẽ chuỗi, các ký tự và các byte.

```
import java.awt.*; public class DrawStrings extends Frame { public DrawStrings() {  
super ("Draw strings, characters, bytes"); setSize (300, 300); setVisible (true); } public  
void paint(Graphics g) { g.drawString ("Good Morning", 50, 50); g.drawString ("Good  
Afternoon", 50, 75); g.drawString ("Good Night", 50, 100); char ch[] =  
{ 'a','b','c','d','e','f' }; g.drawChars(ch,2,4,50,125); byte b[] =  
{ 100,101,102,103,104,105,106,107 }; g.drawBytes(b,1,6,50,150); } public static void  
main (String args[]) { new DrawStrings(); } }
```

Chương trình trên vẽ chuỗi, ký tự từ một mảng ký tự, và vẽ các byte từ mảng các byte. Bạn phải import gói java.awt để sử dụng các đối tượng đồ hoạ có sẵn trong gói này. Ta phải làm điều này vì lớp Graphics nằm trong gói này.

Sau đây là kết quả của chương trình trên:



Strings, characters và bytes

Vẽ đường thẳng (Line) và Oval

Sau đây là cú pháp của các phương thức được sử dụng để vẽ đường thẳng và hình oval:

- `drawLine (int x1, int y1, int x2, int y2);`
- `drawOval (int xCoor, int yCoor, int width, int height);`
- `setColor (Color c);`
- `fillOval (int xCoor, int yCoor, int width, int height);`

Phương thức '`drawLine()`' nhận các tham số sau:

- Toạ độ X, nơi bắt đầu vẽ (x1).
- Toạ độ Y, nơi bắt đầu vẽ (y1).
- Toạ độ X, nơi kết thúc vẽ (x2).
- Toạ độ Y, nơi kết thúc vẽ (y2).

Phương thức này bắt đầu vẽ tại toạ độ 'x1' và 'y1', và kết thúc tại toạ độ 'x2' và 'y2'. Để vẽ nhưng đường thẳng có màu, chúng ta thiết lập một màu nào đó. Phương thức '`setColor`' dùng để thiết lập màu cho hình ảnh đồ hoạ. Trong chương trình này, chúng ta sử dụng câu lệnh sau để chọn màu xanh:

```
g.setColor (Color.blue);
```

Phương thức '**`drawOval()`**' nhận 4 thông số sau:

- Toạ độ X.
- Toạ độ Y.
- Chiều rộng của hình Oval.
- Chiều cao của hình Oval.

Đối với hình oval rộng, thì giá trị của chiều rộng lớn hơn chiều cao, và ngược lại đối với hình oval cao.

Phương thức '`fillOval()`' nhận 4 thông số, nhưng nó sẽ tô hình oval. Sử dụng phương thức `setColor` để đặt màu tô.

```
g.setColor (Color.cyan);
```

Ở đây, hình oval sẽ được tô với màu cyan. Lớp `Color` cung cấp các màu khác nhau mà hệ thống có hỗ trợ.

Vẽ hình chữ nhật (Rectangle) và hình chữ nhật bo góc (Rounded Rectangle)

Sau đây là cú pháp của các phương thức được dùng để vẽ hình chữ nhật và hình chữ nhật bo góc:

- `drawRect (int xCoor, int yCoor, int width, int height);`
- `fillRect (int xCoor, int yCoor, int width, int height);`
- `drawRoundRect (int xCoor, int yCoor, int width, int height, int arcwidth, int archeight);`
- `fillRoundRect (int xCoor, int yCoor, int width, int height, int arcwidth, int archeight);`

Phương thức ‘`drawRect()`’ được dùng để vẽ hình chữ nhật đơn giản. Phương thức này nhận 4 tham số sau:

- Toạ độ X của góc trên bên trái
- Toạ độ Y của góc trên bên trái
- Chiều rộng của hình chữ nhật
- Chiều cao của hình chữ nhật

Phương thức này vẽ hình chữ nhật có chiều rộng và chiều cao cho trước, bắt đầu tại toạ độ X, Y. Chúng ta có thể thiết lập màu của hình chữ nhật. Ở đây, chúng ta chọn màu đỏ. Câu lệnh sẽ như sau:

```
g.setColor (Color.red);
```

Phương thức ‘`drawRoundRect()`’ vẽ hình chữ nhật có các góc tròn. Phương thức này nhận 6 tham số, trong đó 4 tham số đầu thì giống với phương thức `drawRect`. Hai tham số khác là:

- `arcwidth` của hình chữ nhật
- `archeight` của hình chữ nhật

Ở đây, ‘`arcwidth`’ làm tròn góc trái và góc phải của hình chữ nhật. ‘`archeight`’ làm tròn góc trên đỉnh và góc đáy của hình chữ nhật. Ví dụ, `arcwidth = 20` có nghĩa là hình chữ nhật được làm tròn cạnh trái và cạnh phải mỗi cạnh 10 pixel. Tương tự, `archeight = 40` sẽ tạo ra hình chữ nhật được làm tròn từ đỉnh đến đáy 20 pixel.

Pixel là đơn vị đo. Nó là đơn vị nhỏ nhất trong vùng vẽ.

Để tô hay vẽ hình chữ nhật và hình chữ nhật bo góc, chúng ta sử dụng phương thức ‘`fillRect()`’ và ‘`fillRoundRect()`’. Những phương thức này nhận các tham số giống với phương thức `drawRect()` và `drawRoundRect()`. Những phương thức này vẽ các hình ảnh

với một màu cho trước hoặc mới màu hiện hành. Lệnh sau dùng để vẽ hình với màu xanh:

```
g.setColor(Color.green);
```

Vẽ hình chữ nhật 3D và vẽ hình cung (Arc)

Sau đây là cú pháp của các phương thức dùng để vẽ hình chữ nhật 3D và hình cung:

- draw3Drect (int xCoord, int yCoord, int width, int height, boolean raised);
- drawArc(int xCoord, int yCoord, int width, int height, int arcwidth, int archeight);
- fillArc(int xCoord, int yCoord, int width, int height, int arcwidth, int archeight);

Phương thức ‘draw3Drect()’ nhận 5 tham số. 4 tham số đầu thì tương tự với phương thức để vẽ hình chữ nhật. Tuy nhiên, giá trị của tham số thứ 5 quyết định là hình chữ này có 3 chiều hay không. Tham số thứ 5 có kiểu dữ liệu là boolean. Giá trị này True có nghĩa là hình chữ nhật là 3D.

Phương thức ‘drawArc()’ nhận 6 tham số sau:

- Toạ độ x
- Toạ độ y
- Chiều rộng của cung được vẽ.
- Chiều cao của cung được vẽ.
- Góc bắt đầu.
- Độ rộng của cung (góc của cung) so với góc ban đầu.

Phương thức ‘fillArc()’ cũng nhận 6 tham số giống như phương thức drawArc(), nhưng nó vẽ cung và tô cung với màu hiện thời.

Vẽ hình PolyLine

Chương trình sau lấy các điểm từ hai mảng để vẽ một loạt các đường thẳng.

Cú pháp của phương thức này như sau:

- drawPolyline (int xArray[], int yArray[], int totalPoints);
- g.setFont (new Font(“Times Roman”, Font.BOLD, 15));

Phương thức ‘drawPolyline()’ nhận 3 tham số sau:

- Mảng lưu trữ toạ độ x của các điểm.
- Mảng lưu trữ toạ độ y của các điểm.

- Tổng số điểm cần vẽ.

Để vẽ các đường thẳng ta lấy các điểm từ hai mảng như sau:

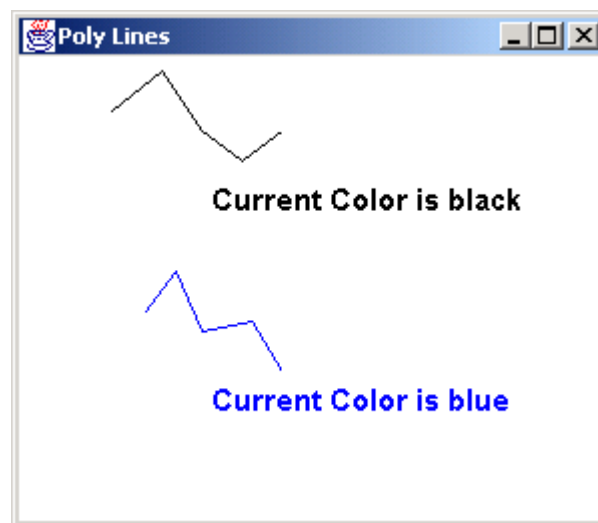
```
(array1[0], array2[0]) (array1[1], array2[1]) (array1[2],  
array2[2]) ...
```

Số đường thẳng vẽ được luôn nhỏ hơn số truyền vào thông số thứ 3 của phương thức `drawPolyline()`. Ví dụ như: `totalPoints - 1`

Chương trình minh họa các vẽ polyline.

```
import java.awt.*; class PolyLines extends Frame { int x1[] = {50, 75, 95, 115, 135};  
int y1[] = {50, 30, 60, 75, 60}; int x2[] = {67, 82, 95, 120, 135}; int y2[] = {150, 130,  
160, 155, 180}; public PolyLines()//constructor { super ("Poly Lines"); setSize (300,  
300); setVisible (true); } public void paint (Graphics g) { g.drawPolyline (x1, y1, 5);  
g.setFont (new Font("Times Roman", Font.BOLD, 15)); g.drawString("Current Color  
is black", 100, 100); g.setColor(Color.blue); g.drawPolyline (x2, y2, 5); g.drawString  
("Current Color is blue", 100, 200); } public static void main (String args[]) { new  
PolyLines(); } }
```

Kết quả của chương trình được minh họa ở [\[link\]](#)



Vẽ và tô đa giác (Polygon)

Lớp Graphics cung cấp hai phương thức để vẽ đa giác. Phương thức đầu tiên nhận một đối tượng của lớp Polygon. Phương thức thứ 2 lấy hai mảng điểm, và tổng số điểm cần vẽ. Chúng ta sẽ sử dụng phương thức 2 để vẽ đa giác.

Cú pháp của **`drawPolygon()`** như sau:

```
drawPolygon(int x[], int y[], int numPoints);
```

Cú pháp của **fillPolygon()** như sau:

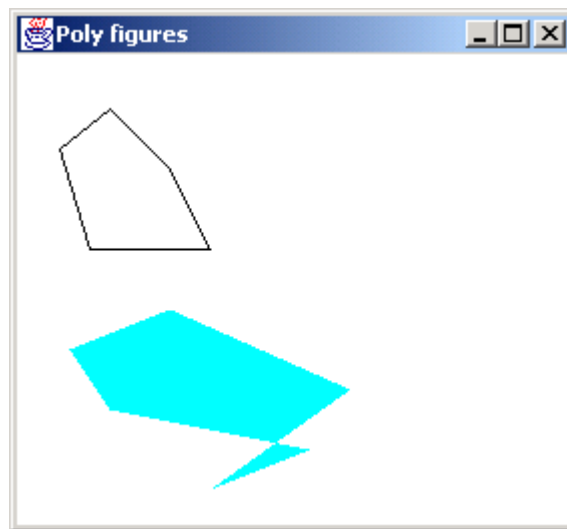
```
fillPolygon (int x[], int y[], int numPoints);
```

Chương trình dưới đây lấy các điểm từ 2 mảng để vẽ đa giác. Phương thức '**drawPolygon()**' nhận 3 tham số sau giống như phương thức **drawPolyline()**

- Mảng lưu trữ tọa độ x của các điểm.
- Mảng lưu trữ tọa độ y của các điểm.
- Tổng số điểm cần vẽ.

```
import java.awt.*; class PolyFigures extends Frame { int
x1[] = {50, 25, 40, 100, 80}; int x2[] = {80, 30, 50, 150,
100, 170}; int y1[] = {50, 70, 120, 120, 80}; int y2[] =
{150, 170, 200, 220, 240,190}; public PolyFigures() {
super ("Poly figures"); setSize(300, 300); setVisible
(true); } public void paint (Graphics g) { g.drawPolygon
(x1, y1, 5); g.setColor (Color.cyan); g.fillPolygon (x2,
y2, 6); } public static void main (String args[]) { new
PolyFigures(); } }
```

Sau đây là kết quả của chương trình trên:



Polygon

Điều khiển màu

Trong Java, chúng ta điều khiển màu bằng cách dùng 3 màu chính là đỏ (red), xanh lá cây (green), xanh dương (blue). Java sử dụng mô hình màu RGB. Đối tượng của lớp

Color chứa 3 số nguyên cho các tham số red, green, blue. Bảng sau trình bày giá trị có thể có của các màu đó:

Phạm vi giá trị của các thành phần màu

Thành phần	Phạm vi
Red	0-255
Green	0-255
Blue	0-255

Sử dụng các giá trị trên để tạo ra một màu tùy thích. Cú pháp của hàm dựng để tạo ra một màu như sau:

```
color (int red, int green, int blue);
```

Bảng sau hiển thị các giá trị của các màu thường gặp:

Các giá trị RGB

Màu	Red	Green	Blue
White	255	255	255
Light Gray	192	192	192
Gray	128	128	128
Dark Gray	64	64	64
Black	0	0	0
Pink	255	175	175
Orange	255	200	0
Yellow	255	255	0
Magenta	255	0	255

Các đối tượng màu khác nhau có thể được tạo bằng những giá trị này. Những đối tượng này có thể được dùng để vẽ hoặc tô các đối tượng đồ họa. Ví dụ, để tạo màu hồng, ta dùng lệnh sau:

```
color c = new Color (255, 175, 175);
```

Ta có thể thiết lập màu bằng cách dùng lệnh sau:

`g.setColor (c);` //g là đối tượng của lớp Graphics

Sử dụng kết hợp các giá trị RGB để tạo ra một màu tùy ý. Để cho dễ hơn, lớp Color cung cấp sẵn một số màu.

Các màu thường gặp

color.white	color.black
color.orange	color.gray
color.lightgray	color.darkgray
color.red	color.green
color.blue	color.pink
color.cyan	color.magenta
color.yellow	

Cách tạo một màu tùy ý

```
Color color1 = new Color (230, 140, 60); Color color4 = new Color (90, 210, 130);  
g.setColor (color1); int myred = color1.getRed (); int mygreen = color1.getGreen (); int  
myblue = color1.getBlue(); color1 = color1.darker(); color4 = color4.brighter();
```

Điều khiển Font

Java cung cấp lớp Font trong gói java.awt cho phép sử dụng các loại font khác nhau. Lớp này bao gồm một số phương thức.

Để sử dụng font, chúng ta nên kiểm tra xem hệ thống có hỗ trợ hay không. Phương thức **getAllFont()** trả về tất cả các font mà hệ thống hỗ trợ.

Trước tiên, khai báo một đối tượng của lớp GraphicsEnvironment như sau:

```
GraphicsEnvironment ge; ge =  
GraphicsEnvironment.getLocalGraphicsEnvironment ();
```

Đối tượng này sử dụng cú pháp sau để lấy tất cả các font có trong mảng Font:

```
Font f[] = ge.getAllFonts();
```

Phương thức getAllFont() được sử dụng ở đây. Phương thức getAllFonts() thuộc lớp GraphicsEnvironment. Đây là lớp trừu tượng, do đó ta không thể khởi tạo lớp này.

để truy cập phương thức **getAllFont()**, chúng ta sử dụng phương thức **getLocalGraphicsEnvironment()** của lớp **GraphicsEnvironment**.

```
ge = GraphicsEnvironment.getLocalGraphicsEnvironment ();
```

Tham chiếu đến lớp này được gán cho biến **ge**. Biến này gọi phương thức **getAllFont()**. Chúng ta sử dụng các font khác nhau để hiển thị các chuỗi khác nhau. Phương thức **getFont()** trả về font mặc định dùng để hiển thị chuỗi, khi không có chọn font nào cả.

```
Font defaultFont = g.getFont (); //g là đối tượng Graphics  
g.drawString ("Default Font is ", 30, 50);
```

Dialog là font mặc định của hệ thống.

Để thay đổi font mặc định của hệ thống thành font khác, chúng ta tạo đối tượng của lớp **Font**. Phương thức khởi tạo của **Font** có 3 tham số sau:

- Tên của font. Ta có thể lấy tên thông qua phương thức **getFontList()**.
- Kiểu của font. Ví dụ: **Font.BOLD**, **Font.PLAIN**, **Font.ITALIC**.
- Kích thước font.

Cú pháp sau minh họa những thông số trên:

```
Font f1 = new Font ("SansSerif", Font.ITALIC, 16);  
g.setFont (f1);
```

Ba tham số được truyền ở đây là: **'SanSerif'** – tên của font, **Font.BOLD** – kiểu font, 14 là kích thước của font. Những thông số này tạo ra đối tượng **f1**. Chúng ta có thể kết hợp 2 kiểu font lại với nhau. Hãy xét ví dụ sau:

```
Font f3 = new Font ("Monospaced", Font.ITALIC+Font.BOLD,  
20);
```

Ở đây kiểu font của **f3** vừa đậm, vừa nghiêng.

Lớp **FontMetrics**

Lớp này xác định kích thước của các ký tự khác nhau thuộc các loại font khác nhau. Xác định kích thước bao gồm chiều cao (**height**), **baseline**, **descent**, và **leading**. Điều này rất cần thiết vì các ký tự khi in đều chiếm một kích thước riêng. Bạn cần tính kích thước cần thiết khi in các ký tự để tránh các ký tự đè lên nhau.

- **Height**: chiều cao của font.

- Baseline (Dòng cơ sở): xác định cơ sở của các ký tự (không kể phần thấp nhất của ký tự)
- Ascent: khoảng cách từ đường baseline đến đỉnh của ký tự.
- Descent: khoảng cách từ baseline đến đáy của ký tự.
- Leading: khoảng cách giữa các chữ.

Chương trình sau sẽ minh họa việc sử dụng các phương thức khác nhau mà lớp FontMetrics có. Trong chương trình này, chúng ta sử dụng các phương thức khác nhau để xem xét chi tiết các loại font khác nhau. Lớp FontMetrics là lớp trừu tượng. Phương thức getFontMetrics() có tham số là đối tượng của lớp Font, vì FontMetrics đi đôi với một font nào đó.

```
FontMetrics fm = g.getFontMetrics (f1);
```

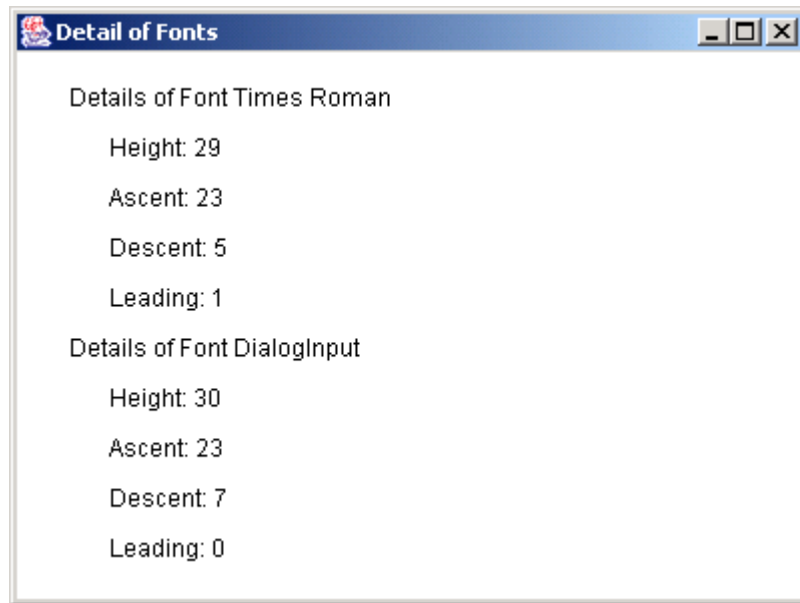
Lệnh này tạo đối tượng fm của lớp FontMetrics, cùng với đối tượng f1. Bây giờ, chúng ta sử dụng fm để lấy chi tiết của font.

Các phương thức **getHeight()**, **getAscent()**, **getDescent()**, và **getLeading()** trả về chi tiết của font. Phương thức **getFont()** của lớp FontMetrics trả về Font mà gắn với đối tượng lớp FontMetrics. Phương thức **getName()** của lớp Font trả về tên Font.

```
import java.awt.*; class FontMetricsUse extends Frame {
public FontMetricsUse() { super ("Detail of Fonts");
setSize (400, 300); setVisible(true); } public void paint
(Graphics g) { Font f1 = new Font ("Times Roman",
Font.PLAIN, 22); FontMetrics fm = g.getFontMetrics (f1);
String name = fm.getFont().getName(); g.drawString
("Details of Font " + name, 30, 50); g.drawString
("Height: " + String.valueOf (fm.getHeight()), 50, 75);
g.drawString ("Ascent: " + String.valueOf
(fm.getAscent()), 50, 100); g.drawString ("Descent: " +
String.valueOf (fm.getDescent()), 50, 125); g.drawString
("Leading: " + String.valueOf (fm.getLeading()), 50, 150);
Font f2 = new Font ("DialogInput", Font.PLAIN, 22); fm =
g.getFontMetrics (f2); name = fm.getFont().getName();
g.drawString ("Details of Font " + name, 30, 175);
g.drawString ("Height: " + String.valueOf
(fm.getHeight()), 50, 200); g.drawString ("Ascent: " +
String.valueOf (fm.getAscent()), 50, 225); g.drawString
("Descent: " + String.valueOf (fm.getDescent()), 50, 250);
g.drawString ("Leading: " + String.valueOf
```

```
(fm.getLeading()), 50, 275); } public static void main
(String args[]) { new FontMetricsUse (); } }
```

Kết quả của chương trình trên:



Lớp FontMetrics

Chương trình sau sẽ minh họa cách lớp FontMetrics được sử dụng để in đoạn văn bản nhiều font, nhiều dòng. Trong chương trình này, chúng ta cần in văn bản nhiều font trên nhiều dòng. Lớp FontMetrics giúp ta xác định khoảng cách cần thiết để in một dòng văn bản cho một font nào đó. Điều này thật cần thiết, bởi vì dòng thứ 2 được in ngay sau dòng thứ nhất.

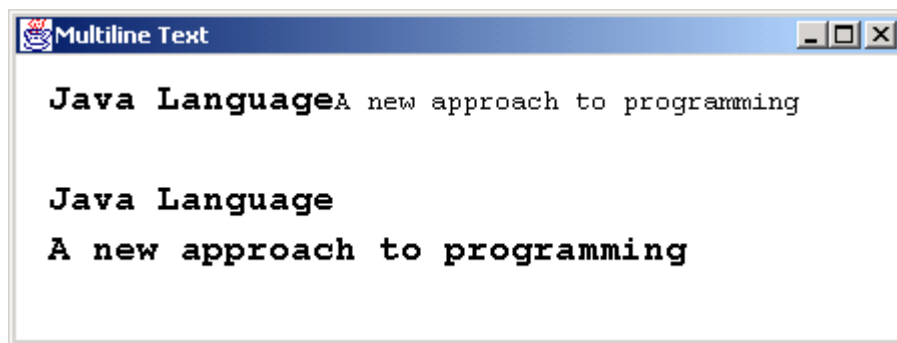
Trước tiên chúng ta in msg1 sử dụng font Monospaced. Sau đó, chúng ta in msg2 sử dụng font DialogInput. Để làm được điều này, chúng ta cần tính khoảng cách cần thiết để in msg1. Phương thức stringWidth() của lớp FontMetrics được dùng để tính ra tổng khoảng cách cần thiết để in msg1. Khi chúng cộng thêm khoảng cách này vào biến x, chúng ta sẽ lấy được vị trí mà chúng ta bắt đầu in đoạn văn bản kế tiếp, msg2. Phương thức setFont() được dùng để thiết lập font để in văn bản.

Kế đó, chúng ta in msg1 và msg2 trên các dòng khác nhau sử dụng chung 1 font Monospaced. Ở đây, chúng ta cần biết khoảng cách chiều cao của font, để in dòng kế tiếp. Phương thức getHeight() được dùng để làm điều này.

```
import java.awt.*; class MultiFontMultiLine extends Frame
{ public MultiFontMultiLine() { super ("Multiline Text");
setSize (450, 200); setVisible (true); } public void paint
(Graphics g) { Font f1 = new Font ("MonoSpaced",
```

```
Font.BOLD, 18); Font f2 = new Font ("DialogInput",
Font.PLAIN, 14); int x = 20; int y = 50; String msg1 =
"Java Language"; String msg2 = "A new approach to
programming"; FontMetrics fm = g.getFontMetrics(f1);
g.setFont(f1); g.drawString (msg1, x, y); x = x +
fm.stringWidth(msg1); g.setFont(f2); g.drawString (msg2,
x, y); g.setFont(f1); y = 100; x = 20; int height =
fm.getHeight(); g.drawString (msg1, x, y); y += height;
g.drawString (msg2, x, y); } public static void main
(String args[]) { new MultiFontMultiLine (); } }
```

Kết quả của chương trình trên:



Văn bản được xuất nhiều font, nhiều dòng

Chọn chế độ (mode) để vẽ

Các đối tượng được vẽ bằng cách sử dụng mode vẽ. Khi một đối tượng mới được vẽ, nó sẽ đè lên các hình đã vẽ trước đây. Tương tự, khi các đối tượng được vẽ đi vẽ lại nhiều lần thì chúng sẽ xóa các đối tượng đã vẽ trước đó. Chỉ hiển thị nội dung của đối tượng mới. Để làm cho nội dung cũ và nội dung mới đều hiển thị trên cùng nền, lớp Graphics cung cấp phương thức **setXORMode (Color c)**;

Chương trình sau sẽ minh họa tiện lợi của của việc sử dụng phương thức **setXORMode()**. Ở đây, chúng ta sử dụng phương thức **setXORMode()** để tô các hình đồ họa khác nhau, mà không đè lên các hình khác. Kết quả là, khi sử dụng mode XOR thì hiển nhiên là tất cả các hình đều hiển thị đầy đủ. Điều này có nghĩa là các hình mới không đè lên các hình cũ. Thay vào đó, phần chung giữa các hình sẽ được hiển thị thành một màu khác. Nhưng khi không sử dụng mode XOR, hình mới hoàn toàn che khuất những hình trước đó.

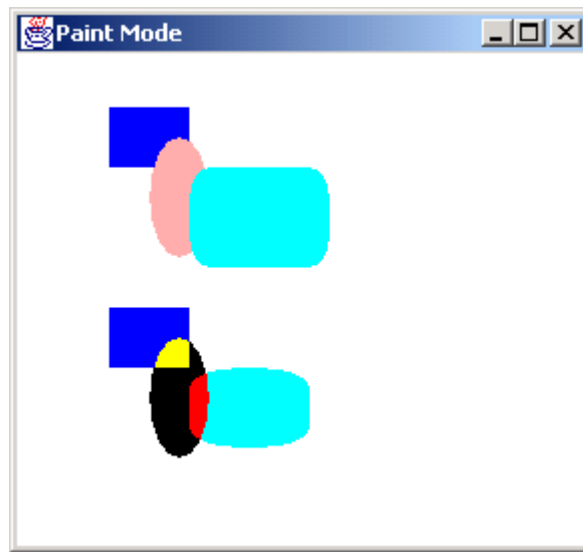
```
import java.awt.*; class PaintMode extends Frame { public
PaintMode() { super ("Paint Mode"); setSize (300, 300);
setVisible (true); } public void paint (Graphics g) {
```

```

g.setPaintMode (); g.setColor (Color.blue); g.fillRect
(50,50,40, 30); g.setColor (Color.pink); g.fillOval (70,
65, 30, 60); g.setColor (Color.cyan); g.fillRoundRect (90,
80, 70, 50, 20, 30); g.setColor (Color.blue); g.fillRect
(50, 150, 40, 30); g.setXORMode (Color.yellow); g.fillOval
(70, 165, 30, 60); g.setXORMode (Color.magenta);
g.fillRoundRect (90, 180, 60, 40, 50, 20); } public static
void main (String args[]) { new PaintMode(); } }

```

Kết quả của chương trình trên:



Paint mode

Tóm tắt

- Applet là chương trình Java chạy trong trình duyệt web.
- Một chương trình Java có thể vừa là applet, vừa là application.
- Lớp Graphics nằm trong gói AWT, bao gồm các phương thức được sử dụng để vẽ các hình đồ họa như hình oval, hình chữ nhật, hình vuông, hình tròn, đường thẳng và văn bản.
- Java sử dụng bảng màu RGB.
- Lớp Font trong gói java.awt cho phép sử dụng nhiều font khác nhau.
- Lớp FontMetrics xác định kích thước của các ký tự.

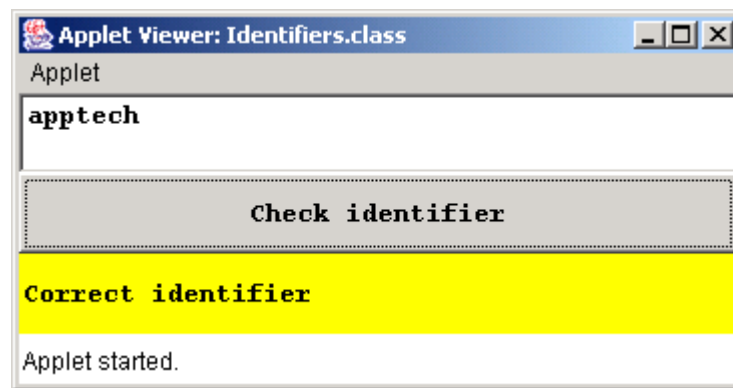
Kiểm tra sự tiến bộ

1. Phương thứcdùng để tạm dừng applet.
2. Phương thứcdùng để cập nhật (update) cửa sổ.
3. Một chương trình Java có thể vừa là applet, vừa là application. **(Đúng/Sai)**

4. Java applet không thể đọc, ghi file trên máy người sử dụng. **(Đúng/Sai)**
5. Phương thứcdùng để tạo tham chiếu đến đối tượng nền đồ hoạ (Graphics).
6. Trong java, điều khiển màu sắc được thực hiện thông qua hai màu cơ bản là trắng và đen. **(Đúng/Sai)**
7. Phương thứcdùng để lấy tất cả các font chữ mà hệ thống hỗ trợ.
8. Trong lớp FontMetrics, 'ascent' là khoảng cách từ 'baseline' đến đáy của ký tự. **(Đúng/Sai)**

Bài tập

Viết applet như sau:

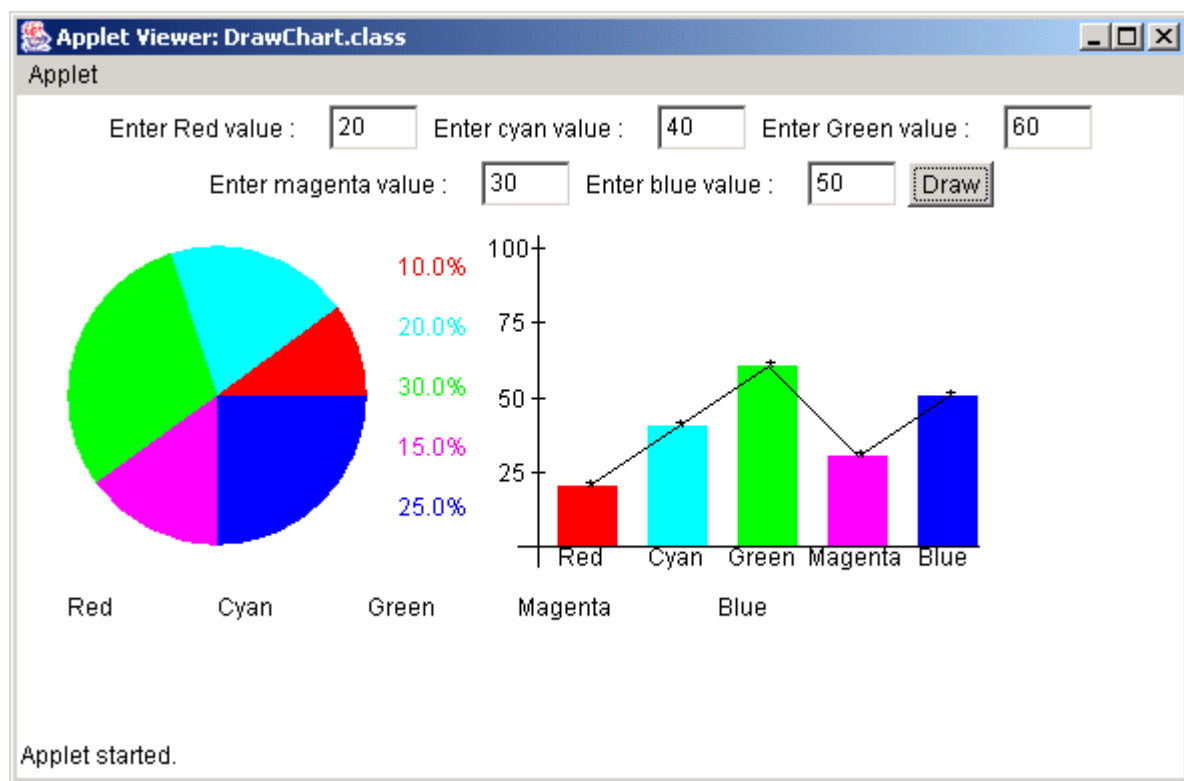


Applet Identifiers

Khi người sử dụng nhập vào ô văn bản và click chuột vào nút 'Check identifier' applet phải kiểm tra xem từ có trong ô văn bản có phải là một từ khoá có trong java không.

Sử dụng các font chữ khác nhau cho ô văn bản, nút lệnh và nhãn.

Viết applet như sau:



Applet DrawChart

Người sử dụng được phép nhập vào giá trị màu trong các ô tương ứng. Khi click nút 'Draw' các biểu đồ dạng đường, bar, pie được hiện ra.

Xử lý ngoại lệ (Exception Handling)

Giới thiệu

Exception là một loại lỗi đặc biệt. Lỗi này xuất hiện vào lúc thực thi chương trình. Các trạng thái không bình thường xảy ra trong khi thi hành chương trình tạo ra các exception. Những trạng thái này không được biết trước trong khi ta đang xây dựng chương trình. Nếu bạn không xử lý các trạng thái này thì chương trình có thể bị kết thúc đột ngột. Ví dụ, việc chia cho 0 sẽ tạo một lỗi trong chương trình. Ngôn ngữ Java cung cấp cơ chế dùng để xử lý ngoại lệ rất hiệu quả. Việc xử lý này làm hạn chế tối đa trường hợp hệ thống bị hỏng (crash) hay hệ thống bị ngắt đột ngột. Tính năng này làm cho Java trở thành một ngôn ngữ lập trình mạnh.

Mục đích của việc xử lý ngoại lệ

Một chương trình nên có cơ chế xử lý ngoại lệ thích hợp. Nếu không, chương trình sẽ bị ngắt khi một ngoại lệ xảy ra. Trong trường hợp đó, tất cả các nguồn tài nguyên mà hệ thống đã cấp không được giải phóng. Điều này gây lãng phí tài nguyên. Để tránh trường hợp này, tất cả các nguồn tài nguyên mà hệ thống cấp nên được thu hồi lại. Tiến trình này đòi hỏi cơ chế xử lý ngoại lệ thích hợp.

Xét thao tác vào ra (I/O) trong một tập tin. Nếu việc chuyển đổi kiểu dữ liệu không thực hiện đúng, một ngoại lệ sẽ xảy ra và chương trình bị hủy mà không đóng tập tin lại. Lúc đó tập tin dễ bị hư hại và các nguồn tài nguyên được cấp phát cho tập tin không được trả lại cho hệ thống.

Xử lý ngoại lệ

Khi một ngoại lệ xảy ra, đối tượng (object) tương ứng với ngoại lệ đó được tạo ra. Đối tượng này sau đó được truyền cho phương thức là nơi mà ngoại lệ xảy ra. Đối tượng này chứa thông tin chi tiết về ngoại lệ. Thông tin này có thể được nhận về và được xử lý. Các môi trường runtime như 'IllegalAccessException', 'EmptyStackException' v.v... có thể tạo ra ngoại lệ. Chương trình đôi khi có thể tự tạo ra ngoại lệ. Lớp 'Throwable' được Java cung cấp là lớp trên cùng của lớp Exception (lớp đầu tiên trong cây thừa kế), lớp này là lớp cha của tất cả các ngoại lệ khác.

Mô hình xử lý ngoại lệ

Trong Java, mô hình xử lý ngoại lệ giám sát việc thực thi mã để phát hiện ngoại lệ. Mô hình xử lý ngoại lệ của Java được gọi là 'catch and throw'. Trong mô hình này, khi một ngoại lệ xảy ra, ngoại lệ sẽ bị chặn và chương trình chuyển đến một khối xử lý ngoại lệ.

Người lập trình phải xử lý các ngoại lệ khác nhau có thể phát sinh trong chương trình. Các ngoại lệ phải được xử lý, hoặc thoát khỏi chương trình khi nó xảy ra.

Ngôn ngữ Java cung cấp 5 từ khoá sau để xử lý các ngoại lệ:

- try
- catch
- throw
- throws
- finally

Dưới đây là cấu trúc của mô hình xử lý ngoại lệ:

```
try { // đoạn mã có khả năng gây ra ngoại lệ }  
catch(Exception e1) { // Nếu các lệnh trong khối 'try' tạo  
ra ngoại lệ có loại e1, thì thực hiện //xử lý ngoại lệ nếu  
không chuyển xuống khối 'catch' tiếp theo }  
catch(Exception e2) { // Nếu các lệnh trong khối 'try' tạo  
ra ngoại lệ có loại e2, thì thực hiện //xử lý ngoại lệ nếu  
không chuyển xuống khối 'catch' tiếp theo }  
catch(Exception eN) { // Nếu các lệnh trong khối 'try' tạo  
ra ngoại lệ có loại eN, thì thực hiện //xử lý ngoại lệ nếu  
không chuyển xuống khối 'catch' tiếp theo } finally { //  
khối lệnh nay luôn được thực hiện cho dù ngoại lệ có xảy  
ra hay không. }
```

Các ưu điểm của mô hình ‘catch và throw’

Mô hình ‘catch và throw’ có hai ưu điểm:

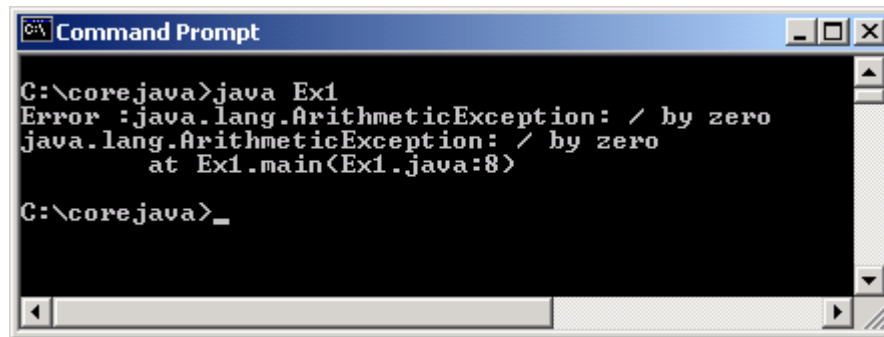
- Người lập trình chỉ phải xử lý ngoại lệ khi cần thiết. Không cần phải thực hiện tại mọi mức.
- Thông báo lỗi có thể được hiện ra khi tiến hành xử lý ngoại lệ.

Các khối ‘try’ và ‘catch’

Khối ‘try-catch’ được sử dụng để thi hành mô hình ‘catch và throw’ của việc xử lý ngoại lệ. Khối ‘try’ chứa một tập lệnh có thể thi hành được. Các ngoại lệ có thể bị chặn khi thi hành tập lệnh này. Phương thức có khả năng tạo ra ngoại lệ có thể được khai báo trong khối ‘try’. Một hay nhiều khối ‘catch’ có thể theo sau một khối ‘try’. Các khối ‘catch’ này bắt các ngoại lệ có khả năng tạo ra trong khối ‘try’. Hãy xem khối ‘try’ dưới đây:

```
try { doFileProcessing(); // phương thức do người sử dụng
định nghĩa displayResults(); } catch (Exception e) // thẻ
hiện của ngoại lệ { System.err.println("Error :" +
e.toString()); e.printStackTrace(); }
```

Ở đây, ‘e’ là đối tượng của lớp ‘**Exception**’. Chúng ta có thể sử dụng đối tượng này để in các chi tiết về ngoại lệ. Các phương thức ‘**toString**’ và ‘**printStackTrace**’ được sử dụng để mô tả các ngoại lệ xảy ra. Hình sau chỉ ra kết xuất của phương thức ‘**printStackTrace**’.



```

C:\corejava>java Ex1
Error :java.lang.ArithmeticException: / by zero
java.lang.ArithmeticException: / by zero
        at Ex1.main(Ex1.java:8)

C:\corejava>_

```

Khởi Try và Catch

Để xử lý được ngoại lệ nào, ta phải chỉ ra kiểu ngoại lệ tương ứng.

catch(Exception e)

Khi ngoại lệ không biết thuộc kiểu nào, chúng ta có thể sử dụng lớp ‘**Exception**’ để bắt ngoại lệ đó.

Khối ‘**catch()**’ bắt giữ bất cứ các lỗi xảy ra trong khi thi hành phương thức ‘**doFileProcessing**’ hay ‘**display**’. Nếu một lỗi xảy ra trong khi thi hành phương thức ‘**doFileProcessing()**’, lúc đó phương thức ‘**displayResults()**’ sẽ không bao giờ được gọi. Chương trình sẽ chuyển đến thực hiện khối ‘**catch**’. Để có nhiều lớp xử lý lỗi hơn, như là ‘**LookupException**’ thay vì một đối tượng ngoại lệ chung (**Exception**), lỗi thực tế có thể là một đối tượng thuộc lớp ‘**LookupException**’ hay một trong số những lớp con của nó. Lỗi sẽ được truyền qua khối ‘**try catch**’ cho tới khi gặp ‘**catch**’ của nó, nếu không tìm thấy chương trình phải dừng thực hiện và thoát.

Các khối chứa nhiều Catch

Nhiều khối ‘**catch**’ xử lý các loại ngoại lệ khác nhau một cách độc lập. Chúng được liệt kê trong đoạn mã sau:

```
try { doFileProcessing(); displayResults(); }
catch (LookupException e) // e - LookupException object {
```

```
handleLookupException(e); // phương thức xử lý lỗi do
người sử dụng //định nghĩa } catch(Exception e) {
System.err.println("Error:" + e.printStackTrace()); } }
```

Trong trường hợp này, khối ‘catch’ đầu tiên sẽ bắt giữ một ‘**LockupException**’. Khối ‘catch’ thứ hai sẽ xử lý kiểu ngoại lệ khác với khối ‘catch’ thứ nhất.

Một chương trình cũng có thể chứa các khối ‘try’ lồng nhau. Ví dụ đoạn mã dưới đây:

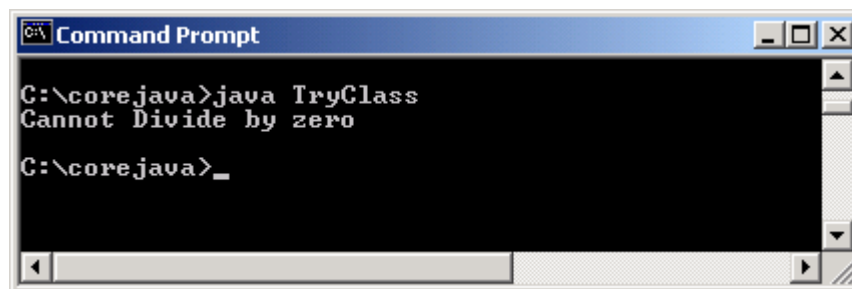
```
try { statement 1; statement 2; try { statement1;
statement2; } catch(Exception e) // của khối try trong { }
} catch(Exception e) // của khối try ngoài { } ...
```

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.

chương trình dưới đây sẽ 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"); } } }
```

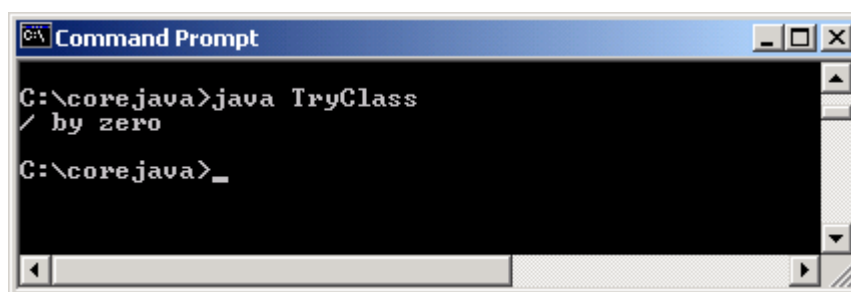
Kết xuất của chương trình:



Cannot divide by zero

Trong chương trình này, một số được chia cho 0. Đây không là phép toán số học hợp lệ. Do đó một ngoại lệ xảy ra và được bắt giữ trong khối catch. Khi nhận biết được loại ngoại lệ nào có thể xảy ra, ta viết lệnh trong khối ‘catch’ tương ứng. Ở đây, ‘a’ được sử dụng như một đối tượng của ArithmeticException để in các chi tiết về ngoại lệ. 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:\core.java>java TryClass
by zero
C:\core.java>_
```

Câu thông báo lỗi

Khi các khối ‘try’ được sử dụng mà không có các khối ‘catch’ nào, chương trình sẽ biên dịch mà không gặp lỗi nào nhưng sẽ bị ngắt khi thực thi. Bởi vì ngoại lệ đã xảy ra khi thực thi chương trình mà không được xử lý.

Khối ‘finally’

Khi một ngoại lệ xuất hiện, phương thức đang được thực thi có thể bị dừng mà không được hoàn thành. Nếu điều này xảy ra, thì các đoạn mã phía sau (ví dụ như đoạn mã có chức năng thu hồi tài nguyên, như các lệnh đóng tập viết ở cuối phương thức) sẽ không bao giờ được gọi. Java cung cấp khối ‘finally’ để giải quyết việc này. Khối ‘finally’ thực hiện tất cả các việc thu dọn khi một ngoại lệ xảy ra. Khối này có thể được sử dụng kết hợp với khối ‘try’. Khối ‘finally’ chứa các câu lệnh thu hồi tài nguyên về cho hệ thống hay lệnh in ra các câu thông báo. Các lệnh này bao gồm:

- Đóng tập tin.
- Đóng ResultSet (được sử dụng trong chương trình cơ sở dữ liệu).
- Đóng lại các kết nối được tạo trong cơ sở dữ liệu.

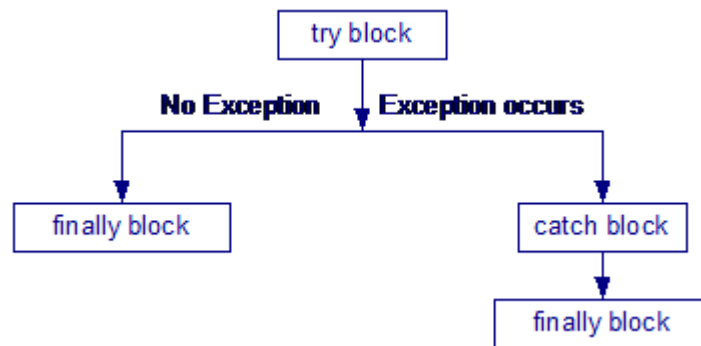
```
try { doSomethingThatMightThrowAnException(); } finally {
cleanup(); }
```

Phương thức ‘**cleanup()**’ được gọi nếu phương thức ‘**doSomethingThatMightThrowAnException()**’ gây ra ngoại lệ. Mặt khác ‘**cleanup()**’ cũng được gọi ngay khi không có ngoại lệ nào xảy ra và thực hiện tiếp phần sau khối lệnh ‘**finally**’.

Khối ‘**finally**’ là tùy ý, không bắt buộc. Khối này được đặt sau khối ‘catch’ cuối cùng. Chương trình sẽ thực thi câu lệnh đầu tiên của khối ‘finally’ ngay sau khi gặp câu lệnh ‘**return**’ hay lệnh ‘**break**’ trong khối ‘**try**’.

Khối ‘**finally**’ bảo đảm lúc nào cũng được thực thi, bất chấp có ngoại lệ xảy ra hay không.

[\[link\]](#) Minh họa sự thực hiện của các khối ‘try’, ‘catch’ và ‘finally’.



Khối lệnh ‘try’, ‘catch’ và ‘finally’

Chương trình dưới đây sẽ sử dụng khối ‘**finally**’. Ở đây, khối ‘**finally**’ được thi hành bất chấp ‘**ArithmeticException**’ có xảy ra hay không. Khối này khai báo các hoạt động thu dọn.

```
class FinallyDemo { String name; int no1,no2;
FinallyDemo(String args[]) { try { name=new String("Aptech
Limited"); no1=Integer.parseInt(args[0]);
no2=Integer.parseInt(args[1]); System.out.println(name);
System.out.println("Division Result is" + no1/no2); }
catch(ArithmeticException i) { System.out.println("Cannot
Divide by zero"); } finally { name=null; // clean up code
System.out.println("Finally executed"); } } public static
void main(String args[]) { new FinallyDemo(args); } }
```

Kết xuất của chương trình:

A screenshot of a Windows Command Prompt window. The title bar reads 'Command Prompt'. The command prompt shows the following sequence of text: 'C:\core.java>java FinallyDemo 12 2', followed by the program's output: 'Aptech Limited', 'Division Result is 6', and 'Finally executed'. The prompt then returns to 'C:\core.java>'.

Khối Finally

Trong ví dụ này, các câu lệnh trong khối ‘**finally**’ luôn luôn thi hành, bất chấp ngoại lệ có xảy ra hay không. Trong kết xuất trên, khối ‘**finally**’ được thi hành mặc dù không có ngoại lệ xảy ra.

Các ngoại lệ được định nghĩa với lệnh ‘throw’ và ‘throws’

Các ngoại lệ có thể được tạo ra bằng cách sử dụng từ khoá ‘**throw**’. Từ khoá ‘**throw**’ chỉ ra một ngoại lệ vừa xảy ra. Toán hạng của throw là một đối tượng thuộc lớp được thừa kế từ ‘**Throwable**’.

Cách sử dụng của lệnh ‘throw’

```
try { if (flag<0) { throw new MyException(); // user-defined } }
```

Một phương thức có thể tạo ra nhiều ngoại lệ. Để làm được điều này này, ta chỉ cần liệt kê danh sách các ngoại lệ mà phương thức có thể tạo ra trong phần định nghĩa phương thức. Giả sử rằng phương thức ‘**x()**’ gọi phương thức ‘**y()**’. Phương thức ‘**y()**’ tạo ra một ngoại lệ nhưng không được xử lý. Trong trường hợp này, phương thức gọi ‘**x()**’ nên khai báo rằng nó có khả năng tạo ra ngoại lệ như ngoại lệ của 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 gọi **x()**).

Đoạn mã minh họa cách sử dụng của từ khoá ‘**throws**’ để tạo nhiều ngoại lệ

```
public class Example { // Các ngoại lệ cách nhau bởi dấu phẩy public void  
exceptionExample() throws ExException, LookupException { try { // các lệnh }  
catch (ExException exmp) { } catch (LookupException lkpex) { } } }
```

Trong ví dụ trên, phương thức ‘**exceptionExample**’ có từ khoá ‘**throws**’. Từ khoá này được theo sau bởi danh sách các ngoại lệ mà phương thức này có thể tạo ra – Trong trường hợp này là ‘**ExException**’ và ‘**LookupException**’. Hàm xử lý ngoại lệ cho các phương thức này nên khai báo các khối ‘catch’ để có thể xử lý tất cả các ngoại lệ mà các phương có thể gây ra.

Lớp ‘**Exception**’ thực thi giao diện ‘**Throwable**’ và cung cấp các tính năng để làm việc với ngoại lệ. Nó có ý nghĩa trong trường hợp các lớp ngoại lệ được định nghĩa bởi người dùng. Để làm điều này, một lớp con của lớp Exception được tạo ra. Ưu điểm của việc thừa kế lớp **Exception** là loại ngoại lệ mới này có thể được ‘catch’ độc lập với các loại **Throwable** khác.

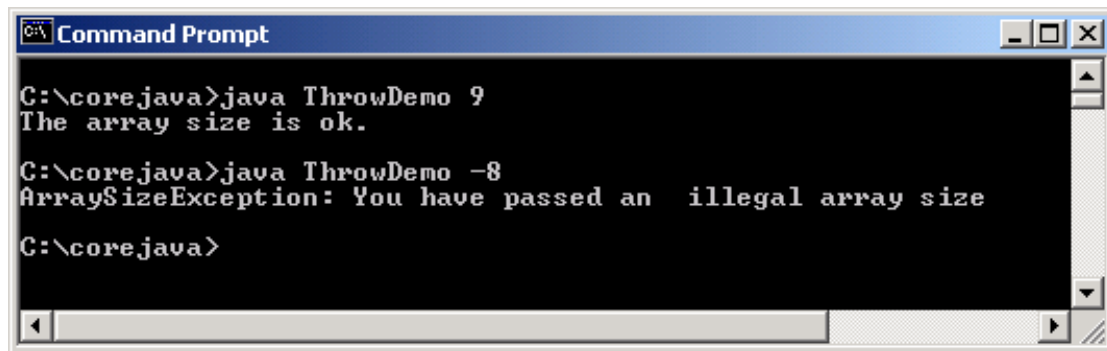
Chương trình minh họa ngoại lệ được định nghĩa bởi người dùng ‘**ArraySizeException**’ :

```
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 tạo ra ngoại lệ ‘**ArraySizeException**’ mà được chỉ ra bởi lệnh ‘**throws**’. Kích thước của mảng được kiểm tra trong cấu trúc ‘**if**’. Nếu kích thước là số âm thì đối tượng của lớp ‘**ArraySizeException**’ được tạo.

Kết xuất của chương trình được chỉ ra ở [\[link\]](#) .



```

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>

```

Ngoại lệ tự định nghĩa

Danh sách các ngoại lệ

Bảng sau đây liệt kê một số ngoại lệ:

Danh sách một số ngoại lệ

Ngoại lệ	Lớp cha của thứ tự phân cấp ngoại lệ
RuntimeException	Lớp cơ sở cho nhiều ngoại lệ java.lang
ArithmeticException	Lỗi về số học, ví dụ như ‘chia cho 0’.
IllegalAccessException	Lớp không thể truy cập.
IllegalArgumentException	Đối số không hợp lệ.
ArrayIndexOutOfBoundsException	Lỗi tràn mảng.
NullPointerException	Khi truy cập đối tượng null.
SecurityException	Cơ chế bảo mật không cho phép thực hiện.
ClassNotFoundException	Không thể nạp lớp yêu cầu.

NumberFormatException	Việc chuyển đổi từ chuỗi sang số thực không thành công.
AWTException	Ngoại lệ về AWT
IOException	Lớp cha của các lớp 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.

Tóm tắt

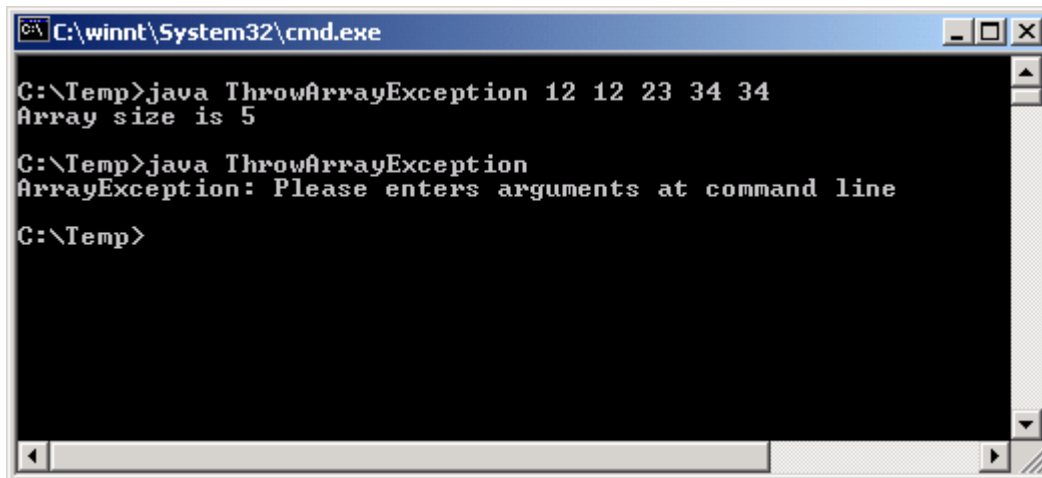
- Bất cứ khi nào một lỗi xuất hiện trong khi thi hành chương trình, nghĩa là một ngoại lệ đã xuất hiện.
- Ngoại lệ phát sinh vào lúc thực thi chương trình theo trình tự mã.
- Mỗi ngoại lệ phát sinh ra phải được xử lý, nếu không ứng dụng sẽ bị ngắt.
- Việc xử lý ngoại lệ cho phép bạn kết hợp tất cả tiến trình xử lý lỗi trong một nơi. Lúc đó đoạn mã của bạn sẽ rõ ràng hơn.
- Java sử dụng các khối 'try' và 'catch' để xử lý các ngoại lệ. Các câu lệnh trong khối 'try' gây ra ngoại lệ còn khối 'catch' xử lý ngoại lệ.
- Nhiều khối catch có thể được sử dụng để xử lý các loại ngoại lệ khác nhau.
- Từ khoá 'throws' liệt kê các ngoại lệ mà phương gây ra.
- Từ khoá 'throw' tạo ra một ngoại lệ.
- Khối 'finally' khai báo các câu lệnh trả về nguồn tài nguyên cho hệ thống và in những thông báo.

Kiểm tra sự tiến bộ

1. Khi một ngoại lệ xảy ra, một đại diện cho ngoại lệ được tạo ra.
2. Lớp là lớp cha của lớp Exception
3. Mô hình xử lý lỗi của Java được gọi là mô hình 'catch and throw'? (**Đúng/Sai**)
4. Ưu điểm của mô hình 'catch and throw' là gì?
5. Khi sử dụng 'try' lồng nhau, 'try' ngoài sẽ được thực hiện trước? (**Đúng/Sai**)
6. Khối 'finally' đảm bảo luôn luôn được thực hiện cho dù có ngoại lệ xảy ra hay không? (**Đúng/Sai**)
7. Mỗi phương thức không có khả năng gây ra nhiều hơn một ngoại lệ. (**Đúng/Sai**)
8. Ngoại lệ được tạo ra khi lớp không thể truy cập được.

Bài tập

Viết chương trình gây ra ngoại lệ khi người sử dụng không nhập tham số nào vào từ dòng lệnh. Chương trình phải hiển thị số tham số nếu có tham số được nhập vào từ dòng lệnh. Đầu ra của chương trình như sau:



```
C:\winnt\System32\cmd.exe

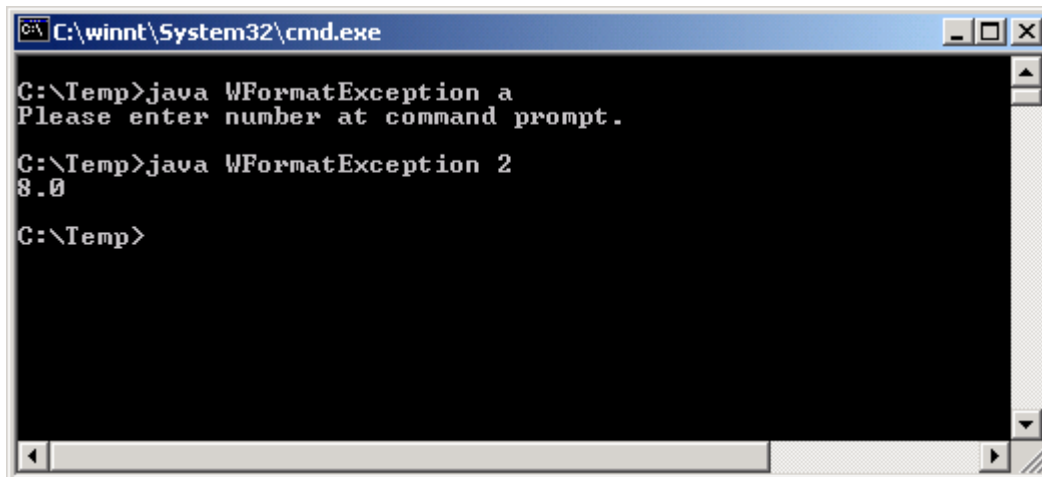
C:\Temp>java ThrowArrayException 12 12 23 34 34
Array size is 5

C:\Temp>java ThrowArrayException
ArrayException: Please enters arguments at command line

C:\Temp>
```

Giao diện chương trình

Viết chương trình gây ra ngoại lệ, nếu không có số nào được nhập vào từ dòng lệnh. Ngược lại, chương trình hiển thị giá trị lập phương của số nhập vào như hình dưới đây:



```
C:\winnt\System32\cmd.exe

C:\Temp>java WFormatException a
Please enter number at command prompt.

C:\Temp>java WFormatException 2
8.0

C:\Temp>
```

Giao diện chương trình

Viết chương trình gây ra ngoại lệ nếu như lớp không thể truy nhập.

Đa luồng (Multithreading)

Giới thiệu

Một luồng là một thuộc tính duy nhất của Java. Nó là đơn vị nhỏ nhất của đoạn mã có thể thi hành được mà thực hiện một công việc riêng biệt. Ngôn ngữ Java và máy ảo Java cả hai là các hệ thống được phân luồng

Đa luồng

Java hỗ trợ đa luồng, mà có khả năng làm việc với nhiều luồng. Một ứng dụng có thể bao hàm nhiều luồng. Mỗi luồng được đăng ký một công việc riêng biệt, mà chúng được thực thi đồng thời với các luồng khác.

Đa luồng giữ thời gian nhàn rỗi của hệ thống thành nhỏ nhất. Điều này cho phép bạn viết các chương trình có hiệu quả cao với sự tận dụng CPU là tối đa. Mỗi phần của chương trình được gọi một luồng, mỗi luồng định nghĩa một đường dẫn khác nhau của sự thực hiện. Đây là một thiết kế chuyên dùng của sự đa nhiệm.

Trong sự đa nhiệm, nhiều chương trình chạy đồng thời, mỗi chương trình có ít nhất một luồng trong nó. Một vi xử lý thực thi tất cả các chương trình. Cho dù nó có thể xuất hiện mà các chương trình đã được thực thi đồng thời, trên thực tế bộ vi xử lý nhảy qua lại giữa các tiến trình.

Tạo và quản lý luồng

Khi các chương trình Java được thực thi, luồng chính luôn luôn đang được thực hiện. Đây là 2 nguyên nhân quan trọng đối với luồng chính:

- Các luồng con sẽ được tạo ra từ nó.
- Nó là luồng cuối cùng kết thúc việc thực hiện. Trong chốc lát luồng chính ngừng thực thi, chương trình bị chấm dứt.

Cho dù luồng chính được tạo ra một cách tự động với chương trình thực thi, nó có thể được điều khiển thông qua một luồng đối tượng.

Các luồng có thể được tạo ra từ hai con đường:

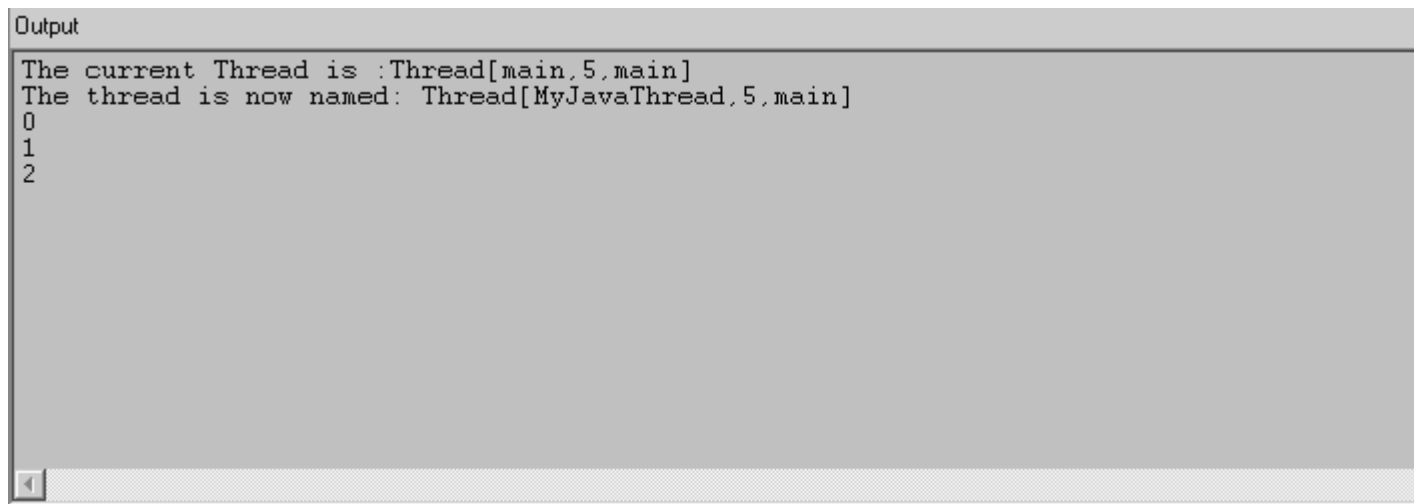
- Trình bày lớp như là một lớp con của lớp luồng, nơi mà phương thức `run()` của lớp luồng cần được ghi đè. Lấy ví dụ: `Class Mydemo extends Thread`

- ```
{ //Class definition public void run() { //thực thi }
}
```
- Trình bày một lớp mà lớp này thực hiện lớp Runnable. Rồi thì định nghĩa phương thức run(). Class Mydemo implements Runnable {  
//Class definition public void run() { //thực thi } }

[\[link\]](#) sẽ chỉ ra sự điều khiển luồng chính như thế nào

```
import java.io.*; public class Mythread extends Thread{
/** * Mythread constructor comment. */ public static void
main(String args[]){ Thread t = Thread.currentThread();
System.out.println("The current Thread is :" + t);
t.setName("MyJavaThread"); System.out.println("The thread
is now named: " + t); try{ for(int i = 0; i <3;i++){
System.out.println(i); Thread.sleep(1500); }
}catch(InterruptedException e){ System.out.println("Main
thread interupted"); } } }
```

Hình sau đây sẽ chỉ ra kết quả xuất ra màn hình của chương trình trên

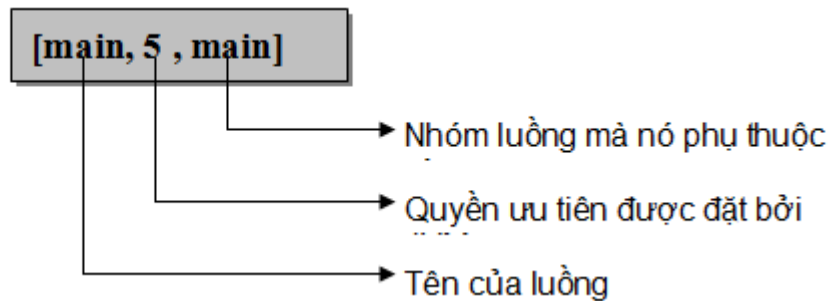


The screenshot shows a window titled "Output" with the following text:

```
The current Thread is :Thread[main,5,main]
The thread is now named: Thread[MyJavaThread,5,main]
0
1
2
```

*Luồng*

Trong kết quả xuất ra ở trên



Mỗi luồng trong chương trình Java được đăng ký cho một quyền ưu tiên. Máy ảo Java không bao giờ thay đổi quyền ưu tiên của luồng. Quyền ưu tiên vẫn còn là hằng số cho đến khi luồng bị ngắt.

Mỗi luồng có một giá trị ưu tiên nằm trong khoảng của một `Thread.MIN_PRIORITY` của 1, và một `Thread.MAX_PRIORITY` của 10. Mỗi luồng phụ thuộc vào một nhóm luồng, và mỗi nhóm luồng có quyền ưu tiên của chính nó. Mỗi luồng được nhận một hằng số ưu tiên của phương thức `Thread.PRIORITY` là 5. Mỗi luồng mới thừa kế quyền ưu tiên của luồng mà tạo ra nó.

Lớp luồng có vài phương thức khởi dựng, hai trong số các phương thức khởi dựng được đề cập đến dưới đây:

- **`public Thread(String threadname)`**

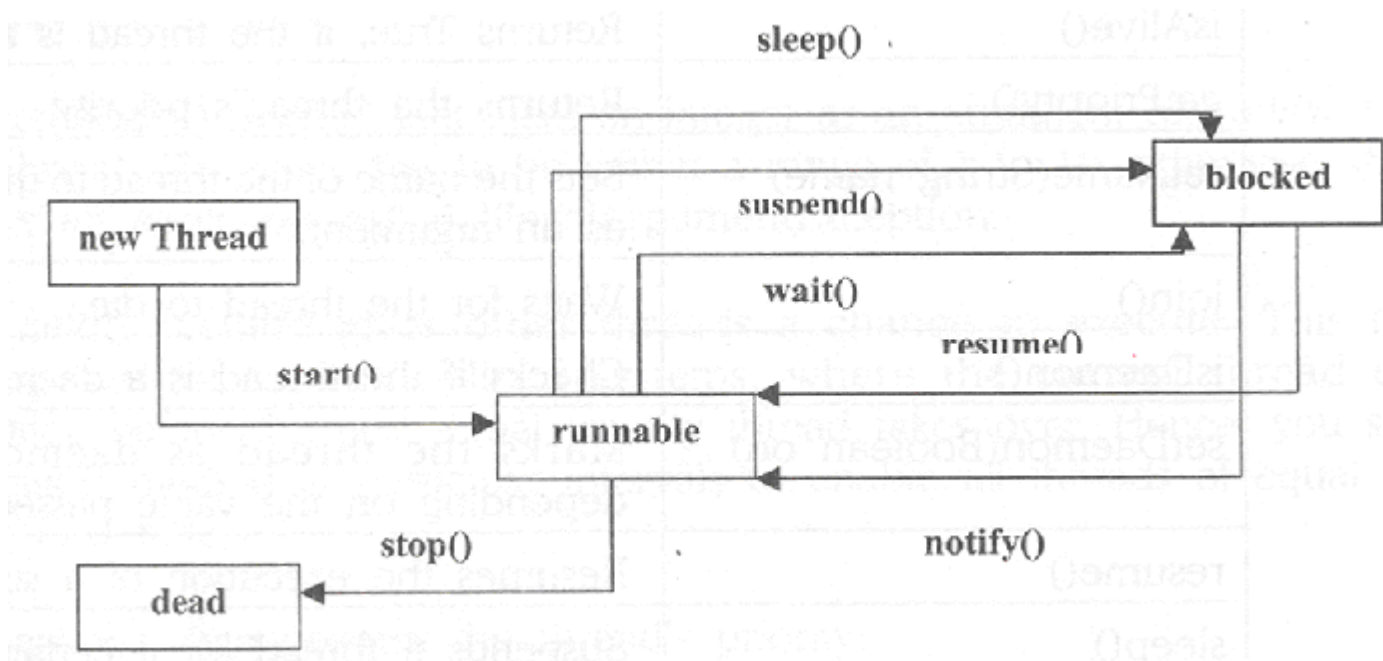
Cấu trúc một luồng với tên là "threadname"

- **`public Thread()`**

Cấu trúc một luồng với tên "Thread, được ràng buộc với một số; lấy ví dụ, Thread-1, Thread-2, v.v...

Chương trình bắt đầu thực thi luồng với việc gọi phương thức `start()`, mà phương thức này phụ thuộc vào lớp luồng. Phương thức này, lần lượt, viện dẫn phương thức `run()`, nơi mà phương thức định nghĩa tác vụ được thực thi. Phương thức này có thể viết đè lên lớp con của lớp luồng, hoặc với một đối tượng `Runnable`.

## Vòng đời của Luồng



*Vòng đời của luồng*

### Phạm vi của luồng và các phương thức của lớp luồng

Một luồng đã được tạo mới gần đây là trong phạm vi “sinh”. Luồng không bắt đầu chạy ngay lập tức sau khi nó được tạo ra. Nó đợi phương thức `start()` của chính nó được gọi. Cho đến khi, nó là trong phạm vi “sẵn sàng để chạy”. Luồng đi vào phạm vi “đang chạy” khi hệ thống định rõ vị trí luồng trong bộ vi xử lý.

Bạn có thể sử dụng phương thức `sleep()` để tạm thời treo sự thực thi của luồng. Luồng trở thành sẵn sàng sau khi phương thức `sleep` kết thúc thời gian. Luồng Sleeping không sử dụng bộ vi xử lý. luồng đi vào phạm vi “waiting” (đợi) luồng đang chạy gọi phương thức `wait()` (đợi).

Khi các luồng khác liên kết với các đối tượng, gọi phương thức `notify()`, luồng đi vào trở lại phạm vi “ready” (sẵn sàng) Luồng đi vào phạm vi “blocked” (khởi) khi nó đang thực thi các phép toán vào/ra (Input/output). Nó đi vào phạm vi “ready” (sẵn sàng) khi các phương thức vào/ra nó đang đợi cho đến khi được hoàn thành. Luồng đi vào phạm vi “dead” (chết) sau khi phương thức `run()` đã được thực thi hoàn toàn, hoặc khi phương thức `stop()` (dừng) của nó được gọi.

Thêm vào các phương thức đã được đề cập trên, Lớp luồng cũng có các phương thức sau:

### Các phương thức của một lớp luồng

| Phương thức                 | Mục đích                                                                                                  |
|-----------------------------|-----------------------------------------------------------------------------------------------------------|
| <b>Enumerate(Thread t)</b>  | Sao chép tất cả các luồng hiện hành vào mảng được chỉ định từ nhóm của các luồng, và các nhóm con của nó. |
| getName()                   | Trả về tên của luồng                                                                                      |
| isAlive()                   | Trả về Đúng, nếu luồng vẫn còn tồn tại (sống)                                                             |
| getPriority()               | Trả về quyền ưu tiên của luồng                                                                            |
| <b>setName(String name)</b> | Đặt tên của luồng là tên mà luồng được truyền như là một tham số.                                         |
| join()                      | Đợi cho đến khi luồng chết.                                                                               |
| isDaemon(Boolean on)        | Kiểm tra nếu luồng là luồng một luồng hiểm.                                                               |
| resume()                    | Đánh dấu luồng như là luồng hiểm hoặc luồng người sử dụng phụ thuộc vào giá trị được truyền vào.          |
| sleep()                     | Hoãn luồng một khoảng thời gian chính xác.                                                                |
| start()                     | Gọi phương thức run() để bắt đầu một luồng.                                                               |

Bảng kế hoạch Round-robin (bảng kiến nghị ký tên vòng tròn) liên quan đến các luồng với cùng quyền ưu tiên được chiếm hữu quyền ưu tiên của mỗi luồng khác. Chúng chia nhỏ thời gian một cách tự động trong theo kiểu kế hoạch xoay vòng này.

Phiên bản mới nhất của Java không hỗ trợ các phương thức **Thread.suspend()** (tri hoãn), **Thread.resume()** (phục hồi) và **Thread.stop()** (dừng), như là các phương thức **resume()** (phục hồi) và **suspend()** (tri hoãn) được thiên về sự đình trệ (**deadlock**), trong khi phương thức **stop()** không an toàn.

### Thời gian biểu luồng

Hầu hết các chương trình Java làm việc với nhiều luồng. CPU chứa đựng cho việc chạy chương trình chỉ một luồng tại một khoảng thời gian. Hai luồng có cùng quyền ưu tiên trong một chương trình hoàn thành trong một thời gian CPU. Lập trình viên, hoặc máy ảo Java, hoặc hệ điều hành chắc chắn rằng CPU được chia sẻ giữa các luồng. Điều này được biết như là bảng thời gian biểu luồng.

Không có máy ảo Java nào thực thi rành mạch cho bảng thời gian biểu luồng. Một số nền Java hỗ trợ việc chia nhỏ thời gian. Ở đây, mỗi luồng nhận một phần nhỏ của thời gian bộ vi xử lý, được gọi là định lượng. Luồng có thể thực thi tác vụ của chính nó trong

suốt khoảng thời gian định lượng đấy. Sau khoảng thời gian này được vượt qua, luồng không được nhận nhiều thời gian để tiếp tục thực hiện, ngay cả nếu nó không được hoàn thành việc thực hiện của nó. Luồng kế tiếp của luồng có quyền ưu tiên bằng nhau này sẽ lấy khoảng thời gian thay đổi của bộ vi xử lý. Java là người lập thời gian biểu chia nhỏ tất cả các luồng có cùng quyền ưu tiên cao.

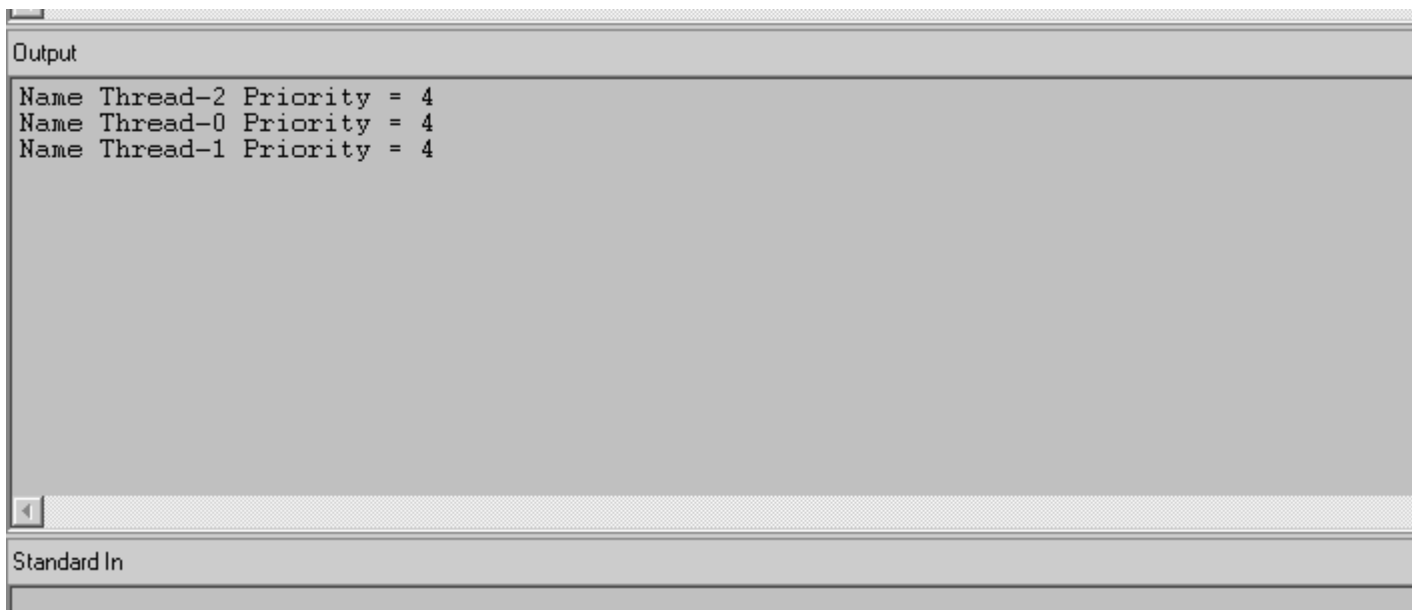
Phương thức **setPriority()** lấy một số nguyên (**integer**) như là một tham số có thể hiệu chỉnh quyền ưu tiên của một luồng. Đây là giá trị có phạm vi thay đổi từ 1 đến 10, mặc khác, phương thức đưa ra một ngoại lệ (bẫy lỗi) được gọi là **IllegalArgumentExpection** (Chấp nhận tham số trái luật)

Phương thức **yield()** (lợi nhuận) đưa ra các luồng khác một khả năng để thực thi. Phương thức này được thích hợp cho các hệ thống không chia nhỏ thời gian (non-time-sliced), nơi mà các luồng hiện thời hoàn thành việc thực hiện trước khi các luồng có quyền ưu tiên ngang nhau kế tiếp tiếp quản. Ở đây, bạn sẽ gọi phương thức **yield()** tại những khoản thời gian riêng biệt để có thể tất cả các luồng có quyền ưu tiên ngang nhau chia sẻ thời gian thực thi CPU.

[\[link\]](#) chứng minh quyền ưu tiên của luồng:

```
class PriorityDemo { Priority t1,t2,t3; public
PriorityDemo(){ t1 = new Priority(); t1.start(); t2 = new
Priority(); t2.start(); t3 = new Priority(); t3.start(); }
public static void main(String args[]){ new
PriorityDemo(); } class Priority extends Thread implements
Runnable{ int sleep; int prio = 3; public Priority(){
sleep += 100; prio++; setPriority(prio); } public void
run(){ try{ Thread.sleep(sleep); System.out.println("Name
"+ getName()+" Priority = "+ getPriority());
}catch(InterruptedException e){
System.out.println(e.getMessage()); } } }
```

Kết quả hiển thị như [\[link\]](#)



### *Quyền ưu tiên luồng*

## **Luồng hiểm**

Một chương trình Java bị ngắt chỉ sau khi tất cả các luồng bị chết. Có hai kiểu luồng trong một chương trình Java:

- Các luồng người sử dụng
- Luồng hiểm

Người sử dụng tạo ra các luồng người sử dụng, trong khi các luồng được chỉ định như là luồng “background” (nền). Luồng hiểm cung cấp các dịch vụ cho các luồng khác. Máy ảo Java thực hiện tiến trình thoát, khi và chỉ khi luồng hiểm vẫn còn sống. Máy ảo Java có ít nhất một luồng hiểm được biết đến như là luồng “garbage collection” (thu lượm những dữ liệu vô nghĩa - dọn rác). Luồng dọn rác thực thi chỉ khi hệ thống không có tác vụ nào. Nó là một luồng có quyền ưu tiên thấp. Lớp luồng có hai phương thức để thỏa thuận với các luồng hiểm:

- `public void setDaemon(boolean on)`
- `public boolean isDaemon()`

## **Đa luồng với Applets**

Trong khi đa luồng là rất hữu dụng trong các chương trình ứng dụng độc lập, nó cũng đáng được quan tâm với các ứng dụng trên Web. Đa luồng được sử dụng trên web, cho ví dụ, trong các trò chơi đa phương tiện, các bức ảnh đầy sinh khí, hiển thị các dòng chữ



chạy qua lại trên biểu ngữ, hiển thị đồng hồ thời gian như là một phần của trang Web v.vv... Các chức năng này cấu thành các trang web làm quyến rũ và bắt mắt.

Chương trình Java dựa trên Applet thường sử dụng nhiều hơn một luồng. Trong đa luồng với Applet, lớp `java.applet.Applet` là lớp con được tạo ra bởi người sử dụng định nghĩa applet. Từ đó, Java không hỗ trợ nhiều kế thừa với các lớp, nó không thể thực hiện được trực tiếp lớp con của lớp luồng trong các applet. Tuy nhiên, chúng ta sử dụng một đối tượng của luồng người sử dụng đã định nghĩa, mà các luồng này, lần lượt, dẫn xuất từ lớp luồng. Một luồng đơn giản xuất hiện sẽ được thực thi tại giao diện (Interface) `Runnable`

[\[link\]](#) chỉ ra điều này thực thi như thế nào:

```
import java.awt.*; import java.applet.*; public class
Myapplet extends Applet implements Runnable { int i;
Thread t; /** * Myapplet constructor comment. */ public
void init(){ t = new Thread(this); t.start(); } public
void paint(Graphics g){ g.drawString(" i = "+i,30,30); }
public void run(){ for(i = 1;i<=20;i++){ try{ repaint();
Thread.sleep(500); }catch(InterruptedException e){
System.out.println(e.getMessage()); } } } }
```

Trong chương trình này, chúng ta tạo ra một Applet được gọi là `Myapplet`, và thực thi giao diện `Runnable` để cung cấp khả năng đa luồng cho applet. Sau đó, chúng ta tạo ra một thể nghiệm (**instance**) cho lớp luồng, với thể nghiệm applet hiện thời như là một tham số để thiết lập (khởi dựng). Rồi thì chúng ta viện dẫn phương thức `start()` của luồng thể nghiệm này. Lần lượt, rồi sẽ viện dẫn phương thức `run()`, mà phương thức này thực sự là điểm bắt đầu cho phương thức này. Chúng ta in số từ 1 đến 20 với thời gian kéo trễ là 500 miligiây giữa mỗi số. Phương thức `sleep()` được gọi để hoàn thành thời gian kéo trễ này. Đây là một phương thức tĩnh được định nghĩa trong lớp luồng. Nó cho phép luồng nằm yên (ngủ) trong khoản thời gian hạn chế.

Xuất ra ngoài có dạng như sau:



*Đa luồng với Applet*

## Nhóm luồng

Một lớp nhóm luồng (**ThreadGroup**) nắm bắt một nhóm của các luồng. Lấy ví dụ, một nhóm luồng trong một trình duyệt có thể quản lý tất cả các luồng phụ thuộc vào một đơn thể applet. Tất cả các luồng trong máy ảo Java phụ thuộc vào các nhóm luồng mặc định. Mỗi nhóm luồng có một nhóm nguồn cha. Vì thế, các nhóm từ một cấu trúc dạng cây. Nhóm luồng “hệ thống” là gốc của tất cả các nhóm luồng. Một nhóm luồng có thể là thành phần của cả các luồng, và các nhóm luồng.

Hai kiểu nhóm luồng thiết lập (khởi dựng) là:

- **public ThreadGroup(String str)**

Ở đây, “str” là tên của nhóm luồng mới nhất được tạo ra.

- **public ThreadGroup(ThreadGroup tgroup, String str)**

Ở đây, “tgroup” chỉ ra luồng đang chạy hiện thời như là luồng cha, “str” là tên của nhóm luồng đang được tạo ra.

Một số các phương thức trong nhóm luồng (ThreadGroup) được cho như sau:

- **public synchronized int activeCount()**

Trả về số lượng các luồng kích hoạt hiện hành trong nhóm luồng

- **public synchronized int activeGroupCount()**

Trả về số lượng các nhóm hoạt động trong nhóm luồng

- **public final String getName()**

Trả về tên của nhóm luồng

- **public final ThreadGroup getParent()**

Trả về cha của nhóm luồng

## Sự đồng bộ luồng

Trong khi đang làm việc với nhiều luồng, nhiều hơn một luồng có thể muốn thâm nhập cùng biến tại cùng thời điểm. Lấy ví dụ, một luồng có thể cố gắng đọc dữ liệu, trong khi luồng khác cố gắng thay đổi dữ liệu. Trong trường hợp này, dữ liệu có thể bị sai lệch.

Trong những trường hợp này, bạn cần cho phép một luồng hoàn thành trọn vẹn tác vụ của nó (thay đổi giá trị), và rồi thì cho phép các luồng kế tiếp thực thi. Khi hai hoặc nhiều hơn các luồng cần thâm nhập đến một tài nguyên được chia sẻ, bạn cần chắc chắn rằng tài nguyên đó sẽ được sử dụng chỉ bởi một luồng tại một thời điểm. Tiến trình này được gọi là “sự đồng bộ” (synchronization) được sử dụng để lưu trữ cho vấn đề này, Java cung cấp duy nhất, ngôn ngữ cấp cao hỗ trợ cho sự đồng bộ này. Phương thức “đồng bộ” (synchronized) báo cho hệ thống đặt một khóa vòng một tài nguyên riêng biệt.

Mấu chốt của sự đồng bộ hóa là khái niệm “monitor” (sự quan sát, giám sát), cũng được biết như là một bảng mã “semaphore” (bảng mã). Một “monitor” là một đối tượng mà được sử dụng như là một khóa qua lại duy nhất, hoặc “mutex”. Chỉ một luồng có thể có riêng nó một sự quan sát (monitor) tại mỗi thời điểm được đưa ra. Tất cả các luồng khác cố gắng thâm nhập vào monitor bị khóa sẽ bị trì hoãn, cho đến khi luồng đầu tiên thoát khỏi monitor. Các luồng khác được báo chờ đợi monitor. Một luồng mà monitor của riêng nó có thể thâm nhập trở lại cùng monitor.

## Mã đồng bộ

Tất cả các đối tượng trong Java được liên kết với các monitor (sự giám sát) của riêng nó. Để đăng nhập vào monitor của một đối tượng, lập trình viên sử dụng từ khóa synchronized (đồng bộ) để gọi một phương thức hiệu chỉnh (**modified**). Khi một luồng đang được thực thi trong phạm vi một phương thức đồng bộ (**synchronized**), bất kỳ luồng khác hoặc phương thức đồng bộ khác mà cố gắng gọi nó trong cùng thể nghiệm sẽ phải đợi.

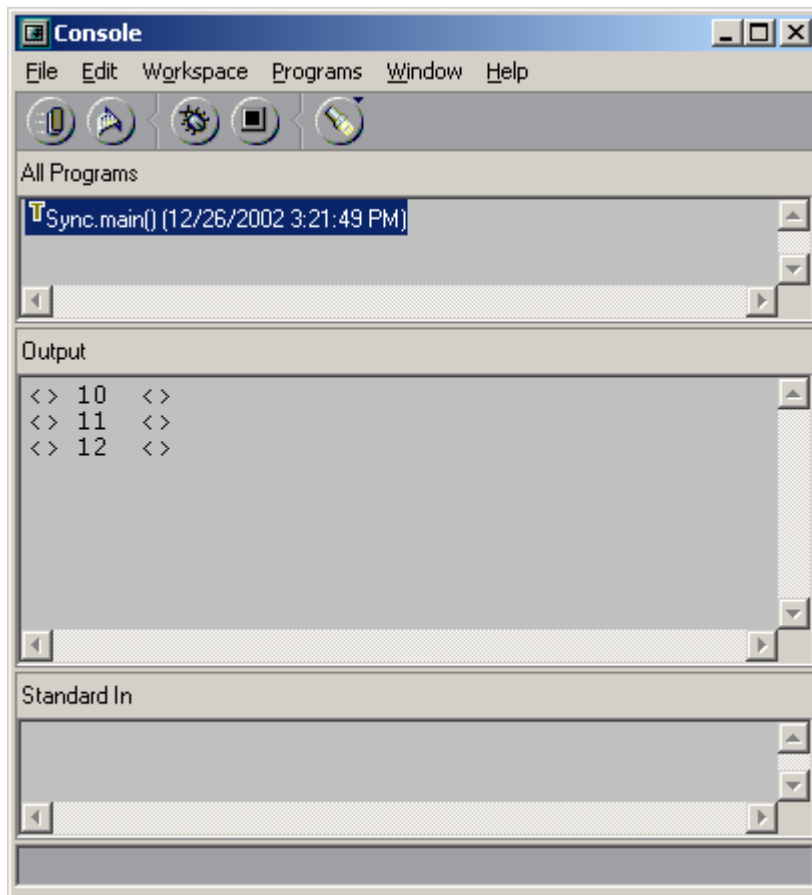
Chương trình 8.4 chứng minh sự làm việc của từ khóa synchronized (sự đồng bộ). Ở đây, lớp “Target” (mục tiêu) có một phương thức “display()” (hiển thị) mà phương thức này lấy một tham số kiểu số nguyên (int). Số này được hiển thị trong phạm vi các cặp

ký tự “< > # số # <>”. Phương thức “Thread.sleep(1000) tạm dừng luồng hiện tại sau khi phương thức “display()” được gọi.

Thiết lập (khởi dựng) của lớp “Source” lấy một tham chiếu đến một đối tượng “t” của lớp “Target”, và một biến số nguyên (integer). Ở đây, một luồng mới cũng được tạo ra. Luồng này gọi phương thức run() của đối tượng “t”. Lớp chính “Synch” thử nghiệm lớp “Target” như là “target (mục tiêu), và tạo ra 3 đối tượng của lớp “Source” (nguồn). Cùng đối tượng “target” được truyền cho mỗi đối tượng “Source”. Phương thức “join()” (gia nhập) làm luồng được gọi đợi cho đến khi việc gọi luồng bị ngắt.

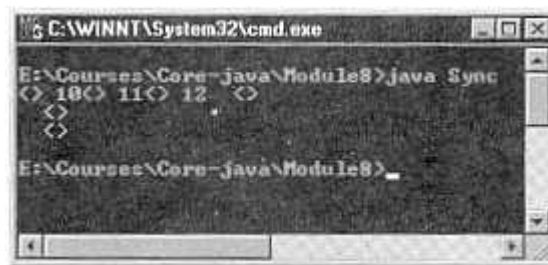
```
class Target { /** * Target constructor comment. */
synchronized void display(int num) { System.out.print("<>
"+num); try{ Thread.sleep(1000);
}catch(InterruptedException e){
System.out.println("Interrupted"); } System.out.println("
<>"); } } class Source implements Runnable{ int number;
Target target; Thread t; /** * Source constructor comment.
*/ public Source(Target targ,int n){ target = targ; number
= n; t = new Thread(this); t.start(); } public void run(){
synchronized(target) { target.display(number); } } } class
Sync { /** * Sync constructor comment. */ public static
void main(String args[]){ Target target = new Target();
int digit = 10; Source s1 = new Source(target,digit++);
Source s2 = new Source(target,digit++); Source s3 = new
Source(target,digit++); try{ s1.t.join(); s2.t.join();
s3.t.join(); }catch(InterruptedException e){
System.out.println("Interrupted"); } } }
```

Kết quả hiện thị như hình cho dưới đây:



Kết quả hiển thị của [\[link\]](#)

Trong chương trình trên, có một “dãy số” đã nhập được hiển thị “**display()**”. Điều này có nghĩa là việc thâm nhập bị hạn chế một luồng tại mỗi thời điểm. Nếu từ khóa `synchronized` đặt trước bị bỏ quên trong phương thức “**display()**” của lớp “**Target**”, tất cả luồng trên có thể cùng lúc gọi cùng phương thức, trên cùng đối tượng. Điều kiện này được biết đến như là “loại điều kiện” (race condition). Trong trường hợp này, việc xuất ra ngoài sẽ được chỉ ra như [\[link\]](#)



Kết quả hiển thị của [\[link\]](#) không có sự đồng bộ

## Sử dụng khối đồng bộ (Synchronized Block)

Tạo ra các phương thức synchronized (đồng bộ) trong phạm vi các lớp là một con đường dễ dàng và có hiệu quả của việc thực hiện sự đồng bộ. Tuy nhiên, điều này không làm việc trong tất cả các trường hợp.

Hãy xem một trường hợp nơi mà lập trình viên muốn sự đồng bộ được xâm nhập vào các đối tượng của lớp mà không được thiết kế cho thâm nhập đa luồng. Tức là, lớp không sử dụng các phương thức đồng bộ. Hơn nữa, mã nguồn là không có giá trị. Vì thế từ khoá synchronized không thể được thêm vào các phương thức thích hợp trong phạm vi lớp.

Để đồng bộ thâm nhập một đối tượng của lớp này, tất cả chúng gọi các phương thức mà lớp này định nghĩa, được đặt bên trong một khối đồng bộ. Tất cả chúng sử dụng chung một câu lệnh đồng bộ được cho như sau:

```
synchronized(object) { // các câu lệnh đồng bộ }
```

Ở đây, “object” (đối tượng) là một tham chiếu đến đối tượng được đồng bộ. Dấu ngoặc móc không cần thiết khi chỉ một câu lệnh được đồng bộ. Một khối đồng bộ bảo đảm rằng nó gọi đến một phương thức (mà là thành phần của đối tượng) xuất hiện chỉ sau khi luồng hiện hành đã được tham nhập thành công vào monitor (sự quan sát) của đối tượng.

[\[link\]](#) chỉ ra câu lệnh đồng bộ sử dụng như thế nào:

```
class Target { /** * Target constructor comment. */
synchronized void display(int num) { System.out.print("<>
"+num); try{ Thread.sleep(1000);
}catch(InterruptedException e){
System.out.println("Interrupted"); } System.out.println("
<>"); } } class Source implements Runnable{ int number;
Target target; Thread t; /** * Source constructor comment.
*/ public Source(Target targ,int n){ target = targ; number
= n; t = new Thread(this); t.start(); } // đồng bộ gọi
phương thức display() public void run(){
synchronized(target) { target.display(number); } } } class
Synchblock { /** * Synchblock constructor comment. */
public static void main(String args[]){ Target target =
new Target(); int digit = 10; Source s1 = new
Source(target,digit++); Source s2 = new
Source(target,digit++); Source s3 = new
Source(target,digit++); try{ s1.t.join(); s2.t.join();
```

```
s3.t.join(); } catch (InterruptedException e) {
System.out.println("Interrupted"); } } }
```

Ở đây, từ khóa `synchronized` không hiệu chỉnh phương thức `display()`. Từ khóa này được sử dụng trong phương thức `run()` của lớp `Target` (mục tiêu). Kết quả xuất ra màn hình tương tự với kết quả chỉ ra ở [\[link\]](#)

### Sự không thuận lợi của các phương thức đồng bộ

Người lập trình thường viết các chương trình trên các đơn thể luồng. Tất nhiên các trạng thái này chắc chắn không lợi ích cho đa luồng. Lấy ví dụ, luồng không tận dụng việc thực thi của trình biên dịch. Trình biên dịch Java từ Sun không chứa nhiều phương thức đồng bộ.

Các phương thức đồng bộ không thực thi tốt như là các phương thức không đồng bộ. Các phương thức này chậm hơn từ ba đến bốn lần so với các phương thức tương ứng không đồng bộ. Trong trạng thái nơi mà việc thực thi là có giới hạn, các phương thức đồng bộ bị ngăn ngừa.

### Kỹ thuật “wait-notify” (đợi – thông báo)

Luồng chia các tác vụ thành các đơn vị riêng biệt và logic (hợp lý). Điều này thay thế các trường hợp (sự kiện) chương trình lặp. Các luồng loại trừ “polling” (kiểm soát vòng).

Một vòng lặp mà lặp lại việc một số điều kiện thường thực thi “**polling**” (kiểm soát vòng). Khi điều kiện nhận giá trị là `True` (đúng), các câu lệnh phức tạp được thực hiện. Đây là tiến trình thường bỏ phí thời gian của CPU. Lấy ví dụ, khi một luồng sinh ra một số dữ liệu, và các luồng khác đang chờ đợi nó, luồng sinh ra phải đợi cho đến khi các luồng sử dụng nó hoàn thành, trước khi phát sinh ra dữ liệu.

Để tránh trường hợp kiểm soát vòng, Java bao gồm một thiết kế tốt trong tiến trình kỹ thuật truyền thông sử dụng các phương thức “**wait()**” (đợi), “**notify()**” (thông báo) và “**notifyAll()**” (thông báo hết). Các phương thức này được thực hiện như là các các phương thức cuối cùng trong lớp đối tượng (**Object**), để mà tất cả các lớp có thể thâm nhập chúng. Tất cả 3 phương thức này có thể được gọi chỉ từ trong phạm vi một phương thức đồng bộ (**synchronized**).

Các chức năng của các phương thức “**wait()**”, “**notify()**”, và “**notifyAll()**” là:

- Phương thức **wait()** nói cho việc gọi luồng trao cho monitor (sự giám sát), và nhập trạng thái “sleep” (chờ) cho đến khi một số luồng khác thâm nhập cùng monitor, và gọi phương thức “**notify()**”.

- Phương thức **notify()** đánh thức, hoặc thông báo cho luồng đầu tiên mà đã gọi phương thức **wait()** trên cùng đối tượng.
- Phương thức **notifyAll()** đánh thức, hoặc thông báo tất cả các luồng mà đã gọi phương thức **wait()** trên cùng đối tượng.
- Quyền ưu tiên cao nhất luồng chạy đầu tiên.

Cú pháp của 3 phương thức này như sau:

```
final void wait() throws IOException final void notify()
final void notifyAll()
```

Các phương thức **wait()** và **notify()** cho phép một đối tượng được chia sẻ để tạm ngừng một luồng, khi đối tượng trở thành không còn giá trị cho luồng. Chúng cũng cho phép luồng tiếp tục khi thích hợp.

Các luồng bản thân nó không bao giờ kiểm tra trạng thái của đối tượng đã chia sẻ.

Một đối tượng mà điều khiển các luồng khách (client) của chính nó theo kiểu này được biết như là một monitor (sự giám sát). Trong các thuật ngữ chặt chẽ của Java, một monitor là bất kỳ đối tượng nào mà có một số mã đồng bộ. Các monitor được sử dụng cho các phương thức **wait()** và **notify()**. Cả hai phương thức này phải được gọi trong mã đồng bộ.

Một số điểm cần nhớ trong khi sử dụng phương thức **wait()**:

- Luồng đang gọi đưa vào CPU
- Luồng đang gọi đưa vào khóa
- Luồng đang gọi đi vào vùng đợi của monitor.

Các điểm chính cần nhớ về phương thức **notify()**

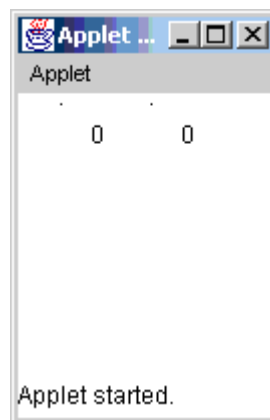
- Một luồng đưa ra ngoài vùng đợi của monitor, và vào trạng thái sẵn sàng.
- Luồng mà đã được thông báo phải thu trở lại khóa của monitor trước khi nó có thể bắt đầu.
- Phương thức **notify()** là không chính xác, như là nó không thể chỉ ra được luồng mà phải được thông báo. Trong một trạng thái đã trộn lẫn, luồng có thể thay đổi trạng thái của monitor trong một con đường mà không mang lại kết quả tốt cho luồng đã được đưa thông báo. Trong một số trường hợp này, các phương thức của monitor đưa ra 2 sự đề phòng:
  - Trạng thái của monitor sẽ được kiểm tra trong một vòng lặp “while” tốt hơn là câu lệnh if
  - Sau khi thay đổi trạng thái của monitor, phương thức **notifyAll()** sẽ được sử dụng, tốt hơn phương thức **notify()**.



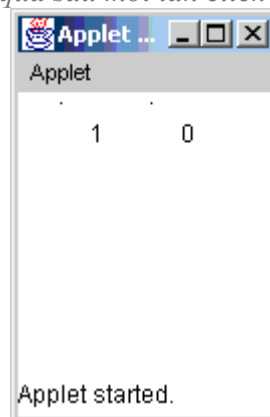
Chương trình biểu thị cho việc sử dụng các phương thức **notify()** và **wait()**

```
import java.applet.*; import java.awt.*; import java.awt.event.*; /*<applet code =
"mouseApplet" width = "100" height = "100"> </applet> */ public class mouseApplet
extends Applet implements MouseListener{ boolean click; int count; public void init()
{ super.init(); add(new clickArea(this)); //doi tuong ve duoc tao ra va them vao
add(new clickArea(this)); //doi tuong ve duoc tao ra va them vao
addMouseListener(this); } public void mouseClicked(MouseEvent e) { } public void
mouseEntered(MouseEvent e) { } public void mouseExited(MouseEvent e) { } public
void mousePressed(MouseEvent e) { synchronized (this) { click = true; notify(); }
count++; //dem viec click Thread.currentThread().yield(); click = false; } public void
mouseReleased(MouseEvent e) { } } //kết thúc Applet class clickArea extends
java.awt.Canvas implements Runnable{ mouseApplet myapp; clickArea(mouseApplet
mapp){ this.myapp = mapp; setSize(40,40); new Thread(this).start(); } public void
paint(Graphics g){ g.drawString(new Integer(myapp.count).toString(),15,20); } public
void run(){ while(true){ synchronized (myapp) { while(!myapp.click){ try{
myapp.wait(); } catch (InterruptedException ie){ } } } repaint(250); } } //end run }
```

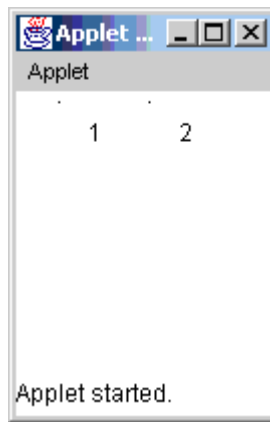
Không cần các phương thức **wait()** và **notify()**, luồng bức vẽ (**canvas**) không thể biết khi nào cập nhập hiển thị. Kết quả xuất ra ngoài của chương trình được đưa ra như sau:



*Kết quả sau mỗi lần click chuột*



*Kết quả sau mỗi lần click chuột*



*Kết quả sau mỗi lần click chuột*

## Sự bế tắc (Deadlocks)

Một “**deadlock**” (sự bế tắc) xảy ra khi hai luồng có một phụ thuộc vòng quanh trên một cặp đối tượng đồng bộ; lấy ví dụ, khi một luồng thâm nhập vào monitor trên đối tượng “**ObjA**”, và một luồng khác thâm nhập vào monitor trên đối tượng “**ObjB**”. Nếu luồng trong “**ObjA**” cố gắng gọi phương thức đồng bộ trên “**ObjB**”, một bế tắc xảy ra.

Nó khó để gỡ lỗi một bế tắc bởi những nguyên nhân sau:

- Nó hiếm khi xảy ra, khi hai luồng chia nhỏ thời gian trong cùng một con đường
- Nó có thể bao hàm nhiều hơn hai luồng và hai đối tượng đồng bộ

Nếu một chương trình đa luồng khóa kín thường xuyên, ngay lập tức kiểm tra lại điều kiện bế tắc.

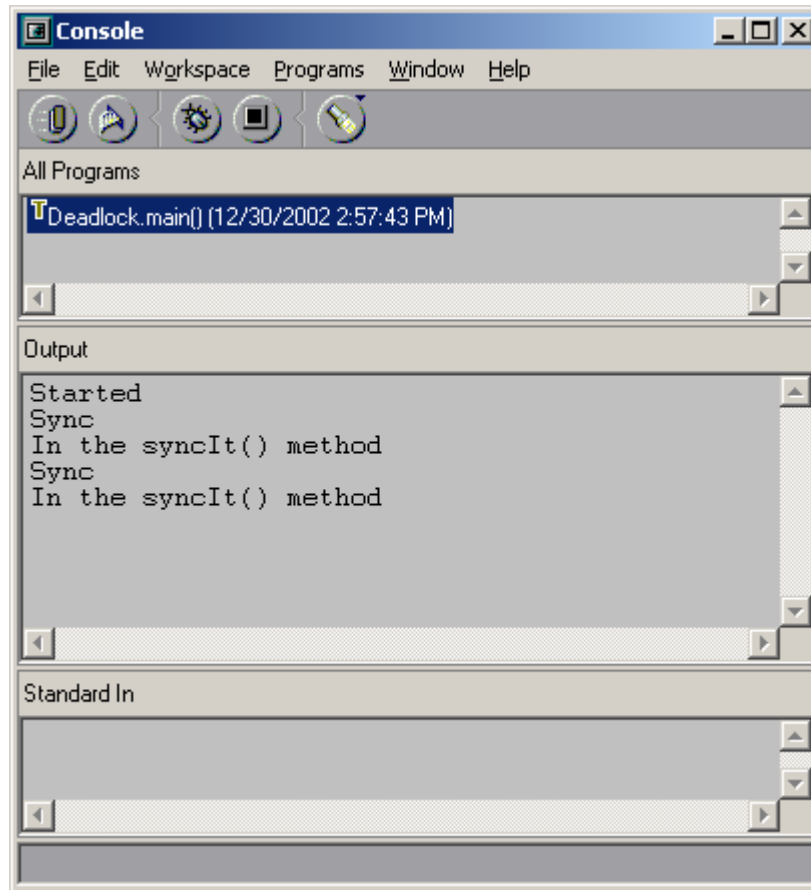
[\[link\]](#) tạo ra điều kiện bế tắc. Lớp chính (main) bắt đầu 2 luồng. Mỗi luồng gọi phương thức đồng bộ run(). Khi luồng “t1” đánh thức, nó gọi phương thức “synchronized” của đối tượng deadlock “dlk1”. Từ đó luồng “t2” một mình giám sát cho “dlk2”, luồng “t1” bắt đầu đợi monitor. Khi luồng “t2” đánh thức, nó cố gắng gọi phương thức “synchronized” của đối tượng Deadlock “dlk2”. Bây giờ, “t2” cũng phải đợi, bởi vì đây là trường hợp tương tự với luồng “t1”. Từ đó, cả hai luồng đang đợi lẫn nhau, cả hai sẽ đánh thức. Đây là điều kiện bế tắc.

Chương trình tạo ra điều kiện bế tắc

```
public class Deadlock implements Runnable{
 public static void main(String args[]){
 Deadlock dlk1= new Deadlock();
 Deadlock dlk2 = new Deadlock();
 Thread t1 = new Thread(dlk1);
 Thread t2 = new Thread(dlk2);
 dlk1.grabIt = dlk1;
 dlk2.grabIt = dlk2;
 t1.start();
 t2.start();
 System.out.println("Started");
 try { t1.join(); t2.join(); }
 catch (InterruptedException e){
 System.out.println("error occurred");
 }
 System.exit(0);
 }
 Deadlock grabIt;
 public synchronized void run() {
 try { Thread.sleep(1500); }
 catch (InterruptedException e){
 System.out.println("error occurred");
 }
 grabIt.syncIt();
 }
}
```

```
} public synchronized void syncIt() { try{ Thread.sleep(1500);
System.out.println("Sync"); } catch(InterruptedException e){ System.out.println("error
occured"); } System.out.println("In the syncIt() method"); } }
```

Kết quả của chương trình này được hiển thị như sau:



*Sự bế tắc*

## Thu dọn “rác” (Garbage collection)

Thu dọn “rác” (Garbage collection) cải tạo hoặc làm trống bộ nhớ đã định vị cho các đối tượng mà các đối tượng này không sử dụng trong thời gian dài. Trong ngôn ngữ lập trình hướng đối tượng khác như C++, lập trình viên phải làm cho bộ nhớ trống mà đã không được yêu cầu trong thời gian dài. Tình trạng không hoạt động để bộ nhớ trống có thể là kết quả trong một số vấn đề. Java tự động tiến hành thu dọn rác để cung cấp giải pháp duy nhất cho vấn đề này. Một đối tượng trở nên thích hợp cho sự dọn rác nếu không có tham chiếu đến nó, hoặc nếu nó đã đăng ký xong.

Sự dọn rác thực thi như là một luồng riêng biệt có quyền ưu tiên thấp. Bạn có thể viện dẫn một phương thức gc() của thể nghiệm để viện dẫn sự dọn rác. Tuy nhiên, bạn không thể dự đoán hoặc bảo đảm rằng sự dọn rác sẽ thực thi một cách trọn vẹn sau đó.

Sử dụng câu lệnh sau để tắt đi sự dọn rác trong ứng dụng:

```
Java -noasyncgc ...
```

Nếu chúng ta tắt đi sự dọn rác, chương trình hầu như chắc chắn rằng bị treo do bởi việc đó.

### **Phương thức finalize() (hoàn thành)**

Java cung cấp một con đường để làm sạch một tiến trình trước khi điều khiển trở lại hệ điều hành. Điều này tương tự như phương thức phân hủy của C++

Phương thức finalize(), nếu hiện diện, sẽ được thực thi trên mỗi đối tượng, trước khi sự dọn rác.

Câu lệnh của phương thức finalize() như sau:

```
protected void finalize() throws Throwable
```

Tham chiếu không phải là sự dọn rác; chỉ các đối tượng mới được dọn rác

Lấy thử nghiệm:

```
Object a = new Object(); Object b = a; a = null;
```

Ở đây, nó sẽ sai khi nói rằng “b” là một đối tượng. Nó chỉ là một đối tượng tham chiếu. Hơn nữa, trong đoạn mã trích trên mặc dù “a” được đặt là rỗng, nó không thể được dọn rác, bởi vì nó vẫn còn có một tham chiếu (b) đến nó. Vì thế “a” vẫn còn với đến được, thật vậy, nó vẫn còn có phạm vi sử dụng trong phạm vi chương trình. Ở đây, nó sẽ không được dọn rác.

Tuy nhiên, trong ví dụ cho dưới đây, giả định rằng không có tham chiếu đến “a” tồn tại, đối tượng “a” trở nên thích hợp cho garbage collection.

```
Object a = new Object(); a = null;
```

Một ví dụ khác:

```
Object m = new Object(); Object m = null;
```

Đối tượng được tạo ra trong sự bắt đầu có hiệu lực cho garbage collection

```
Object m = new Object(); M = new Object();
```

Bây giờ, đối tượng căn nguyên có hiệu lực cho garbage collection, và một đối tượng mới tham chiếu bởi “m” đang tồn tại.

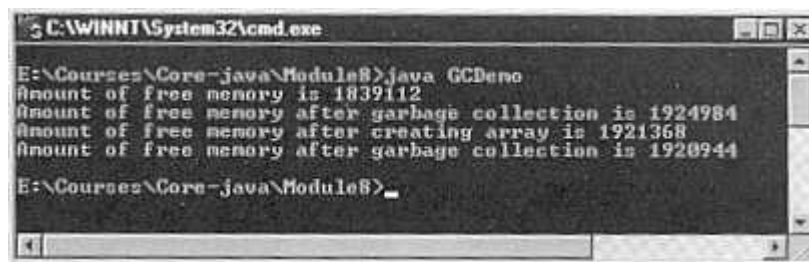
Bạn có thể chạy phương thức garbage collection, nhưng không có bảo đảm rằng nó sẽ xảy ra.

Chương trình điển hình cho Garbage collection

```
class GCDemo { public static void main(String args[]) { int i; long a; , Runtime
r=Runtime.getRuntime(); Long valuesD =new Long[200]; System.out.println
("Amount of free memory is" + r.freeMemory()); r.gc(); System.out.println("Amount
of free memory after garbage collection is " + r.freeMemory()); for
(a=1000000;i=0;i<200;i++) { values[i] =new Long(a); }
System.out.println("Amount of free memory after creating the array " +
r.freeMemory()); for (i=0;i<200;i++) { values[i] =null; } System.out.println("Amount
of free memory after garbage collection is " + r.freeMemory()); }
```

Chúng ta khai một mảng gồm 200 phần tử, trong đó kiểu dữ liệu là kiểu Long. Trước khi mảng được tạo ra, chúng ta phải xác định rõ số lượng bộ nhớ trống, và hiển thị nó. Rồi thì chúng ta viện dẫn phương thức gc() của thể nghiệm Runtime (thời gian thực thi) hiện thời. Điều này có thể hoặc không thể thực thi garbage collection. Rồi thì chúng ta tạo ra mảng, và đăng ký giá trị cho các phần tử của mảng. Điều này sẽ giảm bớt số lượng bộ nhớ trống. Để làm các mảng phần tử thích hợp cho garbage collection, chúng ta đặt chúng rỗng. Cuối cùng, chúng ta sử dụng phương thức gc() để viện dẫn garbage collection lần nữa.

Kết quả xuất ra màn hình của chương trình trên như sau:



*Garbage collection*

## Tổng kết

- Một luồng là đơn vị nhỏ nhất của đoạn mã thực thi được mà một tác vụ riêng biệt.
- Đa luồng giữ cho thời gian rồi là nhỏ nhất. Điều này cho phép bạn viết các chương trình có khả năng sử dụng tối đa CPU.
- Luồng bắt đầu thực thi sau khi phương thức start() được gọi

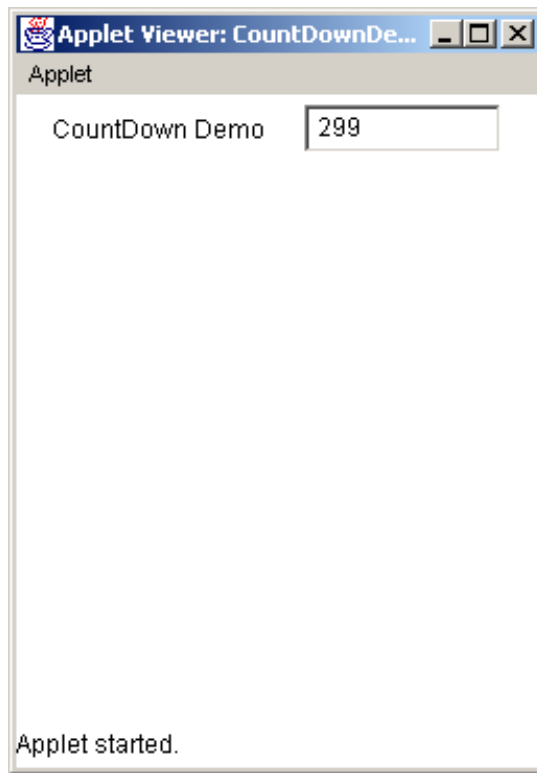
- Lập trình viên, máy ảo Java, hoặc hệ điều hành bảo đảm rằng CPU được chia sẻ giữa các luồng.
- Có hai loại luồng trong một chương trình Java:
  - Luồng người dùng
  - Luồng hiểm.
- Một nhóm luồng là một lớp mà nắm bắt một nhóm các luồng.
- Đồng bộ cho phép chỉ một luồng thâm nhập một tài nguyên được chia sẻ tại một thời điểm.
- Để tránh kiểm soát vòng, Java bao gồm một thiết kế tốt trong tiến trình kỹ thuật truyền thông sử dụng các phương thức “wait()” (đợi), “notify()” (thông báo) và “notifyAll()” (thông báo hết).
- Một “bế tắc” xảy ra khi hai luồng có một phụ thuộc xoay vòng trên một phần của các đối tượng đồng bộ
- Garbage collection là một tiến trình nhờ đó bộ nhớ được định vị để các đối tượng mà không sử dụng trong thời gian dài, có thể cải tạo hoặc làm rảnh bộ nhớ.

### Kiểm tra lại sự hiểu biết của bạn

1. Một ứng dụng có thể chứa đựng nhiều luồng (**Đúng/Sai**)
2. Các luồng con được tạo ra từ luồng chính (**Đúng/Sai**)
3. Mỗi luồng trong một chương trình Java được đăng ký một quyền ưu tiên mà máy ảo Java có thể thay đổi. (**Đúng/Sai**)
4. Phương thức \_\_\_\_\_ có thể tạm thời ngừng việc thực thi luồng
5. Mặc định, một luồng có một quyền ưu tiên \_\_\_\_\_ một hằng số của \_\_\_\_\_
6. \_\_\_\_\_ luồng được dùng cho các luồng “nền”, cung cấp dịch vụ cho luồng khác.
7. Trong luồng đồng bộ, một \_\_\_\_\_ là một đối tượng mà được sử dụng như là một khóa riêng biệt lẫn nhau.
8. \_\_\_\_\_ thường thực thi bởi một vòng lặp mà được sử dụng để lặp lại việc kiểm tra một số điều kiện.

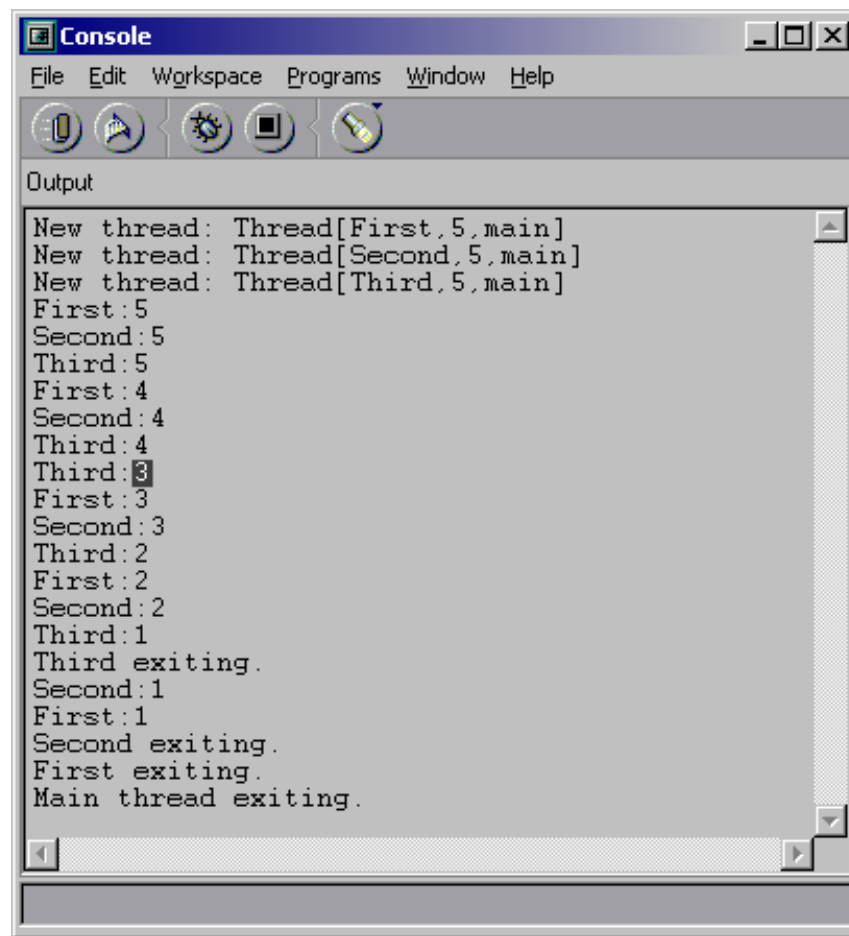
### Bài tập

Viết một chương trình mà hiển thị một sự đếm lùi từng giây cho đến không, như hình sau:



Ban đầu, số 300 sẽ được hiển thị. Giá trị sẽ được giảm dần cho đến 1 đến khi ngoài giá trị 0. Giá trị sẽ được trả lại 300 một lần nữa giảm đến trở thành 0.

Viết một chương trình mà hiển thị như hình dưới đây:

A screenshot of a Java IDE's console window. The window has a title bar labeled 'Console' and a menu bar with 'File', 'Edit', 'Workspace', 'Programs', 'Window', and 'Help'. Below the menu bar is a toolbar with icons for file operations and debugging. The main area is labeled 'Output' and contains a list of text lines representing thread execution. The lines show the creation of three threads (First, Second, Third) and their subsequent execution steps from 5 down to 1, followed by 'exiting.' messages for each thread and the 'Main thread exiting.' message. The text is displayed in a monospaced font with a vertical scrollbar on the right side.

```
New thread: Thread[First,5,main]
New thread: Thread[Second,5,main]
New thread: Thread[Third,5,main]
First:5
Second:5
Third:5
First:4
Second:4
Third:4
Third:3
First:3
Second:3
Third:2
First:2
Second:2
Third:1
Third exiting.
Second:1
First:1
Second exiting.
First exiting.
Main thread exiting.
```

Tạo 3 luồng và một luồng chính trong “main”. Thực thi mỗi luồng như một chương trình thực thi. Khi chương trình kết thúc, các câu lệnh thoát cho mỗi luồng sẽ được hiển thị. Sử dụng kỹ thuật nắm bắt lỗi.



# Dòng vào / ra ( I/O Streams)

## Giới thiệu

Trong buổi học trước, chúng ta đã học về cách tạo các luồng đồng bộ. Ngăn chặn các luồng truy nhập đồng thời các đối tượng dùng chung. Toàn bộ tiến trình này được quản lý bởi cơ chế đợi thông báo (wait-notify). Phương thức wait() báo cho dòng gọi từ bỏ monitor và nhập vào trạng thái ngủ cho đến khi các luồng khác sử dụng xong monitor và gọi phương thức notify(). Phương thức notify() và notifyAll() tạo ra thông báo cho các luồng khác đã gọi phương thức wait() trên cùng đối tượng. Trong bài học trước, chúng ta cũng học về các điều kiện khoá chết là gì và cách tránh chúng.

Chương này giới thiệu khái niệm về dòng. Chúng ta cũng thảo luận các lớp khác nhau trong gói java.io phục vụ vào ra.

## Các dòng (stream)

Theo thuật ngữ chung, stream là một dòng lưu chuyển. trong thuật ngữ về kỹ thuật dòng là một lộ trình mà dữ liệu được truyền trong chương trình. Một ứng dụng về các dòng mà ta đã quen thuộc đó là dòng nhập System.in.

Dòng là những ống (pipelines) để gửi và nhận thông tin trong các chương trình java. Khi một dòng dữ liệu được gửi hoặc nhận, ta tham chiếu nó như đang “ghi” và “đọc” một dòng tương ứng. Khi một dòng được đọc hay ghi, các luồng khác bị có nhu cầu đọc/ghi dòng đó đều phải tạm dừng. Nếu có một lỗi xảy ra khi đọc hay ghi dòng, một ngoại lệ kiểu IOException được tạo ra. Do vậy, các câu lệnh thao tác dòng phải bao gồm khối try-catch.

Lớp ‘java.lang.System’ định nghĩa các dòng nhập và xuất chuẩn. chúng là các lớp chính của các dòng byte mà java cung cấp. Chúng ta cũng đã sử dụng các dòng xuất để xuất dữ liệu và hiển thị kết quả trên màn hình. Dòng vào/ra bao gồm:

- Lớp **System.out**: Dòng xuất chuẩn dùng để hiển thị kết quả trên màn hình.
- Lớp **System.in**: Dòng nhập chuẩn thường đến từ bàn phím và được dùng để đọc các ký tự dữ liệu.
- Lớp **System.err**: Đây là dòng lỗi chuẩn.

Các lớp ‘InputStream’ và ‘OutputStream’ cung cấp nhiều khả năng vào/ra khác nhau. Cả hai lớp này có các lớp thừa kế để thực hiện I/O thông qua các vùng bộ nhớ đệm, các tập tin và ống (pipeline). Các lớp con của lớp InputStream thực hiện vào, trong khi các lớp con của lớp OutputStream thực hiện ra.

## Gói java.io

Các luồng hệ thống rất có ích. Tuy nhiên, chúng không đủ mạnh để dùng khi ứng phó với I/O thực tế. Gói java.io phải được nhập khẩu vì mục đích này. Chúng ta sẽ thảo luận tìm hiểu về các lớp thuộc gói java.io.

### Lớp InputStream

Lớp InputStream là một lớp trừu tượng. Nó định nghĩa cách thức nhận dữ liệu. Điểm quan trọng không nằm ở chỗ dữ liệu đến từ đâu, mà là khả năng truy cập. Lớp InputStream cung cấp một số phương thức để đọc và dùng các dòng dữ liệu để làm đầu vào. Các phương thức này giúp ta tạo, đọc và xử lý các dòng đầu vào. Các phương thức được hiện trong [\[link\]](#)

Các phương thức của lớp InputStream

| Tên phương thức          | Mô tả                                                                                                                                                                                           |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| read()                   | Đọc các byte dữ liệu từ một dòng. Nếu như không có byte dữ liệu nào, nó phải chờ. Khi một phương thức phải chờ, các luồng đang thực hiện phải tạm dừng cho đến khi có dữ liệu.                  |
| read (byte [])           | Trả về số byte đọc được hay '-1' nếu như đã đọc đến cuối dòng. Nó gây ra ngoại lệ IOException nếu có lỗi xảy ra.                                                                                |
| read (byte [], int, int) | Nó cũng đọc vào một mảng byte. Nó trả về số byte thực sự đọc được cho đến khi kết thúc dòng. Nó gây ra ngoại lệ IOException nếu lỗi xảy ra.                                                     |
| available()              | Phương pháp này trả về số lượng byte có thể đọc được mà không phải chờ. Nó trả về số byte hiện tại có trong dòng. Nó không phải là phương thức tin cậy để thực hiện tiến trình xử lý đầu vào.   |
| close()                  | Phương thức này đóng dòng. Nó dùng để giải phóng mọi tài nguyên dòng đã sử dụng. Luôn luôn đóng dòng để chắc chắn rằng dòng xử lý được kết thúc. Nó gây ra ngoại lệ IOException nếu lỗi xảy ra. |
| mark()                   | Đánh dấu vị trí hiện tại của dòng.                                                                                                                                                              |
| markSupported()          | Trả về giá trị boolean chỉ ra rằng dòng có hỗ trợ các khả năng mark và reset hay không. Nó trả về True nếu dòng hỗ trợ ngược lại trả về False.                                                  |
| reset()                  | Phương thức này định vị lại dòng theo vị trí được đánh lần cuối cùng. Nó gây ra ngoại lệ IOException nếu lỗi xảy ra.                                                                            |

|        |                                                                                                                                                                                                          |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| skip() | Phương thức này bỏ qua 'n' byte dòng vào. '-n' chỉ định số byte được bỏ qua. Nó gây ra ngoại lệ IOException nếu lỗi xảy ra. Phương thức này sử dụng để di chuyển tới vị trí đặc biệt bên trong dòng vào. |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Lớp OutputStream

Lớp OutputStream cũng là lớp trừu tượng. Nó định nghĩa cách ghi các kết xuất đến dòng. Nó cung cấp một tập các phương thức trợ giúp tạo ra, ghi và xử lý kết xuất các dòng. Các phương thức bao gồm:

Các phương thức lớp OutputStream

| Tên phương thức       | Mô tả                                                                                                                                                                                |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| write(int)            | Phương thức này ghi một byte                                                                                                                                                         |
| write(byte[])         | Phương thức này phong toả cho đến khi một byte được ghi. dòng phải chờ cho đến khi tác vụ ghi hoàn tất. Nó gây ra ngoại lệ IOException nếu lỗi xảy ra.                               |
| write(byte[],int,int) | Phương thức này ghi mảng các byte. Lớp OutputStream định nghĩa ba dạng khác nhau của phương thức để có thể ghi một byte riêng lẻ, một mảng các byte, hay một đoạn của một mảng byte. |
| flush()               | Phương thức này xả sạch dòng. Dữ liệu được ghi ra dòng. Nó kích hoạt IOException nếu lỗi xảy ra.                                                                                     |
| close()               | Phương thức đóng dòng. Nó được dùng để giải phóng mọi tài nguyên gắn với dòng. Nó kích hoạt IOException nếu lỗi xảy ra.                                                              |

## Vào ra mảng byte

Các lớp '**ByteArrayInputStream**' và '**ByteArrayOutputStream**' sử dụng các bộ đệm. Không cần thiết phải dùng chúng cùng với nhau.

- **Lớp ByteArrayInputStream**

Lớp này tạo dòng đầu vào từ bộ đệm, đó là mảng các byte. Lớp này không hỗ trợ các phương thức mới. Ngược lại nó định nghĩa đè các phương thức của lớp InputStream như 'read()', 'skip()', 'available()' và 'reset()'.

- **Lớp ByteArrayOutputStream**

Lớp này tạo ra dòng ra trên một mảng các byte. Nó cũng cung cấp các khả năng cho phép mản ra tăng trưởng nhằm mục đích tăng kích thước. Lớp này cũng cung cấp thêm các phương thức 'toArray()' và 'toString()'. Chúng được dùng để chuyển đổi dòng thành một mảng byte hay chuỗi.

Lớp `ByteArrayOutputStream` cũng cung cấp hai phương thức thiết lập. Một có một đối số số nguyên dùng để ấn định mảng byte ra theo một kích cỡ ban đầu và thứ hai không có đối số nào, nó thiết lập bộ ra xuất với kích thước mặc định. Lớp này cung cấp vài phương thức bổ sung, không được khai báo trong `OutputStream`:

- **reset()**

Thiết lập lại vùng đệm ra nhằm cho phép ghi lại từ đầu vùng đệm.

- **size()**

Trả về số byte hiện tại đã được ghi tới vùng đệm.

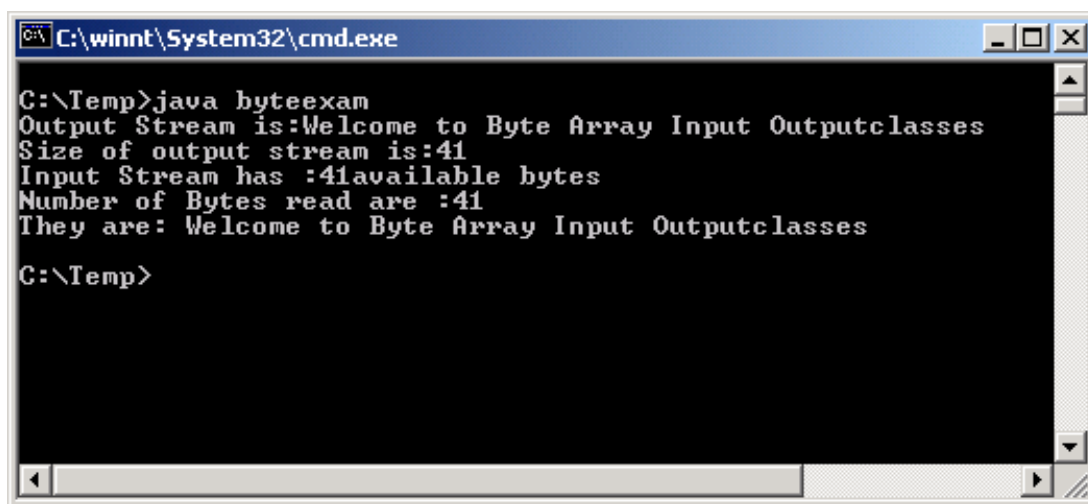
- **writeto()**

Ghi nội dung của vùng đệm ra dòng ra đã chỉ định. Để thực hiệnsử dụng một đối tượng của lớp `OutputStream` làm đối số.

Chương trình sử dụng lớp '`ByteArrayInputStream`' và '`ByteArrayOutputStream`' để nhập và xuất

```
import java.lang.System; import java.io.*; public class byteexam { public static void
main(String args[]) throws IOException { ByteArrayOutputStream os =new
ByteArrayOutputStream(); String s ="Welcome to Byte Array Input Outputclasses";
for(int i=0; i<s.length();i++) os. write (s.charAt(i)) ; System.out.println("Output
Stream is:" + os); System.out.println("Size of output stream is:"+ os.size());
ByteArrayInputStream in; in = new ByteArrayInputStream(os.toByteArray()); int ib =
in.available(); System.out.println("Input Stream has :"+ ib + "available bytes"); byte
ibuf[] = new byte[ib]; int byrd = in.read(ibuf,0,ib); System.out.println("Number of
Bytes read are :"+ byrd); System.out.println("They are: " + new String(ibuf)); } }
```

[\[link\]](#) Xuất hiện kết xuất của chương trình:



```
C:\winnt\System32\cmd.exe

C:\Temp>java byteexam
Output Stream is:Welcome to Byte Array Input Outputclasses
Size of output stream is:41
Input Stream has :41available bytes
Number of Bytes read are :41
They are: Welcome to Byte Array Input Outputclasses

C:\Temp>
```

*Sử dụng lớp 'ByteArrayInputStream' và 'ByteArrayOutputStream' cho nhập và xuất*

## Nhập xuất lọc

Một 'Filter' là một kiểu dòng đã thay đổi cách xử lý dòng hiện có. Các lớp, các dòng nhập xuất lọc của java sẽ giúp ta lọc vào/ra theo một số cách. Về cơ bản, các bộ lọc này dùng để thích nghi các dòng theo các nhu cầu của chương trình cụ thể.

Bộ lọc nằm giữa một dòng nhập và một dòng xuất. Nó thực hiện xử lý một quá trình nào đó trên các byte được truyền từ đầu vào đến đầu ra. Các bộ lọc có thể ghép với nhau khi đó đầu ra của bộ lọc này trở thành đầu vào của bộ lọc kia.

- Lớp FilterInputStream

Đây là lớp trừu tượng. Nó là cha của tất cả các lớp dòng nhập lọc. Lớp này cung cấp khả năng tạo ra một dòng từ dòng khác. Một dòng có thể được đọc và đưa kết quả cho một dòng khác. Biến 'in' được sử dụng để làm điều này. Biến này được dùng để duy trì một đối tượng tách biệt của lớp InputStream. Lớp FilterInputStream được thiết kế sao cho có khả năng kết chuỗi nhiều bộ lọc. Để thực hiện điều này chúng ta dùng vài tầng lồng nhau. Mỗi lớp sẽ truy cập đầu ra của lớp trước đó với sự trợ giúp của biến 'in'.

- Lớp FilterOutputStream

Lớp này là một dạng hỗ trợ cho lớp FilterInputStream. Nó là lớp cha của tất cả các lớp dòng xuất lọc. Lớp này tương tự như lớp FilterInputStream ở chỗ nó duy trì đối tượng của lớp OutputStream làm một biến 'out'. Dữ liệu ghi vào lớp này có thể sửa đổi theo nhu cầu để thực hiện tác vụ lọc và sau đó được chuyển tới đối tượng OutputStream.

## Vào ra tập tin

Java hỗ trợ các tác vụ nhập và xuất tập tin với sự trợ giúp các lớp sau đây:

- File
- FileDescriptor
- FileInputStream
- FileOutputStream

Java cũng hỗ trợ truy cập nhập và xuất trực tiếp hay ngẫu nhiên bằng các lớp ‘File’, ‘FileDescriptor’, và ‘RandomAccessFile’.

- Lớp File

Lớp này được sử dụng để truy cập các đối tượng tập tin và thư mục. Các tập tin đặt tên theo qui ước đặt tên tập tin của hệ điều hành. Các qui ước này được mô tả bằng các hằng số của lớp File. Lớp này cung cấp các phương thức thiết lập các tập tin và các thư mục. Các thiết lập chấp nhận các đường dẫn tập tin tuyệt đối lẫn tương đối của các tập tin và thư mục. Tất cả các thao tác thư mục và tập tin được thực hiện thông qua các phương thức của lớp *File*.

Các phương thức:

- Cho phép bạn tạo, xoá, đổi tên các file.
- Cung cấp khả năng truy cập bằng đường dẫn tập tin.
- Xác định đối tượng là tập tin hay thư mục.
- Kiểm tra quyền truy cập đọc và ghi.

Giống như các phương thức truy cập, các phương thức thư mục cũng cho phép tạo, xoá, đặt tên lại và liệt kê các thư mục. Các phương thức này cho phép truy nhập cây thư mục, cung cấp khả năng truy cập thư mục cha và các thư mục anh em.

- Lớp FileDescriptor

Lớp này cung cấp khả năng truy cập các mô tả tập tin mà hệ điều hành sử dụng khi tập tin và thư mục đang được truy cập. Lớp này không cung cấp các phương thức cho phép xem chi tiết thông tin do hệ điều hành sử dụng. Nó chỉ cung cấp một phương thức duy nhất là ‘valid()’, giúp xác định một đối tượng mô tả tập tin hiện có hợp lệ hay không.

- Lớp FileInputStream

Lớp này cho phép đọc vào từ một tập tin dưới dạng một dòng. Các đối tượng của lớp này được tạo ra nhờ đường dẫn tới file, đối tượng File, hoặc đối tượng FileDescriptor làm một đối số. Lớp này định nghĩa chồng các phương thức của lớp InputStream. Nó cũng cung cấp thêm các phương thức ‘finalize()’ và ‘getFD()’.

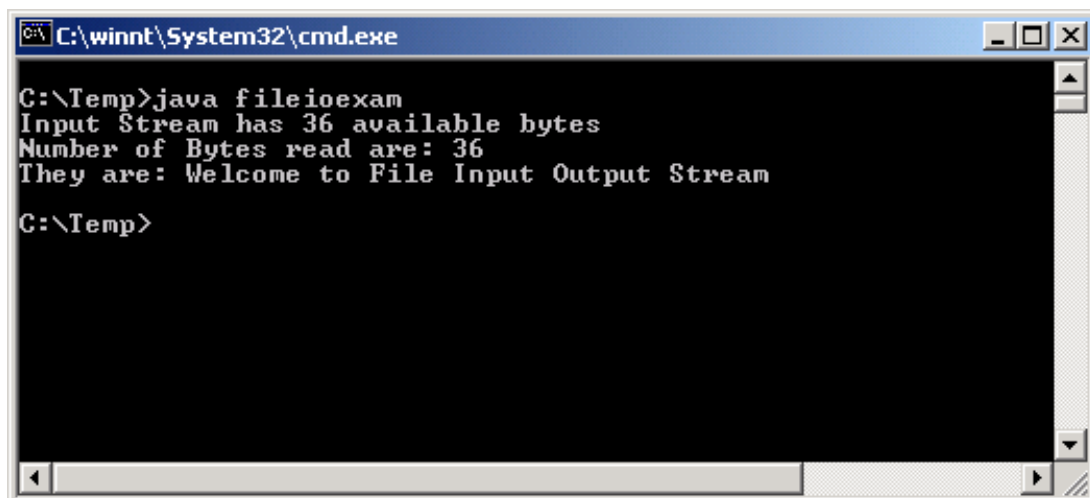
Phương thức 'finalize()' được dùng để đóng dòng khi nó được bộ gom rác Java nhặt. Phương thức 'getFD()' trả về đối tượng FileDescriptor chứa thông tin về sự kết nối thực sự tới file mà 'FileInputStream' đang sử dụng.

- Lớp FileOutputStream

Lớp này cho phép xuất ra một tập tin theo dòng. Các đối tượng của lớp này cũng tạo ra sử dụng đường dẫn của tập tin, FileDescriptor làm tham số thiết lập. Lớp này định nghĩa chồng phương thức của lớp OutputStream và cung cấp thêm các phương thức 'finalize()' và getFD().

```
import java.io.FileOutputStream; import
java.io.FileInputStream; import java.io.File; import
java.io.IOException; public class fileioexam { public
static void main(String args[]) throws IOException {
//tạo file abc.txt FileOutputStream os = new
FileOutputStream("abc.txt"); String s = "Welcome to File
Input Output Stream " ; for(int i = 0; i< s.length();
++i) os. write(s.charAt(i)); os.close(); // mở file
abc.txt để đọc FileInputStream is = new
FileInputStream("abc.txt"); int ibyts = is.available();
System.out.println("Input Stream has " + ibyts + "
available bytes"); byte ibuf[] = new byte[ibyts]; int
byrd = is.read(ibuf, 0, ibyts); System.out.println("Number
of Bytes read are: " + byrd); System.out.println("They
are: " + new String(ibuf)); is.close(); File fl = new
File("abc.txt"); fl.delete(); } }
```

[\[link\]](#) hiện kết xuất của đoạn mã nguồn trên:



The screenshot shows a Windows command prompt window titled "C:\winnt\System32\cmd.exe". The command prompt is at "C:\Temp>". The user has entered the command "java fileioexam". The output of the program is displayed as follows:

```
C:\Temp>java fileioexam
Input Stream has 36 available bytes
Number of Bytes read are: 36
They are: Welcome to File Input Output Stream

C:\Temp>
```

## **Vào/ra có sử dụng bộ đệm**

Vùng đệm là kho lưu trữ dữ liệu. Chúng ta có thể lấy dữ liệu từ vùng đệm thay vì quay trở lại nguồn ban đầu của dữ liệu.

Java sử dụng cơ chế nhập/xuất có lập vùng đệm để tạm thời lập cache dữ liệu vào/ra của một dòng. Nó giúp chương trình đọc/ghi lượng dữ liệu nhỏ không ảnh hưởng lớn đến hiệu năng chung của hệ thống.

Trong khi thực hiện vào có vùng đệm, một số lượng byte lớn được đọc tại một thời điểm, và lưu trữ trong một vùng đệm nhập. Khi chương trình đọc dòng nhập thì thay vì ra dòng vào để đọc nó đọc từ vùng đệm nhập.

Tiến trình lập vùng đệm ra cũng thực hiện tương tự. khi dữ liệu được một chương trình ghi ra dòng ra, dữ liệu ra được lưu trữ trong một vùng đệm ra. Dữ liệu được lưu trữ đến khi vùng đệm đầy hoặc các dòng tra thực hiện xả trống (flush). Cuối cùng liệu ra trong vùng đệm được chuyển đến dòng ra.

Các bộ lọc hoạt động trên vùng đệm. Vùng đệm được đặt giữa chương trình và dòng ra của vùng đệm.

- **Lớp BufferedInputStream**

Lớp này tự động tạo ra và duy trì vùng đệm để hỗ trợ thao tác vào. Nhờ đó chương trình có thể đọc dữ liệu từ dòng từng byte một mà không ảnh hưởng đến tốc độ thực hiện của hệ thống. Bởi lớp 'BufferedInputStream' là một bộ lọc, nên có thể áp dụng nó cho một số đối tượng nhất định của lớp InputStream và cũng có thể phối hợp với các tập tin đầu vào khác.

Lớp này sử dụng vài biến để thực hiện các cơ chế lập vùng đệm đầu vào. Các biến này được khai báo là protected và do đó chương trình không thể truy cập trực tiếp. Lớp này định nghĩa hai phương thức thiết lập. Một cho phép chỉ định kích cỡ của vùng đệm nhập trong khi đó phương thức thiết lập kia thì không. Nhưng cả hai phương thức thiết lập đều tiếp nhận đối tượng của lớp InputStream làm đối số. Lớp này định nghĩa chồng các phương thức truy cập mà InputStream cung cấp và không đưa thêm bất kì phương thức mới nào.

- **Lớp BufferedOutputStream**

Lớp này cũng định nghĩa hai phương thức thiết lập, một cho phép chỉ định kích cỡ của vùng đệm xuất, một sử dụng kích cỡ vùng đệm ngầm định. Lớp này định nghĩa chồng

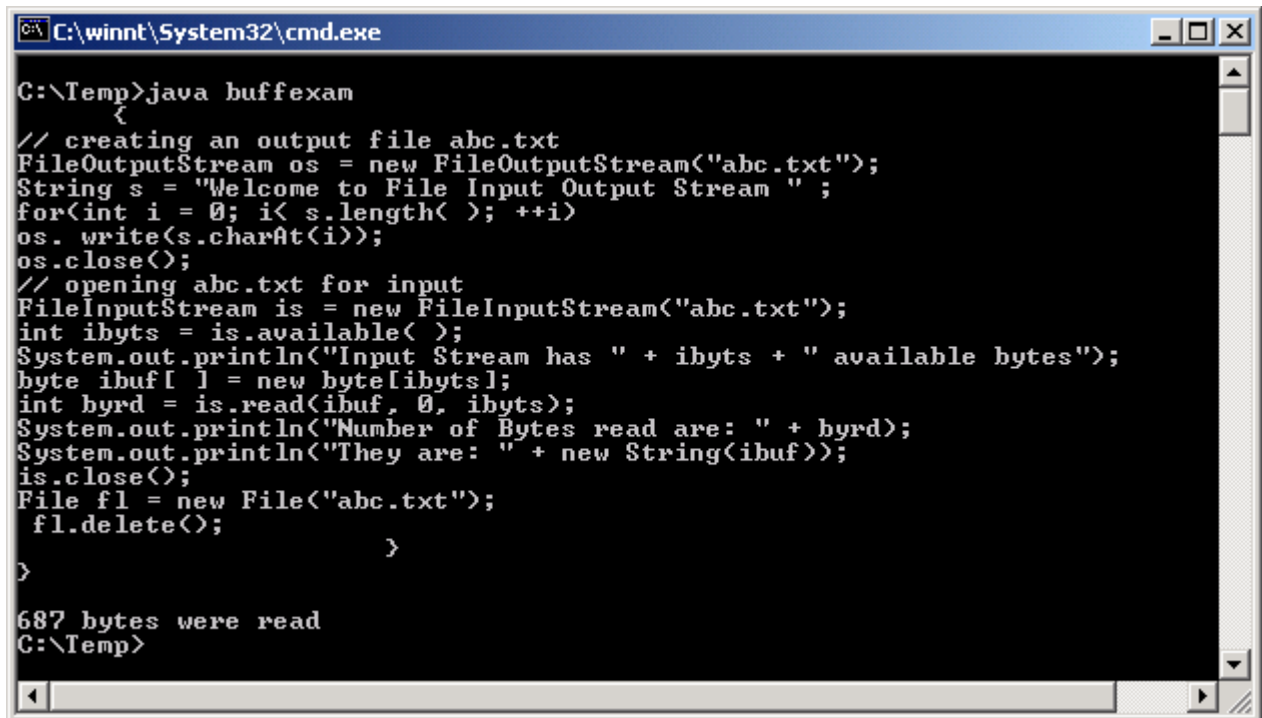


tất cả các phương thức của OutputStream và không đưa thêm bất kì phương thức mới nào.

Chương trình mô tả cách dùng các luồng nhập/xuất có lập vùng đệm

```
import java.io.BufferedInputStream; import java.io.BufferedOutputStream; import
java.io.FileInputStream; import java.io.SequenceInputStream; import
java.io.IOException; public class buffexam { public static void main(String args[])
throws IOException { // defining sequence input stream SequenceInputStream Seq3;
FileInputStream Fis1 ; Fis1 = new FileInputStream("byteexam.java"); FileInputStream
Fis2; Fis2= new FileInputStream("fileioexam.java"); Seq3 = new
SequenceInputStream(Fis1, Fis2); // create buffered input and output streams
BufferedInputStream inst; inst= new BufferedInputStream(Seq3);
BufferedOutputStream oust; oust= new BufferedOutputStream(System.out);
inst.skip(1000); boolean eof = false; int bytent = 0; while(!eof) { int num = inst.read();
if(num==-1) { eof=true; } else { oust.write((char) num); ++bytent; } } String
bytrd=String.valueOf(bytent); bytrd += " bytes were read"; oust.write(bytrd.getBytes(),
0, bytrd.length()); // close all streams. inst.close(); oust.close(); Fis1.close();
Fis2.close(); } }
```

[\[link\]](#) hiện kết xuất của chương trình trên:



```
C:\winnt\System32\cmd.exe

C:\Temp>java buffexam
{
// creating an output file abc.txt
FileOutputStream os = new FileOutputStream("abc.txt");
String s = "Welcome to File Input Output Stream ";
for(int i = 0; i< s.length(); ++i)
os. write(s.charAt(i));
os.close();
// opening abc.txt for input
FileInputStream is = new FileInputStream("abc.txt");
int ibyts = is.available();
System.out.println("Input Stream has " + ibyts + " available bytes");
byte ibuf[] = new byte[ibyts];
int byrd = is.read(ibuf, 0, ibyts);
System.out.println("Number of Bytes read are: " + byrd);
System.out.println("They are: " + new String(ibuf));
is.close();
File fl = new File("abc.txt");
fl.delete();
}

687 bytes were read
C:\Temp>
```

*Sử dụng các lớp vùng đệm luồng nhập và xuất.*

## Lớp Reader và Writer

Đây là các lớp trừu tượng. Chúng là lớp cha của tất cả các lớp đọc và ghi các dòng ký tự unicode. Java 1.1 đã giới thiệu các lớp này.

- Lớp Reader

Lớp này hỗ trợ các phương thức:

- read( )
- reset( )
- skip( )
- mark( )
- markSupported( )
- close( )

Lớp này cũng hỗ trợ phương thức 'ready()'. Phương thức này trả về giá trị kiểu boolean, xem việc đọc có tiếp tục được hay không.

- Lớp Writer

Lớp này hỗ trợ các phương thức:

- write( )
- flush( )
- close( )

## Nhập/ xuất chuỗi và xâu ký tự

Các lớp 'CharArrayReader' và 'CharArrayWriter' cũng tương tự như các lớp ByteArrayInputStream và ByteArrayOutputStream ở chỗ chúng hỗ trợ nhập/xuất từ các vùng đệm. Các lớp CharArrayReader và CharArrayWriter hỗ trợ nhập/ xuất ký tự 8 bit.

CharArrayReader bổ sung thêm phương pháp nào, nó chỉ dùng các phương thức mà lớp Reader cung cấp. Lớp CharArrayWriter bổ sung thêm các phương thức sau đây ngoài các phương thức của lớp Writer.

- reset( )

Thiết lập lại vùng đệm

- size( )

trả về kích cỡ hiện hành của vùng đệm

- `toCharArray( )`

Trả về bản sao mảng ký tự của vùng đệm xuất

- `toString( )`

Chuyển đổi vùng đệm xuất thành một đối tượng `String`

- `writeTo( )`

Ghi vùng đệm ra một luồng xuất khác.

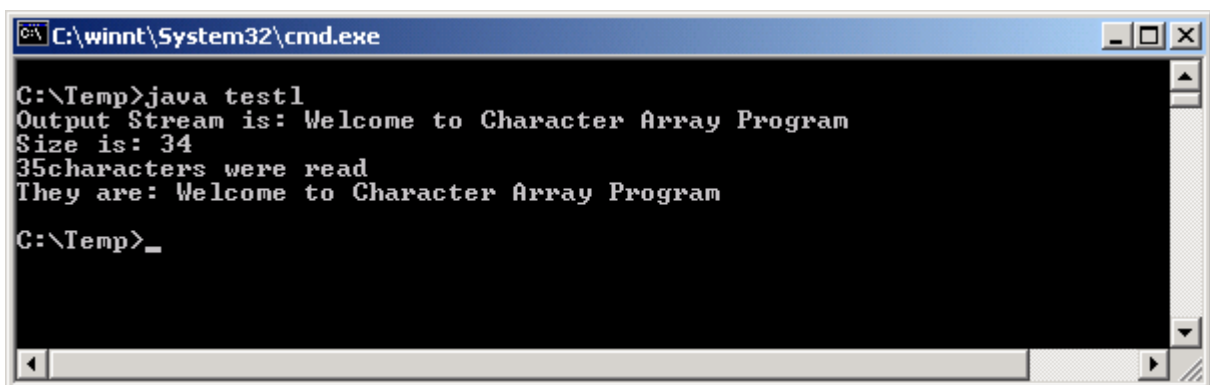
Lớp `StringReader` trợ giúp đọc ký tự từ một chuỗi. Nó không bổ sung phương thức nào ngoài các phương thức của lớp `Reader`.

Lớp `StringWriter` trợ giúp ghi ký tự ra đối tượng `StringBuffer`. Lớp này bổ sung hai phương thức có tên là `getBuffer( )` và `toString( )`. Phương thức `getBuffer( )` trả về đối tượng `StringBuffer` tương ứng với vùng đệm xuất, trong khi đó phương thức `toString( )` trả về một chuỗi chứa bản sao của vùng đệm xuất.

Chương trình thực hiện các tác vụ nhập/xuất mảng ký tự

```
import java.io.CharArrayReader; import java.io.CharArrayWriter; import
java.io.IOException; public class test1 { public static void main(String args[]) throws
IOException { CharArrayWriter ost = new CharArrayWriter(); String s = "Welcome
to Character Array Program"; for(int i= 0; i<s.length(); ++i) ost.write(s.charAt(i));
System.out.println("Output Stream is: " + ost); System.out.println("Size is: " + ost.size(
)); CharArrayReader inst; inst = new CharArrayReader(ost.toCharArray()); int a= 0;
StringBuffer sbI = new StringBuffer(" "); while((a = inst.read()) != -1)
sbI.append((char) a); s = sbI.toString(); System.out.println(s.length() + "characters
were read"); System.out.println("They are:" + s); } }
```

[\[link\]](#) hiện kết xuất chương trình:



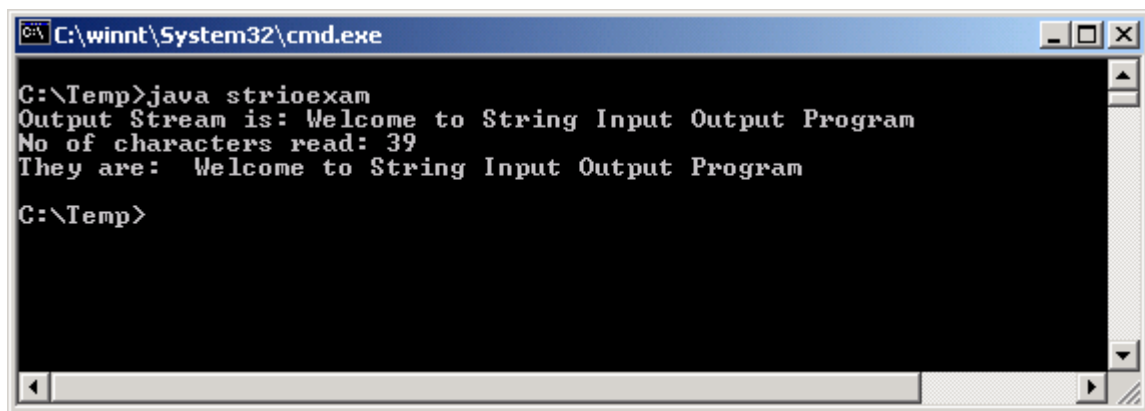
```
C:\winnt\System32\cmd.exe
C:\Temp>java test1
Output Stream is: Welcome to Character Array Program
Size is: 34
34characters were read
They are: Welcome to Character Array Program
C:\Temp>_
```

*Các tác vụ nhập và xuất mảng các ký tự*

Chương trình mô tả tiến trình nhập/xuất chuỗi

```
import java.lang.System; import java.io.StringReader; import java.io.StringWriter;
import java.io.IOException; import java.io. * ; public class strioexam { public static
void main(String args[]) throws IOException { StringWriter ost = new StringWriter();
String s = "Welcome to String Input Output Program"; for(int i= 0; i <s.length(); ++i)
ost.write(s.charAt(i)) ; System.out.println("Output Stream is: " + ost); StringReader
inst; inst = new StringReader(ost.toString()); int a= 0; StringBuffer sb1 = new
StringBuffer(" "); while((a = inst.read())!=-1) sb1.append((char) a); s = sb1.toString();
System.out.println("No of characters read: " +s.length()); System.out.println("They
are: " + s); } }
```

[\[link\]](#) Hiện kết xuất chương trình:



*Nhập và xuất chuỗi*

## Lớp **PrintWriter**

Lớp '**PrintStream**' thực hiện việc kết xuất dữ liệu. Lớp này có các phương thức bổ sung, trợ giúp cho việc in ấn dữ liệu cơ bản.

Lớp '**PrintWriter**' là một phiên bản hướng ký tự của lớp **PrintStream**. Nó thực tế cải thiện lớp **PrintStream** bằng cách dùng dấu tách dòng phụ thuộc hệ điều hành để in các dòng thay vì ký tự '\n'. Lớp này cũng cấp hỗ trợ ký tự Unicode tốt hơn so với **PrintStream**. Phương thức '**checkError()**' được sử dụng kiểm tra kết xuất và xóa sạch các lỗi. Phương thức '**setError()**' được sử dụng để thiết lập lỗi điều kiện. Lớp **PrintWriter** hỗ trợ in ấn các kiểu dữ liệu nguyên thủy, mảng ký tự, chuỗi và đối tượng.

## Giao diện **DataInput**

Giao diện **DataInput** được sử dụng để đọc các byte từ dòng nhị phân và xây dựng lại các kiểu dữ liệu dạng nguyên thủy của Java.

**DataInput** cũng cho phép chúng ta chuyển đổi dữ liệu từ định dạng sửa đổi UTF-8 tới dạng chuỗi. Chuẩn UTF cho định dạng chuyển đổi Unicode. Nó là kiểu định dạng đặt

biệt mã hoá các ký tự Unicode 16 bit . UTF cho rằng trong hầu hết các trường hợp, byte cao trong 2 byte Unicode sẽ là 0. Giao diện `DataInput` được định nghĩa một số các phương thức, các phương thức bao gồm việc đọc các kiểu dữ liệu nguyên thủy trong java.

[\[link\]](#) Tóm lược vài phương thức. Tất cả các phương thức đều có khả năng tạo ra ngoại lệ `IOException` trong trường hợp lỗi:

Các phương thức của giao diện `DataInput`

| Tên phương thức                     | Mô tả                                                                              |
|-------------------------------------|------------------------------------------------------------------------------------|
| <code>boolean readBoolean( )</code> | Đọc một byte nhập, và trả về True nếu byte đó khác 0, và False nếu byte đó bằng 0. |
| <code>byte readByte( )</code>       | Đọc một byte                                                                       |
| <code>char readChar( )</code>       | Đọc và trả về một giá trị ký tự                                                    |
| <code>short readShort( )</code>     | Đọc 2 byte và trả về giá trị kiểu short                                            |
| <code>long readLong( )</code>       | Đọc 8 byte và trả về giá trị kiểu long.                                            |
| <code>float readFloat( )</code>     | đọc 4 byte và trả về giá trị kiểu float                                            |
| <code>int readInt( )</code>         | Đọc 4 byte và trả về giá trị kiểu int                                              |
| <code>double readDouble( )</code>   | Đọc 8 byte và trả về giá trị kiểu double                                           |
| <code>String readUTF( )</code>      | Đọc một chuỗi                                                                      |
| <code>String readLine( )</code>     | Đọc một dòng văn bản                                                               |

## Giao diện `DataOutput`

Giao diện `DataOutput` được sử dụng để tái tạo lại các kiểu dữ liệu nguyên thủy trong java vthành dãy byte. Nó ghi các byte này lên trên dòng nhị phân.

Giao diện `DataOutput` cũng cho phép chúng ta chuyển đổi một chuỗi vào trong Java theo định dạng UTF-8 và ghi nó vào dòng.

Giao diện `DataOutput` định nghĩa một số phương thức được tóm tắt trong [\[link\]](#). Tất cả các phương thức đều có khả năng gây ra ngoại lệ `IOException` trong trường hợp lỗi.

Các phương thức của giao diện `DataOutput`

| Tên phương thức | Mô tả |
|-----------------|-------|
|-----------------|-------|

|                                             |                                               |
|---------------------------------------------|-----------------------------------------------|
| <code>void writeBoolean(boolean b)</code>   | Ghi một giá trị Boolean vào dòng              |
| <code>void writeByte(int value)</code>      | Ghi giá trị 8 bit thấp                        |
| <code>void writeChar(int value)</code>      | Ghi 2 byte giá trị kiểu ký tự vào dòng        |
| <code>void writeShort(int value)</code>     | Ghi 2 byte, biểu diễn lại giá trị kiểu short  |
| <code>void writeLong(long value)</code>     | Ghi 8 byte, biểu diễn lại giá trị kiểu long   |
| <code>void writeFloat(float value)</code>   | Ghi 4 byte, biểu diễn lại giá trị kiểu float  |
| <code>void writeInt(int value)</code>       | ghi 4 byte                                    |
| <code>void writeDouble(double value)</code> | Ghi 8 byte, biểu diễn lại giá trị kiểu double |
| <code>void writeUTF(String value)</code>    | Ghi một chuỗi UTF tới dòng.                   |

### Lớp **RandomAccessFile**

Lớp `RandomAccessFile` cung cấp khả năng thực hiện vào/ra theo một vị trí cụ thể bên trong một tập tin. Trong lớp này, dữ liệu có thể đọc hoặc ghi ở vị trí ngẫu nhiên thay vì liên tục. Vì vậy lớp này có tên là `RandomAccess`. Phương thức `'seek( )'` hỗ trợ truy cập ngẫu nhiên. Vì thế, biến trở tương ứng với tệp tin hiện hành có thể ấn định theo vị trí bất kỳ trong tập tin.

Lớp `RandomAccessFile` thực hiện cả hai việc nhập và xuất. Do vậy, có thể thực hiện I/O bằng các kiểu dữ liệu nguyên thủy. Lớp này cũng hỗ trợ các quyền cơ bản về đọc hoặc ghi tập tin, điều này cho phép đọc tập tin theo chế độ chỉ đọc hoặc đọc-ghi. Tham số `'r'` hoặc `'rw'` được gán cho lớp `RandomAccessFile` chỉ định truy cập `'chỉ đọc'` và `'đọc-ghi'`. Lớp này giới thiệu vài phương thức mới khác với phương thức đã thừa kế từ các lớp `DataInput` và `DataOutput`.

Các phương thức mới thêm vào bao gồm:

- **`seek( )`**

Thiết lập con trỏ tập tin tới vị trí cụ thể bên trong tập tin.

- **`getFilePointer( )`**

Trả về vị trí hiện hành của con trỏ tập tin.

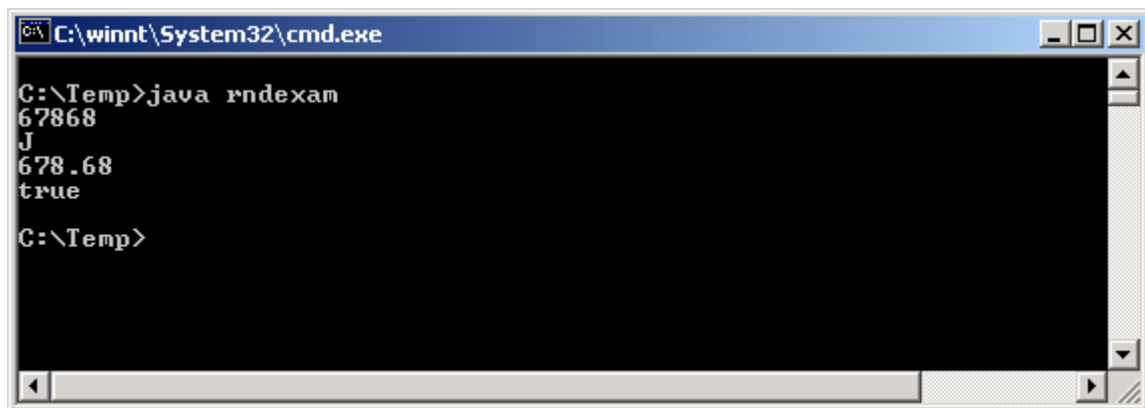
- **`length( )`**

Trả về chiều dài của tập tin tính theo byte.

Chương trình dưới đây minh họa cách dùng lớp `RandomAccessFile`. Nó ghi một giá trị boolean, một int, một char, một double tới một file có tên 'abc.txt'. Nó sử dụng phương pháp `seek()` để tìm vị trí định vị bên trong tập tin (bắt đầu từ 1). Sau đó nó đọc giá trị số nguyên, ký tự và double từ tập tin và hiển thị chúng ra màn hình.

```
import java.lang.System; import java.io.RandomAccessFile;
import java.io.IOException; public class rndexam { public
static void main (String args[]) throws IOException {
RandomAccessFile rf; rf = new RandomAccessFile("abc.txt",
"rw"); rf.writeBoolean(true); rf.writeInt(67868);
rf.writeChars("J"); rf.writeDouble(678.68); //making use
of seek() method to move to a specific file location
rf.seek(1); System.out.println(rf.readInt());
System.out.println(rf.readChar());
System.out.println(rf.readDouble()); rf.seek(0);
System.out.println(rf.readBoolean()); rf.close(); } }
```

[\[link\]](#) Hiện kết xuất chương trình:



*Lớp `RandomAccessFile`*

## Gói `java.awt.print`

Đây là gói mới mà Java 1.2 cung cấp. Nó thay thế khả năng in của JDK 1.1. Nó bao gồm đầy các giao diện:

- `Pageable`
- `Printable`
- `PrinterGraphics`

Giao diện '**Pageable**' định nghĩa các phương thức được sử dụng cho đối tượng mô tả các trang sẽ được in. Nó cũng chỉ ra số lượng trang sẽ được in cũng như sẽ được in trang hiện hành hay một dãy các trang.

Giao diện '**Printable**' chỉ định phương thức `print()` được dùng để in một trạng trên một đối tượng Graphics.

Giao diện '**PrinterGraphics**' cung cấp khả năng truy cập đối tượng '**PrinterJob**'. Nó cung cấp các lớp sau đây:

- Paper
- Book
- PageFormat
- PrinterJob

Lớp '**Page**' định nghĩa các đặc tính vật lý của trang giấy in. Ngoài ra nó cũng cung cấp khổ giấy và vùng vẽ.

Lớp '**Book**' là một lớp con của đối tượng duy trì một danh sách các trang được in. Lớp này cũng cung cấp các phương thức để thêm trang và quản lý trang. Nó cài đặt giao diện Pageable.

Lớp '**PageFormat**' định nghĩa lề của trang như 'Top', 'Bottom', 'Left' và 'Right'. Nó cũng chỉ định kích cỡ và hướng in như 'Portrait' (khổ dọc) hoặc 'Landscape' (khổ ngang).

Lớp '**PrinterJob**' là một lớp con của đối tượng khởi tạo, quản lý, và điều khiển yêu cầu máy in. Lớp này cũng chỉ định các tính chất in.

Dưới đây là ngoại lệ và lỗi mà gói `java.awt.print` kích hoạt:

- PrinterException
- PrinterIOException
- PrinterAbortException

'**PrinterException**' thừa kế `java.lang.Exception` nhằm cung cấp một lớp cơ sở để in các ngoại lệ liên quan.

'**PrinterIOException**' thừa '**PrinterException**' chỉ rõ lỗi trong i/o.

'**PrinterAbortException**' là lớp con của lớp `PrinterException` nêu rõ việc in đã được hủy bỏ.

## Tóm tắt bài học

- Một dòng là một phương pháp qua đó dữ liệu di chuyển trong chương trình java.



- Khi một dòng dữ liệu được gửi hoặc nhận. Chúng ta xem nó như đang ghi và đọc một dòng tương ứng.
- Dòng nhập/xuất bao gồm các lớp sau đây:
  - Lớp System.out
  - Lớp System.in
  - Lớp System.err
- Lớp InputStream là một lớp trừu tượng định nghĩa cách nhận dữ liệu.
- Lớp OutputStream cũng là lớp trừu tượng. Nó định nghĩa ghi ra các dòng được kết xuất như thế nào.
- Lớp ByteArrayInputStream tạo ra một luồng nhập từ vùng đệm bộ nhớ trong khi ByteArrayOutputStream tạo một luồng xuất trên một mảng byte.
- Java hỗ trợ tác vụ nhập/xuất tập tin với sự trợ giúp của các lớp File, FileDescriptor, FileInputStream và FileOutputStream.
- Các lớp Reader và Writer là lớp trừu tượng hỗ trợ đọc và ghi các dòng ký tự Unicode.
- CharArrayReader, CharArrayWriter khác với ByteArrayInputStream, ByteArrayOutputStream hỗ trợ định dạng nhập/xuất 8 bit, Trong khi ByteArrayInputStream, ByteArrayOutputStream hỗ trợ nhập/xuất 16bit.
- Lớp PrintStream cài đặt giao diện xuất, lớp này có bổ sung phương thức giúp ta in các kiểu dữ liệu nguyên thủy.
- Lớp RandomAccessFile cung cấp khả năng thực hiện I/O tới vị trí cụ thể trong một tập tin.

## Kiểm tra mức độ tiến bộ

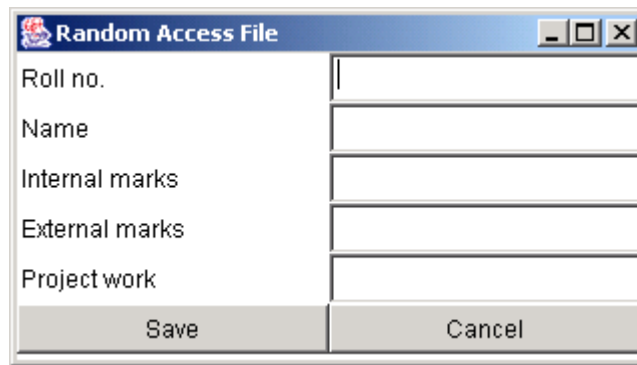
1. ----- là ống (pipeline) để gửi và nhận thông tin trong các chương trình java.
2. ----- là dòng lỗi chuẩn.
3. Phương thức ----- đọc các byte dữ liệu từ một dòng.
4. Phương thức ----- trả về giá trị boolean, kiểm tra xem dòng có hỗ trợ các khả năng mark và reset hay không.
5. Phương thức ----- xả sạch dòng.
6. Nhập/xuất mảng byte sử dụng các lớp ----- và -----
7. Lớp ----- được sử dụng truy cập các đối tượng thư mục và tập tin.
8. -----là một nơi chứa dữ liệu.

## Bài tập

Viết chương trình nhận một dòng văn bản từ người dùng và hiển thị đoạn văn bản đó lên màn hình.

Viết chương trình sao chép nội dung từ một tập tin tới một tập tin khác.

Viết chương trình truy cập ngẫu nhiên tập tin, có giao diện như sau:



A screenshot of a Windows-style dialog box titled "Random Access File". The dialog box has a blue title bar with a small icon on the left and standard window controls (minimize, maximize, close) on the right. The main area contains five text input fields, each preceded by a label: "Roll no.", "Name", "Internal marks", "External marks", and "Project work". At the bottom of the dialog, there are two buttons: "Save" on the left and "Cancel" on the right.

*Giao diện chương trình*

Các bản ghi nên được lưu ở tập tin '.dat', vì vậy người dùng truy cập chúng sau này.

# Cài đặt bảo mật

## Giới thiệu

Trong phần này, chúng ta sẽ tìm hiểu chi tiết về bảo mật Java applet. Chúng ta cũng thảo luận về mô hình bảo mật JDK 1.2 đáp ứng nhu cầu người dùng và người phát triển phần mềm.

Java là một ngôn ngữ lập trình đầu tiên gởi các chương trình tương tác như văn bản, ảnh và các thông tin tĩnh thông qua World Wide Web. Các chương trình này, không giống như chương trình CGI, nó được chạy trên hệ thống của người dùng, thay vì chạy trên máy chủ Web (Web server). Bảo mật Java Applet là sự quan tâm chính giữa người dùng và nhà phát triển applet. Thiếu tính bảo mật trong applet có thể dẫn tới việc sửa đổi hoặc đọc các dữ liệu nhạy cảm trên máy người sử dụng applet. Mô hình bảo mật của Java 2, hoặc JDK 1.2 rất hữu ích cho người dùng, cũng như cho nhà phát triển. Nó giúp người dùng duy trì mức độ bảo mật cao. Trong chương này, chúng ta sẽ học mô hình bảo mật JDK 1.2.

## Công cụ JAR

Một file JAR là một file lưu trữ được nén do công cụ (jar.exe) của Java tạo ra. Nó tương tự như chương trình PKZIP. Nó chứa nhiều file trong một file lưu trữ (.jar). Điều này cho phép tải về trình duyệt hiệu quả. Dùng một jar với applet cải thiện đáng kể hiệu năng thực hiện của trình duyệt. Vì tất cả các tệp các file được biên dịch và để trong một file duy nhất, trình duyệt chỉ cần thiết lập kết nối HTTP với web server. Nén file giảm 50% thời gian tải file.

Để khởi động công cụ JAR, dùng câu lệnh sau tại dấu nhắc lệnh:

```
jar [options][manifest] jar-file input-file(s)
```

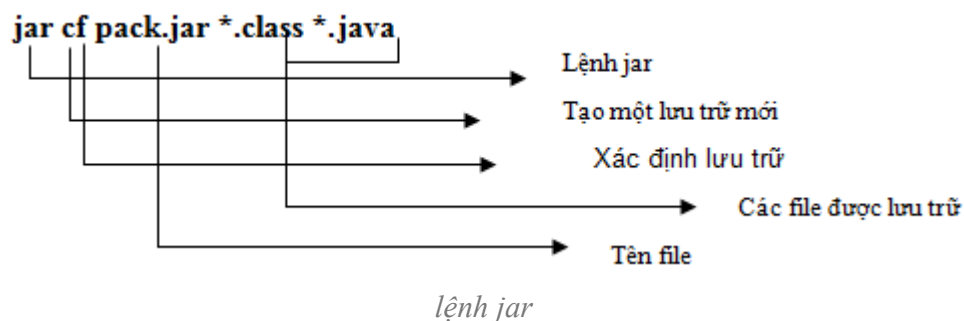
Công cụ jar

| Tuỳ chọn | Mô tả                                                    |
|----------|----------------------------------------------------------|
| c        | Tạo ra một file jar mới                                  |
| t        | Liệt kê nội dung của file jar                            |
| x        | Trích dẫn file có tên từ file jar                        |
| v        | Tạo đầu ra chi tiết (verbose output) trên dòng lỗi chuẩn |
| f        | Xác định tên file jar                                    |

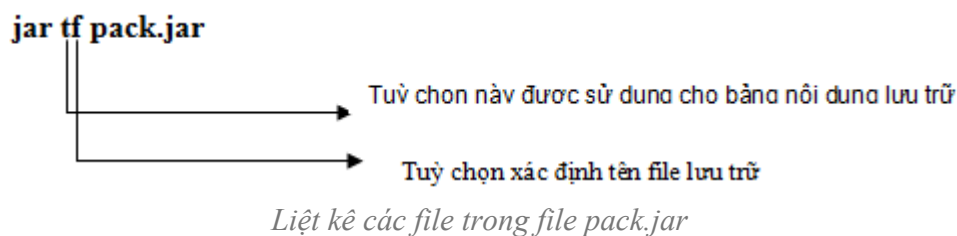
|   |                                                               |
|---|---------------------------------------------------------------|
| m | Bao hàm thông tin chứng thực từ các file chứng thực xác định. |
| o | Chỉ lưu trữ, không nén                                        |
| M | Không tạo các file chứng thực cho các mục (entries).          |

Một file chứng thực chứa thông về các file lưu trữ. File này là một tùy chọn. Thậm chí file không xác định thì JAR cũng tự động tạo ra. File jar được dùng như các lưu trữ. File này phải có phần mở rộng là ‘.jar’ được xác định tại dòng lệnh. File đầu vào (input-file) là danh sách file phân cách bởi dấu phẩy. Netscape Navigator và Internet Explorer hỗ trợ file JAR.

Câu lệnh sau lưu trữ tất cả các file class và file java bao gồm trong một thư mục hiện hành vào một file jar gọi là ‘pack’



Dùng lệnh sau tại dấu nhắc liệt kê các file trong file ‘pack.jar’



Để gộp file lưu trữ ‘pack.jar’ vào trong một applet, mở trang HTML, và thêm thuộc tính ARCHIVE=‘pack.jar’ vào thẻ applet, như sau:

```
<applet code="exr7.class" ARCHIVE="pack.jar" height=125 width=350></applet>
```

Thuộc tính sẽ chỉ cho trình duyệt nạp lưu trữ ‘pack.jar’ để tìm file ‘exr7.class’

Câu lệnh sau trích rút các file được nén trong file pack.jar:

```
jar xvf pack.jar
```

Mục chọn 'x' cho phép bạn trích rút nội dung của file.

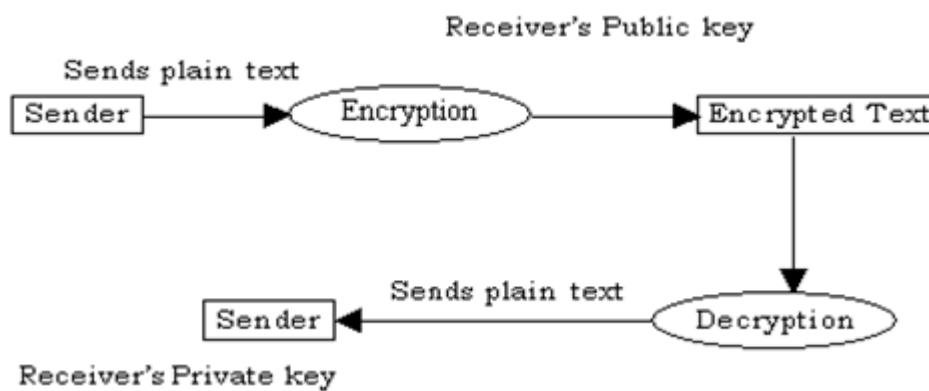
## Chữ ký điện tử (Digital Signature) để định danh các applet

Trong java, bảo mật applet trên web là phần rất quan trọng. Hacker có thể viết các applet nguy hiểm xuyên thủng hàng rào bảo mật. Vì thế, applet hạn chế sự can thiệp của các ngôn ngữ. Applet không hỗ trợ một số thao tác sau:

- Đọc và ghi file từ hệ thống nơi applet đang chạy.
- Lấy thông tin từ file của hệ thống
- Xoá file của hệ thống.

Java 2 có thể thực hiện tất cả các thao tác trên, với các applet cung cấp từ một nhà cung cấp applet tin cậy, và được ký chữ ký điện tử (digitally signed).

[\[link\]](#) sau minh họa quá trình mã hoá khoá



*Mã hoá dựa trên các khoá*

Trong hình trên, khoá công khai (public keys) được dùng mã hoá và giải mã. Cùng ý tưởng được sử dụng cho chữ ký điện tử cộng thêm các tính năng bổ sung.

Một chữ ký điện tử là một file mã hoá đi cùng với chương trình để giúp nhận dạng chính xác nguồn gốc của file. Khóa bí mật tính giá trị từ file applet. Người giữ khoá bí mật kiểm tra nội dung của đối tượng.

Trong chữ ký điện tử, một khóa riêng (private key) được sử dụng để mã hóa, và khóa công khai, được dùng để giải mã. Trong khi ký (sign) trên một đối tượng, phía ký dùng thuật toán Message Digest (như MD5) để tính giá trị digest của đối tượng. Giá trị digest được dùng như là 'dấu vân tay' của đối tượng. Digest sau đó được mã hóa dùng khóa riêng, để tạo ra chữ ký điện tử của đối tượng. Khóa công khai được sử dụng để giải mã và kiểm tra chúng. Kết quả của sự giải mã, giá trị digest được đưa ra. Giá trị digest của đối tượng được tính và so sánh với giá trị digest được giải mã. Nếu giá trị digest của đối

tượng và giá trị digest được mã hoá khớp với nhau, chữ ký được được xác nhận. Tài liệu mô tả chữ ký được gọi là “Chứng thực” (Certificate)

Thiết lập sự tin cậy, nhận dạng applet được chứng nhận. Chứng nhận các thực thể khác sử dụng khóa công khai để giả mạo. Nhà chứng thực (a certificate authority) được dùng để thực hiện chứng nhận. Nhận được được chứng thực từ một CA (Certificate Authority), applet phải đệ trình tài liệu chứng thực sự nhận dạng của nó.

Hiện giờ các công ty đưa ra các dịch vụ xác nhận chứng thực sau:

- VeriSign
- Thawte Certification

Bạn có thể thiết lập các mức bảo mật khác nhau. Một applet có thể đưa ra sự uỷ thác hoàn toàn, hoặc không uỷ thác, với sự giúp đỡ của tập các lớp gọi là permissions. Nhưng nhìn chung, các applet bị giới hạn một các tối đa, trừ khi nhà phát triển ký trên applet. Điều này xác định nhà phát triển là tin cậy.

### **Khoá bảo mật Java (Java Security key).**

Chúng ta cần tạo 3 công cụ, tên là, ‘jar’, ‘jarsigner’, và ‘keytool’, trước khi dùng các applet được ký. Chúng ta cần tạo cặp khóa công public/private, và làm cho nó trở nên sẵn sàng với công cụ jarsigner.

Bây giờ, chúng ta sẽ sử dụng các công cụ để tạo keystore.

- **Keystore** (Lưu trữ khoá)

Keystore là một cơ sở dữ liệu khoá, chứa các chứng thực số dùng để nhận dạng các giá trị khoá công khai (public).

- **Keytool** (Công cụ khoá)

Keytool là công cụ khoá bảo mật của java, tạo và quản lý khóa công khai, khoá riêng (private), và các chứng thực bảo mật. Nó cũng có thể thực hiện:

- Quản lý cặp khoá công public/private
- Lưu trữ các khoá công khai
- Dùng các chứng thực để xác thực chứng thực khác.
- Xác thực (Authenticate) dữ liệu nguồn.

Tất cả thông tin mà keytool quản lý được lưu trữ trong cơ sở dữ liệu gọi là keystore. Sun có một keystore mặc định dùng định dạng file mới gọi là JKS (java key store Lưu trữ

khoá java). Để kiểm tra xem hệ thống bạn có một keystore dưới định dạng này hay chưa hãy thực hiện câu lệnh sau tại dấu nhắc lệnh:

```
keytool -list
```

Thông báo lỗi sau xuất hiện nếu bạn không có gì trong keystore của bạn.

**keytool error: keystore file does not exist: c:\windows\keystore**

JDK tìm keystore chính trong thư mục C:\windows\. Đây là một vị trí chung cho các file hệ thống quan trọng trên windows 95, 98 và NT systems.

Tuỳ chọn keystore cũng có thể được sử dụng trong lệnh keytool, như sau:

```
keytool -list keystore c:\java\try
```

Câu lệnh này chỉ cho JDK tìm keystore trong file được gọi là 'try' trong thư mục 'C:\java\try'. Nếu không tìm thấy, sẽ hiển thị thông báo lỗi như trên.

Lựa chọn '-genkey' có thể được sử dụng cùng với câu lệnh keytool để tạo cặp khoá công public/private. Bạn cũng có thể dùng một số các tuỳ chọn khác. Dạng đơn giản nhất như sau:

```
keytool -genkey -alias "I"
```

Bí danh (alias) có thể được dùng lưu trữ, thay thế hoặc xoá cặp khoá. Các bí danh keytool không phân biệt chữ hoa, thường. Trong lệnh trên, chúng ta không sử dụng tuỳ chọn keystore. Nếu cùng câu lệnh sử dụng tuỳ chọn keystore, sẽ được viết lại như sau:

```
keytool -genkey -alias "I" -keystore "store"
```

Trong lệnh trên, cặp khoá sẽ được lưu trữ trong keystore tên là 'store', và không lưu trong keystore mặc định của hệ thống.

Sau khi nhập lệnh trên vào, và nhấn phím enter, keytool nhắc bạn nhập vào mật khẩu (password) cho keystore, như sau:

**Enter keystore password:**

Nhập vào 'password' là mật khẩu như yêu cầu.

Tiếp theo, keytool nhắc bạn nhập vào các thông tin bổ sung như:

*What is your first and last name? (Tên và họ)*

**[unknown]**

*what is the name of your organization unit?*

**[unknown]: software Development.**

*What is the name of your organization? (Tên của tổ chức)*

**[Unknown]:** ABC Consultants (tư vấn ABC)

*What is the name of your city or Locality? (tên thành phố hoặc địa phương của bạn)*

**[Unknown]:** California

*What is the name of your State or Province? (tên bang hoặc tỉnh của bạn)*

**[Unknown]:**United States of America

*What is the two-letter country code for this unit? (Mã quốc gia với 2 ký tự)*

**[Unknown]:** US

Khi bạn đã nhập vào các thông tin, keytool hiển thị thông tin sau:

**Is <CN=Bob Fernandes, OU=Software Development, O=ABC Consultants, L=California, ST=United States of America, C=US>correct?**

**[no]:**

Cuối cùng, keytool nhắc bạn nhập vào mật khẩu cho khoá riêng của bạn, như:

**Enter key password for <I>**

**(RETURN if same as keystore password):**

Thông tin trên được sử dụng để kết hợp sự phân biệt tên (name) với bí danh (alias). Thông tin trên cũng có thể được đưa vào trực tiếp từ mục chọn ‘-dname’

Mật khẩu sau cùng phân biệt với mật khẩu keystore. Nó được dùng truy cập khoá riêng của cặp khoá. Mật khẩu có thể trực tiếp chỉ rõ bằng cách sử dụng tùy chọn ‘-keypass’. Nếu mật khẩu không chỉ rõ, mật khẩu keystore được sẽ được dùng. Tùy chọn ‘-keypasswd’ dùng thay đổi mật khẩu. Tùy chọn ‘-keyalg’ chỉ rõ thuật toán tạo cặp khoá.



Khi bạn tạo một khoá và bổ sung nó vào trong keystore, bạn có thể dùng tùy chọn ‘**-list**’ của keytool để xem khoá có trong keystore hay không.

Để xoá cặp khoá từ cơ sở liệu, dùng lệnh sau:

```
keytool -delete -alias aliasName
```

‘aliasName’ chỉ tên của khoá được xoá.

Bây giờ, chúng ta tạo cặp khoá riêng/công cộng cho file JAR, chúng ta hãy ký danh nó. Lệnh jarsigner dùng để ký danh một file JAR. Nhập lệnh sau vào dấu nhắc DOS:

```
jarsigner -keystore keyStore -storepass storePassword
-keypass keyPassword
```

Bảng sau cung cấp danh sách của JARFileNames và bí danh:

JARFileNames và bí danh

| Tùy chọn      | Mô tả                         |
|---------------|-------------------------------|
| keyStore      | Tên keystore sử dụng          |
| storePassword | Mật khẩu keystore             |
| keyPassword   | Mật khẩu khoá riêng           |
| JARFileName   | Tên của file JAR được ký danh |
| Alias         | Bí danh của bộ ký danh        |

Để ký danh file JAR ‘pack.jar’, với keystore ‘store’, và mật khẩu để lưu trữ và các khoá riêng là ‘password’, dùng lệnh sau:

```
jarsigner -keystore store -storepass password -keypass
password pack.jar pk
```

‘pk’ nghĩa là tên bí danh.

Nếu tùy chọn ‘**-keystore**’ không chỉ rõ, thì keystore mặc định được dùng.

Để chỉ rõ chữ ký của file JAR được định danh, dùng tùy chọn ‘**-verify**’.

```
jarsigner -verify pack.jar
```

‘pack.jar’ chỉ tên file JAR. Nếu chữ ký không hợp lệ, thì ngoại lệ sau được tạo ra.

**Jarsigner:java.util.zip.ZipException:invalid entry size (expected 900 but got 876 bytes)**

Ngược lại, xuất hiện thông báo “jar verified” (jar được xác minh)

Quá trình xác thực kiểm tra theo các bước sau:

- Có file ‘.DSA’ chứa chữ ký hợp lệ cho file chữ ký .SF không.
- Có các mục trong file chữ ký là các digest cho mỗi mục tương ứng trong file kê khai (manifest file)

## Chữ ký điện tử (Digital Certificates)

Cho đến bây giờ, chúng ta đã học cách tạo và ký một file JAR. Bây giờ, chúng ta sẽ học cách xuất các chữ ký điện tử (digital certificates), nó được sử dụng để xác thực chữ ký của các file JAR. Chúng ta cũng sẽ học cách nhập chữ ký điện tử từ file khác vào.

Chữ ký điện tử là một file, một đối tượng, hoặc một thông báo được ký bởi quyền chứng thực (certificate authority). The CA (Certificate authority) cấp chứng nhận giá trị các khoá công khai. Chứng nhận X.509 của tổ chức International Standards Organization là một dạng chứng nhận số phổ biến. Keytool hỗ trợ những chứng nhận này.

Keytool ở bước đầu tiên cần nhận được một chứng nhận (certificate). Chúng ta dùng chứng nhận đó tạo cặp khoá private/public. Keytool nhập vào các chứng nhận đã được tạo và được ký. Keytool tự động gắn khoá công khai mới với một chứng nhận mới. Cùng thực thể đã tạo khoá công khai ký chứng nhận này. Đó được gọi là **‘self-signed certificates’** (Chứng nhận tự ký). Các chứng nhận này không phải là chứng nhận đáng tin cậy cho định danh. Tuy nhiên, chúng cần để tạo các yêu cầu ký danh chứng nhận (certificate-signing request).

Keytool và tùy chọn được sử dụng để tạo các chứng nhận trên. Câu lệnh sau giúp tạo các chứng nhận trên:

```
keytool -keystore store -alias mykey -certreq -file
mykey.txt
```

Cặp khoá được tạo là ‘mykey’. Tùy chọn ‘-file’ chỉ tên file để lưu certificate-signing.

Dùng tùy chọn ‘-export’ xuất các chứng nhận này như sau:

```
keytool -export -keystore store -alias pk -file mykey
```

Câu lệnh trên hiển thị dấu nhắc sau:

## Enter keystore password

Chúng nhận đã lưu trữ trong <mykey>

Để nhập các chứng nhận khác vào keystore của bạn, nhập câu lệnh sau:

```
keytool -import -keytool keystore -alias alias -file
filename
```

Tên được chỉ như là tên file chứa chứng nhận được nhập vào (imported certificate).

Câu lệnh sau chỉ tên bí danh là 'alice' để nhập chứng nhận trong file 'mykey' vào keystore 'MyStore':

```
keytool -import -keystore MyStore -alias alice -file mykey
```

Câu lệnh trên hiển thị dấu nhắc sau:

**Enter keystore password:** *(Nhập vào mật khẩu keystore)*

Kết quả xuất ra hiển thị hai tùy chọn –Owner và Issuer. Nó hiển thị tên công ty, nghề nghiệp, tổ chức, địa điểm, bang và tiền tệ. Nó cũng hiển thị số serial và thời gian có giá trị. Cuối cùng, nó hỏi là tin cậy chứng nhận này không. Chứng nhận được chấp thuận dựa vào sự tin tưởng cá nhân bạn.

Dùng lệnh '-list' liệt kê nội dung của keystore như sau:

```
keytool -list -keystore Store
```

Câu lệnh trên yêu cầu password keystore

Dùng tùy chọn '-alias' liệt kê một mục. Dùng lệnh **-delete** để xóa bí danh trong keystore, như sau:

```
keytool -delete -keystore Store -alias alias
```

Dùng lệnh **-printcert** in chứng nhận được lưu trữ trong file, theo cách sau:

```
keytool -printcert -file myfile
```

Dùng lệnh **-help** nhận về danh sách tất cả các tùy chọn mà keytool hỗ trợ:

```
keytool -help
```

## Các gói bảo mật java (JAVA Security packages)

Các gói bảo mật Java bao gồm:

- java.security

Đây là gói API bảo mật chính. Chứa các lớp và giao diện hỗ trợ mã hoá, digest và chữ ký điện tử.

- java.security.acl

Chứa các giao diện dùng cài đặt các chính sách điều khiển truy cập

- java.security.cert

Cung cấp sự hỗ trợ cho chứng nhận X.509

- java.security.interfaces

Định nghĩa các giao diện truy nhập Digital Signature Algorithm

- java.security.spec

Cung cấp các lớp độc lập và phụ thuộc vào thuật toán mã hoá sử dụng khoá.

### Tóm tắt

- Nếu khả năng bảo mật trong applet không đảm bảo, các dữ liệu nhạy cảm có thể được sửa đổi hoặc phơi bày.
- Mục đích chính của JAR là kết nối các file mà applet sử dụng trong một file nén duy nhất. Điều này cho phép các applet nạp vào trình duyệt một cách hiệu quả.
- Một file kê khai (manifest file) chứa thông tin về các file lưu trữ.
- Chữ ký điện tử là một mã hoá kèm với chương trình để nhận diện chính xác nguồn gốc của file.
- Keystore là một cơ sở dữ liệu khoá.
- Keytool là công cụ khoá bảo mật của java.
- chứng nhận điện tử là một file, hoặc một đối tượng, hoặc một thông báo được ký bởi quyền chứng nhận (certificate authority)

### Kiểm tra kiến thức

1. File \_\_\_\_\_ là file lưu trữ được nén.

2. Tùy chọn \_\_\_\_\_, khi dùng với công cụ jar, trích rút tên file từ một lưu trữ (.jar)
3. JAR tự động tạo file kê khai, thậm chí nó không được chỉ ra (*Đúng/Sai*)
4. Thuộc tính \_\_\_\_\_, khi dùng trong thẻ applet, chỉ cho trình duyệt nạp file jar, và tìm file class của applet.
5. Trong chữ ký điện tử, \_\_\_\_\_ được dùng cho mã hoá và \_\_\_\_\_ được dùng cho giải mã.
6. Tất cả các thông tin keytool quản lý, được lưu trữ trong một cơ sở dữ liệu gọi là \_\_\_\_\_
7. keytool ở bước đầu tiên cần thiết để nhận được một chứng nhận (**Đúng/Sai**)
8. Gói \_\_\_\_\_ chứa giao diện dùng cài đặt các chính sách điều khiển truy nhập.

## Bài tập

Sử dụng **các câu lệnh java** thực hiện các hành động sau:

1. Tạo một file jar 'core-java.jar' chứa các file lớp (.class) và các file nguồn.
2. Liệt kê nội dung của file jar.
3. Tạo file html nhúng applet CardLayoutDemo.class, file lớp được chứa trong file jar.
4. Trích rút nội dung file jar.
5. Dùng lệnh keytool với tên bí danh và keystore để tạo ra cặp khoá public/private mới.
6. Ký danh file jar mới được tạo.
7. Xác minh chữ ký (signature).
8. Xuất các chứng nhận (certificate)
9. Liệt kê nội dung của keystore.
10. In các chứng nhận được lưu trong file.

## Tham gia đóng góp

Tài liệu: Core Java

Biên tập bởi: Huỳnh Công Pháp

URL: <http://voer.edu.vn/c/95eeacbe>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Lập trình hướng đối tượng

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/d69072fa>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Nhập môn Java

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/45f48591>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Cấu trúc một chương trình Java

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/97ee9ad1>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Chương trình Java đầu tiên

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/284268f9>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Các lớp đối tượng trong Java

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/6aade019>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Kiểu dữ liệu trong Java

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/e96ae52a>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Các biến trong Java

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/4b84429e>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Phương thức trong một lớp (Method)

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/d75e2109>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Các toán tử

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/ef1ad896>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Định dạng dữ liệu xuất và điều khiển luồng

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/a8c61cd4>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Các gói và giao diện

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/99a26993>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Giới thiệu về AWT

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/89fb7cbf>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Container (Vật chứa)

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/1a69826f>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Thành phần (Component)

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/5296baaa>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Thực đơn (Menu)

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/9622d478>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Quản lý cách trình bày (Layout Manager)

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/bb4dcc07>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Xử lý các sự kiện

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/cf92c1e7>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Applets

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/24f359b8>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Xử lý ngoại lệ (Exception Handling)

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/1113670d>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Đa luồng (Multithreading)

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/303a3cc3>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Dòng vào / ra ( I/O Streams)



Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/3670ebde>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: Cài đặt bảo mật

Các tác giả: Huỳnh Công Pháp

URL: <http://www.voer.edu.vn/m/830d33da>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

## **Chương trình Thư viện Học liệu Mở Việt Nam**

Chương trình Thư viện Học liệu Mở Việt Nam (Vietnam Open Educational Resources – VOER) được hỗ trợ bởi Quỹ Việt Nam. Mục tiêu của chương trình là xây dựng kho Tài nguyên giáo dục Mở miễn phí của người Việt và cho người Việt, có nội dung phong phú. Các nội dung đều tuân thủ Giấy phép Creative Commons Attribution (CC-by) 4.0 do đó các nội dung đều có thể được sử dụng, tái sử dụng và truy nhập miễn phí trước hết trong môi trường giảng dạy, học tập và nghiên cứu sau đó cho toàn xã hội.

Với sự hỗ trợ của Quỹ Việt Nam, Thư viện Học liệu Mở Việt Nam (VOER) đã trở thành một cổng thông tin chính cho các sinh viên và giảng viên trong và ngoài Việt Nam. Mỗi ngày có hàng chục nghìn lượt truy cập VOER ([www.voer.edu.vn](http://www.voer.edu.vn)) để nghiên cứu, học tập và tải tài liệu giảng dạy về. Với hàng chục nghìn module kiến thức từ hàng nghìn tác giả khác nhau đóng góp, Thư Viện Học liệu Mở Việt Nam là một kho tàng tài liệu khổng lồ, nội dung phong phú phục vụ cho tất cả các nhu cầu học tập, nghiên cứu của độc giả.

Nguồn tài liệu mở phong phú có trên VOER có được là do sự chia sẻ tự nguyện của các tác giả trong và ngoài nước. Quá trình chia sẻ tài liệu trên VOER trở nên dễ dàng như đếm 1, 2, 3 nhờ vào sức mạnh của nền tảng Hanoi Spring.

Hanoi Spring là một nền tảng công nghệ tiên tiến được thiết kế cho phép công chúng dễ dàng chia sẻ tài liệu giảng dạy, học tập cũng như chủ động phát triển chương trình giảng dạy dựa trên khái niệm về học liệu mở (OCW) và tài nguyên giáo dục mở (OER). Khái niệm chia sẻ tri thức có tính cách mạng đã được khởi xướng và phát triển tiên phong bởi Đại học MIT và Đại học Rice Hoa Kỳ trong vòng một thập kỷ qua. Kể từ đó, phong trào Tài nguyên Giáo dục Mở đã phát triển nhanh chóng, được UNESCO hỗ trợ và được chấp nhận như một chương trình chính thức ở nhiều nước trên thế giới.