

CHƯƠNG 1: TỔNG QUAN VỀ KỸ THUẬT LẬP TRÌNH

1.1. GIỚI THIỆU VỀ CHƯƠNG TRÌNH MÁY TÍNH

Một chương trình máy tính là một dãy hữu hạn các câu lệnh (chỉ dẫn) được viết bằng ngôn ngữ lập trình cho máy tính để thực thi một tác vụ xác định.

Có hai thuật ngữ quan trọng được đề cập trong phân định nghĩa trên:

- Dãy hữu hạn các câu lệnh.
- Ngôn ngữ lập trình cho máy tính.

Ví dụ chương trình máy tính đơn giản dưới đây với một câu lệnh là in ra màn hình dòng chữ: “Xin chào các bạn” được viết bằng ngôn ngữ lập trình Pascal:

```
write("Xin chào các bạn");
```

Khi thực thi câu lệnh trên thì máy tính sẽ in ra dòng chữ "Xin chào các bạn" lên màn hình máy tính.

Như vậy:

- Một chương trình máy tính được gọi là *Phần mềm máy tính*.
- Các câu lệnh cho chương trình máy tính được gọi là *mã nguồn*.

Ngày nay các chương trình máy tính được sử dụng hầu hết trong các lĩnh vực như Y tế, giải trí, an ninh, giao tiếp, ... Danh sách dưới đây là một vài ứng dụng của chương trình máy tính:

MS Word, MS Excel, Adobe Photoshop, Google Chrome, ... là những ví dụ về chương trình máy tính.

Các chương trình máy tính được sử dụng để phát triển đồ họa và những hiệu ứng đặc biệt trong làm phim:

Các chương trình được sử dụng để siêu âm, Chụp X-quang và những dụng y học khác.

Được sử dụng trong điện thoại để nhắn tin SMS, chat và gọi di động.

1.2. NGÔN NGỮ LẬP TRÌNH

1.2.1. Giới thiệu về ngôn ngữ lập trình

Ngôn ngữ lập trình là một ngôn ngữ dùng để viết chương trình cho máy tính. Ta có thể chia ngôn ngữ lập trình thành các loại sau: ngôn ngữ máy, hợp ngữ và ngôn ngữ cấp cao.

- Ngôn ngữ máy (machine language): Là các chỉ thị dưới dạng nhị phân, can thiệp trực tiếp vào trong các mạch điện tử. Chương trình được viết bằng ngôn ngữ máy thì có thể được thực hiện ngay không cần qua bước trung gian nào. Tuy nhiên chương trình viết bằng ngôn ngữ máy dễ sai sót, cồng kềnh và khó đọc, khó hiểu vì toàn những con số 0 và 1.

- Hợp ngữ (assembly language): Bao gồm tên các câu lệnh và quy tắc viết các câu lệnh đó. Tên các câu lệnh bao gồm hai phần: phần mã lệnh (viết tựa tiếng Anh) chỉ phép toán cần thực hiện và địa chỉ chứa toán hạng của phép toán đó. Ví dụ:

INPUT a ; Nhập giá trị cho a từ bàn phím

LOAD a ; Đọc giá trị a vào thanh ghi tổng A

PRINT a ; Hiển thị giá trị của a ra màn hình.

INPUT b

ADD b; Cộng giá trị của thanh ghi tổng A với giá trị b

Trong các lệnh trên thì INPUT, LOAD, PRINT, ADD là các mã lệnh còn a, b là địa chỉ. Để máy thực hiện được một chương trình viết bằng hợp ngữ thì chương trình đó phải được dịch sang ngôn ngữ máy. Công cụ thực hiện việc dịch đó được gọi là Assembler.

- Ngôn ngữ cấp cao (High level language): Ra đời và phát triển nhằm phản ánh cách thức người lập trình nghĩ và làm. Rất gần với ngôn ngữ con người (Anh ngữ) nhưng chính xác như ngôn ngữ toán học. Cùng với sự phát triển của các thế hệ máy tính, ngôn ngữ lập trình cấp cao cũng được phát triển rất đa dạng và phong phú, việc lập trình cho máy tính vì thế mà cũng có nhiều khuynh hướng khác nhau: lập trình cấu trúc, lập trình hướng đối tượng, lập trình logic, lập trình hàm... Một chương trình viết bằng ngôn ngữ cấp cao được gọi là chương trình nguồn (source programs). Để máy tính "hiểu" và thực hiện được các lệnh trong chương trình nguồn thì phải có một chương trình dịch để dịch chương trình nguồn (viết bằng ngôn ngữ cấp cao) thành dạng chương trình có khả năng thực thi.

1.2.2. Một số ngôn ngữ lập trình thông dụng

1.2.2.1. Ngôn ngữ lập trình Pascal

Đây là ngôn ngữ do giáo sư Niklaus Wirth thiết kế vào năm 1970 với mục đích giảng dạy ý niệm lập trình có cấu trúc. Nhưng sau một thời gian do tính ưu việt của nó nên PASCAL đã được sử dụng rộng rãi.

PASCAL là ngôn ngữ lập trình bậc cao. Trước khi PASCAL được phát triển thì việc lập trình được thực hiện trên các ngôn ngữ cấp thấp, các lập trình viên rất khó khăn trong việc xây dựng các chương trình lớn. PASCAL dùng ngôn ngữ sát với ngôn ngữ tự nhiên hơn do đó nó thân thiện với người lập trình hơn. Do vậy nó giảm bớt các công việc nặng nhọc cho người lập trình.

PASCAL kết hợp giữa đặc tính gọn, dễ nhớ, khả năng truy cập cấp thấp, và các cấu trúc giữ liệu đa dạng. PASCAL còn hỗ trợ khả năng đưa các chương trình viết bằng ASSEMBLER vào chương trình của bạn, khả năng đồ họa và hướng đối tượng. PASCAL là ngôn ngữ lập trình có cấu trúc. Tính cấu trúc của PASCAL được thể hiện qua 3 yếu tố: cấu trúc trong dữ liệu, cấu trúc trong các toán tử và cấu trúc trong công cụ thủ tục.

- Tính cấu trúc của dữ liệu được thể hiện qua phần mô tả. Cũng như các ngôn ngữ lập trình khác, PASCAL có một số kiểu dữ liệu được định nghĩa sẵn và các phép toán trên các kiểu dữ liệu này. Từ các kiểu dữ liệu đó, người lập trình có thể xây dựng các kiểu dữ liệu phức tạp hơn. Sau đó để khai báo đối tượng thuộc kiểu dữ liệu phức tạp đó ta không cần trình bày lại cấu trúc thiết lập, mà chỉ cần tham chiếu đến kiểu đó.

- Tính cấu trúc của các toán tử được thể hiện ở chỗ bên trong các toán tử thực hiện một động tác, còn có các toán tử thực hiện nhiều động tác, song sự quan trọng nhất của PASCAL là toán tử hợp thành. Toán tử hợp thành được xây dựng bắt đầu bằng từ khoá BEGIN, sau đó đến đây các toán tử thành phần và kết thúc bằng từ khoá END.

- Tính cấu trúc trong công cụ thủ tục thể hiện thông qua khả năng phân tích chương trình thành các modul độc lập và lời gọi đệ quy thủ tục. PASCAL không phải là ngôn ngữ khó học hơn ngôn ngữ dành cho những người mới bắt đầu làm quen với lập trình (BASIC) nhưng nó lại tỏ ra có những đặc tính cấu trúc hoá tốt hơn và không có những cú pháp mang lỗi.

PASCAL rất thích hợp dùng để giảng dạy trong các nhà trường và cho những người mới bắt đầu học lập trình. Còn đối với những bài toán ứng dụng trong thực tế thì PASCAL ít được sử dụng.

1.2.2.2. Ngôn ngữ lập trình C/C++

Ngôn ngữ C được phát triển từ ngôn ngữ B trên máy UNIX. Đến nay ANSI ban hành chuẩn về C. Cũng giống như PASCAL, C là ngôn ngữ lập trình có cấu trúc. Nhưng nói chặt chẽ về mặt kỹ thuật thì C không phải là ngôn ngữ lập trình có cấu trúc chính cống vì trong C không cho phép các khối giống nhau (chẳng hạn bạn không thể khai báo hàm này trong hàm khác).

C là ngôn ngữ cấp trung vì nó cho phép thao tác trên các bit, byte, và địa chỉ. C kết hợp các yếu tố mềm dẻo của ngôn ngữ bậc cao và khả năng điều khiển mạnh của ASSEMBLER. Do vậy, C tỏ ra thích hợp với lập trình hệ thống.

Chương trình viết bằng C là tập hợp các hàm riêng biệt, giúp cho việc che giấu mã và giữ liệu trở nên dễ dàng. Hàm được viết bởi những người lập trình khác nhau không ảnh hưởng đến nhau và có thể được biên dịch riêng biệt trước khi ráp nối thành chương trình.

So với PASCAL thì C thoáng hơn, chẳng hạn C không kiểm tra kiểu khi chạy, điều này do người lập trình đảm nhiệm.

C tỏ ra ít gắt bó hơn so với các ngôn ngữ bậc cao, nhưng C lại thực tế hơn so với các ngôn ngữ khác.

Một đặc điểm nổi bật của C là C có tính tương thích cao. Chương trình viết bằng C cho một loại máy hoặc hệ điều hành này có thể chuyển dễ dàng sang loại máy hoặc hệ điều hành khác. Hiện nay hầu hết các loại máy tính đều có trình biên dịch C. Một chương trình được viết bằng C sẽ cơ tối ưu, chạy với tốc độ cao và tiết kiệm bộ nhớ. Tuy vậy, C chỉ thích hợp với những chương trình hệ thống hoặc những chương trình đòi hỏi tốc độ. Còn nếu bài toán lớn và phức tạp thì cũng như PASCAL, C cũng rất khó kiểm soát chương trình.

C là một ngôn ngữ lập trình tương đối nhỏ gọn vận hành gần với phần cứng và nó giống với ngôn ngữ Assembler hơn hầu hết các ngôn ngữ bậc cao. Hơn thế, C đôi khi được đánh giá như là "có khả năng di động", cho thấy sự khác nhau quan trọng giữa nó với ngôn ngữ bậc thấp như là Assembler, đó là việc mã C có thể được dịch và thi hành trong hầu hết các máy tính, hơn hẳn các ngôn ngữ hiện tại trong khi đó thì Assembler chỉ có thể chạy trong một số máy tính đặc biệt. Vì lý do này C được xem là ngôn ngữ bậc trung.

1.2.2.3. Ngôn ngữ lập trình Java

JAVA được tạo ra trước năm 1990 bởi nhóm các nhà phát triển của Sun Microsystem có nhiệm vụ phải viết phần mềm hệ thống để nhúng vào các sản phẩm điện tử của khách hàng. Họ đã khắc phục một số hạn chế của C++ để tạo ra ngôn ngữ lập trình JAVA.

Do được phát triển từ C++ nên JAVA rất giống C++. Nhưng JAVA là ngôn ngữ hướng đối tượng hoàn toàn, còn C++ là ngôn ngữ đa hướng. JAVA là ngôn ngữ lập trình mạnh vì nó hội tụ được các yếu tố sau:

- JAVA là ngôn ngữ hướng đối tượng (object oriented programming): Các ngôn ngữ lập trình hướng đối tượng có các modul có thể thay đổi và được xác định trước mà người lập trình có thể gọi ra để thực hiện những nhiệm vụ cụ thể. Trong JAVA các modul này gọi là các lớp (class) và chúng được lưu trữ trong thư viện lớp tạo nên cơ sở của bộ công cụ phát triển JAVA

(Java Development Kit). Trong JAVA tất cả các hàm và biến đều phải là thành phần của một lớp.

- Đơn giản (simple): Mặc dù dựa trên cơ sở của C++ nhưng JAVA đã được lược bỏ các tính năng khó nhất của C++ làm cho ngôn ngữ này dễ dùng hơn. Do vậy việc đào tạo một lập trình viên JAVA ngắn hơn và JAVA trở nên thân thiện với người sử dụng hơn. Trong JAVA không có các con trỏ, không hỗ trợ toán tử Overloading, không có tiền xử lý. Tất cả mọi đối tượng trong một chương trình JAVA đều được tạo trên heap bằng toán tử new - chúng không bao giờ được tạo trên stack. JAVA cũng là ngôn ngữ gom rác (garbage - collected language), vì vậy nó không cần đếm từng new với delete - một nguồn bộ nhớ chung để thất thoát trong các chương trình của C++. Trong thực tế không có toán tử delete trong JAVA.

- Đa luồng (multithread): Có nghĩa là JAVA cho phép xây dựng các trình ứng dụng, trong đó, nhiều quá trình có thể xảy ra đồng thời. Tính đa luồng cho phép các nhà lập trình có thể biên soạn các phần mềm đáp ứng tốt hơn, tương tác hơn và thực hiện theo thời gian thực.

- JAVA độc lập với cấu trúc máy: Đây là thuộc tính đặc sắc nhất của JAVA. Có nghĩa là JAVA không phụ thuộc vào hệ máy, các ứng dụng bằng JAVA có thể dùng được trên hầu như mọi máy tính.

1.2.2.4. Ngôn ngữ lập trình C#

C# (đọc là "C thăng" hay "C sharp" ("xi-sấp")) là một ngôn ngữ lập trình hướng đối tượng được phát triển bởi Microsoft, là phần khởi đầu cho kế hoạch .NET của họ. Tên của ngôn ngữ bao gồm ký tự thăng theo Microsoft nhưng theo ECMA là C#, chỉ bao gồm dấu số thường. Microsoft phát triển C# dựa trên C++ và Java. C# được miêu tả là ngôn ngữ có được sự cân bằng giữa C++, Visual Basic, Delphi và Java.

C# được thiết kế chủ yếu bởi Anders Hejlsberg kiến trúc sư phần mềm nổi tiếng với các sản phẩm Turbo Pascal, Delphi, J++, WFC. Phiên bản gần đây nhất là 8.0, được phát hành vào năm 2019 cùng với Visual Studio 2019 phiên bản 16.3.[14]

Tiêu chuẩn ECMA liệt kê các mục tiêu của việc thiết kế ngôn ngữ C#:

- Ngôn ngữ được dự định là một ngôn ngữ lập trình đơn giản, hiện đại, hướng đến nhiều mục đích sử dụng, và là một ngôn ngữ lập trình hướng đối tượng.
- Ngôn ngữ và việc triển khai đáp ứng các nguyên tắc của ngành kỹ thuật phần mềm như kiểm tra chặt chẽ kiểu dữ liệu, kiểm tra giới hạn mảng, phát hiện các trường hợp sử dụng các biến chưa có dữ liệu, và tự động thu gom rác. Tính mạnh mẽ, sự bền bỉ, và năng suất của việc lập trình là rất quan trọng đối với ngôn ngữ này.
- Ngôn ngữ sẽ được sử dụng để phát triển các thành phần của phần mềm theo hướng thích hợp cho việc triển khai trong các môi trường phân tán.
- Ngôn ngữ sẽ được thiết kế để phù hợp với việc viết các ứng dụng cho cả hai hệ thống: hosted và nhúng, từ các phần mềm quy mô lớn, đến các phần mềm chỉ có các chức năng đơn giản.
- Mặc dù các ứng dụng C# có tính kinh tế đối với các yêu cầu về bộ nhớ và chế độ xử lý, ngôn ngữ này không cạnh tranh trực tiếp về hiệu năng và kích thước đối với ngôn ngữ C hoặc assembly.

1.3. CÁC PHƯƠNG PHÁP LẬP TRÌNH

1.3.1. Phương pháp lập trình tuần tự (Sequential Programming)

Vào thừa sơ khai phương pháp lập trình là lập trình tuần tự. Một chương trình sẽ là một tập hợp các câu lệnh có thứ tự chạy nối tiếp nhau.

Chương trình sẽ chạy từ câu lệnh đầu tiên, và kết thúc ở câu lệnh cuối cùng. Lập trình thế này khó, lâu, và việc xử lý các bài toán lớn sẽ khó khăn, chương trình chỉ có thể viết theo một cấu trúc duy nhất là tuần tự.

1.3.2. Phương pháp lập trình cấu trúc (Structured Programming)

Phương pháp lập trình này ngoài cấu trúc tuần tự, ta có thể viết chương trình theo các cấu trúc rẽ nhánh, cấu trúc vòng lặp và kết hợp của các cấu trúc đó với nhau. Ở phương pháp lập trình này xuất hiện khái niệm về cấu trúc điều khiển (control structure) như if-else, switch case... để khiển rẽ nhánh hay for, while để điều khiển vòng lặp.

Phương pháp này đã khiến việc lập trình trở nên linh hoạt hơn rất nhiều, nhưng nó vẫn thực sự hạn chế khi lập trình các hệ thống lớn. Hiện nay nó không còn là phương pháp lập trình thông dụng. Nhưng nó vẫn tồn tại như là một phần không thể thiếu trong các phương pháp lập trình hiện đại hơn.

Ngôn ngữ hỗ trợ lập trình cấu trúc thông dụng như: Pascal,...

1.3.3. Phương pháp lập trình thủ tục (Procedural Programming)

Phương pháp lập trình thủ tục hay còn gọi là hướng chức năng. Phương pháp này sẽ chia một chương trình thành các khối thủ tục nhỏ, mỗi thủ tục sẽ có dữ liệu và logic riêng. Các thủ tục làm việc độc lập với nhau, dữ liệu sẽ được trao đổi qua các đối số (arguments) và giá trị trả về (returned value). Việc chia thủ tục sẽ có nhiều lợi thế ví dụ như việc sử dụng lại mã (reuse code) để dễ dàng chia công việc cho nhiều người.

Lập trình thủ tục đi liền với các khái niệm khái niệm hàm (function), thủ tục (procedure), tham số (parameter), đối số (argument), trả về (return).. Ở một số ngôn ngữ như C, C++ không phân biệt hàm với thủ tục, ở một số ngôn ngữ khác như pascal thì hàm sẽ có giá trị trả về và sử dụng từ khóa function, còn thủ tục không có giá trị trả về và sử dụng từ khóa procedure.

Cách lập trình này đã trừu tượng hóa công việc lập trình hơn rất nhiều. Người lập trình có thể quan sát tổng thể một chương trình qua các chức năng của nó mà không cần quan tâm tới chi tiết trong nó. Hiểu được đặc điểm này là rất quan trọng, nó sẽ giúp bạn giải quyết các bài toán lớn một cách dễ dàng và chính xác hơn rất nhiều.

Các ngôn ngữ hỗ trợ lập trình hướng thủ tục thông dụng như: C, C++, Pascal, PHP...

1.3.4. Phương pháp lập trình hướng đối tượng (Object-Oriented Programming)

Lập trình hướng đối tượng (object-oriented programming viết tắt là OOP), hay còn gọi là lập trình định hướng đối tượng, là kỹ thuật lập trình hỗ trợ công nghệ đối tượng. OOP được xem là giúp tăng năng suất, đơn giản hóa độ phức tạp khi bảo trì cũng như mở rộng phần mềm bằng cách cho phép lập trình viên tập trung vào các đối tượng phần mềm ở bậc cao hơn. Ngoài ra, nhiều người còn cho rằng OOP dễ tiếp thu hơn cho những người mới học về lập trình hơn là các phương pháp trước đó. Một cách giản lược, đây là khái niệm và là một nỗ lực nhằm giảm nhẹ các thao tác viết mã cho người lập trình, cho phép họ tạo ra các ứng dụng mà các yếu

tổ bên ngoài có thể tương tác với các chương trình đó giống như là tương tác với các đối tượng vật lý.

Những đối tượng trong một ngôn ngữ OOP là các kết hợp giữa mã và dữ liệu mà chúng được nhìn nhận như là một đơn vị duy nhất. Mỗi đối tượng có một tên riêng biệt và tất cả các tham chiếu đến đối tượng đó được tiến hành qua tên của nó. Như vậy, mỗi đối tượng có khả năng nhận vào các thông báo, xử lý dữ liệu (bên trong của nó) và gửi ra hay trả lời đến các đối tượng khác hay đến môi trường.

Trong OOP, việc lập trình dựa trên cơ chế kế thừa, tận dụng mọi đặc trưng đã được mô tả cho các lớp có sẵn để tạo ra lớp mới. Các đối tượng trong OOP dùng các thông báo gửi tới các đối tượng khác để thực hiện yêu cầu tính toán cần thiết. OOP quan tâm nhiều tới việc nhìn nhận khía cạnh tĩnh của đối tượng - dữ liệu, và coi chương trình xử lý là một thành phần của dữ liệu đó phản ánh mặt động của đối tượng.

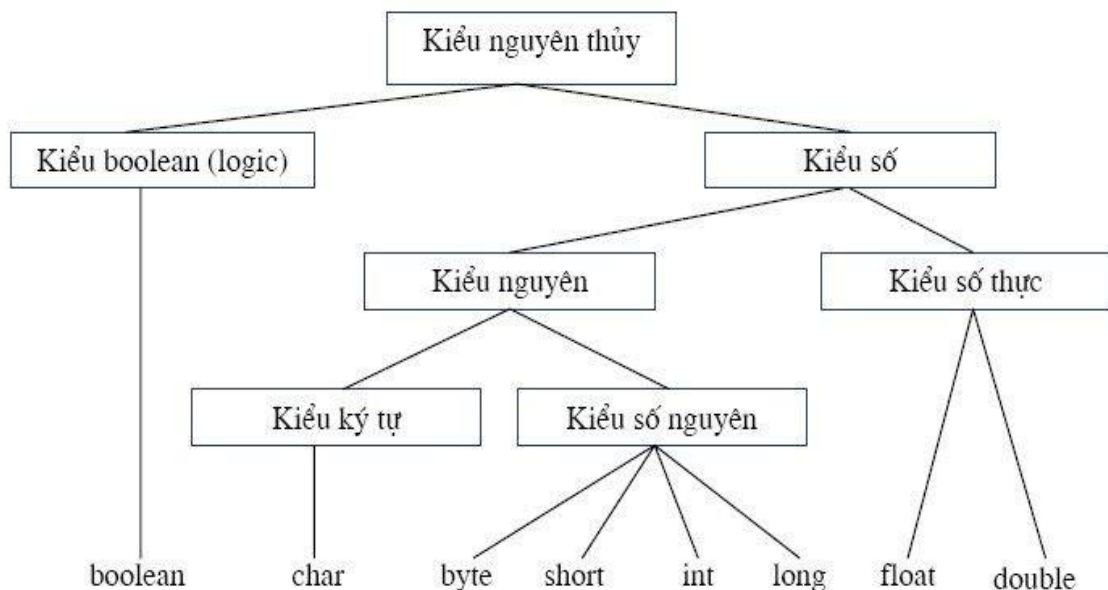
Hầu hết các ngôn ngữ lập trình hiện đại đều có xu hướng hỗ trợ lập trình hướng đối tượng. Các ngôn ngữ lập trình hướng đối tượng phổ biến: Objective-C, C++, JAVA, C#...

1.4. CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

1.4.1. Cấu trúc dữ liệu

1.4.1.1. Khái niệm cấu trúc dữ liệu

Dữ liệu nguyên thủy là kiểu dữ liệu được cung cấp sẵn trong các ngôn ngữ lập trình, nó dùng để lưu trữ các giá trị đơn giản.



Các bạn có thể thấy có 8 kiểu dữ liệu nguyên thủy được chia ra thành 4 nhóm chính đó là:

- Kiểu logic(boolean): lưu giá trị logic nhận giá trị true hoặc false
- Kiểu ký tự(char): lưu các thông tin là ký tự: ví dụ 'a', 'c', 'd' ...
- Kiểu số nguyên(byte, short, int, long): lưu các thông tin là kiểu số nguyên như 1,2,3,4,5,6,7,8...
- Kiểu số thực(float, double): lưu các thông tin là kiểu số thực như 1.4, 2.5, 2.6...

Cấu trúc dữ liệu là cách lưu trữ, tổ chức dữ liệu một cách logic để dữ liệu có thể được sử dụng và quản lý một cách hiệu quả.

Giả sử như bạn đang tháo một thiết bị điện tử mà bạn chưa tháo lần nào trước đó, thì làm sao để các bạn có thể lắp ráp lại sau khi tháo ra?

Chúng ta sẽ sử dụng các truyền thống và đơn giản nhất, đó là đặt từng bộ phận được tháo ra theo thứ tự, sau đó, khi lắp vào thì chỉ việc cái nào ra sau thì vào trước. Đó cũng là nguyên tắc mà “stack” một loại cấu trúc dữ liệu sử dụng để hoạt động, dữ liệu vào sau sẽ được lấy ra trước.

1.4.1.2. Một số loại cấu trúc dữ liệu

Cấu trúc dữ liệu Array:

Array (hay là Mảng) là loại cấu trúc dữ liệu đầu tiên bạn cần phải biết. Đây cũng là loại cấu trúc dữ liệu bạn thường xuyên sử dụng nhất.

Value:	32	52	8				
Index:	0	1	2	3	4	5	6

Value:	Đỏ	Xanh	Vàng	Lam			
Index:	0	1	2	3	4	5	6

Chúng là các cấu trúc dữ liệu giống như danh sách cho phép bạn lưu trữ nhiều phần tử, nhiều loại trong một biến.

Các phần tử này có thể là các kiểu tham chiếu như: Đối tượng, các mảng khác, các kiểu dữ liệu nguyên thủy ...

Cấu trúc dữ liệu Linked list:

Linked list là một tập hợp tuyến tính các phần tử. Mỗi phần tử trong đó sẽ chỉ đến phần tử tiếp theo.



Nó là một tập hợp các nút, trong đó chứa: Dữ liệu và một tham chiếu đến nút kế tiếp trong dãy.

Và vì Linked list không cần được lưu trữ liên tục trong bộ nhớ (Không có vị trí vật lý) nên nó có thể dễ dàng bị chèn hoặc xóa mà không cần sắp xếp lại toàn bộ cấu trúc.

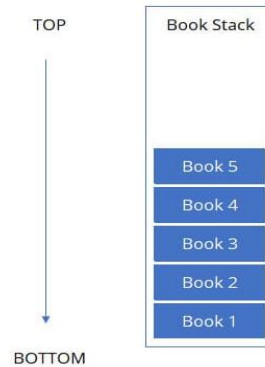
Cấu trúc Linked list cũng tuân theo nguyên tắc LIFO (Last-in-first-out), có nghĩa là vào sau thì ra trước.

Bạn hãy tưởng tượng như là ông C sử đó. Bạn muốn lấy ra cái bên dưới thì phải lấy cái ở ngoài cùng ra trước. (Theo cách thông thường)

Cấu trúc dữ liệu Stack:

Chắc bạn đã nghe đến Stack Overflow rồi nhỉ? Dịch sát nghĩa thì có nghĩa là "Ngăn xếp tràn".

Trong đó, Stack chính là loại cấu trúc dữ liệu ngăn xếp.



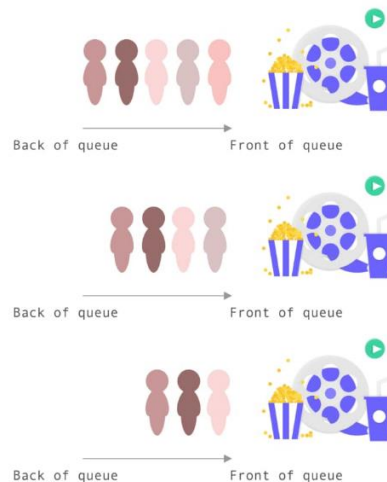
Stack cũng tuân theo nguyên tắc LIFO.

Stack cho phép thêm và xóa liên tục một mục nhưng không may là chúng không cung cấp quyền truy cập liên tục vào mục thứ n trong ngăn xếp.

Cấu trúc dữ liệu Queues:

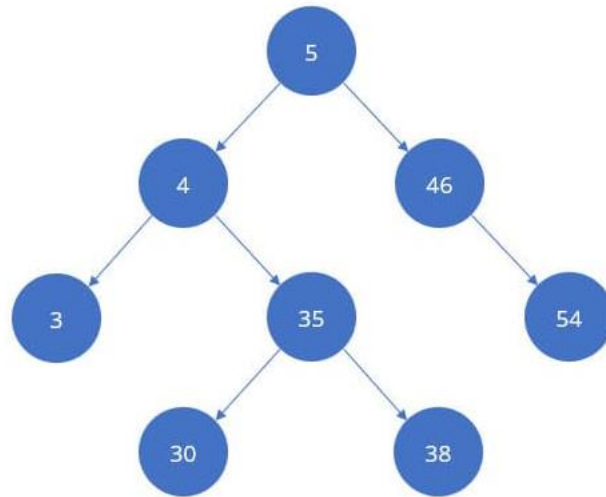
Cấu trúc dữ liệu Queues là một hàng đợi. Một hàng đợi tương tự như một Stack, nhưng sử dụng phương thức FIFO (first-in-first-out), hoặc nhập trước xuất trước.

Giống như xếp hàng check-in vậy đó, ai đến trước thì check-in trước



Cấu trúc dữ liệu Binary Trees:

Binary Tress (Cây nhị phân) là một cấu trúc dữ liệu bao gồm một loạt các nút trong đó mỗi nút có thể có tối đa hai nút con, mỗi nút một giá trị.



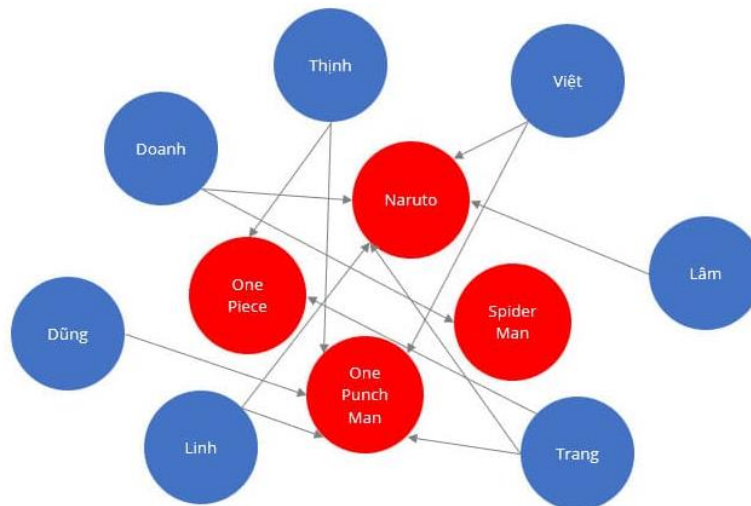
Root là nút trên cùng trong cấu trúc Binary Tree, không có nút cha.

Binary Search Tree là một loại cây nhị phân khác trong đó tất cả các giá trị nút là khác nhau, mỗi nút cha có một 2 nút con, trong đó:

- Nút con bên trái có giá trị nhỏ hơn nút cha
- Nút con bên phải có giá trị lớn hơn nút cha

Cấu trúc dữ liệu Graphs:

Graphs (Đồ thị) là một cấu trúc dữ liệu bao gồm các nút có các cạnh.



Nếu một biểu đồ được định hướng, điều đó có nghĩa là mỗi cạnh có một hướng được liên kết với nó (giống như đường một chiều). Ngược lại, vô hướng không có hướng cạnh liên quan.

Bạn có thể có một biểu đồ về người (màu xanh) và phim (màu đỏ) trong đó mỗi người có thể có một vài bộ phim yêu thích nhưng phim không thể không có người yêu thích.

1.4.2. Giải thuật

1.4.2.1. Khái niệm giải thuật

“Giải thuật, còn gọi là thuật toán, là một tập hợp hữu hạn hay một dãy các quy tắc chặt chẽ của các chỉ thị, phương cách hay 1 trình tự các thao tác trên một đối tượng cụ thể được xác

định và định nghĩa rõ ràng cho việc hoàn tất một số sự việc từ một trạng thái ban đầu cho trước; khi các chỉ thị này được áp dụng triệt để thì sẽ dẫn đến kết quả sau cùng như đã dự đoán trước.”

Thuật toán, giải thuật hay tiếng Anh gọi là algorithms là tập hợp các bước, thao tác để giải quyết một vấn đề gì đó.

Để dễ hiểu thì mình sẽ liên hệ thực tế cho các bạn dễ hình dung.

Giả sử như bạn có một vấn đề cần giải quyết đó là biến gạo thành cơm để ăn (hay nói cách khác là nấu cơm), bạn sẽ phải thực hiện những bước sau đây:

Lấy gạo - vo gạo - đóng nước - đặt vào nồi - đậy nắp - cắm điện - bật nút - ngồi đợi nó chín.

Đấy, các bạn vừa thực hiện một “thuật toán nấu cơm” rồi đấy.

Tương tự như vậy trong lập trình cũng có rất nhiều thuật toán để giải quyết nhiều vấn đề khác nhau ví dụ như: Quy hoạch động, Tiệm cận, Tham lam, Chia để trị,...

1.4.2.2. Vai trò của giải thuật

Với mỗi vấn đề cần giải quyết, ta có thể tìm ra nhiều thuật toán khác nhau. Có những thuật toán thiết kế đơn giản, dễ hiểu, dễ lập trình và sửa lỗi, tuy nhiên thời gian thực hiện lớn và tiêu tốn nhiều tài nguyên máy tính. Ngược lại, có những thuật toán thiết kế và lập trình rất phức tạp, nhưng cho thời gian chạy nhanh hơn, sử dụng tài nguyên máy tính hiệu quả hơn. Khi đó, câu hỏi đặt ra là ta nên lựa chọn giải thuật nào để thực hiện?

Đối với những chương trình chỉ được thực hiện một vài lần thì thời gian chạy không phải là tiêu chí quan trọng nhất. Đối với bài toán kiểu này, thời gian để lập trình viên xây dựng và hoàn thiện thuật toán đáng giá hơn thời gian chạy của chương trình và như vậy những giải thuật đơn giản về mặt thiết kế và xây dựng nên được lựa chọn.

Đối với những chương trình được thực hiện nhiều lần thì thời gian chạy của chương trình đáng giá hơn rất nhiều so với thời gian được sử dụng để thiết kế và xây dựng nó. Khi đó, lựa chọn một giải thuật có thời gian chạy nhanh hơn (cho dù việc thiết kế và xây dựng phức tạp hơn) là một lựa chọn cần thiết. Trong thực tế, trong giai đoạn đầu của việc giải quyết vấn đề, một giải thuật đơn giản thường được thực hiện trước như là 1 nguyên mẫu (prototype), sau đó nó sẽ được phân tích, đánh giá, và cải tiến thành các phiên bản tốt hơn.

1.4.2.3. Mối quan hệ giữa cấu trúc dữ liệu và giải thuật

Thực hiện một đề án tin học là chuyển bài toán thực tế thành bài toán có thể giải quyết trên máy tính. Một bài toán thực tế bất kỳ đều bao gồm các đối tượng dữ liệu và các yêu cầu xử lý trên những đối tượng đó.

Tuy nhiên khi giải quyết một bài toán trên máy tính, chúng ta thường có khuynh hướng chỉ chú trọng đến việc xây dựng giải thuật mà quên đi tầm quan trọng của việc tổ chức dữ liệu trong bài toán. Giải thuật phản ánh các phép xử lý, còn đối tượng xử lý của giải thuật lại là dữ liệu, chính dữ liệu chứa đựng các thông tin cần thiết để thực hiện giải thuật. Để xác định được giải thuật phù hợp cần phải biết nó tác động đến loại dữ liệu nào (ví dụ để làm nhuyễn các hạt đậu, người ta dùng cách xay chứ không băm bằng dao, vì đậu sẽ văng ra ngoài) và khi chọn lựa cấu trúc dữ liệu cũng cần phải hiểu rõ những thao tác nào sẽ tác động đến nó (ví dụ để biểu diễn các điểm số của sinh viên người ta dùng số thực thay vì chuỗi ký tự vì còn phải thực hiện thao tác tính trung bình từ những điểm số đó).

Như vậy trong một đề án tin học, giải thuật và cấu trúc dữ liệu có mối quan hệ chặt chẽ với nhau, được thể hiện qua công thức: **Cấu trúc dữ liệu + Giải thuật = Chương trình**

Với một cấu trúc dữ liệu đã chọn, sẽ có những giải thuật tương ứng, phù hợp. Khi cấu trúc dữ liệu thay đổi thường giải thuật cũng phải thay đổi theo để tránh việc xử lý gượng ép, thiếu tự nhiên trên một cấu trúc không phù hợp. Hơn nữa, một cấu trúc dữ liệu tốt sẽ giúp giải thuật xử lý trên đó có thể phát huy tác dụng tốt hơn, vừa đáp ứng nhanh vừa tiết kiệm vật tư, giải thuật cũng dễ hiểu và đơn giản hơn.

1.5. BIỂU DIỄN GIẢI THUẬT BẰNG LƯU ĐỒ THUẬT TOÁN

Lưu đồ thuật toán là công cụ dùng để biểu diễn thuật toán, mô tả nhập (input), dữ liệu xuất (output) và luồng xử lý thông qua các ký hiệu hình học. Công cụ này rất thích hợp để bạn học cách tư duy phân tích bài toán.

1.5.1. Giới thiệu

Để vẽ lưu đồ thuật toán, bạn cần nhớ và tuân thủ các ký hiệu sau đây:

STT	Ký hiệu	Ý nghĩa
1		Bắt đầu / Kết thúc chương trình
2		Điều kiện rẽ nhánh (lựa chọn)
3		Luồng xử lý
4		Nhập
5		Xuất
6		Xử lý / tính toán / gán
7		Trả về giá trị (return)
8		Điểm nối liên kết

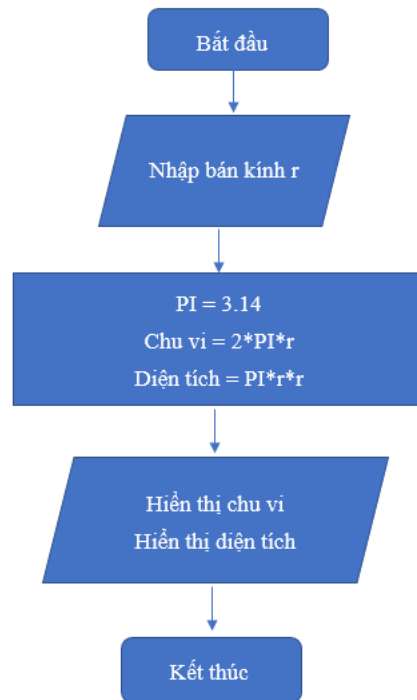
Lưu đồ thuật toán được duyệt lưu đồ thuật toán theo trình tự sau:

- Duyệt từ trên xuống.
- Duyệt từ trái sang phải.

1.5.2. Ví dụ về lưu đồ thuật toán

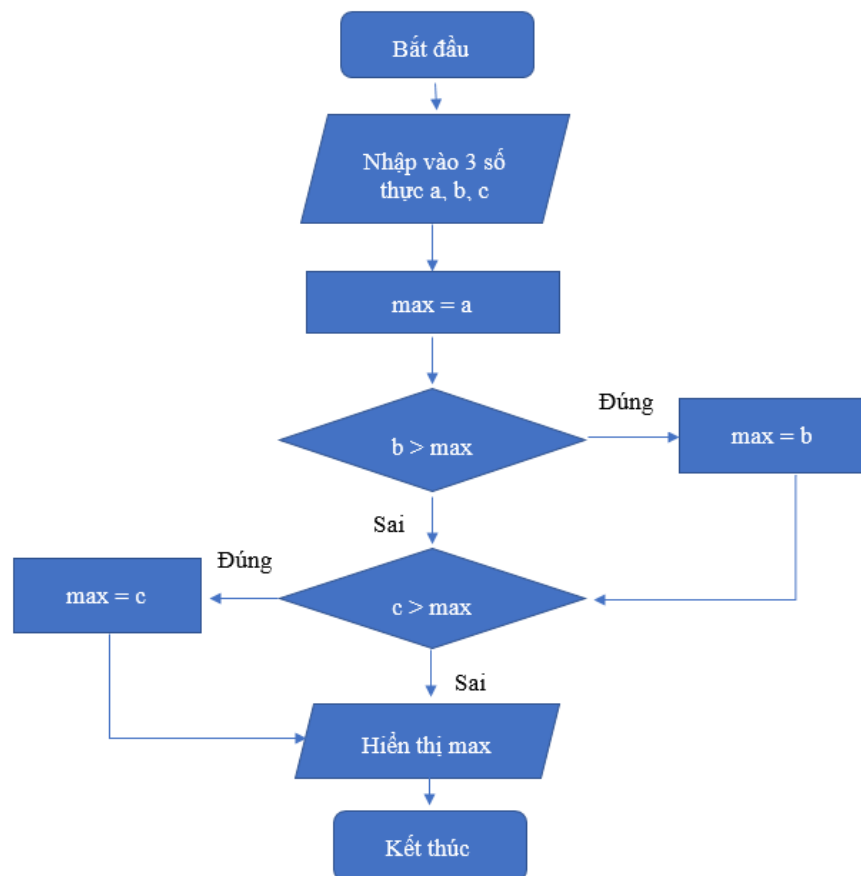
Ví dụ 1.1: Vẽ lưu đồ cho thuật toán tính chu vi, diện tích hình tròn. Hiển thị chu vi và diện tích sau khi tính.

Phân tích: Nhập vào bán kính r ; Xử lý là tính chu vi $= 2*PI*r$, diện tích $= PI*r*r$; Hiển thị chu vi và diện tích



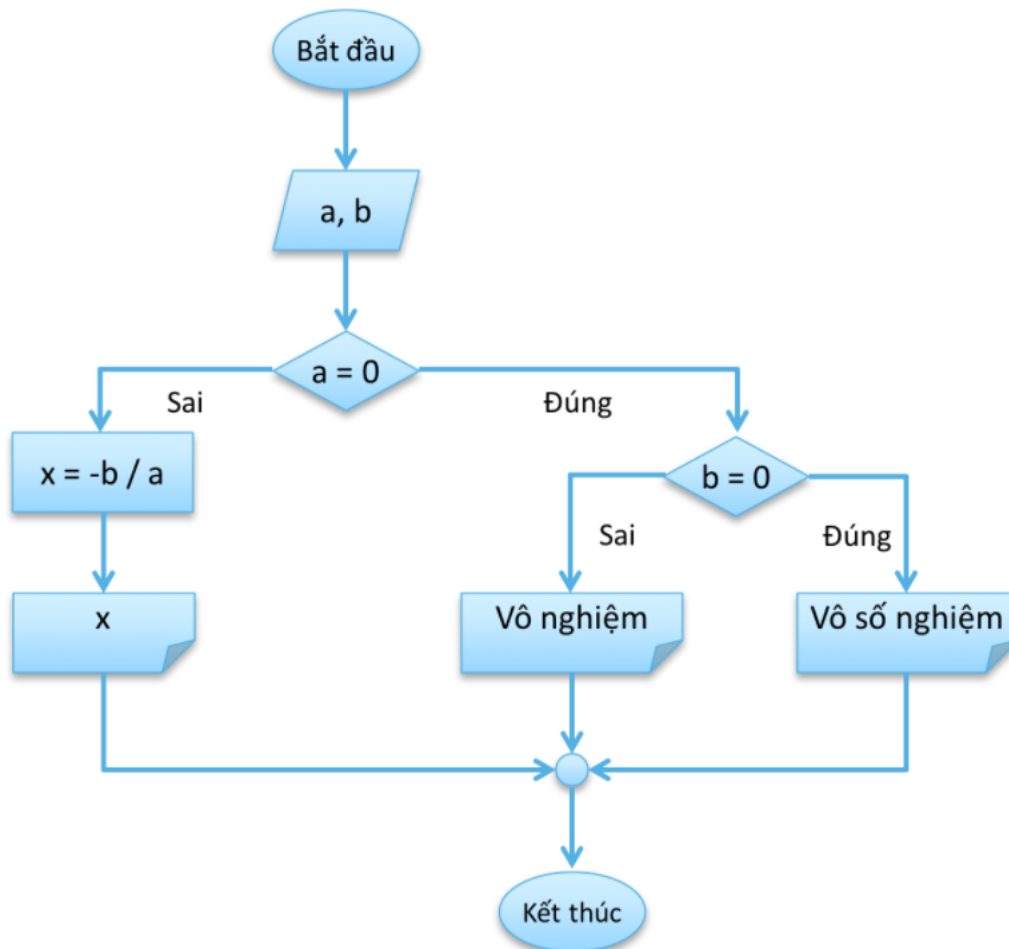
Ví dụ 1.2: Vẽ lưu đồ cho thuật toán tìm số lớn nhất trong 3 số a , b và c .

Phân tích: Nhập vào 3 số thực; Xử lý tìm số lớn nhất bằng cách so sánh; Hiển thị số lớn nhất



Ví dụ 1.3: Giải và biện luận phương trình bậc nhất: $ax + b = 0$

Phân tích: Đầu vào: hai số nguyên a và b . Đầu ra: nghiệm của phương trình.



1.6. NHỮNG TỐ CHẤT CẦN THIẾT ĐỂ TRỞ THÀNH LẬP TRÌNH VIÊN

Nghề lập trình đòi hỏi sự sáng tạo cũng như các kỹ năng đánh giá, phân tích yêu cầu của dự án, đưa ra các giải pháp thiết kế hoặc cách tiếp cận công nghệ mới. Do vậy, để trở thành một lập trình viên tốt thì cần phải có các tố chất cơ bản như sau:

- *Suy nghĩ một cách logic*: Logic là điều quan trọng nhất trong lập trình. Bạn phải có đủ nhạy bén, linh hoạt và khả năng phán xét cao để giải quyết một vấn đề triệt để bằng phương pháp logic. Vì thế, nếu không có khả năng suy luận logic thì chắc chắn một điều nghề lập trình không thích hợp với bạn. Bạn sẽ cảm thấy nhàm chán và nhức đầu khi theo đuổi các đoạn code của chương trình, các vấn đề về debug (gỡ rối), về lỗi, về dấu chấm, dấu phẩy...

- *Tiếp cận vấn đề có thứ tự và luôn chú ý tới chi tiết*: Các lập trình viên nên tập cho mình thói quen cẩn thận và luôn chú ý đến từng chi tiết. Đôi khi những chi tiết rất nhỏ, bạn vô tình bỏ qua, thì bạn phải ân hận khi mất hàng ngàn giờ chỉ để tìm những lỗi nhỏ đó. Bạn cần có kỹ năng truyền đạt thông tin tốt cũng như viết chương trình của mình một cách mạch lạc, có cấu trúc để đồng nghiệp của bạn có thể biết được tại sao bạn lại viết đoạn mã như vậy và cái gì tiếp theo sẽ xảy ra trong chương trình của bạn.

- *Làm việc nhóm*: Đa số, công việc lập trình đều làm việc theo nhóm. Khả năng để bạn thích ứng, và chia sẻ những ý kiến của bạn tại công ty chiếm vị trí rất quan trọng. Bạn phải biết cách

phối hợp công việc với cộng sự, khả năng thuyết trình, giao tiếp, ứng xử của bạn sẽ được sử dụng tối đa trong môi trường làm việc này.

- *Làm việc một mình trong thời gian dài*: Thời hạn của dự án luôn làm bạn đau đầu. Có đôi lúc, bạn phải ngồi làm việc một mình, do đó, bạn cần phải có tính độc lập cao hơn, biết tổ chức và sắp xếp thời gian để hoàn thành dự án đúng lúc. Để được như vậy, bạn cần phải ghi danh sách những việc bạn phải làm và có ý chí quyết tâm cao khi làm việc một mình.

- *Kỹ năng thiết kế*: Công việc phân tích và thiết kế luôn là công việc rất quan trọng của lập trình. Bạn có thể phải thiết kế toàn bộ một hệ thống cho kinh doanh, bao gồm các bảng lưu trữ thông tin, các giao diện để nhập xuất thông tin hay các tài liệu kỹ thuật liên quan đến chương trình... Bạn phải giỏi trong việc lắng nghe và chuyển đổi các yêu cầu của các khách hàng đơn lẻ, các nhóm khách hàng và thậm chí cả việc kinh doanh thành các ứng dụng. Các chương trình của bạn phải dễ dùng và có hiệu quả cao. Do vậy, bất kỳ kỹ năng thiết kế nào của bạn cũng sẽ rất hữu ích trong lĩnh vực này.

- *Kiên nhẫn*: Các vấn đề mà các lập trình viên phải giải quyết thường là các vấn đề khó có thể giải quyết ngay lập tức. Nó mất nhiều giờ, nhiều ngày, thậm chí nhiều tháng làm việc một cách cẩn thận để giải quyết, tìm hướng đi. Nhiều khi bạn đi sai hướng lại phải quay lại giải quyết từ phần đã giải quyết đúng và bắt đầu lại.

- *Tự học*: Không trường lớp nào có thể đào tạo cho bạn tất cả những thứ bạn cần cho công việc lập trình sau này. Chính vì thế, khả năng tự học qua sách vở, tài liệu, internet và qua cả bạn bè nữa là không thể thiếu. Kết hợp với những dự án làm việc trong thực tế, bạn sẽ dần dần thành thạo những gì mình đã tự học được.

1.7. BÀI TẬP CHƯƠNG 1

Bài tập 1.1: Nêu các đặc điểm của ngôn ngữ lập trình Java?

Bài tập 1.2: Nêu vai trò của cấu trúc dữ liệu và giải thuật?

Bài tập 1.3: Vẽ lưu đồ thuật toán nhập vào số nguyên n, xuất ra màn hình từ 1 đến n.

Bài tập 1.4: Vẽ lưu đồ thuật toán giải phương trình bậc 2?

Bài tập 1.5: Vẽ lưu đồ thuật toán tính tổng:

$$S(n) = \frac{1}{2} + \frac{3}{4} + \frac{5}{6} + \dots + \frac{2n+1}{2n+2}, \text{ với } n > 0$$

CHƯƠNG 2: GIỚI THIỆU VỀ NGÔN NGỮ LẬP TRÌNH JAVA

2.1. LỊCH SỬ PHÁT TRIỂN NGÔN NGỮ JAVA

Năm 1991, một nhóm kỹ sư của Sun Microsystems muốn lập trình để điều khiển các thiết bị điện tử như tivi, máy giặt, lò nướng... Ban đầu, họ định dùng C và C++ nhưng trình biên dịch C/C++ lại phụ thuộc vào từng loại CPU. Do đó, họ đã bắt tay vào xây dựng một ngôn ngữ chạy nhanh, gọn, hiệu quả, độc lập thiết bị và ngôn ngữ “Oak” ra đời và vào năm 1995, sau đó được đổi tên thành Java.

Ngôn ngữ lập trình Java được Sun Microsystems đưa ra giới thiệu vào tháng 6 năm 1995 và đã nhanh chóng trở thành một ngôn ngữ lập trình của các lập trình viên chuyên nghiệp. Java được xây dựng dựa trên nền tảng của C và C++ nghĩa là Java sử dụng cú pháp của C và đặc trưng hướng đối tượng của C++. Java là ngôn ngữ vừa biên dịch vừa thông dịch. Đầu tiên mã nguồn được biên dịch thành dạng bytecode. Sau đó được thực thi trên từng loại máy nhờ 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 các nền phần cứng khác nhau.

Ngày nay, Java được sử dụng rộng rãi, không chỉ để viết các ứng dụng trên máy cục bộ hay trên mạng mà còn để xây dựng các trình điều khiển thiết bị di động, ...



2.2. ĐẶC TRƯNG CỦA NGÔN NGỮ JAVA

Ngôn ngữ Java có những đặc trưng cơ bản sau:

- Đơn giản
- Hướng đối tượng
- Độc lập phần cứng và hệ điều hành
- Mạnh mẽ
- Bảo mật
- Phân tán
- Đa luồng
- Linh 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:

- Loại bỏ thao tác con trỏ, thao tác định nghĩa chồng toán tử (operator overloading)...
- Không cho phép đa kế thừa (Multi-inheritance) mà sử dụng các giao diện (interface)
- Không sử dụng lệnh “goto” cũng như file header (.h).

- Loại bỏ cấu trúc “struct” và “union”.

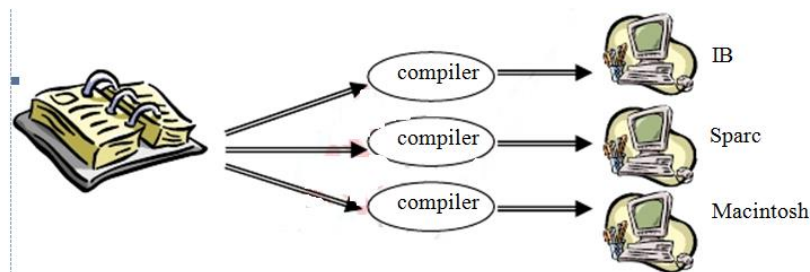
Hướng đối tượng:

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

- Mọi thực thể trong hệ thống đều được coi là một đối tượng, tức là một thể hiện cụ thể của một lớp xác định.
- Tất cả các chương trình đều phải nằm trong một lớp (class) nhất định.
- Không thể dùng Java để viết một chức năng mà không thuộc vào bất kì một lớp nào. Tức là Java không cho phép định nghĩa dữ liệu và hàm tự do trong chương trình.

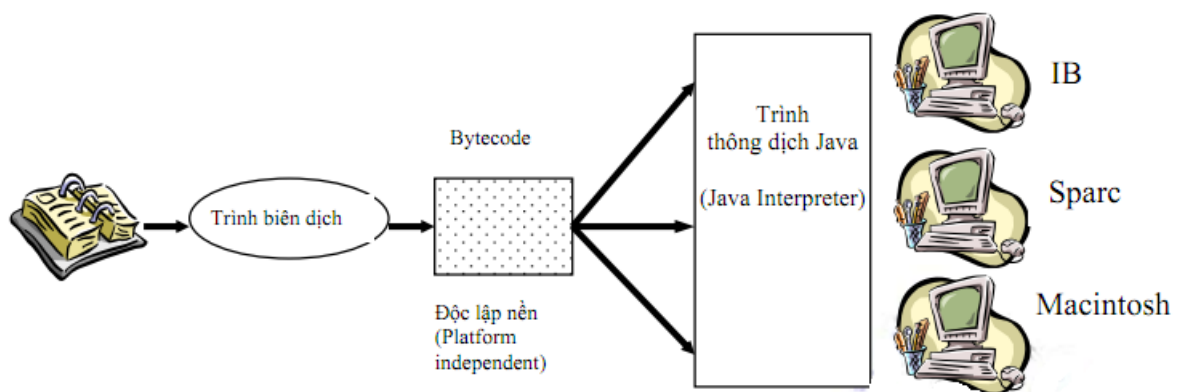
Độc lập phần cứng và hệ điều hành:

Đối với các ngôn ngữ lập trình truyền thống như C/C++, phương pháp biên dịch được thực hiện như hình bên dưới: Với mỗi một nền phần cứng khác nhau, có một trình biên dịch khác nhau để biên dịch mã nguồn chương trình cho phù hợp với nền phần cứng ấy. Do vậy, khi chạy trên một nền phần cứng khác, bắt buộc phải biên dịch lại mã nguồn.



Cách biên dịch truyền thống.

Đối các chương trình viết bằng Java, trình biên dịch Javac sẽ biên dịch mã nguồn thành dạng bytecode. Sau đó, khi chạy chương trình trên các nền phần cứng khác nhau, máy ảo Java dùng trình thông dịch Java để chuyển mã bytecode thành dạng chạy được trên các nền phần cứng tương ứng. Do vậy, khi thay đổi nền phần cứng, không phải biên dịch lại mã nguồn Java. Hình sau minh họa quá trình biên dịch và thông dịch mã nguồn Java.



Biên dịch chương trình Java.

Mạnh mẽ:

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

- Kiểu dữ liệu phải được khai báo tường minh.

- Java không sử dụng con trỏ và các phép toán con trỏ.
- Java kiểm tra việc truy nhập đến mảng, chuỗi khi thực thi để đảm bảo rằng các truy nhập đó không ra ngoài giới hạn kích thước mảng.
- Quá trình cấp phát, giải phóng bộ nhớ cho biến được thực hiện tự động, nhờ dịch vụ thu nhặt những đối tượng không còn sử dụng nữa (garbage collection).
- Cơ chế bẫy lỗi của Java giúp đơn giản hóa quá trình xử lý lỗi và hồi phục sau lỗi.

Bảo mật:

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

- Ở mức thứ nhất, dữ liệu và các phương thức được đóng gói bên trong lớp. Chúng chỉ được truy xuất thông qua các giao diện mà lớp cung cấp.
- Ở mức thứ hai, trình biên dịch kiểm soát để đảm bảo mã là an toàn, và tuân theo các nguyên tắc của Java.
- Mức thứ ba được đảm bảo bởi trình thông dịch. Chúng kiểm tra xem bytecode có đảm bảo các qui tắc an toàn trước khi thực thi.
- Mức thứ tư kiểm soát việc nạp các lớp vào bộ nhớ để giám sát việc vi phạm giới hạn truy xuất trước khi nạp vào hệ thống.

Phân tán:

Java được thiết kế để hỗ trợ các ứng dụng chạy trên mạng bằng các lớp Mạng (java.net). Hơn nữa, Java hỗ trợ nhiều nền chạy khác nhau nên chúng được sử dụng rộng rãi như là công cụ phát triển trên Internet, nơi sử dụng nhiều nền khác nhau.

Đa luồng:

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

Linh động:

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

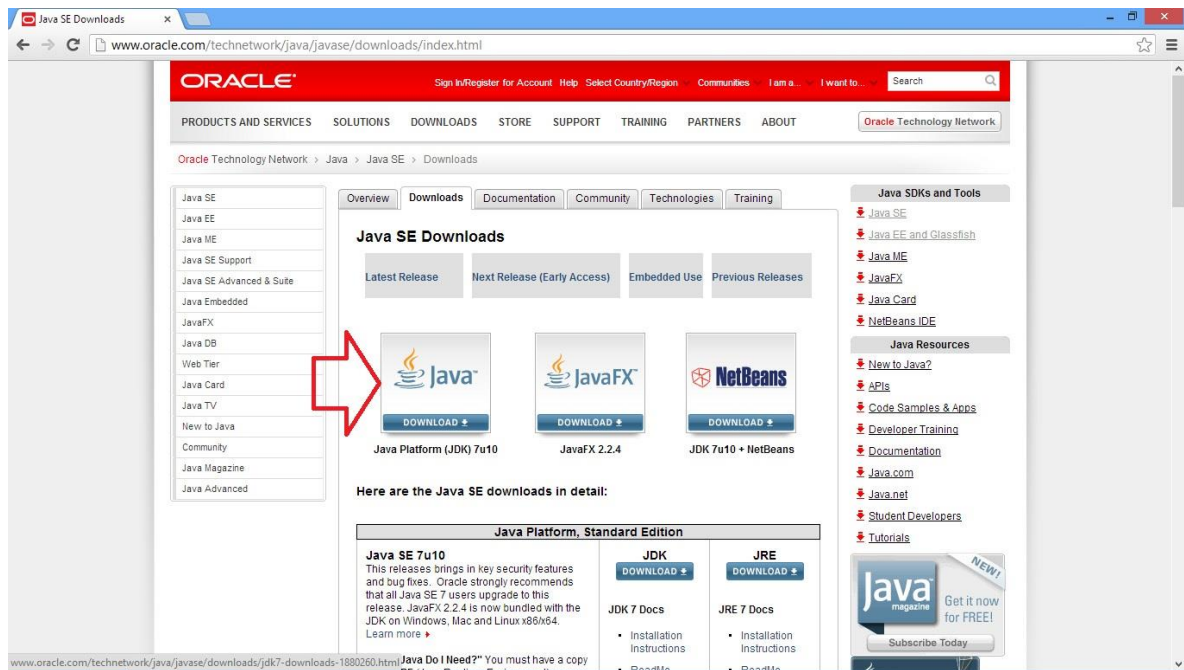
2.3. CÀI ĐẶT JAVA VÀO MÁY TÍNH

JDK (Java Development Kit) là bộ công cụ cho người phát triển ứng dụng bằng ngôn ngữ lập trình Java, là một tập hợp những công cụ phần mềm được phát triển bởi Sun Microsystems. Bộ công cụ này được phát hành miễn phí gồm có trình biên dịch, trình thông dịch, trình giúp sửa lỗi, trình chạy applet và tài liệu nghiên cứu.

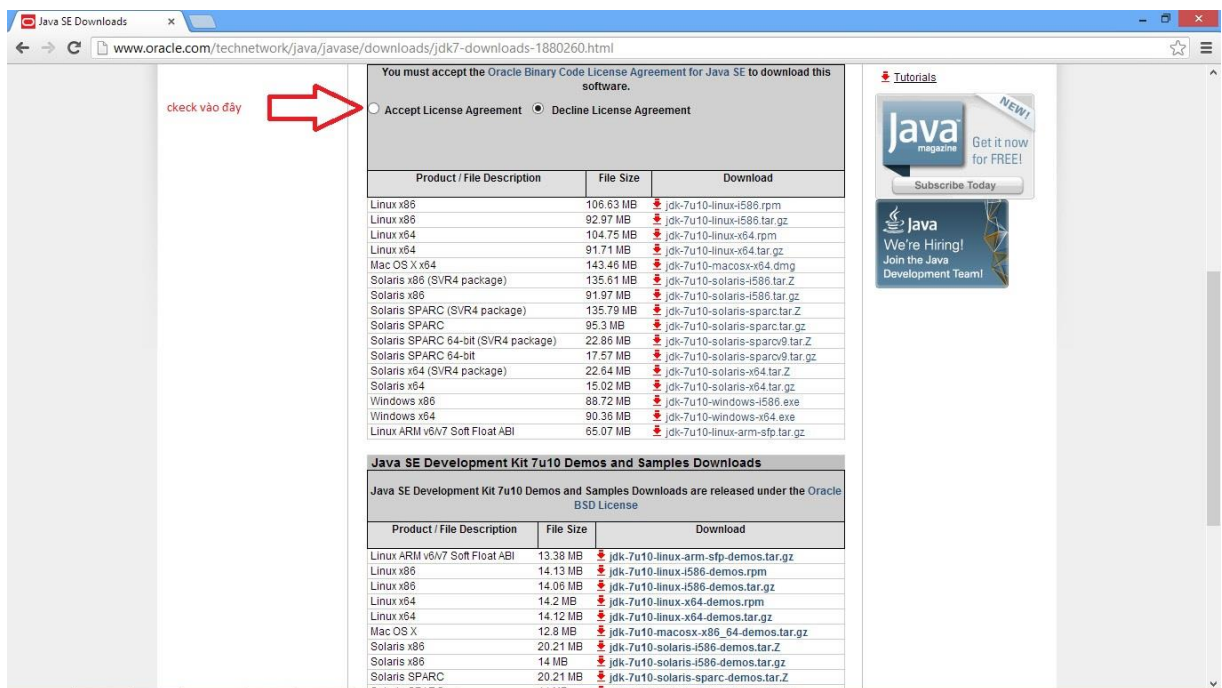
Cài đặt JDK:

Các bạn vào địa chỉ sau để tải bộ cài đặt về:

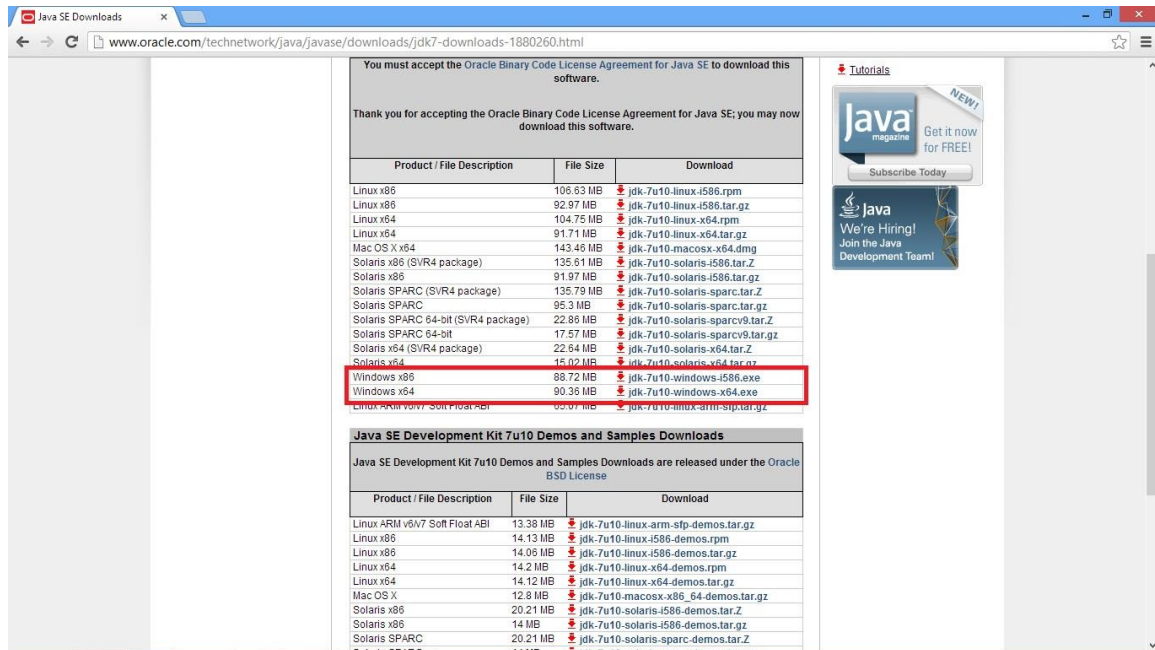
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>



Chọn Accept License Agreement.



Chọn tải phiên bản JDK tương ứng cho loại Windows mình đang dùng là 32bit hay 64bit.



Sau khi tải về thành công, ví dụ tải về phiên bản JDK 1.7 là tập tin jdk-7u25-windows-x64.exe về ổ đĩa cứng máy tính của mình.

Các bạn Click đôi vào tập tin ấy tiến hành cài đặt. Công việc cài đặt của bạn lúc này chỉ là next, next ... finish mà thôi.

Sau khi cài đặt xong các bạn sẽ đến thư mục chứa các chương trình của bộ JDK ở đường dẫn: C:\Program Files (x86)\Java\jdk1.7.0\bin (Đường dẫn này có thể thay đổi tùy theo phiên bản Windows bạn đang dùng hoặc khi các bạn lựa chọn trong quá trình cài đặt)

Các chương trình cơ bản của Java trong thư mục bin này như:

- javac: Biên dịch chương trình nguồn (.java) thành tập tin nhị phân (mã byte code) (.class)
- java: Bộ thông dịch, thực thi tập tin .class với ứng dụng độc lập, chạy chương trình.
- appletviewer: chạy các ứng dụng nhúng applet.

2.4. VIẾT CHƯƠNG TRÌNH JAVA ĐẦU TIÊN

Sau đây chúng ta sẽ cùng viết 1 chương trình java đơn giản hiển thị dòng chữ: “Chao mừng den voi lop hoc Java” và quá trình biên dịch chương trình như thế nào?

Các bạn có thể dùng chương trình notepad của Windows để viết mã nguồn. Mở chương trình notepad lên, viết mã nguồn như sau:

```
class Xinchao
{
    public static void main(String args[])
    {
        System.out.println("Chao mung den voi lop hoc Java");
    }
}
```

Sau khi viết xong mã lệnh các bạn lưu tập tin mã nguồn lại với tên là: Xinchao.java vào thư mục bin của Java đã cài đặt ở trên. (Lưu ý: Tên lớp giống tên của tập tin chương trình)

Tiếp tục, chúng ta biên dịch chương trình Xinchao.java thành Xinchao.class bằng cách như sau:

- Bấm và giữ phím Window, bấm thêm phím R để chạy chương trình Run của Window, gõ cmd vào mục Open, chọn Ok để chạy chương trình DOS trong windows:

```
C:\Users\user>
```

- Chuyển đến đường dẫn thư mục bin của Java bằng cách gõ lệnh: cd C:\Program Files (x86)\java\jdk1.7.0\bin

```
C:\Users\user>cd C:\Program Files (x86)\java\jdk1.7.0\bin
C:\Program Files (x86)\Java\jdk1.7.0\bin>
```

- Gõ lệnh biên dịch: javac Xinchao.java

```
C:\Program Files (x86)\Java\jdk1.7.0\bin>javac Xinchao.java
```

- Quá trình biên dịch sẽ biên dịch được File Xinchao.class ở cùng thư mục bin.
- Chạy chương trình bằng cách gõ vào lệnh: java Xinchao

```
C:\Program Files (x86)\Java\jdk1.7.0\bin>java Xinchao
Chào mừng đến với lớp học Java
```

2.5. CHƯƠNG TRÌNH ECLIPSE

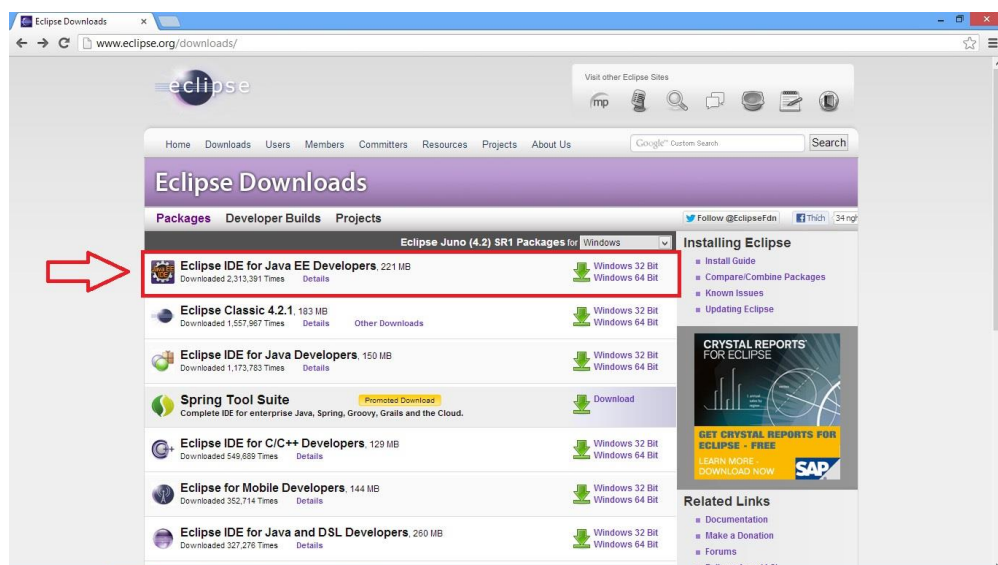
2.5.1. Cài đặt chương trình Eclipse

Eclipse là một phần mềm mã nguồn mở được thiết kế để hỗ trợ các lập trình viên viết các ứng dụng bằng nhiều ngôn ngữ khác nhau trong đó có ngôn ngữ Java.

Eclipse tạo ra một môi trường lập trình đơn giản, trực quan với việc hỗ trợ rất nhiều plugin (phần thêm vào) để các lập trình viên đơn giản và tiện ích hơn trong việc lập trình.

Để cài đặt Eclipse vào máy tính, đầu tiên phải cài đặt bộ biên dịch Java JDK như hướng dẫn ở trên, sau đó các bạn tải bộ nguồn Eclipse về từ Internet và chạy chương trình như sau:

- Các bạn vào địa chỉ: <http://www.eclipse.org/downloads/>
- Sau đó chọn như hình dưới, nhớ chọn theo đúng hệ điều hành mình đang sử dụng nhé:



- Sau khi download về, các bạn chỉ cần giải nén ra chỗ nào đó trên ổ cứng, ví dụ: D:\eclipse

- Vào thư mục eclipse, click đúp vào file eclipse.exe là có thể chạy được luôn.

Nếu vẫn không chạy được thì thiết lập lại biến môi trường như sau:

- Bấm chuột phải vào biểu tượng MyComputer trên màn hình Desktop và chọn “Properties” -> chọn tab “Advanced” -> chọn nút “Environment Variables”

- Tiếp theo chỗ “System variables” ta chọn New để bắt đầu tạo biến môi trường “JAVA_HOME”, bạn sẽ thấy dialog hiện ra để nhập thông tin vào bao gồm: Variable name (bạn gõ vào là “JAVA_HOME”) và Variable value (là thư mục bạn đã cài đặt jdk trước đó, thông thường sẽ nằm ở thư mục “C:\Program Files\Java\jdk1.7.0_25”) sau đó bạn chọn OK.

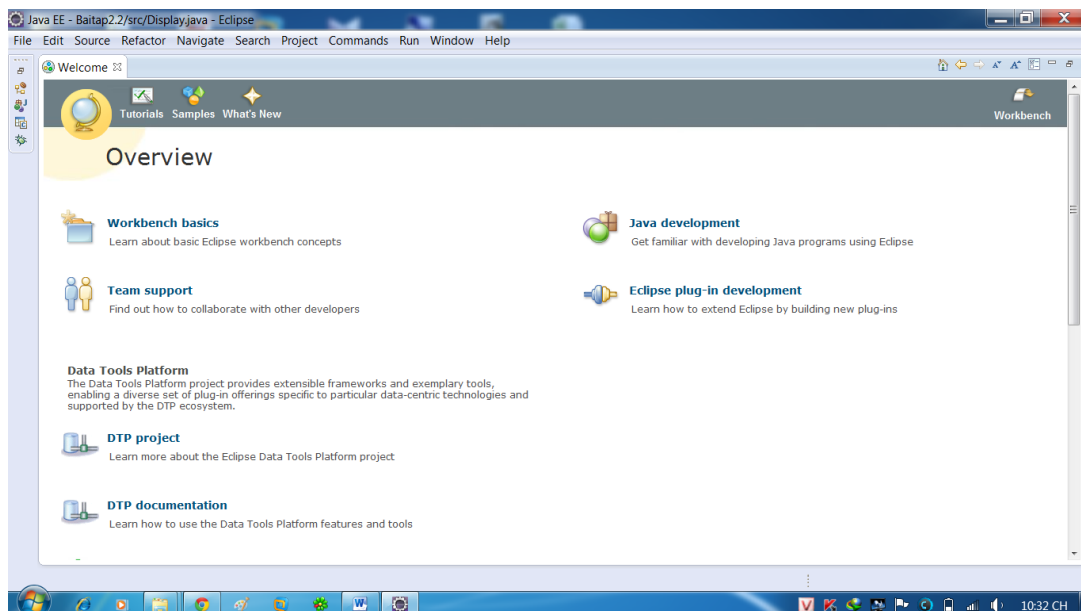
- Tiếp theo bạn cần cập nhật thông tin của biến “JAVA_HOME” vào hệ thống để sử dụng chung cho các ứng dụng cần chạy trên nền máy ảo JVM bằng cách tại dialog của “System variables” bạn kéo thanh scroll xuống và tìm đến “Path” -> chọn nút “Edit” và có một dialog hiện ra để bạn cập nhật thông tin cho “Path”, tại phần “Variable value” bạn di chuyển đến cuối và gõ vào “;%JAVA_HOME%\bin;” (lưu ý là dấu “;” ở đầu biến “JAVA_HOME” sẽ không cần nếu trước đó đã có rồi) -> sau đó bạn chọn các nút “OK” ở các dialog để hoàn tất.

- Để xác nhận lại việc thiết lập biến môi trường có đúng không thì tại màn hình CMD (Terminal) bạn dùng lệnh: echo %JAVA_HOME%, nếu bạn nhận được kết quả là đường dẫn tới thư mục cài đặt jdk là đúng.

- Chạy lại file eclipse.exe để chạy chương trình.

- Nếu vẫn không chạy được thì vào đường dẫn trong thư mục eclipse: eclipse\configuration\org.eclipse.osgi\manager, xóa file: .fileTableLock.

- Chạy lại file eclipse.exe để chạy chương trình. Giao diện chương trình như sau:



- Chúng ta sẽ bắt đầu dùng chương trình eclipse này để viết code cho tất cả các chương trình Java sau này.

Lưu ý: Nếu không tải và cài đặt được JDK và phần mềm eclipse, các bạn có thể liên hệ địa chỉ email: **khuongdx@gmail.com** để copy bộ JDK và phần mềm eclipse.

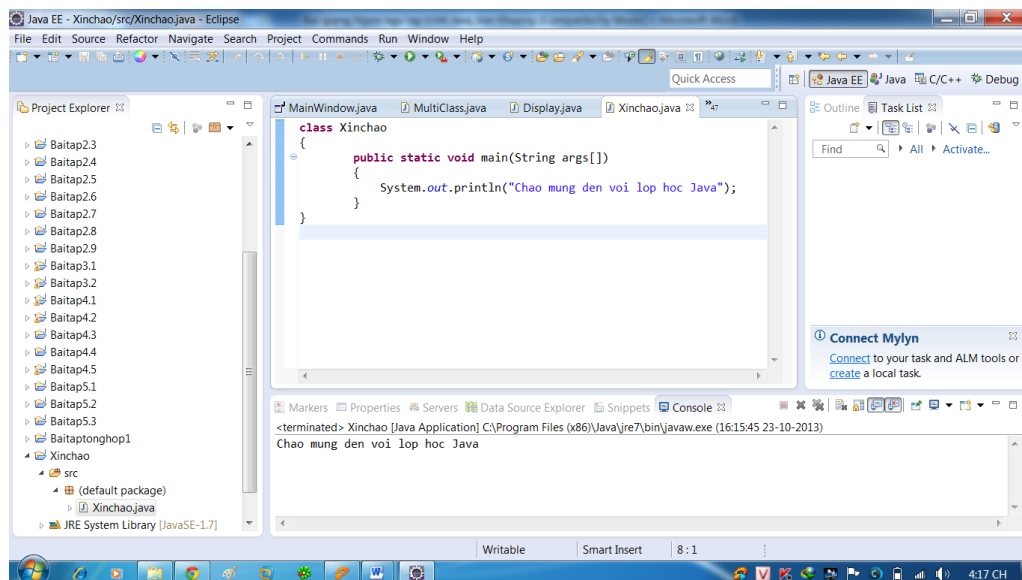
2.5.2. Viết chương trình Java bằng Eclipse

Bây giờ chúng ta viết lại chương trình hiển thị dòng chữ: “Chao mung den voi lop hoc Java” bằng chương trình Eclipse nhé. Các bước như sau:

- Khởi động Eclipse.
- Chọn File / New/ Java Project để tạo dự án mới. Chọn Next
- Gõ tên Xinchao vào mục Project Name. Chọn Next, Chọn Finish.
- Click chuột phải vào dự án: Xinchao phía bên trái cửa sổ làm việc, chọn New / Class để tạo lớp.
- Gõ tên lớp là Xinchao vào mục Name, đánh dấu chọn vào public static void main (String[] args) để tạo phương thức chính của lớp.
- Chọn Finish.
- Viết lệnh cho chương trình như sau:

```
class Xinchao
{
    public static void main(String args[])
    {
        System.out.println("Chao mung den voi lop hoc Java");
    }
}
```

- Biên dịch và chạy chương trình: Nhấn **Ctrl + F11**, chọn OK. Chương trình Eclipse sẽ tự động kết nối đến các chương trình của bộ JDK để biên dịch và chạy chương trình.



- Nếu quá trình biên dịch mà có lỗi xảy ra, các bạn dựa vào thông báo lỗi bên dưới để sửa lỗi và biên dịch lại chương trình.

- Dự án chương trình của chúng ta sẽ lưu vào thư mục ở đường dẫn: C:\Users\user\workspace\Xinchao (Đường dẫn này cũng có thể thay đổi tùy vào thiết lập của từng máy tính).

- Tập tin mã nguồn Xinchao.java lưu ở thư mục **src**, tập tin biên dịch Xinchao.class lưu ở thư mục **bin** trong thư mục **Xinchao**.

2.6. CÁC DẠNG CHƯƠNG TRÌNH ỨNG DỤNG CỦA NGÔN NGỮ JAVA

2.6.1. Chương trình ứng dụng độc lập

Chương trình ứng dụng độc lập là chương trình sau khi dịch có thể chạy trực tiếp được từ dòng lệnh.

Cách tạo ra chương trình ứng dụng độc lập là phải định nghĩa một lớp có chứa hàm main.

2.6.2. Chương trình ứng dụng nhúng *APPLET*

Là loại chương trình mà khi thực thi phải được nhúng vào các ứng dụng khác, trình duyệt Web Browser. Trình appletviewer của Java cũng thực thi loại chương trình này.

2.6.3. Chương trình chạy được cả ứng dụng độc lập và *Web Browser*

Dùng hàm main để chạy độc lập.

Ứng dụng mở rộng, kế thừa từ lớp Applet.

Dùng lớp Frame để tạo ra một khung khi chạy ứng dụng độc lập.

2.7. BÀI TẬP CHƯƠNG 2

Bài tập 2.1: Nêu những đặc trưng của ngôn ngữ lập trình Java.

Bài tập 2.2: Tải và cài đặt JDK vào máy tính.

Bài tập 2.3: Tải và cài đặt Eclipse vào máy tính.

Bài tập 2.4: Viết chương trình trong Eclipse hiển thị:

- Tôi tên là: Nguyễn Văn A.
- Lớp: Công nghệ thông tin.
- Trường: ...

CHƯƠNG 3: CÁC THÀNH PHẦN CƠ SỞ CỦA JAVA

3.1. CÁC PHẦN TỬ CƠ SỞ CỦA JAVA

3.1.1. Định danh/ tên gọi (Identifier)

- Xác định biến, phương thức, đối tượng, lớp, kiểu, giao diện, gói (package)...Đặt tên bằng các ký tự, số, các ký tự đặc biệt (\$,..), không bắt đầu bằng số, độ dài không giới hạn.
- Java phân biệt chữ hoa và chữ thường: ví dụ : hocSinh và hocsinh là 2 định danh khác nhau.
- Một chuẩn qui định về cách đặt tên:
 - Định danh cho lớp, giao diện: Chữ cái đầu mỗi từ viết hoa, ví dụ: HocSinh, KhachHang, InputStream, OutputStream...
 - Định danh cho biến, phương thức, đối tượng: Chữ cái đầu mỗi từ viết hoa trừ chữ cái đầu tiên, ví dụ: hocSinh, khachHang, diChuyen...

3.1.2. Các từ khoá

Ngôn ngữ lập trình đã định danh sẵn các từ để biểu diễn ngôn ngữ, các từ khóa này không được sử dụng cho định danh.

Ví dụ:

- Định nghĩa lớp: class, extends, interface, implements
- Từ khoá cho việc dùng các biến, các lớp: public, private, protected, static, String
- Tổ chức các lớp: package, import
- Các kiểu dữ liệu cơ bản: long, int, short, byte, double, char, boolean
- Giá trị hằng gán cho biến: true, false, this, super, null
- Xử lý ngoại lệ: throw, throws, try, catch, finally
- Tạo lập và kiểm tra đối tượng: new, instanceof
- Các dòng điều khiển chương trình: if, else, case, while, switch, default, break, continue, do, return, for
- Các từ khoá khác: const, goto...

3.1.3. Chú thích (Comment)

- Giải thích trong chương trình, chương trình dịch sẽ bỏ qua:
- Chú thích trên một dòng: dùng dấu //dòng chú thích
- Chú thích trên nhiều dòng: dùng cặp dấu:
/* dòng thứ nhất
dòng thứ hai
....
*/

3.2. CÁC KIỂU DỮ LIỆU NGUYÊN THỦY

Các kiểu dữ liệu cơ bản (nguyên thủy) đã định nghĩa sẵn, các kiểu dữ liệu khác (người dùng định nghĩa, kiểu dữ liệu trừu tượng: mảng, chuỗi, lớp...) được xây dựng trên các kiểu dữ liệu này.

3.2.1. Kiểu nguyên

Gồm kiểu số nguyên và kiểu ký tự:

Kiểu số nguyên:

Khai báo với các kiểu sau:

- byte (8 bits: từ -256 đến 255)
- short (16 bits: từ -32768 đến 32767)
- int (32 bits)
- long (64 bits)

Cách khai báo:

- `int soNguyen=18; //số nguyên 18`
- `int soNguyen=0123; // số nguyên hệ bát phân, System.out.println(soNguyen) cho kết quả 83`
- `int soNguyen=0x12; //số nguyên hệ thập lục phân, System.out.println(soNguyen) cho kết quả 18`

Kiểu ký tự:

Java biểu diễn các ký tự với mã Unicode (dùng 16 bits để biểu diễn ký tự), có 2^{16} ký tự với 128 ký tự đầu trùng với bảng mã ASCII.

Khai báo:

- `char ch='A'; //System.out.println(ch) cho kết quả A`
- `char ch1=65; //truyền theo mã ASCII, System.out.println(ch1) cho kết quả A`
- `char ch2=65535; //truyền theo mã Unicode`
- `char ch3='\u0041' //truyền theo hệ thập lục phân, System.out.println(ch2) cho kết quả A`

Các ký tự điều khiển:

- `'\n'`: bắt đầu một dòng mới
- `'\\'`: chèn dấu \
- `'\"'`: chèn dấu “
- `'\'''`: chèn dấu ‘

3.2.2. Kiểu số thực

Có hai dạng khai báo:

• float (dấu phẩy động, 32 bits): Biểu diễn qua lũy thừa của cơ số 10, dùng E để chỉ lũy thừa 10.

Ví dụ:

$1000=1*10^3=1E3$, $-42000=-0.42*10^5=-0.42E5$

$3.14=314*10^{-2}=314E-2$, $0.023=0.23*10^{-1}=0.23E-1$

• double: Dấu chấm thập phân, 64 bits:

Chú ý: Một hằng ngầm định là kiểu double và không thể gán kiểu double cho kiểu float.

3.2.3. Kiểu boolean

Nhận một trong hai giá trị true hoặc false

Khai báo: `boolean bien=true;`

3.3. KHAI BÁO BIẾN

Trong Java, có 3 kiểu biến trong Java bao gồm:

- Biến local (biến cục bộ)
- Biến instance (biến toàn cục)
- Biến static

Cách khai báo biến trong Java:

Để khai báo biến, ta sử dụng cú pháp cơ bản sau

`kieu_du_lieu bien [= giatri][, bien [= giatri] ...] ;`

Ví dụ:

```
public class Demo {  
    private static final int CONST = 1; // biến static  
    private int a = 2; // biến toàn cục  
  
    public Demo() {  
        int b = 3; //biến cục bộ  
    }  
}
```

3.3.1. Biến cục bộ

Biến cục bộ là biến được khai báo trong một khối lệnh (đó có thể các phương thức, hàm constructor hoặc một block):

- Các biến cục bộ được khởi tạo khi các hàm, hàm tạo hoặc các khối lệnh được tạo ra và sẽ bị hủy bỏ một khi các hàm, hàm tạo hoặc các khối lệnh chứa nó kết thúc.
- Không sử dụng phạm vi truy cập khi khai báo biến cục bộ
- Cần phải khởi tạo giá trị mặc định cho biến cục bộ khi khai báo chúng,
- Biến cục bộ được lưu trong vùng nhớ stack của bộ nhớ.

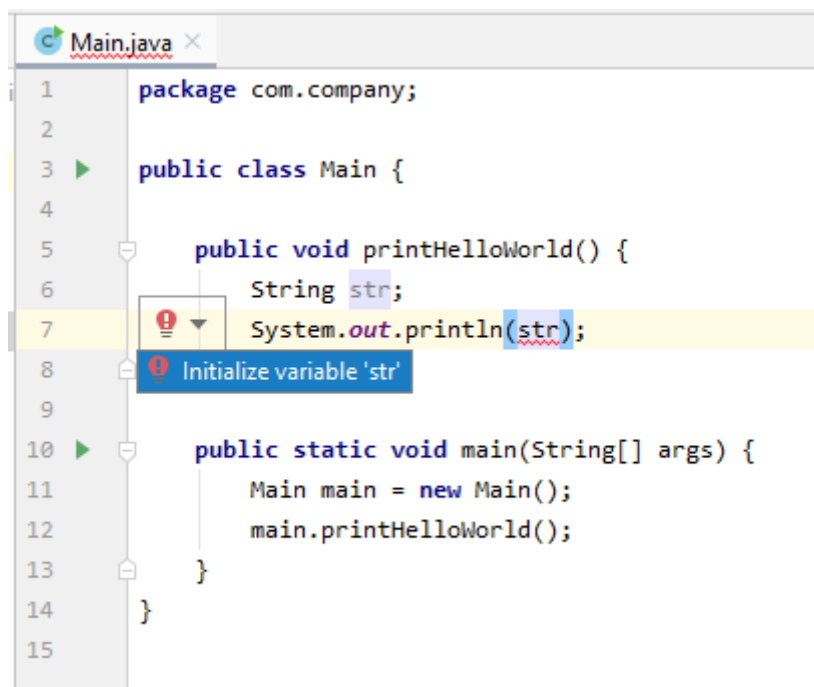
Ví dụ 3.1:

```
public class Main {  
  
    public void printHelloWorld() {  
        String str = "Hello World!";  
        System.out.println(str);  
    }  
  
    public static void main(String[] args) {  
        Main main = new Main();  
        main.printHelloWorld();  
    }  
}
```

Kết quả thu được:

Hello World!

Tuy nhiên cũng trong trường hợp này, nếu ta không khởi tạo giá trị cho biến `str` của phương thức `printHelloWorld()` thì chương trình sẽ ngay lập tức báo lỗi.



3.3.2. Biến toàn cục

Các biến toàn cục hay còn gọi là thuộc tính, được khai báo trong một lớp, nhưng ở bên ngoài một phương thức, constructor hoặc bất kỳ khối lệnh nào:

- Biến toàn cục được tạo khi một đối tượng được tạo bằng cách sử dụng từ khóa `new` và sẽ bị phá hủy khi đối tượng bị phá hủy.
- Biến instance có thể được sử dụng bởi các phương thức, constructor, block, ... Nhưng nó phải được sử dụng thông qua một đối tượng cụ thể.
- Bạn được phép sử dụng "access modifier" khi khai báo biến instance, mặc định là "default".

- Biến instance có giá trị mặc định phụ thuộc vào kiểu dữ liệu của nó. Ví dụ nếu là kiểu int, short, byte thì giá trị mặc định là 0, kiểu double thì là 0.0d, ... Vì vậy, bạn sẽ không cần khởi tạo giá trị cho biến instance trước khi sử dụng.
- Bên trong class mà bạn khai báo biến instance, bạn có thể gọi nó trực tiếp bằng tên khi sử dụng ở khắp nơi bên trong class đó.
- Biến toàn cục được lưu trữ trong bộ nhớ heap.

Các đặc điểm của biến toàn cục có liên quan mật thiết đến phần lập trình hướng đối tượng, tạm thời chúng ta chưa đi sâu, trong phần sau của khóa học sẽ tìm hiểu chi tiết và cụ thể hơn.

Ví dụ 3.2:

```
public class Employee {
    private String name; //biến toàn cục

    public Employee (String name){
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public static void main(String[] args){
        Employee bkitEmployee = new Employee ("Huy Nguyen");
        System.out.println("Tên: "+bkitEmployee.getName());
    }
}
```

Kết quả thu được:

Tên: Huy Nguyen

3.3.3. Biến static

Các biến static được khai báo với từ khóa static trong một lớp, nhưng ở bên ngoài một phương thức, constructor hoặc một khối:

- Sẽ chỉ có một bản sao của mỗi biến class cho mỗi lớp, bất chấp việc bao nhiêu đối tượng được tạo từ nó.
- Các biến static được lưu giữ trong bộ nhớ static.
- Các biến static được tạo khi chương trình bắt đầu và bị hủy khi chương trình kết thúc.
- Biến static được truy cập thông qua tên của class chứa nó, với cú pháp: TenClass.tenBien.
- Trong class, các phương thức sử dụng biến static bằng cách gọi tên của nó khi phương thức đó cũng được khai báo với từ khóa "static" và ngược lại.

Ví dụ 3.3:

```
public class Employee {  
    // biến static khai báo tên  
    public static String name = "Huy Nguyen";  
  
    public static void main(String args[]) {  
        // Gọi trực tiếp  
        System.out.println("Ten : " + name);  
  
        // Gọi thông qua tên class  
        System.out.println("Ten : " + Employee.name);  
    }  
}
```

Kết quả thu được:

Ten : Huy Nguyen

Ten : Huy Nguyen

3.4. CẤU TRÚC TẬP TIN CHƯƠNG TRÌNH JAVA

//Phần 1: Định nghĩa gói – Tùy chọn package <tên gói>
//Phần 2: Nhập gói thư viện cần sử dụng - không có hoặc có nhiều import java.io.*;
//Phần 3: Định nghĩa các lớp và các giao diện - không có hoặc có nhiều public class <lớp 1> {...} class <lớp 2> {...} ... interface <giao diện 1> {...} interface <giao diện 2> {...}

- Trong một tệp chương trình java cho phép định nghĩa nhiều lớp hoặc nhiều giao diện.
- Tên tập tin chương trình java trùng lớp tên lớp được khai báo là public và lớp này chứa hàm main() nếu là ứng dụng độc lập. Tập tin chương trình nguồn có phần mở rộng .java
- Khi biên dịch thì mỗi lớp hoặc giao diện trong tập tin sẽ tạo ra tập tin byte code có tên trùng với tên lớp và giao diện, có phần mở rộng là .class

Hàm main() của chương trình:

Cú pháp: public static void main(String args[])

Ví dụ:

```
//Khai báo thư viện  
import java.io.*;  
//Khai báo lớp HamMain
```

```
public class HamMain
{
    //Khai bao ham main
    public static void main(String args[])
    {
        //Hien thi dong chu ra man hinh
        System.out.println("Day la mot chuong trinh Java");
    }
}
```

- Lưu tập tin chương trình bằng tên: HamMain.java
- Một ứng dụng độc lập luôn bắt đầu bằng hàm main()
- Các khai báo trong hàm main():
 - public: Các thành viên đều truy được từ hàm này
 - static: Đây là một hàm tĩnh, truy cập được mà không cần tạo ra đối tượng
 - void: Hàm không trả về giá trị
 - Đối số của hàm main(String args[]): là một mảng các xâu ký tự, nhận được từ đối số dòng lệnh. Khi thực thi chương trình ta có thể chuyển đối số từ dòng lệnh vào cho hàm main xử lý, cú pháp: java <tên tập tin> <đối số 1> <đối số 2>...

3.5. CÁC PHÉP TOÁN TRONG JAVA (TOÁN TỬ)

3.5.1. Các phép toán số học

3.5.1.1. Phép toán một ngôi

Phép toán thực hiện trên một đối số (toán hạng):

Tên gọi	Phép toán	Ví dụ	Kết quả
Tăng	++	int a=5; ++a; //hoặc a++, cả hai đều tăng giá trị của a, a=a+1 //Chú ý trong phép gán int i=100, j; j=i++ //j mang giá trị của i trước khi tăng i j=++i //j mang giá trị của i sau khi tăng i	//a=6 //j=100,i=101 //j=101,i=101
Giảm	--	int b=5; --b; //hoặc b--, cả hai đều giảm giá trị của b, b=b-1 //Chú ý trong phép gán int i=100, j ; j=i-- //j mang giá trị của i trước khi giảm i j=--i //j mang giá trị của i sau khi giảm i	//b=4 //j=101, i=99 //j=99, i=99
Lấy số đối	-	int c= 6; c= -c	//c= -6

3.5.1.2. Phép toán hai ngôi

Phép toán thực hiện trên nhiều toán hạng

Thực hiện trên kiểu dữ liệu số (số nguyên, số thực)

Tên gọi	Phép toán	Ví dụ	Kết quả	Chú ý
Cộng	+	int a=5, b=4, c; c=a+b;	//c=9	Phép chia và lấy số dư cũng được áp dụng cho kiểu số thực float a=9f,b=4f,thuong, soDu; thuong=a/b; soDu=a%b; System.out.println(thuong);//2.25 System.out.println(soDu);//1.0
Trừ	-			
Nhân	*			
Chia	/	int a=9, b=4,c; c=a/b;	//c=2	
Modulo (lấy số dư)	%	c=a%b	//c=1	

3.5.1.3. Phép toán gán

Cú pháp: <biến> <toán tử>=<biểu thức>

Các toán tử dùng trong biểu thức: +, -, *, /, %

Thay đổi giá trị của biến hiện thời với một lượng của <biểu thức> theo toán tử

Phép toán	Ví dụ	Kết quả
=	a=6;	
+=	a+=3; //a=a+3	//9
-=	a-=3; //a=a-3	//3
=	a=3; //a=a*3	//18
/=	a/=3 ;//a=a/3	//2
%=	a%=3; //a=a%3	//0

3.5.2. Các phép toán lô-gic

Các phép toán lô-gic được áp dụng trên các toán hạng là các mệnh đề:

Phép toán	Tên gọi	Ví dụ	Kết quả
&&	và	boolean va =(6==6) (9<2);	//true
	hoặc	System.out.println("ket qua "+va);	
!	phủ định		

Mệnh đề biểu diễn bằng các phép toán quan hệ:

Phép toán quan hệ	Tên gọi	Ví dụ mệnh đề	Kết quả
==	bằng	9==9	True
>=	lớn hơn hoặc bằng	9>4	True
<=	nhỏ hơn hoặc bằng	5<=5	True
>	lớn hơn	3>9	False
<	nhỏ hơn	5<2	False
!=	không bằng	0!=1	True

3.5.3. Thứ tự ưu tiên của các phép toán

STT	Tên gọi	Các phép toán	Qui tắc kết hợp thực hiện
1	Phép toán 1 ngôi hậu tố (postfix)	exp++ exp--	Thực hiện từ trái qua phải
2	Phép toán 1 ngôi tiền tố (prefix)	++exp --exp	Thực hiện từ phải qua trái
3	Loại phép nhân	* / %	Thực hiện từ trái qua phải
4	Loại phép cộng	+ -	Thực hiện từ trái qua phải
5	Phép toán quan hệ	< <= > >=	Thực hiện từ trái qua phải
6	Phép so sánh đẳng thức	== !=	Thực hiện từ trái qua phải
7	Phép và (AND)	&	Thực hiện từ trái qua phải
8	Phép hoặc (OR)		Thực hiện từ trái qua phải
9	Các phép gán	= += -= *= /= %=	Thực hiện từ phải qua trái

3.6. CÁC QUI TẮC CHUYỂN ĐỔI KIỂU

3.6.1. Chuyển đổi kiểu dữ liệu

Trong một biểu thức:

- Toán tử và toán hạng để cho kết quả là một giá trị tương ứng với một kiểu dữ liệu
- Phép gán giá trị của biểu thức cho một biến

Trình biên dịch của java sẽ kiểm tra sự tương thích giữa các kiểu khi gán giá trị trong biểu thức:

Ví dụ:

```
int c;
```

```
float a=4.5f, b=1.2f;
```

```
c=a+b ;    // thông báo lỗi vì a+b kết quả là kiểu float và c là kiểu int
```

3.6.2. Ép kiểu

Chuyển đổi kết quả của biểu thức (kiểu rộng hơn) thành một kiểu dữ liệu khác (hẹp hơn)

Cú pháp: <(kiểu mới)><biểu thức>

Ví dụ:

```
int soNguyen=1.5f;                //thông báo lỗi
```

```
int soNguyen=(int)1.5f            //soNguyen=1
```

```
float soThuc=3.4d;                //thông báo lỗi
```

```
float soThuc=(float)3.4d;
```

```
//kiểu char có hai cách khai báo, độ rộng 16 bits
```

```
char ch='A';
```

```
char ch=65;
```

```
int soNguyen=65;
```

```
char ch=soNguyen;                 //thông báo lỗi
```

```
char ch=char(soNguyen);           //chuyển đổi kiểu
```

Chú ý: Ép kiểu làm mất thông tin về độ rộng cũng như độ chính xác

3.7. CÁC HÀM TOÁN HỌC

- Các hàm toán học nằm trong gói java.lang.Math
- Hàm toán học là các hàm tĩnh (không tạo ra đối tượng) nằm trong lớp Math
- Cách truy cập hàm <tên lớp>. <tên hàm>[(tham số)]

Một số hàm toán học cơ bản trong Java:

- Hàm abs(đối số): Cho giá trị tuyệt đối của một số

```
int soNguyen=-5;
```

- ```
System.out.println("gia tri tuyet doi "+Math.abs(soNguyen)); //5
double soThuc=-5.4d;
System.out.println("gia tri tuyet doi "+Math.abs(soNguyen)); //5.4
```
- Hàm round(đối số): trả lại số nguyên gần nhất của đối số
 

```
float so=3.7f
Math.round(so) //4
float so=-3.2f
Math.round(so) //-3
```
  - Hàm ceil(x) : x là số thực (float, double), trả về giá trị nhỏ nhất kiểu double mà không nhỏ hơn đối số và bằng một số nguyên.
 

```
System.out.println(Math.ceil(3.14)); //4.0
System.out.println(Math.ceil(-3.14)); //-3.0
```
  - Hàm floor(x) : x là số thực (float, double), trả về giá trị lớn nhất kiểu double mà không lớn hơn đối số và bằng một số nguyên.
 

```
System.out.println(Math.floor(3.14)); //3.0
System.out.println(Math.floor(-3.14)); //-4.0
```
  - Hàm pow(a,n): trả lại giá trị  $a^n$  với giá trị là số thực
 

```
int a=2,n=3
Math.pow(a,n) //8.0
```
  - Hàm exp(x): trả về giá trị của hàm ex
 

```
int x=1
Math.exp(x) //cho giá trị số e=2.71828
```
  - Hàm log(x) // cho lô-ga-rit cơ số e của x:  $\ln(x)$ 

```
int x=1;
Math.log(x)//0.0
Math.log(Math.exp(x))//1.0 ;lne=1
```
  - Hàm sqrt(x): cho giá trị căn bậc 2 của x
 

```
int x=16
Math.sqrt(x)//4.0
```
  - Hàm sin(x), cos(x), tan(x) : cho giá trị sin, cos, tan của một góc với đơn vị đo là radian, 900 có đơn vị radian là  $\pi/2$
  - Hằng PI: Math.PI // cho giá trị của số  $\pi$ 

```
Math.sin(Math.PI/2)//1.0 sin45°=1
Math.cos(Math.PI/6)// 0.866 cos30°= $\sqrt{3}/2$
```



- Hàm random(): trả về một số ngẫu nhiên nằm trong đoạn 0.0 đến 1.0 kiểu double  
double soNgauNhiem;  
soNgauNhiem=Math.random();  
System.out.println("ket qua la"+soNgauNhiem);

### 3.8. CÂU LỆNH ĐIỀU KHIỂN Rẽ NHÁNH

#### 3.8.1. Câu lệnh if / else

Câu lệnh if - else có cách sử dụng linh hoạt trong nhiều hoàn cảnh khác nhau, giúp rút ngắn thời gian code và tăng hiệu năng của chương trình.

##### Sử dụng mệnh đề if:

Mệnh đề If được sử dụng khi kiểm tra giá trị dạng boolean của điều kiện. Khối lệnh sau if được thực thi nếu giá trị của điều kiện là True

Ví dụ:

```
public class Main {

 public static void main(String args[]) {
 int number = 5;
 if (number < 10) {
 System.out.println("Số đã cho nhỏ hơn 10");
 }
 }
}
```

##### Sử dụng mệnh đề if - else:

Mệnh đề If-Else cũng kiểm tra giá trị dạng boolean của điều kiện. Nếu giá trị điều kiện là True thì chỉ có khối lệnh sau If sẽ được thực hiện, nếu là False thì chỉ có khối lệnh sau Else được thực hiện.

Ví dụ 3.4:

```
public class Main {

 public static void main(String args[]) {
 int number = 10;
 if (number < 10) {
 System.out.println("Số đã cho nhỏ hơn 10");
 } else {
 System.out.println("Số đã cho lớn hơn hoặc bằng 10");
 }
 }
}
```

Kết quả chương trình: Số đã cho lớn hơn hoặc bằng 10

##### Sử dụng mệnh đề if-else-if:

Sau khi kiểm tra điều kiện if thứ nhất, nếu trả về True thì khối lệnh sau if thứ nhất được thực hiện. Còn nếu trả về false, lại xét tiếp đến điều kiện if thứ 2, và cứ thế cho đến khi trả về true hoặc không còn điều kiện if nào nữa.

Việc sử dụng mệnh đề if-else-if sẽ tránh được trường hợp kiểm tra thừa điều kiện.

*Ví dụ 3.5:*

```
public class Main {

 public static void main(String args[]) {
 int number = 2;
 if (number < 5) {
 System.out.println("Số đã cho nhỏ hơn 5");
 System.out.println("Số đã cho nhỏ hơn 10");
 } else if (number < 10) {
 System.out.println("Số đã cho không nhỏ hơn 5");
 System.out.println("Số đã cho nhỏ 10");
 }
 }
}
```

Trong trường hợp này, ta muốn kiểm tra xem biến number có nhỏ hơn 5 và nhỏ hơn 10 không. Thay vì viết 2 lệnh if riêng rẽ, ta sử dụng cấu trúc trên sẽ giảm được 1 lần kiểm tra với 10 nếu number nhỏ hơn 5.

Kết quả chương trình:

Số đã cho nhỏ hơn 5

Số đã cho nhỏ hơn 10

### 3.8.2. Câu lệnh switch - case

Mệnh đề switch-case trong java được sử dụng để thực thi 1 hoặc nhiều khối lệnh từ nhiều điều kiện.

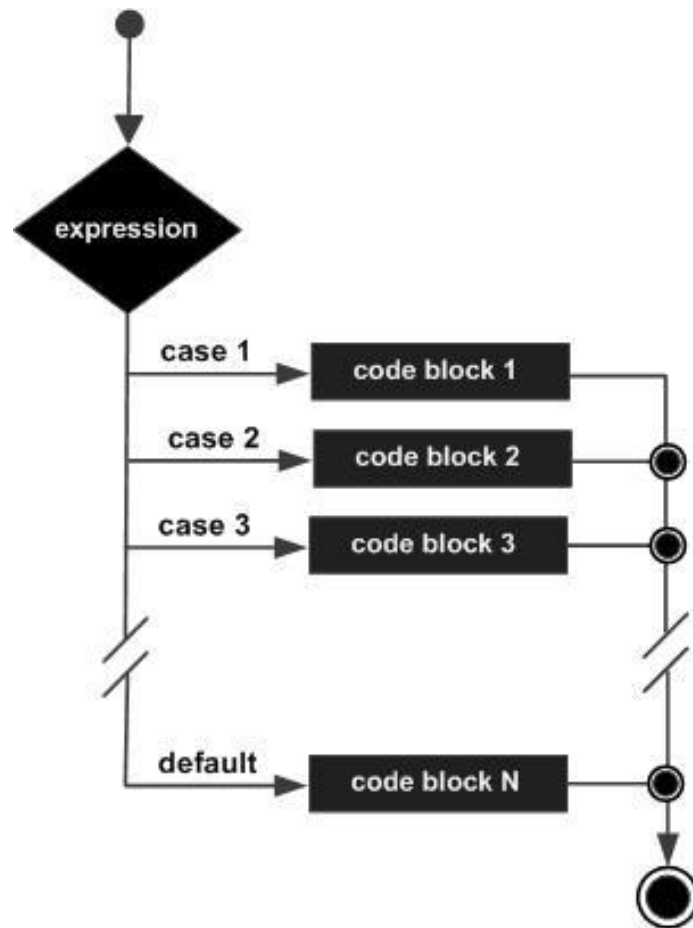
Cú pháp:

```
switch(expression) {
 case value :
 // Statements
 break; // optional

 case value :
 // Statements
 break; // optional

 // You can have any number of case statements.
 default : // Optional
 // Statements
}
```

Luồng hoạt động của switch - case



Ví dụ 3.6:

```
public class Main {

 public static void main(String args[]) {
 int number = 2;
 switch (number) {
 case 1:
 System.out.println("Number = 1");
 break;
 case 2:
 System.out.println("Number = 2");
 break;
 case 3:
 System.out.println("Number = 3");
 break;
 default:
 System.out.println("Number > 3");
 }
 }
}
```

Kết quả thu được:

Number = 2

Khi không sử dụng từ khóa 'break' trong mệnh đề switch-case, có nghĩa là các khối lệnh sau case có giá trị phù hợp sẽ được thực thi.

Ví dụ 3.7:

```
public class Main {

 public static void main(String args[]) {
 int number = 2;
 switch (number) {
 case 1:
 System.out.println("Number = 1");
 case 2:
 System.out.println("Number = 2");
 case 3:
 System.out.println("Number = 3");
 default:
 System.out.println("Number > 3");
 }
 }
}
```

Kết quả thu được:

Number = 2  
Number = 3  
Number > 3

## 3.9. CÂU LỆNH ĐIỀU KHIỂN LẶP

### 3.9.1. 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 vòng 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

Ví dụ 3.8:

```
public class Main {

 public static void main(String args[]) {
 int i=0;
 while(i<5){
 System.out.println("Vị trí thứ "+i);
 i++;
 }
 }
}
```

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

Kết quả thu về:

Vị trí thứ 0

Vị trí thứ 1

Vị trí thứ 2

Vị trí thứ 3

Vị trí thứ 4

### 3.9.2. Vòng lặp do...while

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

Cú pháp:

```
do{
 action statement
 :
 :
}
```

```
while(condition)
```

- condition: Biểu thức Boolean, nó trả về giá trị True hoặc False.

- action statement: Các câu lệnh được thực hiện 1 lần sau đó kiểm tra điều kiện, nếu đúng thì thực hiện tiếp và ngược lại.

Ví dụ 3.9:

```
public class Main {

 public static void main(String args[]) {
 int i = 0;
 do {
 System.out.println("Vị trí thứ : " + i);
 i++;
 } while (i < 5);
 }
}
```

Kết quả thu được:

Vị trí thứ 0

Vị trí thứ 1

Vị trí thứ 2

Vị trí thứ 3

Vị trí thứ 4

### 3.9.3. Vòng lặp for

Một vòng lặp for cho phép bạn viết một vòng lặp một cách hiệu quả để cần thực thi một số lần lặp cụ thể nào đó.

Một vòng lặp for là hữu ích khi bạn biết số lần một tác vụ được lặp lại.

Cú pháp:

```
for(initialization; condition; update){
 action statement
 :
 :
}
```

- initialization: được thực thi đầu tiên, và chỉ một lần. Bước này cho phép bạn khai báo và khởi tạo bất kỳ biến điều khiển vòng lặp. Bạn không được yêu cầu đặt một lệnh ở đây, miễn là một dấu chấm phẩy xuất hiện.

- condition: Nếu nó là true, phần thân vòng lặp được thực thi. Nếu nó là false, phần thân vòng lặp không thực thi và luồng điều khiển nhảy tới lệnh tiếp theo sau vòng lặp for.

- update: Thay đổi (tăng hoặc giảm) giá trị của biến sau mỗi lần lặp  
action statement: Các câu lệnh được thực hiện trong mỗi lần lặp

*Ví dụ 3.10:*

```
public class Main {

 public static void main(String args[]) {
 for (int i = 0; i < 5; i++) {
 System.out.println("Vị trí thứ: " + i);
 }
 }
}
```

Kết quả thu được:

Vị trí thứ 0

Vị trí thứ 1

Vị trí thứ 2

Vị trí thứ 3

Vị trí thứ 4

## 3.10. HÀM TRONG JAVA

### 3.10.1. Giới thiệu về hàm

Một method trong java là một tập các câu lệnh được nhóm lại với nhau để thực hiện một hành động cụ thể.

**Cú pháp hàm:**

```
modifier returnType nameOfMethod (Parameter List) {
 // method body
}
```



Giải thích:

- Modifiers (tạm dịch là phạm vi sửa đổi và truy cập): public hoặc static
- returnType: kiểu dữ liệu trả về
- nameOfMethod: tên của hàm(method)
- Parameter là các tham số đầu vào của hàm(có thể có nhiều tham số với nhiều kiểu dữ liệu khác nhau)
- // body: là các mã code bên trong hàm

Ví dụ: Viết hàm tính tổng 2 chữ số a và b

```
public static int sum(int a, int b) {
 return a + b;
}
```

Khi chúng ta đã viết xong một hàm rồi làm sao để chúng ta có thể sử dụng chúng đây ta! Trước tiên chúng ta sẽ có 2 loại hàm, mỗi loại sẽ có cách gọi khác nhau:

- Hàm có trả về kết quả
- Hàm không trả về kết quả

### 3.10.2. Hàm có trả về kết quả

Đối với hàm có kết quả trả về, chúng ta cần dùng từ khoá return để trả về kết quả mà nó đã tính toán được.

Khai báo biến có kiểu dữ liệu tương ứng với kết quả trả về của hàm để nhận giá trị trả về.

Như ví dụ tìm sum() của chúng ta, nhiệm vụ của nó là tính tổng của 2 số nguyên a, b nhập vào.

Ví dụ 3.11:

```
public class Main {

 public static int sum(int a, int b) {
 return a + b;
 }

 public static void main(String[] args) {

 int sum = sum(2,5);
 System.out.println(sum);

 }
}
```

Chúng ta sẽ dùng biến có kiểu dữ liệu tương ứng để nhận kết quả trả về từ hàm. Hàm sum() trả về int chúng ta dùng biến int sum để nhận kết quả.

### 3.10.3. Hàm không trả về kết quả

Đối với loại hàm này chúng ta chỉ cần gọi để sử dụng.

Ví dụ:

```
public static void printHello() {
 System.out.println("Hello");
}
```

Hàm `printHello()` là một hàm không có kiểu trả về nên được thay thế thành `void`. Từ khoá `void` cho chúng ta biết là method đó sẽ không có kết quả trả về.

#### 3.10.4. Truyền tham số khi gọi hàm(method)

Khi bạn gọi đến một hàm đã được định và nó có danh sách các thông số nhất định. Thì khi gọi các bạn sẽ phải truyền đúng thứ tự và kiểu dữ liệu cho từng tham số. Nếu sai sót trong quá trình này có thể dẫn đến biên dịch lỗi hoặc là kết quả không đúng như mong muốn.

Ví dụ 3.12:

```
public class Main {

 public static void swap(int a, int b) {
 int tam = a;
 a = b;
 b = tam;
 System.out.println("Ket qua swap: " + "a = " + a + " b= " + b);
 }

 public static void main(String[] args) {

 int a = 5, b = 3;
 swap(a, b);
 System.out.println("a sau khi swap: " + a);
 System.out.println("b sau khi swap: " + b);

 }
}
```

Output:

Ket qua swap: a = 3 b= 5

a sau khi swap: 5

b sau khi swap: 3

Nhìn vào kết quả trên, rõ ràng hàm `swap` đã được `a` thành `b`, `b` thành `a`, sao trong hàm `main` giá trị lại như cũ nhỉ? Đó chính là truyền thông số theo giá trị đó các bạn, các tham số truyền vào hàm sẽ chỉ nhận giá trị mà không tham chiếu đến 2 biến `a`, `b` của chúng ta. Thế nên đừng có viết hàm `swap()` như trên nữa nhe, hồi mình chuyển từ C/C++ sang java cũng đã cố gắng viết hàm `swap` mà không thành.

### 3.11. BÀI TẬP CHƯƠNG 3

**Bài tập 3.1:** Viết chương trình Java nhập 2 số thực `a`, `b` từ bàn phím và hiển thị tổng, hiệu, tích, thương của 2 số đó ra màn hình.

*Bài giải:*

- Khởi động Eclipse.
- Chọn File / New/ Java Project để tạo dự án mới. Chọn Next
- Gõ tên Baitap3.1 vào mục Project Name. Chọn Next, Chọn Finish.
- Click chuột phải vào dự án: Baitap3.1 phía bên trái cửa sổ làm việc, chọn New / Class để tạo lớp.

- Gõ tên lớp là Baitap3\_1 vào mục Name, đánh dấu chọn vào public static void main (String[] args) để tạo phương thức chính của lớp.
- Chọn Finish.
- Viết lệnh cho chương trình như sau:

```
//Khai bao cac thu vien
import java.util.Scanner;

public class Baitap3_1 {
 //Khai bao doi tuong input cua lop Scanner nhan du lieu nhap vao
 private static Scanner input;

 //Chương trình chính
 public static void main(String[] args) {
 //Khoi tao doi tuong input
 input = new Scanner(System.in);
 //Nhap gia tri cho bien a, b kieu so thuc tu ban phim
 System.out.print("Nhap a: ");
 float a = input.nextFloat();
 System.out.print("Nhap b: ");
 float b = input.nextFloat();
 //Hien thi ket qua tinh toan ra man hinh
 System.out.println("Ket qua bai toan a + b la: " + (a+b));
 System.out.println("Ket qua bai toan a - b la: " + (a-b));
 System.out.println("Ket qua bai toan a * b la: " + (a*b));
 System.out.println("Ket qua bai toan a / b la: " + (a/b));
 }
}
```

- Để biên dịch và chạy chương trình, chọn Ctrl + F11, chọn lớp để chạy, chọn OK.
- *Lưu ý: Chương trình trên vẫn chưa hoàn chỉnh vì sẽ gặp lỗi nếu thực hiện phép chia mà b=0, các bạn hãy xử lý lỗi trên?*

### **Bài tập 3.2:** Viết chương trình Java giải phương trình bậc 2. (Sử dụng lệnh if...else)

*Bài giải:*

- Khởi động Eclipse.
- Chọn File / New/ Java Project để tạo dự án mới. Chọn Next
- Gõ tên Baitap3.2 vào mục Project Name. Chọn Next, Chọn Finish.
- Click chuột phải vào dự án: Baitap3.2 phía bên trái cửa sổ làm việc, chọn New / Class để tạo lớp.
- Gõ tên lớp là Baitap3\_2 vào mục Name, đánh dấu chọn vào public static void main (String[] args) để tạo phương thức chính của lớp.
- Chọn Finish.
- Viết lệnh cho chương trình như sau:

```
//Khai bao cac thu vien
import java.util.Scanner;

public class Baitap3_2 {
```

```
//Khai bao doi tuong input cua lop Scanner nhan du lieu nhap vao
private static Scanner input;

//Chương trình chính
public static void main(String[] args){
 //Khoi tao doi tuong input
 input = new Scanner(System.in);
 //Nhap gia tri cho he so a, b, c kieu so thuc tu ban phim
 System.out.print("Nhập a: ");
 float a = input.nextFloat();
 System.out.print("Nhập b: ");
 float b = input.nextFloat();
 System.out.print("Nhập c: ");
 float c = input.nextFloat();
 //Tinh gia tri delta
 float delta=b*b-4*a*c;
 //Xet cac truong hop cua delta de cho ra nghiem
 if (delta<0)
 System.out.println("Phương trình vô nghiệm.");
 else if (delta==0)
 System.out.println("Phương trình có nghiệm kép: "+(-b/(2*a)));
 else{
 System.out.println("Phương trình có 2 nghiệm:");
 System.out.println("X1 = "+(-b+Math.sqrt(delta))/(2*a));
 System.out.println("X2 = "+(-b-Math.sqrt(delta))/(2*a));
 }
}
```

**Bài tập 3.3:** Viết chương trình Java nhập số nguyên n, và hiển thị tổng các số chẵn từ 0 đến n. (Dùng vòng lặp for)

*Bài giải:*

- Khởi động Eclipse.
- Chọn File / New/ Java Project để tạo dự án mới. Chọn Next
- Gõ tên Baitap3.3 vào mục Project Name. Chọn Next, Chọn Finish.
- Click chuột phải vào dự án: Baitap3.3 phía bên trái cửa sổ làm việc, chọn New / Class để tạo lớp.
- Gõ tên lớp là Baitap3\_3 vào mục Name, đánh dấu chọn vào public static void main (String[] args) để tạo phương thức chính của lớp.
- Chọn Finish.
- Viết lệnh cho chương trình như sau:

```
//Khai bao cac thu vien
import java.util.Scanner;

public class Baitap3_3 {
 //Khai bao doi tuong input cua lop Scanner nhan du lieu nhap vao
 private static Scanner input;

 //Chương trình chính
```

```
public static void main(String[] args){
 //Khởi tạo đối tượng input
 input = new Scanner(System.in);
 //Nhập giá trị cho biến n
 System.out.print("Nhập n: ");
 int n = input.nextInt();
 //Khởi tạo biến tổng
 int tong=0;
 //Thực hiện vòng lặp để tính tổng các số chẵn
 for(int i=0;i<=n;i++)
 {
 if(i%2==0)
 tong=tong +i;
 }
 System.out.println("Tổng các số chẵn từ 0 đến n là: "+tong);
}
```

- Biên dịch và chạy chương trình: Ctrl + F11.
- *Lưu ý:* để đơn giản chương trình các bạn có thể dùng vòng lặp for với bước lặp là 2:

```
for(int i=0;i<=n;i+=2);
```

**Bài tập 3.4:** Viết chương trình Java trò chơi Oẳn tù tì, tương ứng các lựa chọn (Búa – B; Kéo – K; Giấy – G) như hình bên dưới: (Sử dụng lệnh if...else, switch)

```
<terminated> Baitap3_4 [Java Application] C:\Program Files (x86)\Java\jre7\bin\jav
Lựa chọn của người chơi 1 (Búa-1, Kéo-2, Giấy-3): 1
Lựa chọn của người chơi 2 (Búa-1, Kéo-2, Giấy-3): 2
Kết quả: Người 1 thắng
```

*Bài giải:*

- Khởi động Eclipse.
- Chọn File / New/ Java Project để tạo dự án mới. Chọn Next
- Gõ tên Baitap3.4 vào mục Project Name. Chọn Next, Chọn Finish.
- Click chuột phải vào dự án: Baitap3.4 phía bên trái cửa sổ làm việc, chọn New / Class để tạo lớp.
- Gõ tên lớp là Baitap3\_4 vào mục Name, đánh dấu chọn vào public static void main (String[] args) để tạo phương thức chính của lớp.
- Chọn Finish.
- Viết lệnh cho chương trình như sau:

```
//Khởi tạo các thư viện
import java.util.Scanner;

public class Baitap3_4 {
 //Khởi tạo đối tượng input của lớp Scanner nhận dữ liệu nhập vào
 private static Scanner input;

 //Chương trình chính
 public static void main(String[] args){
```

```
//Khoi tao doi tuong input
input = new Scanner(System.in);
//Nhap gia tri cho bien n
System.out.print("Lựa chọn của người chơi 1 (Búa-1, Kéo-2, Giấy-3): ");
int luachon1 = input.nextInt();
System.out.print("Lựa chọn của người chơi 2 (Búa-1, Kéo-2, Giấy-3): ");
int luachon2 = input.nextInt();
//Kiem tra gia tri cua luachon1, luachon2 de cho ra ket qua tro choi
if(luachon1==1)
{
 switch(luachon2)
 {
 case 1: System.out.print("Ket qua: Hoa nhau");
 break;
 case 2: System.out.print("Ket qua: Nguoi 1 thang");
 break;
 case 3: System.out.print("Ket qua: Nguoi 2 thang");
 break;
 }
}
else if(luachon1==2)
{
 switch(luachon2)
 {
 case 1: System.out.print("Ket qua: Nguoi 2 thang");
 break;
 case 2: System.out.print("Ket qua: Hoa nhau");
 break;
 case 3: System.out.print("Ket qua: Nguoi 1 thang");
 break;
 }
}
else if(luachon1==3)
{
 switch(luachon2)
 {
 case 1: System.out.print("Ket qua: Nguoi 1 thang");
 break;
 case 2: System.out.print("Ket qua: Nguoi 2 thang");
 break;
 case 3: System.out.print("Ket qua: Hoa nhau");
 break;
 }
}
}
```

**Bài tập 3.5:** Viết chương trình Java nhập tên và mật khẩu (tối đa nhập 3 lần). Nếu nhập đúng tên là “khuong”, mật khẩu là “123” thì hiển thị các số từ 100 đến 200 mà chia hết cho 3, 5 và tính tổng các số đó.

```
<terminated> Baitap3_5 [Java Application] C:\Program I
Nhập tên người dùng: khuong
Nhập mật khẩu: 123
105 120 135 150 165 180 195
Tổng các số thỏa điều kiện là: 1050
```



*Bài giải:*

- Khởi động Eclipse.
- Chọn File / New/ Java Project để tạo dự án mới. Chọn Next
- Gõ tên Baitap3.5 vào mục Project Name. Chọn Next, Chọn Finish.
- Click chuột phải vào dự án: Baitap3.5 phía bên trái cửa sổ làm việc, chọn New / Class để tạo lớp.
- Gõ tên lớp là Baitap3\_5 vào mục Name, đánh dấu chọn vào public static void main (String[] args) để tạo phương thức chính của lớp.
- Chọn Finish.
- Viết lệnh cho chương trình như sau:

```
//Khai bao cac thu vien
import java.util.Scanner;

public class Baitap3_5 {
 //Khai bao doi tuong input cua lop Scanner nhan du lieu nhap vao
 private static Scanner input;

 //Chương trình chính
 public static void main(String[] args){
 //Khoi tao doi tuong input
 input = new Scanner(System.in);
 //Khai bao 2 bien ten, matkhau
 String ten=null, matkhau=null;
 int dem=1;
 //Thuc hien nhap ten, matkhau toi da 3 lan
 while(dem<=3)
 {
 System.out.print("Nhập tên người dùng: ");
 ten = input.next();
 System.out.print("Nhập mật khẩu: ");
 matkhau = input.next();
 dem=dem+1;
 //Neu ten va matkhau nhap vao dung thi thoat vong lap
 if((ten.equals("khuong")) && (matkhau.equals("123")))
 break;
 }
 //Khai bao bien tong
 int tong=0;
 //Thuc hien vong lap forkiem tra tung so tu 100 den 200
 for (int i=100; i<=200; i++)
 {
 //Neu i thoa dieu kien thi hien thi va cong i vao bien tong
 if((i%3==0)&&(i%5==0))
 {
 System.out.print(i + " ");
 tong=tong+i;
 }
 }
 System.out.println("\nTổng các số thỏa điều kiện là: " + tong);
 }
}
```

**Bài tập 3.6:** Viết chương trình Java đếm xem trong các số nguyên  $\leq 100$ :

- Có bao nhiêu số chia hết cho 5.
- Chia cho 5 dư 1
- Chia cho 5 dư 2
- Chia cho 5 dư 3

*Bài giải:*

- Khởi động Eclipse. Chọn File / New/ Java Project để tạo dự án mới. Chọn Next
- Gõ tên Baitap3.6 vào mục Project Name. Chọn Next, Chọn Finish.
- Click chuột phải vào dự án: Baitap3.6 phía bên trái cửa sổ làm việc, chọn New / Class để tạo lớp.
- Gõ tên lớp là Baitap3\_6 vào mục Name, đánh dấu chọn vào public static void main (String[] args) để tạo phương thức chính của lớp.
- Chọn Finish.
- Viết lệnh cho chương trình như sau:

```
public class Baitap3_6 {

 public static void main(String[] args) {
 //Khai bao cac bien de dem
 int dem0=0, dem1=0, dem2=0, dem3=0;
 //Duyet tat ca cac so tu 0 den 100
 for (int i=0; i<=100; i++)
 {
 //Kiem tra gia tri i chia lay du cho 5
 switch (i%5)
 {
 case 0:
 dem0++;
 break;
 case 1:
 dem1++;
 break;
 case 2:
 dem2++;
 break;
 case 3:
 dem3++;
 break;
 }
 }
 //Hien thi ket qua
 System.out.println("Số các số chia hết cho 5 là: " + dem0);
 System.out.println("Số các số chia cho 5 dư 1 là: " + dem1);
 System.out.println("Số các số chia hết cho 5 dư 2 là: " + dem2);
 System.out.println("Số các số chia hết cho 5 dư 3 là: " + dem3);
 }
}
```

**Bài tập 3.7:** Viết chương trình Java giải phương trình bậc nhất:  $ax + b = 0$ .

**Bài tập 3.8:** Viết chương trình nhập một số nguyên dương  $n$  và thực hiện các yêu cầu sau:

- Kiểm tra số  $n$  có phải là số nguyên tố không?
- Liệt kê các ước số của  $n$ . Có bao nhiêu ước số.
- Liệt kê các ước số là nguyên tố của  $n$ .

**Bài tập 3.9:** Viết chương trình Java nhập 1 số nguyên  $N$  và tính tổng sau: (Dùng hàm, nên xây dựng 2 hàm)

$$S = 1 + 1/(1+2!) + 1/(1+2!+3!) + \dots + 1/(1+2!+3!+\dots+N!)$$

**Bài tập 3.10:** Viết chương trình nhập 1 số nguyên từ bàn phím và kiểm tra số đó có phải là số nguyên tố hay không? (Ví dụ số nguyên tố: 1, 3, 5, 7, 11,...)

**Bài tập 3.11:** Viết chương trình nhập 1 số nguyên từ bàn phím và kiểm tra số đó có phải là số chính phương hay không? (Ví dụ số chính phương:  $4=2 \times 2$ )

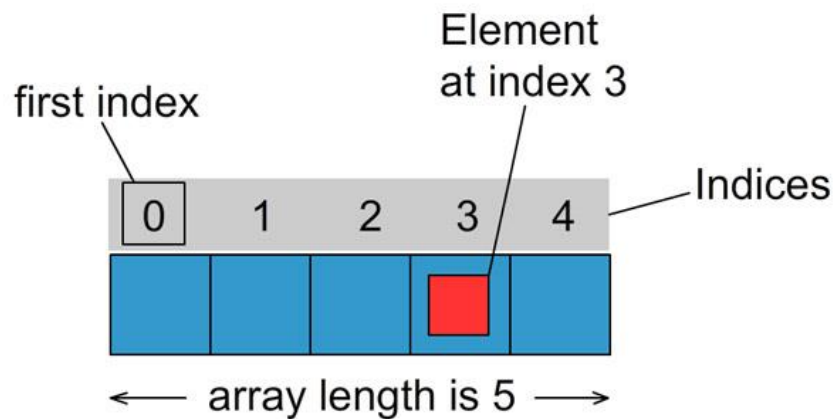
**Bài tập 3.12:** Viết chương trình nhập 2 số nguyên khác nhau từ bàn phím và hiển thị ước chung lớn nhất của 2 số đó?

## CHƯƠNG 4: MẢNG TRONG JAVA

### 4.1. KHÁI NIỆM VỀ MẢNG

Trong Java, mảng (array) là một tập hợp các phần tử có cùng kiểu dữ liệu, có địa chỉ tiếp nhau trên bộ nhớ (memory). Mảng có số phần tử cố định và bạn không thể thay đổi kích thước của nó. Có 2 loại mảng sau:

**Mảng 1 chiều:** Các phần tử của mảng được đánh chỉ số (index), bắt đầu từ chỉ số 0. Bạn có thể truy cập vào các phần tử của nó thông qua chỉ số.



Hãy xem một hình ảnh mảng 5 phần tử, chứa các số int.

|   | Array_Values |
|---|--------------|
| 0 | 10           |
| 1 | 14           |
| 2 | 36           |
| 3 | 27           |
| 4 | 43           |

**Mảng đa chiều:** Ngoài các mảng 1 chiều, đôi khi bạn cũng phải làm việc với mảng 2 chiều hoặc nhiều chiều.

Một mảng 2 chiều, các phần tử của nó sẽ được đánh hai chỉ số, chỉ số hàng và chỉ số cột. Dưới đây là hình ảnh một mảng 2 chiều.

| Index | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
|-------|----|----|----|----|----|----|----|----|----|----|
| 0     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| 1     | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 2     | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 3     | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 4     | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |

## 4.2. MỘT SỐ THAO THÁC CƠ BẢN TRÊN MẢNG

### 4.2.1. Thao tác trên mảng 1 chiều

#### 4.2.1.1. Khai báo mảng một chiều trong Java

Mảng một chiều là một tập hợp của nhiều phần tử có kiểu dữ liệu giống nhau.

Để khai báo mảng, chúng ta cần phải xác định trước **3 thông tin cần thiết** sau:

- Kiểu dữ liệu của mảng.
- Tên của mảng.
- Số lượng các phần tử (hay kích thước) của mảng.

**Câu lệnh khai báo mảng sẽ có 2 dạng như sau:**

Dạng 1:

```
[Kiểu_dữ_liệu] tên_mảng[];
```

Dạng 2:

```
[Kiểu_dữ_liệu][] tên_mảng;
```

trong đó, [Kiểu\_dữ\_liệu] mô tả kiểu của mỗi phần tử thuộc mảng (*như int, char, double, String,...*), tên\_mảng là tên của mảng và quy tắc đặt tên phải tuân theo quy tắc đặt tên biến trong Java.

**Ví dụ:** `int[] a;` : khai báo mảng tên a và có kiểu dữ liệu là int.

**Cấp phát bộ nhớ cho mảng:**

Tương tự như chuỗi, bản chất của mảng là 1 đối tượng, vì vậy mảng cần phải được cấp phát bộ nhớ trước khi sử dụng. Để cấp phát bộ nhớ cho mảng thì chúng ta có 2 cách như sau:

**Cách 1:**

```
[Kiểu_dữ_liệu] tên_mảng[] = new [Kiểu_dữ_liệu] [Số_phần_tử_của_mảng];
```

**Cách 2:**

```
[Kiểu_dữ_liệu][] tên_mảng = new [Kiểu_dữ_liệu] [Số_phần_tử_của_mảng];
```

trong đó, [Số\_phần\_tử\_của\_mảng] chỉ ra số lượng phần tử tối đa mà mảng có thể lưu trữ, giá trị này phải là một số nguyên dương.

Ngoài ra, Java còn cho phép chúng ta vừa có thể khai báo mảng và vừa khởi tạo giá trị cho mảng, ví dụ:

```
int[] a = new int[] {2, 10, 4, 8, 5};
```

: khai báo mảng một chiều có tên là a, kiểu dữ liệu là int và mảng này chứa 5 phần tử có giá trị lần lượt là 2, 10, 4, 8, 5.

#### 4.2.1.2. Truy xuất các phần tử của mảng 1 chiều

Đối với mảng thì chúng ta có thể truy xuất các phần tử của mảng thông qua các **chỉ số của phần tử đó**. Cú pháp như sau:

```
Tên_mảng[Chỉ_số_phần_tử];
```

trong đó, [Chỉ\_số\_phần\_tử] là số thứ tự của các phần tử trong mảng và bắt đầu từ 0. Như vậy, mảng có n phần tử thì các phần tử của nó có chỉ số lần lượt là 0, 1, 2,..., n - 1.

Ví dụ 4.1: Chúng ta có đoạn chương trình sau:

```
public static void main(String[] args) {
 // Khai báo và khởi tạo giá trị ban đầu cho mảng
 char[] kyTu = new char[] {'a', 'b', 'c', 'd', 'e'};

 // hiển thị ký tự tại vị trí thứ 2 trong mảng
 System.out.println("Ký tự tại vị trí thứ 2 trong mảng là " + kyTu[2]);
}
```

Kết quả sau khi biên dịch chương trình:



Console

<terminated> DuyetMang [Java Application] C:\Program Files\Java  
Ký tự tại vị trí thứ 2 trong mảng là c

#### 4.2.1.3. Nhập xuất các phần tử cho mảng 1 chiều

Chương trình dưới đây sẽ minh họa cách nhập các phần tử cho mảng một chiều từ bàn phím và sau đó hiển thị các phần tử đó ra màn hình.

Ví dụ 4.2:

```
public static void main(String[] args) {
 int size; // kích thước của mảng
 Scanner scanner = new Scanner(System.in);

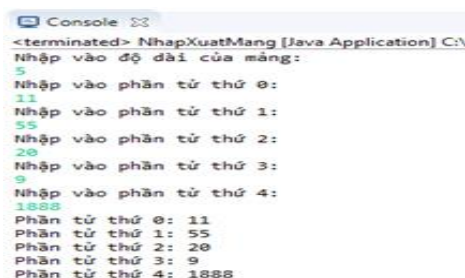
 System.out.println("Nhập vào độ dài của mảng: ");
 size = scanner.nextInt();

 // khai báo và cấp phát bộ nhớ cho mảng
 // mảng này có tên là array và kích thước = size
 int[] array = new int[size];

 // array.length: trả về kích thước của mảng
 // vòng lặp này sẽ duyệt i từ 0 đến chiều dài của mảng - 1
 for (int i = 0; i < size; i++) {
 System.out.println("Nhập vào phần tử thứ " + i + ": ");
 array[i] = scanner.nextInt(); // nhập giá trị cho phần tử
 }

 // hiển thị giá trị các phần tử trong mảng
 for (int i = 0; i < size; i++) {
 System.out.println("Phần tử thứ " + i + ": " + array[i]);
 }
}
```

Kết quả sau khi biên dịch chương trình:



Console

<terminated> NhapXuatMang [Java Application] C:\  
Nhập vào độ dài của mảng:  
5  
Nhập vào phần tử thứ 0:  
11  
Nhập vào phần tử thứ 1:  
55  
Nhập vào phần tử thứ 2:  
20  
Nhập vào phần tử thứ 3:  
9  
Nhập vào phần tử thứ 4:  
1888  
Phần tử thứ 0: 11  
Phần tử thứ 1: 55  
Phần tử thứ 2: 20  
Phần tử thứ 3: 9  
Phần tử thứ 4: 1888



#### 4.2.1.4. Một số thao tác đối với mảng 1 chiều

Gán giá trị cho phần tử của mảng:

Cho 2 mảng array1 và array2 là các mảng có cùng kiểu dữ liệu. Giả sử các phần tử trong mảng array1 đã có giá trị, khi đó chúng ta có thể gán giá trị của một phần tử trong array1 cho một phần tử trong mảng array2 như sau:

Ví dụ 4.3:

```
public static void main(String[] args) {
 // khai báo mảng array1 và array2
 int[] array1 = {2, 10, 3, 9, 8};
 int array2[] = new int[5];

 // gán giá trị của phần tử thứ 2 trong mảng array1
 // cho phần tử thứ 3 trong mảng array2
 array2[3] = array1[2];

 System.out.println("Giá trị của phần tử thứ 3 trong mảng array2 = " +
 array2[3]);
}
```

Kết quả sau khi biên dịch chương trình:



#### 4.2.2. Thao tác trên mảng 2 chiều

##### 4.2.2.1. Khai báo mảng một 2 trong Java

Như đã nói trong bài trước, mảng hai chiều là mảng có 2 chỉ số để lưu trữ các giá trị (*chẳng hạn giá trị của một bảng có m dòng, n cột*).

**Cú pháp khai báo mảng:**

Tương tự như khai báo mảng 1 chiều, cú pháp khai báo mảng 2 chiều có 2 dạng như sau:

Dạng 1:

```
[Kiểu_dữ_liệu] Tên_mảng[][];
```

Dạng 2:

```
[Kiểu_dữ_liệu] [][] Tên_mảng;
```

trong đó: [Kiểu\_dữ\_liệu] mô tả kiểu của mỗi phần tử thuộc mảng (*như int, char, double, String,...*), tên\_mảng là tên của mảng và quy tắc đặt tên phải tuân theo quy tắc đặt tên biến trong Java.

**Ví dụ:** `int a[][];` khai báo mảng hai chiều a có kiểu dữ liệu là int.

**Cấp phát bộ nhớ cho mảng:**

Để cấp phát bộ nhớ cho mảng 2 chiều thì chúng ta sử dụng từ khóa new, trong đó [Số\_dòng], [Số\_cột]: là hai số nguyên dương chỉ ra số lượng dòng và số lượng cột của mảng hai chiều và trong Java có 2 cách để cấp phát bộ nhớ như sau:

### Cách 1:

```
[Kiểu_dữ_liệu] Tên_mảng[][] = new [Kiểu_dữ_liệu] [Số_dòng][Số_cột];
```

Ví dụ: khai báo và cấp phát bộ nhớ cho mảng number có 2 dòng, 3 cột:

```
int number[][] = new int[2][3];
```

### Cách 2:

```
[Kiểu_dữ_liệu][][] Tên_mảng = new [Kiểu_dữ_liệu] [Số_dòng][Số_cột];
```

Ví dụ: khai báo và cấp phát bộ nhớ cho mảng A có 3 dòng, 5 cột:

```
String[][] A = new String[3][5];
```

Khi trình biên dịch gặp lệnh trên thì nó sẽ cấp phát vùng nhớ để chứa mảng hai chiều có 3 dòng, 5 cột với **số phần tử trong mảng = số dòng \* số cột = 15**. Hình ảnh minh họa của mảng hai chiều trên như là một bảng gồm có các dòng và các cột như sau:

|        | Cột 0   | Cột 1   | Cột 2   | Cột 3   | Cột 4   |
|--------|---------|---------|---------|---------|---------|
| Dòng 0 | A[0][0] | A[0][1] | A[0][2] | A[0][3] | A[0][4] |
| Dòng 1 | A[1][0] | A[1][1] | A[1][2] | A[1][3] | A[1][4] |
| Dòng 2 | A[2][0] | A[2][1] | A[2][2] | A[2][3] | A[2][4] |

Bản chất của mảng 2 chiều là mỗi dòng của nó chính là một mảng một chiều. Ví dụ: với mảng hai chiều a có 3 dòng, 5 cột, mỗi phần tử của mảng có kiểu int thì a được xem như mảng một chiều có 3 phần tử, mỗi phần tử này là một mảng một chiều có 5 phần tử.

Ngoài ra, Java còn cho phép chúng ta vừa có thể khai báo mảng và vừa khởi tạo giá trị cho mảng. Ví dụ để khai báo mảng một chiều có tên là diem, kiểu dữ liệu là int và mảng này chứa 6 phần tử có giá trị lần lượt là 1, 2, 3, 4, 5, 6 thì chúng ta làm như sau:

```
// khai báo một mảng 2 chiều có 3 dòng và 2 cột
int diem[][] = {{1, 2}, {3, 4}, {5, 6}};
```

Bảng dưới đây minh họa mảng hai chiều trên:

| Dòng | Cột |   |
|------|-----|---|
|      | 0   | 1 |
| 0    | 1   | 2 |
| 1    | 3   | 4 |
| 2    | 5   | 6 |

#### 4.2.2.2. Truy xuất các phần tử của mảng 2 chiều

Mỗi phần tử của mảng 2 chiều được truy xuất thông qua **tên mảng cùng với chỉ số dòng và chỉ số cột của phần tử đó**. Tương tự như mảng một chiều, nếu một mảng hai chiều có m dòng và n cột thì chỉ số của dòng sẽ chạy từ 0, 1, 2,..., m - 1 và chỉ số của cột sẽ chạy từ 0, 1, 2,..., n - 1.

### Cú pháp như sau:

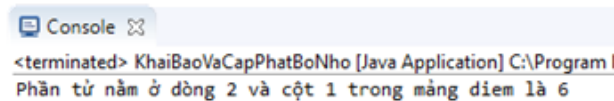
Tên\_mảng[Chỉ\_số\_dòng][Chỉ\_số\_cột]

*Ví dụ 4.4:* Để truy cập đến phần tử nằm ở dòng 2, cột 1 của mảng diem được khai báo ở trên thì chúng ta làm như sau:

```
public static void main(String[] args) {
 // khai báo một mảng 2 chiều có 3 dòng và 2 cột
 int diem[][] = {{1, 2}, {3, 4}, {5, 6}};

 System.out.println("Phần tử nằm ở dòng 2 và cột 1 trong mảng diem là " +
diem[2][1]);
}
```

Nhìn vào bảng minh họa bên trên thì chúng ta nhận thấy phần tử nằm ở dòng 2 và cột 1 trong mảng diem là 6. Kết quả biên dịch chương trình cũng cho chúng ta thấy được điều đó:



```
<terminated> KhaiBaoVaCapPhatBoNho [Java Application] C:\Program I
Phần tử nằm ở dòng 2 và cột 1 trong mảng diem là 6
```

#### 4.2.2.3. Nhập xuất các phần tử cho mảng 2 chiều

Chương trình dưới đây sẽ minh họa cách nhập các phần tử cho mảng hai chiều từ bàn phím và sau đó hiển thị các phần tử đó ra màn hình.

*Ví dụ 4.5:*

```
public static void main(String[] args) {
 // khai báo số dòng và số cột cho mảng
 int soDong, soCot;

 Scanner scanner = new Scanner(System.in);

 System.out.println("Nhập vào số dòng của mảng: ");
 soDong = scanner.nextInt();
 System.out.println("Nhập vào số cột của mảng: ");
 soCot = scanner.nextInt();

 // khai báo và cấp phát bộ nhớ cho mảng
 int[][] A = new int[soDong][soCot];

 // Để nhập giá trị các phần tử cho mảng
 // chúng ta sẽ sử dụng 2 vòng lặp for
 // vòng lặp for bên ngoài sẽ duyệt i từ 0 đến soDong - 1
 // và vòng lặp for bên trong sẽ duyệt j từ 0 đến soCot - 1
 // mỗi lần như vậy thì sẽ nhập vào phần tử tại vị trí i, j
 for (int i = 0; i < soDong; i++) {
 for (int j = 0; j < soCot; j++) {
 System.out.print("Nhập phần tử thứ [" + i + ", " + j + "]: ");
 A[i][j] = scanner.nextInt();
 }
 }

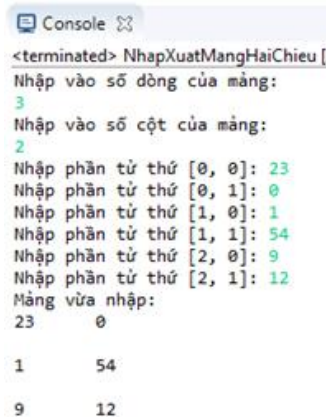
 // hiển thị các phần tử trong mảng vừa nhập
 // chúng ta cũng sử dụng 2 vòng lặp for như khi nhập
```

```

System.out.println("Mảng vừa nhập: ");
for (int i = 0; i < soDong; i++) {
 for (int j = 0; j < soCot; j++) {
 System.out.print(A[i][j] + "\t");
 }
 // sau khi viết xong 1 dòng thì xuống hàng
 System.out.println("\n");
}
}

```

Kết quả sau khi biên dịch chương trình:



```

Console
<terminated> NhapXuatMangHaiChieu [
Nhập vào số dòng của mảng:
3
Nhập vào số cột của mảng:
2
Nhập phần tử thứ [0, 0]: 23
Nhập phần tử thứ [0, 1]: 0
Nhập phần tử thứ [1, 0]: 1
Nhập phần tử thứ [1, 1]: 54
Nhập phần tử thứ [2, 0]: 9
Nhập phần tử thứ [2, 1]: 12
Mảng vừa nhập:
23 0
1 54
9 12

```

#### 4.2.2.4. Một số thao tác đối với mảng 2 chiều

Ví dụ 4.6:

Viết chương trình thực hiện các công việc sau:

- Nhập m, n là số dòng và 2 số cột của 2 ma trận 2 chiều A và B từ bàn phím.
- Nhập giá trị cho các phần tử cho 2 ma trận này.
- Tính và in ra màn hình ma trận C là tổng của 2 ma trận này.
- Yêu cầu kỹ thuật: Kiểm tra số dòng, số cột nhập vào không được nhỏ hơn 1.

```

public static void main(String[] args) {
 int m; // số dòng của ma trận
 int n; // số cột của ma trận
 Scanner scanner = new Scanner(System.in);

 do {
 System.out.println("Nhập vào số dòng của ma trận:");
 m = scanner.nextInt();
 System.out.println("Nhập vào số cột của ma trận:");
 n = scanner.nextInt();
 } while (m < 1 || n < 1);

 // khai báo 2 ma trận A và B có m dòng và n cột
 int A[][] = new int[m][n];
 int B[][] = new int[m][n];

 // ma trận tổng C
 int C[][] = new int[m][n];

 System.out.println("Nhập các phần tử cho ma trận A:");
}

```

```
for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 System.out.print("A[" + i + "," + j + "] = ");
 A[i][j] = scanner.nextInt();
 }
}

System.out.println("Nhập các phần tử cho ma trận B:");
for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 System.out.print("B[" + i + "," + j + "] = ");
 B[i][j] = scanner.nextInt();
 }
}

System.out.println("Ma trận A:");
for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 System.out.print(A[i][j] + "\t");
 }
 System.out.println("\n");
}

System.out.println("Ma trận B:");
for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 System.out.print(B[i][j] + "\t");
 }
 System.out.println("\n");
}

// Để tính tổng hai ma trận
// ta sẽ sử dụng 2 vòng lặp for
// để duyệt i từ 0 đến m và j từ 0 đến n
// sau đó tính tổng hai phần tử
// tại vị trí i, j tương ứng của 2 ma trận A, B
for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 C[i][j] = A[i][j] + B[i][j];
 }
}

// hiển thị ma trận tổng C
System.out.println("Ma trận tổng C:");
for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 System.out.print(C[i][j] + "\t");
 }
 System.out.println("\n");
}
}
```

Kết quả sau khi biên dịch chương trình:

```

Console
<terminated> ViDu1 [Java Application] C:\Progra
Nhập vào số dòng của ma trận:
2
Nhập vào số cột của ma trận:
3
Nhập các phần tử cho ma trận A:
A[0,0] = 1
A[0,1] = 0
A[0,2] = 12
A[1,0] = 4
A[1,1] = 9
A[1,2] = 43
Nhập các phần tử cho ma trận B:
B[0,0] = 20
B[0,1] = 4
B[0,2] = 94
B[1,0] = 89
B[1,1] = 5
B[1,2] = 10
Ma trận A:
1 0 12
4 9 43
Ma trận B:
20 4 94
89 5 10
Ma trận tổng C:
21 4 106
93 14 53

```

Ví dụ 4.7:

Một ma trận được gọi là ma trận thưa nếu số phần tử có giá trị bằng 0 nhiều hơn số phần tử khác 0. Viết chương trình thực hiện các công việc sau:

- Nhập m, n là số dòng và số cột của ma trận hai chiều A từ bàn phím.
- Nhập giá trị các phần tử của ma trận A từ bàn phím.
- Kiểm tra và thông báo lên màn hình ma trận vừa nhập là ma trận thưa hay ma trận không thưa.
- Yêu cầu kỹ thuật: Kiểm tra số dòng, số cột nhập vào không được nhỏ hơn 1.

```

public static void main(String[] args) {
 int m, n;
 int soPhanTu0 = 0; // số phần tử = 0 trong ma trận
 int soPhanTuKhac0 = 0; // số phần tử khác 0 trong ma trận
 Scanner scanner = new Scanner(System.in);

 do {
 System.out.println("Nhập vào số dòng của ma trận:");
 m = scanner.nextInt();
 System.out.println("Nhập vào số cột của ma trận:");
 n = scanner.nextInt();
 } while (m < 1 || n < 1);

 // khai báo ma trận A có m dòng, n cột
 int A[][] = new int[m][n];

 System.out.println("Nhập các phần tử cho ma trận A:");
 for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 System.out.print("A[" + i + "," + j + "] = ");
 A[i][j] = scanner.nextInt();
 }
 }
}

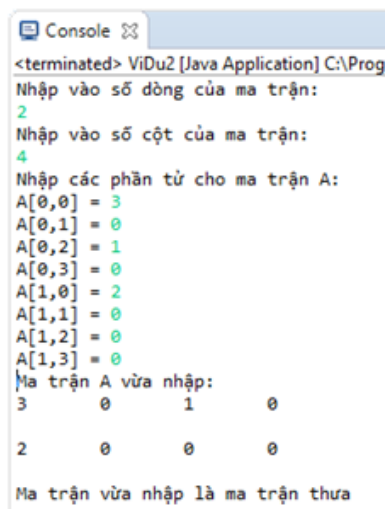
```

```
System.out.println("Ma trận A vừa nhập:");
for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 System.out.print(A[i][j] + "\t");
 }
 System.out.println("\n");
}

// kiểm tra ma trận thưa hay không thưa
for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 // kiểm tra nếu phần tử tại i, j bằng 0
 // thì tăng biến soPhanTu0 lên 1
 // ngược lại tăng biến soPhanTuKhac0 lên 1
 if (A[i][j] == 0) {
 soPhanTu0++;
 } else {
 soPhanTuKhac0++;
 }
 }
}

// nếu biến soPhanTu0 lớn hơn soPhanTuKhac0
// thì ma trận đó là ma trận thưa
// ngược lại là ma trận không thưa
if (soPhanTu0 > soPhanTuKhac0) {
 System.out.println("Ma trận vừa nhập là ma trận thưa");
} else {
 System.out.println("Ma trận vừa nhập là ma trận không thưa");
}
}
```

Kết quả sau khi biên dịch chương trình:



```
Console
<terminated> ViDu2 [Java Application] C:\Prog
Nhập vào số dòng của ma trận:
2
Nhập vào số cột của ma trận:
4
Nhập các phần tử cho ma trận A:
A[0,0] = 3
A[0,1] = 0
A[0,2] = 1
A[0,3] = 0
A[1,0] = 2
A[1,1] = 0
A[1,2] = 0
A[1,3] = 0
Ma trận A vừa nhập:
3 0 1 0
2 0 0 0
Ma trận vừa nhập là ma trận thưa
```

Ví dụ 4.8:



Một ma trận được gọi là ma trận đối xứng trước hết nó phải là ma trận vuông (có số dòng và số cột bằng nhau) và các phần tử của nó đối xứng nhau qua đường chéo chính. Viết chương trình nhập từ bàn phím các phần tử của ma trận A, kích thước m dòng, n cột ( $1 \leq m, n \leq 5$ ). Kiểm tra xem ma trận vừa nhập có phải là ma trận đối xứng hay không?

**Hướng dẫn:** Giả sử chúng ta có một ma trận vuông có 3 dòng, 3 cột thì chúng ta gọi ma trận này là ma trận vuông bậc 3. Hình dưới đây minh họa đường chéo phụ và đường chéo chính như sau:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

đường chéo phụ

đường chéo chính

$a_{11}, a_{12}, \dots, a_{nn}$  : các phần tử chéo

Các phần tử nằm trên đường chéo chính có đặc điểm chỉ số dòng bằng chỉ số cột. Ví dụ trong hình trên chúng ta thấy có các phần tử nằm trên đường chéo chính như  $a_{11}, a_{12}, \dots, a_{nn}$ , các phần tử này được gọi là các phần tử chéo.

Để kiểm tra ma trận A có phải là ma trận đối xứng hay không thì cần thực hiện các bước sau:

Ma trận A nhập vào phải luôn luôn là một ma trận vuông.

Kiểm tra phần tử nằm dưới đường chéo chính có bằng phần tử đối xứng với nó qua đường chéo chính hay không (tức là kiểm tra các phần tử  $A[i][j]$  có bằng  $A[j][i]$  không, với  $i, j$  chạy từ 0 đến  $n$ ).

```
public static void main(String[] args) {
 int n; // bậc của ma trận
 int kt = 0;
 Scanner scanner = new Scanner(System.in);

 do {
 System.out.println("Nhập vào số bậc của ma trận: ");
 n = scanner.nextInt();
 } while (n < 1);

 // ma trận A là ma trận vuông
 // có n dòng và n cột
 int[][] A = new int[n][n];

 System.out.println("Nhập các phần tử cho ma trận A:");
 for (int i = 0; i < n; i++) {
 for (int j = 0; j < n; j++) {
```

```

 System.out.print("A[" + i + "," + j + "] = ");
 A[i][j] = scanner.nextInt();
 }
}

System.out.println("Ma trận A vừa nhập:");
for (int i = 0; i < n; i++) {
 for (int j = 0; j < n; j++) {
 System.out.print(A[i][j] + "\t");
 }
 System.out.println("\n");
}
// kiểm tra các phần tử nằm dưới đường chéo chính
// và phần tử đối xứng với nó qua đường chéo chính
// có bằng nhau hay không
// nếu bằng nhau thì ma trận đó là ma trận đối xứng
for (int i = 0; i < n; i++) {
 for (int j = 0; j < i; j++) {
 if (A[i][j] == A[j][i]) {
 kt = 1;
 } else {
 kt = 0;
 }
 }
}

if (kt == 1) {
 System.out.println("Ma trận vừa nhập là ma trận đối xứng");
} else {
 System.out.println("Ma trận vừa nhập là ma trận không đối xứng");
}
}

```

### Giải thích hoạt động của chương trình trên như sau:

Giả sử khi biên dịch chương trình, tôi nhập vào một ma trận vuông bậc 3 như sau:

|   |   |   |
|---|---|---|
| 3 | 1 | 2 |
| 1 | 0 | 3 |
| 2 | 3 | 0 |

Thì chúng ta thấy các phần tử nằm trên đường chéo chính bao gồm 3 phần tử đó là  $A_{00} = 3$ ,  $A_{11} = 0$  và  $A_{22} = 0$ . Hoạt động của vòng lặp for kiểm tra mảng đó có phải là mảng đối xứng trải qua các bước như sau:

Bước 1: Khởi tạo  $i = 0 < n$  nhưng  $j = 0 = i$  nên không thực hiện lệnh trong thân vòng lặp for.

Bước 2: Tăng  $i$  lên 1, lúc này  $i = 1 < n$  và  $j = 0 < i$  nên thực hiện lệnh trong thân vòng lặp for thì thấy  $A[1][0] = A[0][1] = 1$  nên lúc này biến  $kt = 1$ .

Bước 3: Tăng  $j$  lên 1, lúc này  $j = 1 = i$  nên không thực hiện lệnh trong thân vòng lặp for.

Bước 4: Quay lại vòng lặp for, lúc này  $i = 2 < n$  và  $j = 0 < i$  nên thực hiện lệnh trong thân vòng lặp for thì thấy  $A[2][0] = A[0][2] = 2$  nên lúc này biến  $kt = 1$ .

Bước 5: Tăng  $j$  lên 1, lúc này  $j = 1 < i$  nên thực hiện lệnh trong thân vòng lặp for thì thấy  $A[2][1] = A[1][2] = 3$  nên lúc này biến  $kt = 1$ .

Bước 6: Tăng j lên 1, lúc này  $j = 2 = i$  nên không thực hiện lệnh trong thân vòng lặp for.

Bước 7: Tăng i lên 1, lúc này  $i = 3 = n$  nên kết thúc vòng lặp for.

Bước 8: Sau khi ra khỏi vòng lặp for thì lúc này biến  $kt = 1$  nên sẽ hiển thị thông báo "*Ma trận vừa nhập là ma trận đối xứng*" ra màn hình.

#### 4.2.3. Độ dài của mảng

Để biết có bao nhiêu phần tử một mảng, sử dụng thuộc tính length :

Ví dụ 4.9:

```
package vn.viettuts.array;
```

```
public class DodaiArray1 {
 public static void main(String[] args) {
 String[] cars = { "Honda", "BMW", "Ford", "Mazda" };
 System.out.println("Độ dài của mảng cars là: " + cars.length);
 }
}
```

Kết quả: 4

### 4.3. HẠN CHẾ CỦA MẢNG

Vì khi khai báo mảng, chúng ta cần phải khai báo kích thước cố định cho mảng nên sẽ xảy ra 2 trường hợp như sau: Nếu khai báo mảng với kích thước lớn mà không sử dụng hết sẽ gây lãng phí bộ nhớ, ngược lại nếu khai báo mảng với kích thước quá nhỏ thì chúng ta sẽ không thể mở rộng mảng được.

Vì các phần tử trong mảng được sắp xếp liên tục nên việc chèn hoặc xóa một phần tử trong mảng cũng sẽ gặp nhiều khó khăn.

### 4.4. BÀI TẬP CHƯƠNG 4

**Bài tập 4.1:** Viết chương trình nhập mảng số nguyên 1 chiều gồm n phần tử. Hiển thị mảng vừa nhập và hiển thị các phần tử có giá trị lẻ của mảng.

**Bài tập 4.2:** Bài tập 4.1: Viết chương trình nhập mảng số nguyên 2 chiều gồm n hàng và m cột. Hiển thị mảng vừa nhập và hiển thị tổng giá trị các phần tử theo hàng.

**Bài tập 4.3:** Viết chương trình nhập mảng số nguyên 1 chiều gồm n phần tử. Hiển thị mảng vừa nhập và hiển thị các phần tử là số chính phương của mảng (số chính phương là số bằng bình phương của một số khác, ví dụ;  $4=2 \times 2$ )

**Bài tập 4.4:** Bài tập 4.1: Viết chương trình nhập mảng số nguyên 2 chiều gồm n hàng và m cột. Hiển thị mảng vừa nhập và hiển thị các giá trị trên đường chéo chính của mảng.

## CHƯƠNG 5: THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM

### 5.1. THUẬT TOÁN SẮP XẾP

#### 5.1.1. Giới thiệu

Sắp xếp là sắp đặt các phần tử của một cấu trúc theo thứ tự tăng dần (hay giảm dần). Với một cấu trúc đã được sắp xếp chúng ta rất thuận tiện khi thực hiện các tác vụ trên cấu trúc như tìm kiếm, trích lọc, duyệt cấu trúc ....

Có 2 loại giải thuật sắp xếp được dùng phổ biến trong khoa học máy tính là internal sorting và external sorting

Với internal sorting thì toàn bộ dữ liệu cần sắp xếp được đưa vào bộ nhớ trong, do vậy kích thước dữ liệu cần sắp xếp nhỏ và thời gian sắp xếp được thực hiện rất nhanh.

Với external sorting thì chỉ một phần nhỏ dữ liệu cần sắp xếp được đưa vào bộ nhớ trong, phần lớn dữ liệu còn lại được lưu ở bộ nhớ ngoài, kích thước dữ liệu cần sắp xếp lúc này rất lớn, và thời gian sắp xếp thực hiện rất chậm.

Trong khuôn khổ bài này mình chỉ xin giới thiệu về internal sorting để các bạn có một cái nhìn khoa học hơn về giải thuật sắp xếp. Internal sorting bao gồm:

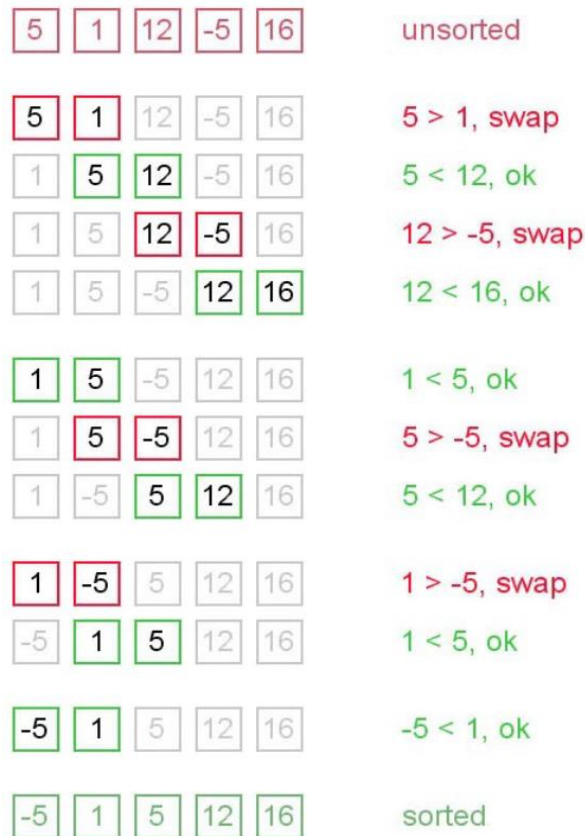
- Bubble sort.
- Quick sort.
- Simple selection sort.
- Heap sort.
- Simple insertion sort.
- Shell sort.
- Merge sort.
- Radix sort.

Tôi sẽ giới thiệu 3 thuật toán sắp xếp đầu tiên thường sử dụng đến các bạn.

#### 5.1.2. Bubble sort

Bubble sort hay là sắp xếp bằng cách đổi chỗ hay, hay còn gọi là sắp xếp nổi bọt.

Nhìn vào hình mô tả bên dưới chắc các bạn đã hình dung ra các mà phương pháp này sử dụng để sắp xếp các phần như nào rồi nhỉ. Phương pháp này sẽ duyệt danh sách nhiều lần, trong mỗi lần duyệt sẽ lần lượt so sánh cặp nút thứ  $i$  và thứ  $i+1$  và đổi chỗ hai nút này nếu chúng không đúng thứ tự. Sau đây là code cài đặt giải thuật bubble sort.



Ví dụ 5.1: Chương trình thuật toán Bubble sort:

```
public class BubbleSort {
 // Sort the original data
 private static final int [] array = {5,1,12,-5,16}

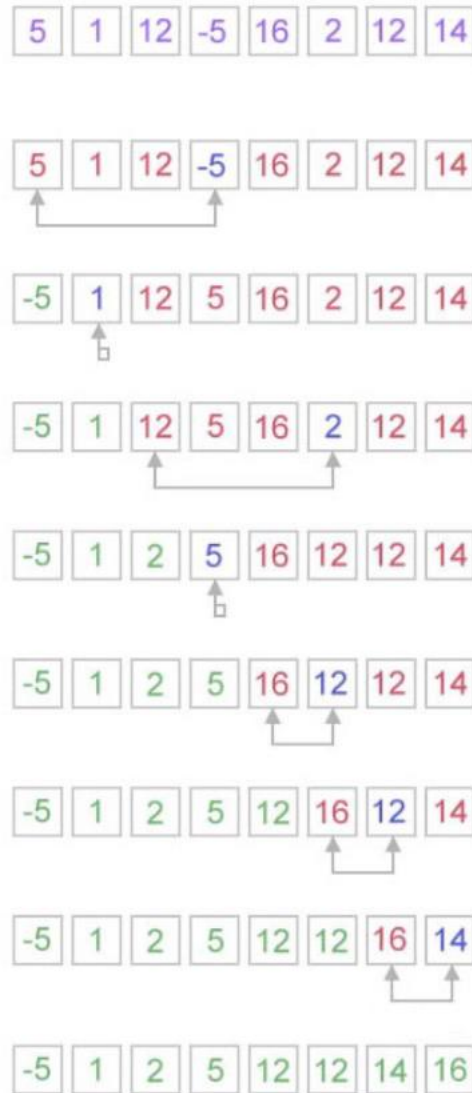
 public static void main(String[] args) {
 // TODO Auto-generated method stub
 int temp;
 for (int i = 0 ; i < array.length - 1 ; i ++) {
 for (int j = 0 ; j < array.length - 1 - i ; j ++) {
 if (array[j] > array[j + 1]) {
 temp = array[j];
 array[j] = array[j + 1];
 array[j + 1] = temp;
 }
 }
 }
 System.out.println("\nMảng sau khi sắp xếp: ");
 for (int i = 0; i < array.length; i++) {
 System.out.print(array[i] + " ");
 }
 }
}
```

Kết quả chương trình:

```
Mảng sau khi sắp xếp:
-5 1 5 12 16
```

### 5.1.3. Simple selection sort

Ý tưởng thuật toán Simple selection sort – sắp xếp lựa chọn là: Chọn phần tử nhỏ nhất trong  $n$  phần tử ban đầu, đưa phần tử này về vị trí đúng là đầu tiên của dãy hiện hành. Sau đó không quan tâm đến nó nữa, xem dãy hiện hành chỉ còn  $n-1$  phần tử của dãy ban đầu, bắt đầu từ vị trí thứ 2. Lặp lại quá trình trên cho dãy hiện hành đến khi dãy hiện hành chỉ còn 1 phần tử. Dãy ban đầu có  $n$  phần tử. Vậy tóm tắt ý tưởng thuật toán là thực hiện  $n-1$  lượt việc đưa phần tử nhỏ nhất trong dãy hiện hành về vị trí đúng ở đầu dãy.



Ví dụ 5.2: Chương trình thuật toán SimpleSelectSort:

```
public class SimpleSelectSort {
 public void selectionSort(int arr[]) {
 int indexMin, i, j;

 // lap qua tat ca cac so
 for (i = 0; i < arr.length - 1; i++) {
 // thiet lap phan tu hien tai la min
 indexMin = i;

 // kiem tra phan tu hien tai co phai la nho nhat khong
 for (j = i + 1; j < arr.length; j++) {
```

```
 if (arr[j] < arr[indexMin]) {
 indexMin = j;
 }
 }

 if (indexMin != i) {
 System.out.println(" ==> Trao doi phan tu: [" + arr[i]
 + ", " + arr[indexMin] + "]");
 // Trao doi cac so
 int temp = arr[indexMin];
 arr[indexMin] = arr[i];
 arr[i] = temp;
 }

 System.out.println("Vong lap thu " + (i + 1));
 display(arr);
}

public void display(int arr[]) {
 int i;
 System.out.print("[");

 // Duyet qua tat ca phan tu
 for (i = 0; i < arr.length; i++) {
 System.out.print(arr[i] + " ");
 }

 System.out.print("]\n");
}

public static void main(String[] args) {
 // khoi tao mang arr
 int arr[] = { 6, 7, 0, 2, 8, 1, 3, 9, 4, 5 };

 SimpleSelectSort sapXepChon = new SimpleSelectSort();
 System.out.println("Mang du lieu dau vao: ");
 sapXepChon.display(arr);
 System.out.println("-----");
 sapXepChon.selectionSort(arr);
 System.out.println("-----");
 System.out.println("\nMang sau khi da sap xep: ");
 sapXepChon.display(arr);
}
}
```

Kết quả chạy chương trình:



```

Mang du lieu dau vao:
[6 7 0 2 8 1 3 9 4 5]

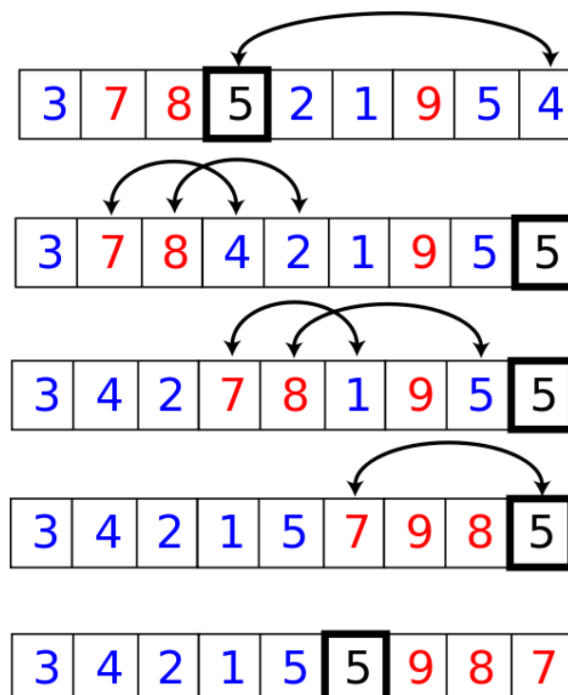
==> Trao doi phan tu: [6, 0]
Vong lap thu 1
[0 7 6 2 8 1 3 9 4 5]
==> Trao doi phan tu: [7, 1]
Vong lap thu 2
[0 1 6 2 8 7 3 9 4 5]
==> Trao doi phan tu: [6, 2]
Vong lap thu 3
[0 1 2 6 8 7 3 9 4 5]
==> Trao doi phan tu: [6, 3]
Vong lap thu 4
[0 1 2 3 8 7 6 9 4 5]
==> Trao doi phan tu: [8, 4]
Vong lap thu 5
[0 1 2 3 4 7 6 9 8 5]
==> Trao doi phan tu: [7, 5]
Vong lap thu 6
[0 1 2 3 4 5 6 9 8 7]
Vong lap thu 7
[0 1 2 3 4 5 6 9 8 7]
==> Trao doi phan tu: [9, 7]
Vong lap thu 8
[0 1 2 3 4 5 6 7 8 9]
Vong lap thu 9
[0 1 2 3 4 5 6 7 8 9]

Mang sau khi da sap xep:
[0 1 2 3 4 5 6 7 8 9]

```

#### 5.1.4. Quick Sort

Quick sort - Sắp xếp nhanh: là phương pháp đổi chỗ từng phần (partition exchange), đây là phương pháp rất hiệu quả, rất thông dụng.



Nội dung của phương pháp này như sau:

Chọn một nút bất kỳ trong danh sách (Giả sử là nút số 5 như trên hình) gọi là nút làm trục (pivot node), xác định vị trí hợp lệ của nút này trong danh sách (gọi là vị trí pivot).

Tiếp theo chúng ta phân hoạch các nút còn lại trong danh sách cần sắp xếp sao cho từ vị trí 0 đến vị trí pivot-1 đều có nội dung nhỏ hơn hoặc bằng nút làm trục, các nút từ vị trí pivot+1 đến n-1 đều có nội dung lớn hơn nút làm trục.

Quá trình lại tiếp tục như thế với hai danh sách con từ vị trí 0 đến vị trí pivot-1 và từ vị trí pivot+1 đến vị trí n-1, ... Sau cùng chúng ta sẽ được danh sách có thứ tự. Sau đây là code cài đặt giải thuật quick sort.

Ví dụ 5.3: Chương trình thuật toán Quick sort:

```
public class QuickSort {
 // ham de trao doi gia tri
 public void swap(int arr[], int num1, int num2) {
 int temp = arr[num1];
 arr[num1] = arr[num2];
 arr[num2] = temp;
 }

 // ham de chia mang thanh 2 phan, su dung phan tu chot (pivot)
 public int partition(int arr[], int left, int right, int pivot) {
 int leftPointer = left - 1;
 int rightPointer = right;

 while (true) {
 while (arr[++leftPointer] < pivot) {
 // khong lam gi
 }

 while (rightPointer > 0 && arr[--rightPointer] > pivot) {
 // khong lam gi
 }

 if (leftPointer >= rightPointer) {
 break;
 } else {
 System.out.println(" Phan tu duoc trao doi: " + arr[leftPointer] + ", "
 + arr[rightPointer]);
 swap(arr, leftPointer, rightPointer);
 }
 }

 System.out.println(" Phan tu chot duoc trao doi: " + arr[leftPointer] + ", " +
 arr[right]);
 swap(arr, leftPointer, right);
 display(arr);
 return leftPointer;
 }

 // ham sap xep
 public void quickSort(int arr[], int left, int right) {
 if (right - left <= 0) {
 return;
 } else {

```

```

 int pivot = arr[right];
 int partitionPoint = partition(arr, left, right, pivot);
 quickSort(arr, left, partitionPoint - 1);
 quickSort(arr, partitionPoint + 1, right);
 }
}

public void display(int arr[]) {
 int i;
 System.out.print("[");

 // Duyệt qua tất cả phần tử
 for (i = 0; i < arr.length; i++) {
 System.out.print(arr[i] + " ");
 }

 System.out.print("]\n");
}

public static void main(String[] args) {
 // TODO Auto-generated method stub
 // Sort the original data
 int [] arr = {3,7,8,5,2,1,9,5,4};
 QuickSort sapXepNhanh = new QuickSort();
 System.out.println("Mang du lieu dau vao: ");
 sapXepNhanh.display(arr);
 System.out.println("-----");
 sapXepNhanh.quickSort(arr, 0, arr.length - 1);
 System.out.println("-----");
 System.out.println("\nMang sau khi da sap xep: ");
 sapXepNhanh.display(arr);
}
}

```

Kết quả chạy chương trình:

```

Mang du lieu dau vao:
[3 7 8 5 2 1 9 5 4]

Phan tu duoc trao doi: 7, 1
Phan tu duoc trao doi: 8, 2
Phan tu chot duoc trao doi: 5, 4
[3 1 2 4 8 7 9 5 5]
Phan tu duoc trao doi: 3, 1
Phan tu chot duoc trao doi: 3, 2
[1 2 3 4 8 7 9 5 5]
Phan tu duoc trao doi: 8, 5
Phan tu chot duoc trao doi: 7, 5
[1 2 3 4 5 5 9 8 7]
Phan tu chot duoc trao doi: 9, 7
[1 2 3 4 5 5 7 8 9]
Phan tu chot duoc trao doi: 9, 9
[1 2 3 4 5 5 7 8 9]

Mang sau khi da sap xep:
[1 2 3 4 5 5 7 8 9]

```

## 5.2. THUẬT TOÁN TÌM KIẾM

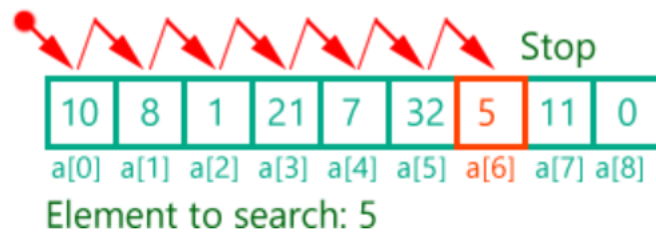
### 5.2.1. Tìm kiếm tuyến tính

#### Giải thuật:

Tìm tuyến tính (LinearSearch) là một kỹ thuật tìm kiếm rất đơn giản và cổ điển. Ý tưởng là tìm kiếm từ đầu cho đến cuối mảng (hoặc ngược lại).

- Nếu tìm thấy trả vị trí phần tử có kết quả tìm kiếm và dừng.
- Nếu không tìm thấy thì trả về -1.

Ví dụ: Cho mảng như bên dưới và tìm vị trí phần tử có giá trị là: 5



#### Cài đặt thuật toán:

Ví dụ 5.4: Từ mô tả trên đây của thuật toán tìm tuyến tính, có thể cài đặt hàm LinearSearch để xác định vị trí của phần tử có giá trị x trong mảng a:

```
import java.util.Scanner;

public class TimTuyenTinh {

 //Xây dựng hàm tìm tuyến tính
 static int LinearSearch(int a[], int N, int x)
 {
 for (int i = 0; i < N; i++)
 if (a[i] == x)
 return i;
 return -1;
 }

 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);
 int a[] = {10, 8, 1, 21, 7, 32, 5, 11, 0};

 System.out.print("Nhập giá trị cần tìm: ");
 int giatri = scanner.nextInt();

 int ketqua = LinearSearch(a, a.length, giatri);
 if (ketqua == -1)
 System.out.println("Không tìm thấy giá trị: " + giatri);
 else
 System.out.println("Giá trị cần tìm ở chỉ số mảng: " + ketqua);
 }
}
```

Chương trình khi chạy:

```
Nhập giá trị cần tìm: 5
Giá trị cần tìm ở chỉ số mảng: 6
```

Trong cài đặt trên đây, nhận thấy mỗi lần lặp của vòng lặp while phải tiến thành kiểm tra 2 điều kiện ( $i < N$ ) - điều kiện biên của mảng - và ( $a[i] != x$ ) - điều kiện kiểm tra chính.

### Đánh giá giải thuật:

Có thể ước lượng độ phức tạp của giải thuật tìm kiếm qua số lượng các phép so sánh được tiến hành để tìm ra x. Trường hợp giải thuật tìm tuyến tính, có:

| Trường hợp | Số lần so sánh | Giải thích                                                         |
|------------|----------------|--------------------------------------------------------------------|
| Tốt nhất   | 1              | Phần tử đầu tiên có giá trị x                                      |
| Xấu nhất   | $n+1$          | Phần tử cuối cùng có giá trị x                                     |
| Trung bình | $(n+1)/2$      | Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau. |

Vậy giải thuật tìm tuyến tính có độ phức tạp tính toán cấp n:  $T(n) = O(n)$

Trong trường hợp tốt nhất, phần tử cần tìm nằm ở vị trí đầu tiên, thuật toán sử dụng 1 lần so sánh.

Trong trường hợp xấu nhất, phần tử cần tìm nằm ở vị trí cuối hoặc không nằm trong mảng, thuật toán sử dụng  $n - 1$  lần so sánh.

Linear Search là một giải thuật đơn giản khi hiện thực và khá hiệu quả với danh sách đủ nhỏ hoặc một danh sách chưa được sắp xếp.

### 5.2.2. Tìm kiếm nhị phân

#### Giải thuật:

Yêu cầu của thuật toán tìm kiếm nhị phân là mảng đã được sắp xếp tăng dần. Cách thực hiện rất đơn giản, đầu tiên ta kiểm tra số chính giữa mảng có phải là số cần tìm hay không. Nếu số cần tìm lớn hơn số chính giữa thì ta có thể loại bỏ 1/2 số phần tử trước số chính giữa và tiếp tục tìm kiếm trong 1/2 còn lại. Nếu số cần tìm nhỏ hơn số chính giữa thì ta có thể loại bỏ 1/2 số phần tử sau số chính giữa và tiếp tục tìm kiếm trong 1/2 còn lại. Cứ mỗi lần chọn phần tử để so sánh, ta sẽ loại bỏ được 1/2 số phần tử không phù hợp.

Các bước thực hiện giải thuật tìm kiếm nhị phân:

1. Chọn min là 0 và max =  $n - 1$  (n là số phần tử của mảng)
2. Nếu max nhỏ hơn min, số cần tìm không tồn tại, trả về -1
3. Chọn mid là số chính giữa mảng
4. Nếu mid là số cần tìm (target), ta trả về giá trị cần tìm
5. Nếu  $mid < target$ , ta set lại min = mid + 1
6. Nếu  $mid > target$ , ta set lại max = mid - 1;
7. Lặp lại bước 2

#### Cài đặt thuật toán:

Ví dụ 5.5: Từ mô tả trên đây của thuật toán tìm kiếm nhị phân, có thể cài đặt hàm BinarySearch để xác định vị trí của phần tử có giá trị x trong mảng a:

```
import java.util.Scanner;

public class TimNhiPhan {

 private static int BinarySearch(int a[], int x) {
 int min = 0;
 int max = a.length - 1;
 int mid = 0;

 while (min <= max) {
 mid = (min + max) / 2;
 if (a[mid] == x) {
 return mid;
 } else if (a[mid] < x) {
 min = mid + 1;
 } else {
 max = mid - 1;
 }
 }

 return -1;
 }

 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);
 int a[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 27, 31};

 System.out.print("Nhập giá trị cần tìm: ");
 int giatri = scanner.nextInt();

 System.out.println("Giá trị cần tìm ở chỉ số mảng: " + BinarySearch(a,
giatri));
 }
}
```

Chương trình khi chạy:

```
Nhập giá trị cần tìm: 13
Giá trị cần tìm ở chỉ số mảng: 5
```

### Đánh giá giải thuật:

Trường hợp giải thuật tìm nhị phân, có bảng phân tích sau:

| Trường hợp | Số lần so sánh | Giải thích                                                        |
|------------|----------------|-------------------------------------------------------------------|
| Tốt nhất   | 1              | Phần tử giữa của mảng có giá trị x                                |
| Xấu nhất   | $\log_2 n$     | Không có x trong mảng                                             |
| Trung bình | $\log_2 n/2$   | Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau |

Vậy giải thuật tìm nhị phân có độ phức tạp tính toán cấp n:  $T(n) = O(\log_2 n)$

Giải thuật tìm nhị phân dựa vào quan hệ giá trị của các phần tử mảng để định hướng trong quá trình tìm kiếm, do vậy chỉ áp dụng được cho những dãy đã có thứ tự.

Giải thuật tìm nhị phân tiết kiệm thời gian hơn rất nhiều so với giải thuật tìm tuyến tính do  $T_{nhị\ phân}(n) = O(\log 2 n) < T_{tuyến\ tính}(n) = O(n)$ . Tuy nhiên khi muốn áp dụng giải thuật tìm nhị phân cần phải xét đến thời gian sắp xếp dãy số để thỏa điều kiện dãy số có thứ tự. Thời gian này không nhỏ, và khi dãy số biến động cần phải tiến hành sắp xếp lại. Tất cả các nhu cầu đó tạo ra khuyết điểm chính cho giải thuật tìm nhị phân. Ta cần cân nhắc nhu cầu thực tế để chọn một trong hai giải thuật tìm kiếm trên sao cho có lợi nhất.

### 5.3. BÀI TẬP CHƯƠNG 5

**Bài tập 5.1:** Viết chương trình nhập mảng số nguyên  $n$  phần tử và sắp xếp các phần tử theo thứ tự giảm dần.

**Bài tập 5.2:** Viết chương trình nhập mảng số nguyên  $n$  phần tử và tìm kiếm vị trí của phần tử có giá trị nhập vào từ bàn phím.

**Bài tập 5.3:** Viết chương trình nhập số nguyên  $n$  phần tử và hiển thị phần tử có giá trị lớn nhất của mảng.

**Bài tập 5.4:** Viết chương trình nhập một mảng số nguyên 2 chiều gồm  $m$  hàng và  $n$  cột. Sắp xếp các phần tử của từng hàng theo thứ tự tăng dần và hiển thị mảng trước và sau khi sắp xếp.

**Bài tập 5.5:** Viết chương trình nhập một mảng số nguyên 2 chiều gồm  $m$  hàng và  $n$  cột. Hiển thị mảng và hiển thị phần tử có giá trị lớn nhất của mảng.

**Bài tập 5.6:** Viết chương trình nhập một mảng số nguyên 2 chiều gồm  $m$  hàng và  $n$  cột. Hiển thị mảng và hiển thị các phần tử có giá trị lớn nhất của từng hàng trong mảng.

**Bài tập 5.7:** Viết chương trình nhập một mảng số nguyên 2 chiều gồm  $m$  hàng và  $n$  cột. Hoán vị hàng số 2 và 3 của mảng. Hiển thị mảng trước và sau khi hoán vị.



## CHƯƠNG 6: KỸ THUẬT ĐỆ QUY VÀ CHUỖI TRONG JAVA

### 6.1. KỸ THUẬT ĐỆ QUY

#### 6.1.1. Giới thiệu về đệ quy

Đệ quy trong java là quá trình trong đó một phương thức gọi lại chính nó một cách liên tiếp. Một phương thức trong java gọi lại chính nó được gọi là phương thức đệ quy.

Sử dụng đệ quy giúp code chặt chẽ hơn nhưng sẽ khó để hiểu hơn.

Cú pháp:

```
returntype methodname() {
 // code
 methodname();
}
```

- returntype: Kiểu dữ liệu trả về

- methodname(): Tên phương thức (hàm)

#### 6.1.2. Ví dụ về đệ quy trong java

Dưới đây là các ví dụ về cách sử dụng đệ quy trong java.

Ví dụ 6.1: Vòng lặp vô tận

```
public class Vidu6_1 {
 static void p() {
 System.out.println("hello");
 p();
 }

 public static void main(String[] args) {
 p();
 }
}
```

Khi chạy chương trình thì sẽ báo lỗi lặp vô hạn:

```
Vidu6_1 [Java Application]
hello
hello
hello
*** java.lang.instrument ASSERTION FAILED ***: "!errorOutstanding"
```

Ví dụ 6.2: Vòng lặp có hạn

```
public class Vidu6_2 {
 static int count = 0;

 static void p() {
 count++;
 if (count <= 5) {
 System.out.println("hello " + count);
 p();
 }
 }
}
```

```
 public static void main(String[] args) {
 p();
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu6_2 |
hello 2
hello 3
hello 4
hello 5
```

Ví dụ 6.3: Tính N giai thừa ( $n! = 1 \times 2 \times \dots \times n$ )

```
public class Vidu6_3 {
 static int factorial(int n) {
 if (n == 1)
 return 1;
 else
 return (n * factorial(n - 1));
 }

 public static void main(String[] args) {
 System.out.println("Giai thua cua 5 là: " + factorial(5));
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu6_3 [Java Application] C
Giai thua cua 5 là: 120
```

Chương trình trên hoạt động như sau:

```
1 | factorial(5)
2 | factorial(4)
3 | factorial(3)
4 | factorial(2)
5 | factorial(1)
6 | return 1
7 | return 2*1 = 2
8 | return 3*2 = 6
9 | return 4*6 = 24
10 | return 5*24 = 120
```

Ví dụ 6.4: Hiển thị 15 số đầu của dãy số Fibonacci

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

```
public class Vidu6_4 {
```

```
static int n1 = 0, n2 = 1, n3 = 0;

static void printFibo(int count) {
 if (count > 0) {
 n3 = n1 + n2;
 n1 = n2;
 n2 = n3;
 System.out.print(" " + n3);
 printFibo(count - 1);
 }
}

public static void main(String[] args) {
 int count = 15;
 System.out.print(n1 + " " + n2); // in 0 và 1
 printFibo(count - 2); // n-2 vì 2 số 0 và 1 đã được in ra
}
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu6_4 [Java Application] C:\Program Files\Ja
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

## 6.2. CHUỖI TRONG JAVA

### 6.2.1. Khai báo chuỗi ký tự

Trong Java, để khai báo 1 chuỗi ký tự thì chúng ta có 2 cách như sau:

Cách 1:

Cú pháp:

```
String tenChuoi = "giá_trị_khởi_tạo";
```

Trong đó, giá\_trị\_khởi\_tạo của chuỗi có thể có hoặc không và nếu có thì phải đặt trong cặp dấu ". Nếu một chuỗi có giá\_trị\_khởi\_tạo = " " thì chuỗi đó được gọi là chuỗi rỗng.

Ví dụ 6.5:

```
public class Vidu6_5 {
 public static void main(String[] args) {
 // khai báo chuỗi rỗng
 String chuoi1 = "";

 // khai báo chuỗi có nội dung là "Welcome"
 String chuoi2 = "Welcome";

 // hiển thị giá trị của 2 chuỗi trên ra màn hình
 System.out.println("Chuỗi rỗng có giá trị = " + chuoi1);
 System.out.println("Chuỗi 2 có giá trị = " + chuoi2);
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_5 [Java Application]
Chuỗi rỗng có giá trị =
Chuỗi 2 có giá trị = Welcome
```

Cách 2: Sử dụng từ khóa new.

Cú pháp:

```
String tenChuoi = new String("giá_trị");
```

trong đó giá\_trị là một chuỗi bất kỳ và phải đặt trong cặp dấu " ".

Ví dụ 6.6:

```
public class Vidu6_6 {
 public static void main(String[] args) {
 // khai báo một chuỗi có nội dung là "Welcome to Java!"
 String chuoi = new String("Welcome to Java!");
 System.out.println(chuoi);
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_6 [Java Application]
Welcome to Java!
```

## 6.2.2. Các hàm xử lý chuỗi ký tự

### 6.2.2.1. Hàm xác định độ dài chuỗi ký tự

Cú pháp:

```
int length = tên_chuỗi.length();
```

Chức năng: Hàm trả về độ dài chuỗi ký tự bằng cách đếm các ký tự trong chuỗi.

Ví dụ 6.7:

```
import java.util.Scanner;

public class Vidu6_7 {
 public static void main(String[] args) {
 String chuoi;
 int doDai;
 Scanner scanner = new Scanner(System.in);

 System.out.println("Nhập vào chuỗi bất kỳ từ bàn phím: ");
 chuoi = scanner.nextLine();

 // tính độ dài chuỗi
 doDai = chuoi.length();

 System.out.println("Chuỗi " + chuoi + " có độ dài = " + doDai);
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_7 [Java Application] C:\Progr
Nhập vào chuỗi bất kỳ từ bàn phím:
hello
Chuỗi hello có độ dài = 5
```

#### 6.2.2.2. Hàm nối 2 chuỗi ký tự

Trong Java, để nối 2 chuỗi ký tự lại với nhau thì chúng ta có 2 cách. Cách thứ nhất là chúng ta dùng dấu + để nối chuỗi và cách thứ hai là dùng phương thức concat().

Cú pháp:

```
String string3 = string1.concat(String string2);
```

Chức năng: Hàm có tác dụng nối chuỗi string2 vào string1 và trả về chuỗi string3.

Ví dụ 6.8:

```
public class Vidu6_8 {
 public static void main(String[] args) {
 String chuoi1 = "Happy ", chuoi2 = "new year!";

 /*
 * nối chuỗi
 */
 String chuoi3 = chuoi1.concat(chuoi2);
 System.out.println(chuoi3);
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_8 [Java Application] C
Happy new year!
```

#### 6.2.2.3. Hàm trả về một ký tự trong chuỗi

Một chuỗi là tập hợp các ký tự và ký tự đầu tiên trong chuỗi sẽ có chỉ số (index) là 0. Ví dụ: một chuỗi có chiều dài là 10 thì chỉ số của các ký tự trong chuỗi đó sẽ được đánh số từ 0 đến 9.

Cú pháp:

```
char character = chuoi.charAt(int index);
```

Chức năng: Hàm trả về ký tự character trong chuỗi có chỉ số là index.

Ví dụ 6.9:

```
public class Vidu6_9 {
 public static void main(String[] args) {
 String chuoi = "Happy new year!";

 /*
 * trả về ký tự có chỉ số là 4 trong chuỗi
 * ký tự 'H' có chỉ số là 0, nên ký tự có chỉ số 4 là ký tự 'y'
 */
 char character = chuoi.charAt(4);
 System.out.println(character);
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_9 [J
y
```

#### 6.2.2.4. Hàm so sánh 2 chuỗi ký tự

Cú pháp:

```
int result = string1.compareTo(String string2);
```

Chức năng: Hàm có tác dụng so sánh hai chuỗi string1, string2 và trả về kết quả:

- Nếu result = 0 thì hai chuỗi đó bằng nhau.
- Nếu result < 0 thì chuỗi string1 < string2.
- Nếu result > 0 thì chuỗi string1 > string2.

Ví dụ 6.10:

```
public class Vidu6_10 {
 public static void main(String[] args) {
 int result;
 String string1 = "Happy new year!";
 String string2 = "Happy new year!";

 result = string1.compareTo(string2);
 if (result == 0) {
 System.out.println("Chuỗi " + string1 + " = " + string2);
 } else if (result < 0) {
 System.out.println("Chuỗi " + string1 + " < " + string2);
 } else {
 System.out.println("Chuỗi " + string1 + " > " + string2);
 }
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_10 [Java Application] C:\Program Files\J
Chuỗi Happy new year! = Happy new year!
```

#### 6.2.2.5. Hàm trả về vị trí xuất hiện đầu tiên của một chuỗi trong chuỗi khác

Cú pháp:

```
int result = string1.indexOf(String string2);
```

Chức năng: Hàm trả về vị trí xuất hiện đầu tiên của chuỗi string2 trong string1. Nếu chuỗi string2 không có trong chuỗi string1 thì kết quả trả về result = -1.

Ví dụ 6.11:

```
public class Vidu6_11 {
 public static void main(String[] args) {
 int result;
 String string1 = "Happy new year!";
 String string2 = "new year!";

 result = string1.indexOf(string2);
 System.out.println("Vị trí đầu tiên xuất hiện chuỗi " + string2 +
 " trong chuỗi " + string1 + " = " + result);
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_11 [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (Sep 14, 20:
Vị trí đầu tiên xuất hiện chuỗi new year! trong chuỗi Happy new year! = 6
```

#### 6.2.2.6. Hàm trả về vị trí xuất hiện cuối cùng của một chuỗi trong chuỗi khác

Cú pháp:

```
int result = string1.lastIndexOf(String string2);
```

Chức năng: Hàm trả về vị trí xuất hiện cuối cùng của chuỗi string2 trong string1. Nếu chuỗi string2 không có trong chuỗi string1 thì kết quả trả về result = -1.

Ví dụ 6.12:

```
public class Vidu6_12 {
 public static void main(String[] args) {
 int result;
 String string1 = "Happy new year and new year!";
 String string2 = "new year!";

 result = string1.lastIndexOf(string2);
 System.out.println("Vị trí cuối cùng xuất hiện chuỗi " + string2 +
 " trong chuỗi " + string1 + " = " + result);
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_12 [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (Sep 14, 2020 5:24:22 PM – 5:24:22 PM)
Vị trí cuối cùng xuất hiện chuỗi new year! trong chuỗi Happy new year and new year! = 19
```

#### 6.2.2.7. Hàm thay thế một chuỗi con trong chuỗi ký tự bằng chuỗi khác

Cú pháp:

```
string1.replace(char oldChar, char newChar);
```

Chức năng: Hàm sẽ thay thế ký tự oldChar bằng ký tự newChar trong chuỗi string1. Nếu ký tự cần thay thế không có trong chuỗi string1 thì chương trình sẽ trả về chuỗi string1.

Ví dụ 6.13:

```
public class Vidu6_13 {
 public static void main(String[] args) {
 String string1 = new String("Happy new year!");

 // ký tự thay thế 'l' không có trong chuỗi string1
 System.out.println(string1.replace('l', 'r'));

 // thay thế ký tự 'y' trong string1 thành 'r'
 System.out.println("Chuỗi sau khi thay thế là " +
 string1.replace('y', 'r'));
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_13 [Java Application] C:\Program Files\
Happy new year!
Chuỗi sau khi thay thế là Haprr new rear!
```



#### 6.2.2.8. Hàm loại bỏ các khoảng trắng thừa ở đầu và cuối chuỗi

Cú pháp:

```
String string1 = string1.trim();
```

Chức năng: Hàm sẽ loại bỏ các khoảng trắng thừa ở đầu và cuối chuỗi string1. Nếu chuỗi đó không có khoảng trắng thừa thì chương trình sẽ trả về chuỗi gốc.

Ví dụ 6.14:

```
public class Vidu6_14 {
 public static void main(String[] args) {
 String string1 = new String(" Welcome to Freetuts.net! ");

 // loại bỏ các khoảng trắng thừa trong chuỗi string1
 string1 = string1.trim();

 System.out.println("Chuỗi sau khi loại bỏ khoảng trắng thừa là " + string1);
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_14 [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (Se
Chuỗi sau khi loại bỏ khoảng trắng thừa là Welcome to Freetuts.net!
```

#### 6.2.2.9. Hàm tạo một chuỗi con từ vị trí index trong chuỗi cha

Cú pháp:

```
chuoiCon = chuoiCha.substring(int beginIndex);
String chuoiCon = chuoiCha.substring(int beginIndex, int endIndex);
```

Chức năng: Hàm sẽ tạo một chuỗi con từ vị trí có chỉ số là beginIndex trong chuỗi cha. Trong cú pháp thứ 2, thì hàm sẽ tạo một chuỗi con bắt đầu từ vị trí có chỉ số là beginIndex và kết thúc tại vị trí có chỉ số endIndex - 1 trong chuỗi cha.

Ví dụ 6.15:

```
public class Vidu6_15 {
 public static void main(String[] args) {
 String chuoiCha = new String("Welcome to Freetuts.net!");

 // tạo một chuỗi con từ vị trí 11 trong chuỗi string1
 String chuoiCon1 = chuoiCha.substring(11); // Freetuts.net!
 System.out.println(chuoiCon1);

 /*
 * tách một chuỗi con bắt đầu từ vị trí 11
 * và kết thúc tại vị trí 19 trong chuỗi cha
 */
 String chuoiCon2 = chuoiCha.substring(11, 19); // Freetuts
 System.out.println(chuoiCon2);
 }
}
```

Kết quả sau khi biên dịch chương trình:

```
<terminated> Vidu6_15 |
Freetuts.net!
Freetuts
```

### 6.2.3. Một số ví dụ về chuỗi

Ví dụ 6.16: Viết chương trình nhập từ bàn phím một chuỗi không quá 80 ký tự và một ký tự bất kỳ. Đếm và in ra màn hình số lần xuất hiện của ký tự đó trong chuỗi vừa nhập.

```
import java.util.Scanner;

public class Vidu6_16 {
 public static void main(String[] args) {
 String chuoi;
 char kyTu;
 int count = 0;
 Scanner scanner = new Scanner(System.in);

 // nếu độ dài chuỗi nhập vào còn lớn hơn 80 thì phải nhập lại
 do {
 System.out.println("Nhập vào 1 chuỗi bất kỳ: ");
 chuoi = scanner.nextLine();
 } while (chuoi.length() > 80);

 System.out.println("Nhập vào ký tự cần đếm số lần xuất hiện: ");
 kyTu = scanner.next().charAt(0);

 /*
 * đếm và in ra số lần xuất hiện của ký tự đó trong chuỗi
 * duyệt từ đầu đến cuối chuỗi
 * nếu có ký tự nào tại vị trí i bằng với ký tự ch thì tăng biến count lên 1
 */
 for (int i = 0; i < chuoi.length(); i++) {
 if (kyTu == chuoi.charAt(i)) {
 count++;
 }
 }

 System.out.println("Số lần xuất hiện của ký tự " + kyTu +
 " trong chuỗi " + chuoi + " = " + count);
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu6_16 [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe
Nhập vào 1 chuỗi bất kỳ:
nguyen van khuong
Nhập vào ký tự cần đếm số lần xuất hiện:
n
Số lần xuất hiện của ký tự n trong chuỗi nguyen van khuong = 4
```

Ví dụ 6.17: Viết chương trình nhập vào một chuỗi bất kỳ bao gồm cả số, ký tự thường và ký tự hoa từ bàn phím. Sau đó đếm và in ra số ký tự thường và ký tự hoa và số có trong chuỗi đó.

```
import java.util.Scanner;

public class Vidu6_17 {
 public static void main(String[] args) {
 String chuoi;
 int soKyTuInHoa = 0, soKyTuInThuong = 0, soChuSo = 0;
 Scanner scanner = new Scanner(System.in);

 // nếu độ dài chuỗi nhập vào còn lớn hơn 80 thì phải nhập lại
```

```
do {
 System.out.println("Nhập vào 1 chuỗi bất kỳ: ");
 chuoi = scanner.nextLine();
} while (chuoi.length() > 80);

// đếm và in ra số lần xuất hiện của ký tự đó trong chuỗi
// duyệt từ đầu đến cuối chuỗi
// nếu có ký tự nào tại vị trí i bằng với ký tự ch thì tăng biến count lên 1
for (int i = 0; i < chuoi.length(); i++) {
 // hàm isUpperCase() là hàm dùng để kiểm tra ký tự tại vị trí i
 // có phải là ký tự in hoa hay không.
 if (Character.isUpperCase(chuoi.charAt(i))) {
 soKyTuInHoa++;
 }

 // hàm isLowerCase() là hàm dùng để kiểm tra ký tự tại vị trí i
 // có phải là ký tự in thường hay không.
 if (Character.isLowerCase(chuoi.charAt(i))) {
 soKyTuInThuong++;
 }

 // hàm isDigit() là hàm dùng để kiểm tra ký tự tại vị trí i
 // có phải là số hay không.
 if (Character.isDigit(chuoi.charAt(i))) {
 soChuSo++;
 }
}

System.out.println("Trong chuỗi " + chuoi +
 " có " + soKyTuInHoa + " ký tự in hoa," +
 " có " + soKyTuInThuong + " ký tự in thường" +
 " và có " + soChuSo + " số.");
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu6_17 [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (Sep 14, 2020 5:40:57 PM - 5:4
Nhập vào 1 chuỗi bất kỳ:
Nguyen Van Khuong 1983
Trong chuỗi Nguyen Van Khuong 1983 có 3 ký tự in hoa, có 12 ký tự in thường và có 4 số.
```

### 6.3. BÀI TẬP CHƯƠNG 6

**Bài tập 6.1:** Tìm ước chung lớn nhất của 2 số nguyên dương theo thuật toán đệ quy?

**Bài tập 6.2:** Tính biểu thức sau theo thuật toán đệ quy:  $P(n)=1.3.5...(2n+1)$  với  $n \geq 0$

**Bài tập 6.3:** Tính biểu thức sau theo thuật toán đệ quy:  $S(n) = 1 + 1/2 + 1/3 + ... + 1/n$  với  $n > 0$

**Bài tập 6.4:** Viết chương trình Đổi sang hệ nhị phân của số nguyên dương  $n$  theo thuật toán đệ quy?

**Bài tập 6.5:** Viết chương trình đếm số lượng chữ số nguyên dương  $n$  theo thuật toán đệ quy?

**Bài tập 6.6:** Viết chương trình đếm số từ trong một chuỗi trong Java. Mỗi từ cách nhau bởi một khoảng trắng (tab, space, ...) Ví dụ ” học java co ban den nang cao ” có 7 từ.

**Bài tập 6.7:** Viết chương trình java liệt kê số lần xuất hiện của các từ trong một chuỗi.

## CHƯƠNG 7: NGOẠI LỆ TRONG JAVA

### 7.1. TỔNG QUAN VỀ NGOẠI LỆ TRONG JAVA

#### 7.1.1. Khái niệm ngoại lệ

Ngoại lệ (*Exception*) là một sự kiện chỉ xảy ra trong quá trình chương trình Java thực thi một câu lệnh nào đó và thông thường nó sẽ phá vỡ luồng làm việc của chương trình, tức là chương trình đang chạy sẽ lập tức ngừng lại và xuất hiện thông báo lỗi. Đó chính là Exception (ngoại lệ).

Ví dụ dễ hiểu nhất về Exception đó chính là khi chúng ta tiến hành thực hiện phép chia một số nguyên dương cho số 0 thì khi biên dịch chương trình sẽ làm phát sinh lỗi và đó được coi là ngoại lệ.

#### 7.1.2. Các loại Exception trong Java

Trong Java có 2 loại Exception là Checked Exception và Unchecked Exception.

- Checked Exception: là các Exception xảy ra tại thời điểm Compile time (là thời điểm chương trình đang được biên dịch). Những Exception này thường liên quan đến lỗi cú pháp (syntax) và bắt buộc chúng ta phải "bắt" (catch) nó.

- Unchecked Exception: là các Exception xảy ra tại thời điểm Runtime (là thời điểm chương trình đang chạy). Những Exception này thường liên quan đến lỗi logic và không bắt buộc chúng ta phải "bắt" (catch) nó.

Sau đây tôi sẽ đưa ra 2 ví dụ minh họa Checked Exception và Unchecked Exception:

##### CheckedException.java:

```
public class CheckedException {

 public static void main(String[] args) {
 System.out.println(ABC);
 }
}
```

Lúc này, ngay tại dòng code `System.out.println(ABC);` sẽ bị lỗi. Lý do là vì ví dụ này mục đích là để hiển thị một chuỗi, mà đã hiển thị chuỗi thì bắt buộc chuỗi đó phải nằm trong cặp dấu `" "`. Đây chính là Checked Exception, lúc này nếu chúng ta tiến hành chạy chương trình thì sẽ có thông báo lỗi hiển thị trong cửa sổ Console. Kết quả biên dịch chương trình như sau:

| 1 error, 11 warnings, 0 others       |                       |
|--------------------------------------|-----------------------|
| Description                          | Resource              |
| ▼  Errors (1 item)                   |                       |
| ABC cannot be resolved to a variable | CheckedException.java |

##### UncheckedException.java:

```
public class UncheckedException {

 public static void main(String[] args) {
 int a = 5, b = 0;
 System.out.println(a/b);
 }
}
```

Lúc này, chương trình sẽ không báo lỗi gì trong đoạn code của chúng ta nhưng khi biên dịch thì sẽ có thông báo lỗi “/ by zero” (lỗi chia cho 0) trong màn hình Console. Đây chính là Unchecked Exception. Kết quả sau khi biên dịch chương trình:

```
<terminated> UncheckedException [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\java.exe
Exception in thread "main" java.lang.ArithmeticException: / by zero
 at vidu.UncheckedException.main(UncheckedException.java:7)
```

### 7.1.3. Các cách xử lý Exception

Trong Java, để xử lý ngoại lệ chúng ta sẽ có các cách được liệt kê dưới đây và chi tiết về các cách này tôi sẽ giới thiệu trong phần sau.

- Sử dụng khối try...catch để xử lý.
- Sử dụng multicatch để bắt nhiều ngoại lệ.
- Sử dụng khối try...catch...finally.
- Sử dụng try with resource.
- Sử dụng Nested try (lồng một khối try trong một try khác).
- Sử dụng từ khóa throw và throws.

## 7.2. TRY – CATCH XỬ LÝ NGOẠI LỆ

### 7.2.1. Cú pháp try - catch trong Java

```
try
{
 // Các câu lệnh có thể tạo ra exception
}
catch (exception(type) e(object))
{
 // Code xử lý exception e
}
```

Try block chứa tập hợp các câu lệnh có thể xảy ra ngoại lệ.

Catch block là nơi bạn xử lý các exception, nó phải đi kèm với try block.

### 7.2.2. Ví dụ về try catch

Nếu một exception xảy ra trong try block thì chương trình sẽ thực thi các câu lệnh trong catch block tương ứng. Một try block có thể có nhiều catch block được liên kết với nó, bạn nên sắp xếp các catch block hợp lý sao cho catch block xử lý exception chung ở cuối cùng (xem trong ví dụ bên dưới).

Một exception chung có thể xử lý tất cả các ngoại lệ nhưng bạn nên đặt ở cuối, nếu bạn đặt nó ở trước tất cả các catch block khác thì nó sẽ hiển thị một thông báo chung cho tất cả trường hợp. Chắc chắn bạn sẽ không muốn điều đó xảy ra đâu.

Ví dụ 7.1:

```
class Vidu7_1 {
 public static void main(String args[]) {
 int num1, num2;
 try {
 /* Ta nghi ngờ rằng khối lệnh này sẽ throw ra
 * exception vì vậy ta sẽ xử lý nó bằng cách đặt các câu lệnh này
 * bên trong try block và xử lý exception trong catch block
 */
```

```
 num1 = 0;
 num2 = 62 / num1;
 System.out.println(num2);
 System.out.println("Kết thúc try block.");
 }
 catch (ArithmeticException e) {
 /* Catch block này sẽ chỉ thực hiện nếu có ngoại lệ số học
 * xảy ra trong try block
 */
 System.out.println("Lỗi: Số bị chia không thể là số 0");
 }
 catch (Exception e) {
 /* Đây là một exception chung, nó có thể xử lý
 * tất cả các exception. Catch block này sẽ thực thi nếu ngoại lệ không
 * được xử lý bởi catch block bên trên
 */
 System.out.println("Lỗi: một ngoại lệ đã xảy ra");
 }
 System.out.println("Ra khỏi try catch block.");
}
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu7_1 [Java Application] C:\Program I
Lỗi: Số bị chia không thể là số 0
Ra khỏi try catch block.
```

### 7.2.3. Nhiều catch block trong Java

Sau đây là vài quy tắc cần lưu ý khi sử dụng try catch:

Quy tắc 1. Một try block có thể nhiều catch block.

Quy tắc 2. Một catch block xử lý exception chung có thể xử lý tất cả các exception, cho dù đó là `ArrayIndexOutOfBoundsException` hay `ArithaturesException` hay `NullPulumException` hay bất kỳ loại exception nào.

```
catch(Exception e){
 // catch block này có thể xử lý tất cả các exception
}
```

Nếu bạn đang thắc mắc tại sao chúng ta cần phải xử lý các exception khác trong khi có thể code gọn hơn bằng cách catch exception chung cho tất cả trường hợp. Thì bạn phải nhớ rằng việc catch một exception chung sẽ chỉ hiển thị cùng một thông báo cho tất cả các ngoại lệ, và người dùng hoặc có thể là bạn sẽ không biết được exception nào đang xảy ra. Đó là lý do bạn nên đặt catch block xử lý exception chung ở cuối.

Quy tắc 3. Nếu không có exception xảy ra trong try block thì catch block hoàn toàn bị bỏ qua.

Quy tắc 4. Các catch block tương ứng thực thi cho exception cụ thể đó:

- catch (`ArithaturesException e`) dùng để xử lý ngoại lệ `ArithaturesException`
- catch (`NullPulumException e`) dùng để xử lý ngoại lệ `NullPulumException`

Quy tắc 5. Bạn cũng có thể throw exception, đây là phần nâng cao.

Ví dụ 7.2: Sử dụng nhiều catch block

```
class Vidu7_2{
 public static void main(String args[]){
 try{
 int a[]=new int[7];
 a[4]=30/0;
 System.out.println("Câu lệnh in đầu tiên trong try block");
 }
 catch(ArithmeticException e){
 System.out.println("Cảnh báo: ngoại lệ ArithmeticException");
 }
 catch(ArrayIndexOutOfBoundsException e){
 System.out.println("Cảnh báo: ngoại lệ ArrayIndexOutOfBoundsException");
 }
 catch(Exception e){
 System.out.println("Cảnh báo: ngoại lệ khác");
 }
 System.out.println("Ra khỏi try-catch block...");
 }
}
```

Kết quả chạy chương trình:

```
<terminated> Vidu7_2 [Java Application] C:\Program Files\
Cảnh báo: ngoại lệ ArithmeticException
Ra khỏi try-catch block...
```

Trong ví dụ trên có nhiều catch block và các catch block này thực hiện tuần tự khi có exception xảy ra trong try block. Điều đó có nghĩa là nếu bạn đặt catch block cuối cùng catch(Exception e) ở vị trí đầu tiên, ngay sau try block thì trong trường hợp có bất kỳ exception nào, khối này sẽ thực thi vì nó có thể xử lý tất cả các exception. Catch block này nên được đặt ở cuối cùng để tránh những tình huống như vậy.

## 7.3. FINALLY XỬ LÝ NGOẠI LỆ

### 7.3.1. Cú pháp của finally block

```
try {
 // Các câu lệnh có thể tạo ra exception
}
catch {
 // Xử lý exception
}
finally {
 // Các câu lệnh sẽ thực thi bất kể có exception xảy ra hay không
}
```

Finally block, nó được sử dụng cùng với try-catch. Các câu lệnh có trong finally block sẽ luôn thực thi bất kể exception có xảy ra trong try catch hay không, chẳng hạn như việc ngắt kết nối, ngắt luồng,...v.v.. là những việc cần phải làm bất kể có exception xảy ra hay không.

### 7.3.2. Ví dụ về finally block

Ví dụ 7.3:

```
class Vidu7_3
{
 public static void main(String args[]) {
 try{
 int num=121/0;
```



```
 System.out.println(num);
 }
 catch(ArithmeticException e){
 System.out.println("Lỗi: không thể chia cho số 0");
 }
 /* Finally block sẽ luôn được thực thi
 * dù có exception hay không
 */
 finally{
 System.out.println("Đây là finally block");
 }
 System.out.println("Ra khỏi try-catch-finally");
}
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu7_3 [Java Application]
Lỗi: không thể chia cho số 0
Đây là finally block
Ra khỏi try-catch-finally
```

### 7.3.3. Try-catch-finally block

Try block nên được liên kết với catch block hoặc finally block.

Vì catch block thực hiện xử lý exception và finally block thực hiện việc dọn dẹp, nên cách tốt nhất là sử dụng cả hai.

Cú pháp:

```
try {
 // Các câu lệnh có thể tạo ra exception
}
catch (...) {
 // Xử lý exception
}
finally {
 // Các câu lệnh được thực thi bất kể có exception hay không
}
```

Ví dụ 7.4: Ví dụ minh họa hoạt động của finally block khi không có exception xảy ra.

```
class Vidu7_4{
 public static void main(String args[]){
 try{
 System.out.println("Câu lệnh đầu tiên của try block");
 int num=45/3;
 System.out.println(num);
 }
 catch(ArrayIndexOutOfBoundsException e){
 System.out.println("ArrayIndexOutOfBoundsException");
 }
 finally{
 System.out.println("Finally block");
 }
 System.out.println("Ra ngoài try-catch-finally block");
 }
}
```



Kết quả khi chạy chương trình:

```
<terminated> Vidu7_4 [Java Application] C:\Program |
Câu lệnh đầu tiên của try block
15
Finally block
Ra ngoài try-catch-finally block
```

Ví dụ 7.5: Ví dụ minh họa hoạt động của finally block khi có exception xảy ra trong try block nhưng không được xử lý trong catch block.

```
class Vidu7_5{
 public static void main(String args[]){
 try{
 System.out.println("Câu lệnh đầu tiên của try block");
 int num=45/0;
 System.out.println(num);
 }
 catch(ArrayIndexOutOfBoundsException e){
 System.out.println("ArrayIndexOutOfBoundsException");
 }
 finally{
 System.out.println("Finally block");
 }
 System.out.println("Ra ngoài try-catch-finally block");
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu7_5 [Java Application] C:\Prc
Câu lệnh đầu tiên của try block
Finally block
```

Ví dụ 7.6: Khi xảy ra exception trong try block và cách xử lý trong catch block.

```
class Vidu7_6{
 public static void main(String args[]){
 try{
 System.out.println("Câu lệnh đầu tiên của try block");
 int num=45/0;
 System.out.println(num);
 }
 catch(ArithmeticException e){
 System.out.println("ArithmeticException");
 }
 finally{
 System.out.println("Finally block");
 }
 System.out.println("Ra ngoài try-catch-finally block");
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu7_6 [Java Application] C:\Pro
Câu lệnh đầu tiên của try block
ArithmeticException
Finally block
Ra ngoài try-catch-finally block
```

#### 7.3.4. Vài lưu ý về *finally* block

Finally block phải được dùng chung với try block, bạn không thể sử dụng nó mà không có try block. Bạn nên đặt các câu lệnh phải được thực thi bất chấp có exception hay không bên trong finally block.

Finally block có thể có hoặc không, try-catch block là đủ để xử lý exception, tuy nhiên nếu bạn đặt finally block thì nó sẽ luôn chạy sau khi thực hiện try block.

Trong trường hợp bình thường khi không có exception trong try block thì finally block được thực thi sau try block. Tuy nhiên, nếu một exception xảy ra thì catch block được thực thi trước finally block.

Một exception trong finally block có thể chạy như bất kỳ exception nào khác bên ngoài nó.

Các câu lệnh có trong finally block thực thi ngay cả khi try block chứa các câu lệnh chuyển điều khiển như return, break hoặc continue.

### 7.4. THROW EXCEPTION TRONG JAVA

#### 7.4.1. Cú pháp của từ khóa *throw* trong Java

Java đã định nghĩa sẵn các class exception như ArithmeticException, NullPointerException, hay ArrayIndexOutOfBoundsException,...v.v.. Những exception này được dùng để catch các trường hợp khác nhau.

Ví dụ khi chúng ta chia một số cho 0, exception ArithmeticException sẽ được ném ra (throw), hoặc là khi chúng ta cố gắng truy cập phần tử của mảng mà phần tử đó không nằm trong giới hạn của nó thì chúng ta sẽ nhận được exception ArrayIndexOutOfBoundsException.

Chúng ta có thể ném một exception bằng cách sử dụng từ khóa throw:

```
throw new exception_class("error message");
```

#### 7.4.2. Ví dụ về từ khóa *throw* trong Java

Giả sử chúng ta đang cần chọn các học sinh có tuổi trên 12 hoặc cân nặng trên 40 để tham dự giải hội thao, nếu bất kỳ điều kiện nào kể trên không được đáp ứng thì từ chối học sinh đó, lúc đó ta sẽ throw ArithmeticException với cảnh báo "Student is not eligible for registration".

Chúng ta kiểm tra bằng method checkEligibility(int stuage, int stuweight) bên dưới.

Ví dụ 7.7:

```
public class Vidu7_7 {
 static void checkEligibility(int stuage, int stuweight){
 if(stuage<12 && stuweight<40) {
 throw new ArithmeticException("Student is not eligible for registration");
 }
 else {
 System.out.println("Student Entry is Valid!!");
 }
 }

 public static void main(String args[]){
 System.out.println("Welcome to the Registration process!!");
 checkEligibility(10, 39);
 System.out.println("Have a nice day..");
 }
}
```

Kết quả chạy chương trình:

```
Vidu7_7 [Java Application] C:\Program Files\Java\jre1.8.0_2
Welcome to the Registration process!!
```

Sẽ thông báo lỗi ở dòng: `throw new ArithmeticException("Student is not eligible for registration");`

Trong ví dụ trên, chúng ta đã tự throw ra một unchecked exception, loại exception này được bỏ qua trong quá trình biên dịch, không bắt buộc ta phải xử lý nó. Bạn cũng có thể throw một user-defined exception bằng cách tương tự như ví dụ trên.

## 7.5. CUSTOM EXCEPTION TRONG JAVA

### 7.5.1. Giới thiệu

Trong phần trước chúng ta đã học cách sử dụng throw để ném ra exception, ở phần này ta sẽ tiếp tục dùng đến nó để ném ra các custom exception mà người dùng tự định nghĩa.

Chúng ta có thể tự tạo ra các exception class của riêng mình và ném exception đó bằng cách sử dụng từ khóa throw. Những exception này được gọi là user-defined exception hoặc custom exception.

### 7.5.2. Ví dụ về user-defined exception trong Java

Ví dụ 7.8:

```
/* This is my Exception class, I have named it MyException
 * you can give any name, just remember that it should, extend Exception class
 */
class MyException extends Exception{
 String str1;
 /* Constructor of custom exception class
 * here I am copying the message that we are passing while
 * throwing the exception to a string and then displaying
 * that string along with the message.
 */
 MyException(String str2) {
 str1=str2;
 }
 public String toString(){
 return ("MyException Occurred: "+str1) ;
 }
}

class Vidu7_8{
 public static void main(String args[]){
 try{
 System.out.println("Starting of try block");
 // I'm throwing the custom exception using throw
 throw new MyException("This is My error Message");
 }
 catch(MyException exp){
 System.out.println("Catch Block") ;
 System.out.println(exp) ;
 }
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu7_8 [Java Application] C:\Program Files\Java\jre1.8
Starting of try block
Catch Block
MyException Occurred: This is My error Message
```

Bạn có thể thấy rằng tôi đã tự throw ra một user-defined exception bằng câu lệnh `throw new MyException("This is My error Message")`.

*Chú ý rằng:*

- User-defined exception nghĩa phải extend từ Exception class.
- Exception được ném ra bằng cách sử dụng từ khóa `throw`.

## 7.6. BÀI TẬP CHƯƠNG 7

**Bài tập 7.1:** Cho một ví dụ về Ngoại lệ số học (ArithmeticException)

Hướng dẫn:

```
class Baitap7_1
{
 public static void main(String args[])
 {
 try{
 int num1=30, num2=0;
 int output=num1/num2;
 System.out.println ("Result: "+output);
 }
 catch(ArithmeticException e){
 System.out.println ("You Shouldn't divide a number by zero");
 }
 }
}
```

**Bài tập 7.2:** Cho một ví dụ về Ngoại lệ ArrayIndexOutOfBoundsException

Hướng dẫn:

```
class Baitap7_2
{
 public static void main(String args[])
 {
 try{
 int a[]=new int[10];
 //Array has only 10 elements
 a[11] = 9;
 }
 catch(ArrayIndexOutOfBoundsException e){
 System.out.println ("ArrayIndexOutOfBoundsException");
 }
 }
}
```

**Bài tập 7.3:** Cho một ví dụ về NumberFormatException

**Bài tập 7.4:** Cho một ví dụ về Ngoại lệ StringIndexOutOfBoundsException

**Bài tập 7.5:** Cho một ví dụ về Ngoại lệ Ngoại lệ `StringIndexOutOfBoundsException`

Viết chương trình nhập vào 2 số thực. Bắt ngoại lệ để khi nhập vào không phải là số.

Cài đặt hàm chia, trong đó bắt ngoại lệ nếu số chia là 0 thì thông báo phép chia không hợp lệ và kết thúc chương trình.

**Bài tập 7.6:** Cho một ví dụ về Ngoại lệ Ngoại lệ `StringIndexOutOfBoundsException`

Khai báo 1 mảng có n phần tử các số nguyên, viết hàm nhập các phần tử cho mảng. Bắt ngoại lệ nếu nhập phần tử có giá trị là 100 thì in ra các phần tử đã nhập và kết thúc chương trình.

## CHƯƠNG 8: LUỒNG VÀ TẬP TIN TRONG JAVA

### 8.1. LUỒNG TRONG JAVA

#### 8.1.1. Giới thiệu về luồng

Trong Java, luồng (thread) về cơ bản là một tiến trình con (sub-process). Luồng là đơn vị nhỏ nhất trong chương trình có thể thực hiện được một công việc riêng biệt và các luồng này được quản lý bởi máy ảo Java. Một luồng gồm có 4 thành phần chính đó là: định dạng, một bộ đếm chương trình, một tập thanh ghi và ngăn xếp. Một ứng dụng Java ngoài luồng chính có thể có các luồng khác thực thi đồng thời.

Multithreading được gọi là đa luồng, còn Multitasking được gọi là đa tiến trình. Đa luồng và đa tiến trình được sử dụng để tạo ra hệ thống đa nhiệm (multitasking). Một chương trình được gọi là đa luồng khi chương trình đó có 2 luồng trở lên chạy song song với nhau và một luồng (thread) là đơn vị nhỏ nhất của tiến trình (process). Đa luồng trong Java giúp công việc được hoàn thành một cách nhanh chóng.

Tóm lại, đa luồng trong Java có thể hiểu đơn giản là xử lý nhiều luồng dữ liệu song song với nhau để thực hiện các nhiệm vụ khác nhau cùng một lúc. Tuy nói là cùng một lúc nhưng thời gian để chuyển qua lại giữa các luồng đó vẫn có độ trễ nhưng rất ngắn (chỉ tính bằng đơn vị nano giây).

Để có thể hiểu về đa luồng, tôi có một ví dụ đơn giản như sau: Ta có 2 tiến trình 1 và 2, khi chạy thì tiến trình 1 sẽ được chạy trong một khoảng thời gian nhất định rồi tạm dừng rồi chuyển sang chạy tiến trình 2 và tiến trình 2 cũng chạy trong một khoảng thời gian nhất định và chuyển về tiến trình 1. Quá trình này được thực hiện liên tục đến khi nào 1 trong 2 tiến trình kết thúc. Lưu ý là quá trình chuyển đổi qua lại giữa 2 tiến trình này rất nhanh, thời gian để thực thi 2 tiến trình này cũng rất nhanh nên chúng ta sẽ có cảm giác hai tiến trình 1 và 2 chạy song song với nhau nhưng thực tế nó vẫn có 1 khoảng chênh lệch thời gian nhất định.

#### Các cách tạo luồng trong Java:

Java cung cấp cho chúng ta 1 lớp có tên là Thread. Thread là 1 lớp có thể tạo ra 1 lớp chạy đa tiến trình được. Trong Java, chúng ta có 2 cách chính để tạo luồng đó là:

- Tạo 1 đối tượng của lớp kế thừa lớp Thread.
- implements từ 1 Interface có tên là Runnable.

Hai cách trên có những điểm giống nhau nhưng cũng có những điểm khác nhau như sau:

- Giống nhau: đều cùng được dùng để tạo luồng.
- Khác nhau: Nếu chúng ta tạo luồng bằng cách tạo 1 lớp kế thừa từ lớp Thread thì chúng ta sẽ không thể kế thừa thêm 1 lớp nào khác vì ngôn ngữ lập trình Java không hỗ trợ tính đa kế thừa. Còn nếu chúng ta tạo luồng bằng cách implements Interface Runnable thì chúng ta có thể kế thừa một lớp khác ngoài lớp Thread.

#### 8.1.2. Tạo và quản lý luồng trong Java

##### 8.1.2.1. Tạo luồng bằng cách kế thừa từ lớp Thread

Để tạo luồng bằng cách kế thừa từ lớp Thread, chúng ta phải tạo một lớp kế thừa từ lớp Thread. Như đã nói trong bài trước, Thread là một lớp có thể tạo ra 1 lớp chạy đa tiến trình

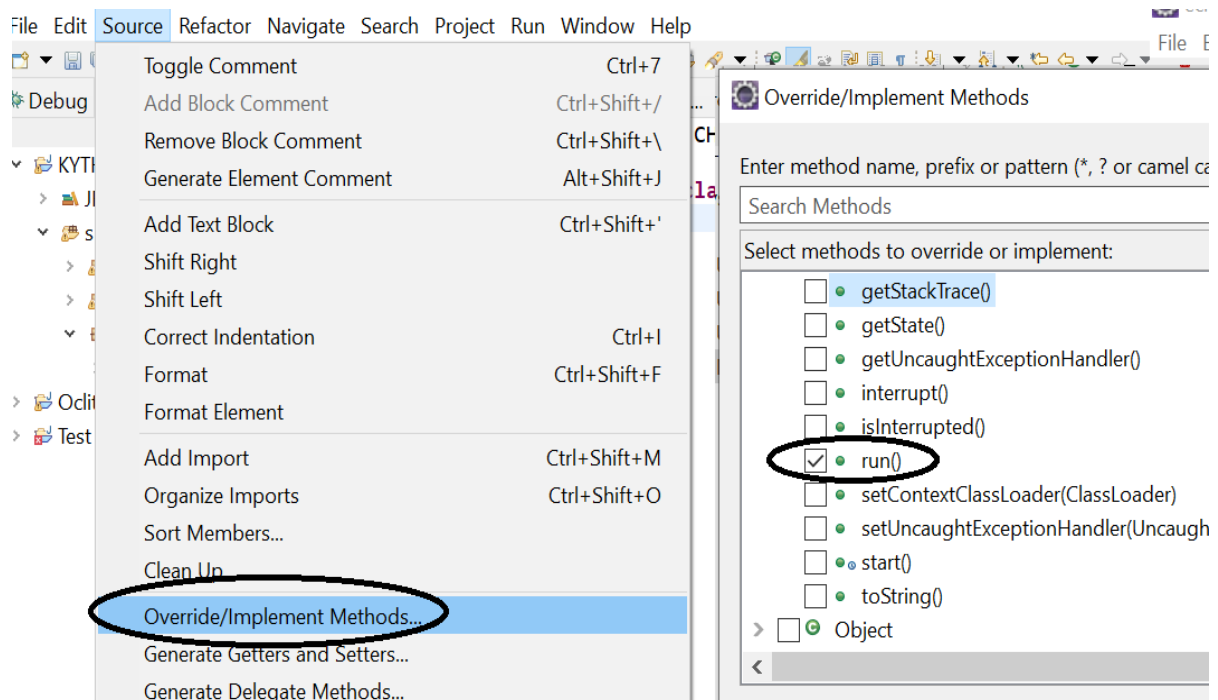
được. Trong ví dụ dưới đây tôi tạo ra một lớp có tên là MyThread kế thừa từ lớp Thread và dĩ nhiên lúc này lớp MyThread sẽ là một lớp có thể tạo luồng được.

### MyThread.java:

```
public class MyThread extends Thread {

}
```

Sau đó chúng ta sẽ tiến hành ghi đè phương thức run() của lớp Thread. Những gì có trong phương thức run() này sẽ được thực thi khi luồng bắt đầu chạy. Để ghi đè phương thức này, chúng ta vào Source → Override/Implement Methods, sau đó chọn phương thức run() và nhấn OK để kết thúc.



Lúc này lớp MyThread sẽ có nội dung như sau:

```
public class MyThread extends Thread {

 @Override
 public void run() {
 // TODO Auto-generated method stub
 super.run();
 }

}
```

Trong phương thức run(), để đơn giản tôi sẽ viết 1 vòng lặp for duyệt i từ 0 đến 4. Mỗi lần chạy sẽ hiển thị tên luồng đang chạy thông qua dòng: System.out.println(Thread.currentThread().getName()); với kết quả hiển thị không giống nhau. Tức là luồng trong Java sẽ được chạy dưới dạng bất đồng bộ, chúng ta không biết được luồng nào chạy trước và luồng nào chạy sau (phụ thuộc vào hệ điều hành. Hệ điều hành sẽ quyết định tiến trình nào được chạy trước và với mỗi lần chạy thì kết quả sẽ khác nhau, do đó đối với đa luồng thì chúng ta rất khó sửa lỗi).

Lúc này lớp MyThread sẽ có nội dung như sau:

```
public class MyThread extends Thread {

 @Override
 public void run() {
 // TODO Auto-generated method stub
 super.run();
 for (int i = 0; i < 5; i++) {
 // Thread.currentThread().getName(): cho chúng ta biết tên luồng đang chạy
 // và tên luồng này có thể thay đổi được.
 System.out.println(Thread.currentThread().getName());
 }
 }
}
```

Ví dụ 8.1: Tạo chương trình **Vidu8\_1.java** để kiểm tra luồng đã tạo

```
public class Vidu8_1{

 public static void main(String[] args) {
 // Tạo ra luồng myThread0 từ lớp MyThread
 MyThread myThread0 = new MyThread();
 myThread0.start(); // kích hoạt luồng

 // Tạo ra luồng myThread1 từ lớp MyThread
 MyThread myThread1 = new MyThread();
 myThread1.start();

 // Tạo ra luồng myThread2 từ lớp MyThread
 MyThread myThread2 = new MyThread();
 myThread2.setName("Luồng 2"); // thay đổi tên luồng thành Luồng 2
 myThread2.start();
 }
}
```

Kết quả chạy chương trình:

```
<terminated> Vidu8_1 [Java Application]
Thread-0
Thread-0
Thread-0
Luồng 2
Luồng 2
Luồng 2
Luồng 2
Luồng 2
Thread-1
Thread-1
Thread-1
Thread-1
Thread-1
Thread-0
Thread-0
```

Trong lớp Vidu8\_1.java tôi có dòng code `myThread2.setName("Luồng 2");`, đây là dòng lệnh dùng để thay đổi tên của luồng `myThread2` thành "Luồng 2", mặc định thì tên gọi của mỗi luồng sẽ là Thread-số thứ tự, với số thứ tự của luồng được bắt đầu từ 0.



*Lưu ý:* Mọi câu lệnh, nhiệm vụ mà chúng ta muốn luồng thực thi thì chúng ta phải khai báo trong phương thức run() (trong ví dụ này thì nhiệm vụ của mỗi luồng là hiển thị tên của luồng đó 5 lần) nhưng nếu chúng ta muốn thực thi luồng đó thì chúng ta phải gọi phương thức start(). Phương thức start() là phương thức dùng để cấp phát tài nguyên cho luồng rồi mới gọi phương thức run() để chạy. Nếu chúng ta không gọi phương thức start() thì những câu lệnh có trong run() sẽ không được chạy.

#### 8.1.2.2. Tạo luồng bằng cách implement Interface Runnable

Để tạo luồng bằng cách implement Interface Runnable, chúng ta phải tạo một lớp implement Interface này. Trong ví dụ dưới đây tôi tạo ra một lớp có tên là DemoThread implement Interface Runnable và dĩ nhiên lúc này lớp DemoThread sẽ là một lớp có thể tạo luồng được.

##### **DemoThread.java:**

```
public class DemoThread implements Runnable {

 @Override
 public void run() {
 // TODO Auto-generated method stub
 }

}
```

Tương tự như khi chúng ta tạo luồng bằng cách kế thừa từ lớp Thread thì khi tạo luồng bằng cách implement Interface Runnable thì chúng ta cũng có phương thức run() và những gì có trong phương thức run() này sẽ được thực thi khi luồng bắt đầu chạy, chỉ khác là khi tạo từ lớp Thread thì chúng ta phải tiến hành override lại phương thức này còn đối với tạo từ Runnable thì phương thức run() này đã được tự động override lại.

Trong phương thức run() của lớp DemoThread chúng ta thêm vào đoạn code như sau:

```
public class DemoThread implements Runnable {

 @Override
 public void run() {
 // TODO Auto-generated method stub
 for (int i = 0; i < 3; i++) {
 // Thread.currentThread().getId(): lấy id của luồng đang chạy
 // dùng để phân biệt với các luồng khác cùng tiến trình or cùng tập luồng.
 // Đây là thông số mà máy ảo java tự tạo ra khi ta tạo luồng
 // nên ta không thể sửa đổi cũng như áp đặt thông số này khi tạo luồng.
 System.out.println(Thread.currentThread().getId() + "\t" +
 Thread.currentThread().getName());
 }
 }

}
```

Ví dụ 8.2: Viết chương trình Vidu8\_2.java để sử dụng luồng đã tạo

```
package CHUONG8;

public class Vidu8_2 {

 public static void main(String[] args) {
 DemoThread demoThread0 = new DemoThread();
 Thread thread0 = new Thread(demoThread0);
 thread0.start();
 }
}
```

```
DemoThread demoThread1 = new DemoThread();
Thread thread1 = new Thread(demoThread1);
thread1.setName("Luồng 1");
thread1.start();

DemoThread demoThread2 = new DemoThread();
Thread thread2 = new Thread(demoThread2);
thread2.start();
}

}
```

Kết quả chạy chương trình:

```
<terminated> Vidu8_2 [Java Application]
14 Thread-0
14 Thread-0
15 Luồng 1
16 Thread-2
16 Thread-2
16 Thread-2
15 Luồng 1
14 Thread-0
15 Luồng 1
```

Chúng ta sẽ sử dụng cách tạo luồng bằng cách implement Interface Runnable khi chúng ta muốn chia sẻ thuộc tính giữa các luồng trong chương trình. Ví dụ dưới đây sẽ minh họa điều này:

Ví dụ 8.3: Tạo luồng **ShareThread.java**:

```
public class ShareThread implements Runnable {

 private int shareVariable = 0; // thuộc tính sử dụng chung

 public int getShareVariable() {
 return shareVariable;
 }

 @Override
 public void run() {
 for (int i = 0; i < 3; i++) {
 System.out.println("ID:" + Thread.currentThread().getId() +
 ", Name: " + Thread.currentThread().getName()
 + ", shareVariable = " + shareVariable);
 shareVariable += 2;
 }
 }
}
```

Viết chương trình Vidu8\_3.java để sử dụng luồng đã tạo:

```
package CHUONG8;

public class Vidu8_3 {

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

```

 ShareThread shareThread = new ShareThread();

 Thread thread0 = new Thread(shareThread);
 thread0.setName("Luồng 1");
 thread0.start();

 Thread thread1 = new Thread(shareThread);
 thread1.setName("Luồng 2");
 thread1.start();

 Thread thread2 = new Thread(shareThread);
 thread2.setName("Luồng 3");
 thread2.start();

 System.out.println("Giá trị thuộc tính shareVariable = " +
shareThread.getShareVariable());
 }

}

```

Kết quả khi chạy chương trình:

```

<terminated> Vidu8_3 [Java Application] C:\Program Files'
Giá trị thuộc tính shareVariable = 0
ID:16, Name: Luồng 3, shareVariable = 0
ID:14, Name: Luồng 1, shareVariable = 0
ID:15, Name: Luồng 2, shareVariable = 0
ID:15, Name: Luồng 2, shareVariable = 6
ID:14, Name: Luồng 1, shareVariable = 4
ID:16, Name: Luồng 3, shareVariable = 2
ID:16, Name: Luồng 3, shareVariable = 12
ID:14, Name: Luồng 1, shareVariable = 10
ID:15, Name: Luồng 2, shareVariable = 8

```

### 8.1.3. Đồng bộ hóa đa luồng

#### 8.1.3.1. Giới thiệu về đồng bộ hóa đa luồng

Để các bạn hiểu về đồng bộ luồng trong Java, tôi đưa ra một ví dụ đơn giản cần sử dụng đến đồng bộ như sau: Giả sử một công ty cấp quyền cho 10 nhân viên có thể thực hiện giao dịch rút tiền và chuyển tiền từ tài khoản ngân hàng chung của công ty và số dư của tài khoản này hiện đang là 10.000\$. Vào lúc 9h sáng, giám đốc của công ty này đi giao dịch với khách hàng và quyết định rút 2.000\$ từ tài khoản. Trong khi hành động rút tiền của vị giám đốc này đang được thực hiện và vẫn chưa kết thúc thì vào lúc này, người kế toán trưởng của công ty bắt đầu thực hiện kiểm tra số dư tài khoản và tiến hành chuyển tiền cho đối tác của công ty với số tiền là 9.000\$.

Lúc này, vì giao dịch rút tiền của người giám đốc vẫn chưa kết thúc nên khi người kế toán trưởng kiểm tra số dư thì máy ATM vẫn trả lại số dư là 10.000\$ và giao dịch rút 9.000\$ của người này vẫn được chấp nhận. Và nếu trong trường hợp này mà không có sự đồng bộ thì cả 2 hành động rút tiền và chuyển tiền đều được thực hiện thành công và vì vậy ngân hàng sẽ mất đi số tiền là 1.000\$ → trong trường hợp này bắt buộc cần đến sự đồng bộ hóa giữa các luồng.

Tóm lại, đồng bộ luồng trong Java chính là việc sắp xếp thứ tự các luồng khi truy xuất vào cùng một đối tượng (trong ví dụ trên là số tiền trong tài khoản) sao cho không có sự xung đột

dữ liệu. Cơ chế đồng bộ này sẽ khóa hay đồng bộ dữ liệu sử dụng chung để tại một thời điểm chỉ có một luồng được thực thi (rút tiền hoặc chuyển tiền). Chỉ khi nào việc thực thi luồng này kết thúc thì luồng khác mới được thực hiện. Đây chính là cơ chế đồng bộ luồng trong Java.

#### 8.1.3.2. Đồng bộ luồng sử dụng từ khóa *synchronized*

Để đồng bộ luồng trong Java, chúng ta sẽ sử dụng từ khóa `synchronized`. Từ khóa này sẽ đứng trước kiểu trả về và đứng sau phạm vi truy cập của phương thức đó.

Ví dụ: 8.4: Để minh họa cách sử dụng từ khóa này, tôi có một ví dụ minh họa bài toán ngân hàng như sau:

```
public class Vidu8_4 {
 private int taiKhoan = 10000;

 public Vidu8_4() {
 System.out.println("Tài khoản hiện có = " + taiKhoan);
 }

 private synchronized void rutTien (int soTienRut) {
 System.out.println("Giao dịch rút tiền đang được thực hiện với" +
 " số tiền = " + soTienRut + "...");

 if(taiKhoan < soTienRut) {
 System.out.println("Số tiền trong tài khoản không đủ!");
 try {
 wait(); // phương thức wait sẽ đưa Thread rơi vào trạng thái
sleeping
 } catch (InterruptedException ie) {
 System.out.println(ie.toString());
 }
 }

 taiKhoan -= soTienRut;
 System.out.println("Rút tiền thành công. Số tiền hiện có trong tài khoản = "
+ taiKhoan);
 }

 private synchronized void nopTien(int soTienNop) {
 System.out.println("Giao dịch nộp tiền đang được thực hiện với" +
 " số tiền nộp = " + soTienNop + "...");
 taiKhoan += soTienNop;
 System.out.println("Nộp tiền thành công. Số tiền hiện có trong tài khoản = "
+ taiKhoan);
 notify();
 }

 public static void main(String[] args) {

 final Vidu8_4 customer = new Vidu8_4();

 Thread t1 = new Thread(){

 public void run() {
 customer.rutTien(20000);
 }

 };

 t1.start();
 }
}
```

```
Thread t2 = new Thread(){
 public void run() {
 customer.nopTien(30000);
 }
};

t2.start();
}
```

Kết quả chạy chương trình:

```
<terminated> Vidu8_4 [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (S
Tài khoản hiện có = 10000
Giao dịch rút tiền đang được thực hiện với số tiền = 20000...
Số tiền trong tài khoản không đủ!
Giao dịch nộp tiền đang được thực hiện với số tiền nộp = 30000...
Nộp tiền thành công. Số tiền hiện có trong tài khoản = 40000
Rút tiền thành công. Số tiền hiện có trong tài khoản = 20000
```

### Giải thích hoạt động của chương trình trên:

Trong ví dụ trên, tôi giả sử số tiền hiện có trong tài khoản ngân hàng là 10.000\$. Trong hàm main(), chúng ta có 2 luồng: luồng t1 sẽ thực hiện việc rút tiền và luồng t2 sẽ thực hiện việc nộp tiền vào tài khoản. Trong Thread t1, khách hàng rút tiền với số tiền là 20.000\$ thông qua phương thức rutTien(), lúc này số dư trong tài khoản không đủ nên hành động rút tiền này sẽ được đưa vào trạng thái sleeping thông qua dòng lệnh wait(). Sau đó Thread t2 sẽ được thực thi, lúc này khách hàng sẽ nộp vào tài khoản với số tiền là 40.000\$ thông qua phương thức nopTien().

Trong phương thức nopTien() này, khi đã nộp tiền thành công thì dòng lệnh notify() sẽ đánh thức Thread đứng trước nó đang ở trạng thái sleeping vì phương thức wait() bị gọi (ở đây là Thread t1). Lúc này phương thức rutTien() sẽ kiểm tra số dư tài khoản là 40.000\$ > số tiền cần rút là 20.000\$ thì hành động rút tiền này sẽ thành công và số dư tài khoản còn lại là 20.000\$.

## 8.2. TẬP TIN TRONG JAVA

### 8.2.1. Cách tạo một file

Trong bài này chúng ta sẽ học cách tạo một file trong Java bằng phương thức createNewFile(). Phương thức này tạo ra một file trống, nếu file không tồn tại ở vị trí đã chỉ định và trả về giá trị true. Nếu tập tin đã có thì phương thức này trả về false.

Nó ném ra ngoại lệ:

- IOException - Nếu xảy ra lỗi Input / Output trong quá trình tạo file.
- SecurityException - Nếu một trình quản lý bảo mật tồn tại và phương thức SecurityManager.checkWrite (java.lang.String) của nó từ chối quyền truy cập ghi vào file.

Đoạn mã dưới đây sẽ tạo ra một file txt có tên là newfile.txt nằm trong ổ đĩa D. Bạn có thể thay đổi đường dẫn trong đoạn mã dưới đây để tạo file trong thư mục khác hoặc trong ổ đĩa khác.

Ví dụ 8.5:

```
import java.io.File;
import java.io.IOException;

public class Vidu8_5
{
 public static void main(String[] args)
 {
 try {
 File file = new File("D:\\newfile.txt");
 /*If file gets created then the createNewFile()
 * method would return true or if the file is
 * already present it would return false
 */
 boolean fvar = file.createNewFile();
 if (fvar){
 System.out.println("File has been created successfully");
 }
 else{
 System.out.println("File already present at the specified location");
 }
 } catch (IOException e) {
 System.out.println("Exception Occurred:");
 e.printStackTrace();
 }
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu8_5 [Java Application] C:\Program F
File has been created successfully
```

### 8.2.2. Cách đọc file

Chúng ta sẽ đọc một file trong Java bằng FileInputStream và BufferedInputStream. Dưới đây là các bước chi tiết mà tôi đã thực hiện trong đoạn mã dưới đây:

- Tạo một file bằng cách cung cấp đường dẫn đầy đủ của file (mà chúng ta sẽ đọc) File file = new File("C://myfile.txt");.
- Truyền đối tượng file làm tham số cho đối tượng FileInputStream fis = new FileInputStream(file);
- Truyền đối tượng FileInputStream cho BufferedInputStream để tạo BufferedInputStream.
- Sử dụng vòng lặp while để đọc file. Phương thức có available() được sử dụng để kiểm tra phần cuối của file vì nó trả về 0 khi con trỏ đến cuối file. Đọc nội dung của file bằng phương thức read () của FileInputStream.

Ví dụ 8.6: Đã có tập tin myfile.txt ở ổ đĩa D. Viết chương trình đọc tập tin đó.

```
import java.io.*;
public class Vidu8_6 {
 public static void main(String[] args) {
 //Specify the path of the file here
 File file = new File("D://myfile.txt");
 BufferedInputStream bis = null;
 FileInputStream fis= null;

 try
```

```

{
 //FileInputStream to read the file
 fis = new FileInputStream(file);

 /*Passed the FileInputStream to BufferedInputStream
 For Fast read using the buffer array./
 bis = new BufferedInputStream(fis);

 /*available() method of BufferedInputStream
 * returns 0 when there are no more bytes
 * present in the file to be read*/
 while(bis.available() > 0){
 System.out.print((char)bis.read());
 }

}catch(FileNotFoundException fnfe)
{
 System.out.println("The specified file not found" + fnfe);
}
catch(IOException ioe)
{
 System.out.println("I/O Exception: " + ioe);
}
finally
{
 try{
 if(bis != null && fis!=null)
 {
 fis.close();
 bis.close();
 }
 }catch(IOException ioe)
 {
 System.out.println("Error in InputStream close(): " + ioe);
 }
}
}
}

```

Kết quả chạy chương trình:

```

<terminated> Vidu8_6 [Java Application]
Nguyen Van Khuong

```

### 8.2.3. Cách ghi file

Chúng ta sẽ sử dụng phương thức write () của FileOutputStream để ghi nội dung vào file đã chỉ định. Đây là cú pháp của phương thức write ().

```
public void write(byte[] b) throws IOException
```

Nó ghi các byte b.length từ mảng byte của tham số đầu vào. Như bạn có thể thấy phương thức này cần truyền tham số định dạng byte để ghi chúng vào một file. Do đó, chúng ta sẽ cần phải chuyển đổi nội dung của mình thành mảng byte trước khi ghi nó vào file.

Ví dụ 8.7: Trong ví dụ dưới đây, chúng ta đang ghi một chuỗi vào một file. Để chuyển đổi chuỗi thành một mảng byte, chúng ta sử dụng phương thức getBytes () của lớp String.

```
import java.io.File;
import java.io.FileOutputStream;
```



```
import java.io.IOException;

public class Vidu8_7 {
 public static void main(String[] args) {
 FileOutputStream fos = null;
 File file;
 String mycontent = "This is my Data which needs" + " to be written into the
file";

 try {
 //Specify the file path here
 file = new File("D:/myfile.txt");
 fos = new FileOutputStream(file);

 /* This logic will check whether the file
 * exists or not. If the file is not found
 * at the specified location it would create
 * a new file*/
 if (!file.exists()) {
 file.createNewFile();
 }

 /*String content cannot be directly written into
 * a file. It needs to be converted into bytes
 */
 byte[] byteArray = mycontent.getBytes();

 fos.write(byteArray);
 fos.flush();
 System.out.println("File Written Successfully");
 }
 catch (IOException ioe) {
 ioe.printStackTrace();
 }
 finally {
 try {
 if (fos != null)
 {
 fos.close();
 }
 }
 catch (IOException ioe) {
 System.out.println("Error in closing the Stream");
 }
 }
 }
}
```

Kết quả chạy chương trình:

```
<terminated> Vidu8_7 [Java Application] t
File Written Successfully
```

#### 8.2.4. Cách ghi nối thêm nội dung vào file

Có hai cách để nối thêm:

- Sử dụng FileWriter và BufferedWriter: Trong cách này chúng ta sẽ có một hoặc nhiều chuỗi, và sẽ nối các chuỗi đó vào file. File có thể được ghi thêm chỉ bằng đối tượng FileWriter, tuy nhiên nếu sử dụng thêm BufferedWriter thì sẽ tăng tốc độ ghi file.



- Sử dụng `PrintWriter`: Đây là một trong những cách tốt nhất để chắp thêm nội dung vào một file. Bất cứ điều gì bạn viết bằng cách sử dụng đối tượng `PrintWriter` sẽ được thêm vào file.

Ví dụ 8.8: Sử dụng `PrintWriter` để ghi nối tập tin

```
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.BufferedWriter;
import java.io.IOException;

class Vidu8_8{
 public static void main(String[] args)
 {
 try{
 File file =new File("D://myfile.txt");
 if(!file.exists()){
 file.createNewFile();
 }
 FileWriter fw = new FileWriter(file,true);
 BufferedWriter bw = new BufferedWriter(fw);
 PrintWriter pw = new PrintWriter(bw);
 //This will add a new line to the file content
 pw.println("");
 /* Below three statements would add three
 * mentioned Strings to the file in new lines.
 */
 pw.println("This is first line");
 pw.println("This is the second line");
 pw.println("This is third line");
 pw.close();

 System.out.println("Data successfully appended at the end of file");

 }catch(IOException ioe){
 System.out.println("Exception occurred:");
 ioe.printStackTrace();
 }
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu8_8 [Java Application] C:\Program Files\Java\j
Data successfully appended at the end of file
```

Giả sử ban đầu file có nội dung là:

This is the already present content of my file

Sau khi chạy chương trình thì file có nội dung như sau:

This is the already present content of my file

This is first line

This is the second line

This is third line

### 8.2.5. Cách xóa file

Để xóa một file nào đó bằng ngôn ngữ Java thì bạn sử dụng phương thức `Delete()`, nó được tích hợp sẵn trong đối tượng `File`.

Phương thức này sẽ trả về `TRUE` nếu xóa thành công, và `false` nếu xóa thất bại.

Ví dụ 8.9: Xóa một tập tin đã có

```
import java.io.File;
public class Vidu8_9
{
 public static void main(String[] args)
 {
 try{
 //Specify the file name and path
 File file = new File("D:\\myfile.txt");
 /*the delete() method returns true if the file is
 * deleted successfully else it returns false
 */
 if(file.delete()){
 System.out.println(file.getName() + " is deleted!");
 }else{
 System.out.println("Delete failed: File didn't delete");
 }
 }catch(Exception e){
 System.out.println("Exception occurred");
 e.printStackTrace();
 }
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu8_9 [Java Application]
myfile.txt is deleted!
```

## 8.3. BÀI TẬP CHƯƠNG 8

**Bài tập 8.1:** Hãy liệt kê sự khác biệt cơ bản giữa tiến trình (process) và luồng (thread).

**Bài tập 8.2:** Tên phương thức được sử dụng khi luồng thực thi?

**Bài tập 8.3:** Khi nào gọi một phương thức để ngăn chặn trực tiếp luồng không được thực thi?

**Bài tập 8.4:** Viết chương trình đếm số lần xuất hiện của một ký tự nào đó trong file. Ký tự do người dùng nhập vào từ bàn phím, file đã có sẵn.

## CHƯƠNG 9: ĐỐI TƯỢNG VÀ LỚP TRONG JAVA

### 9.1. TỔNG QUAN VỀ ĐỐI TƯỢNG VÀ LỚP

#### 9.1.1. Đối tượng

Các bạn có thể hình dung về khái niệm đối tượng (**Object**) như sau: Trong lập trình hướng đối tượng, một thực thể được gọi là đối tượng nếu thực thể đó có 2 đặc điểm: trạng thái (state) và hành vi (behavior).

Ví dụ: là con người thì chúng ta sẽ có các trạng thái như: màu da, màu mắt, màu tóc, chiều cao, cân nặng,..., có các hành vi như: ăn, uống, làm việc,... ⇒ con người là 1 đối tượng.

Như vậy, thông qua ví dụ trên chúng ta có thể định nghĩa đối tượng như sau: đối tượng là một khái niệm được dùng để chỉ một thực thể cụ thể có trạng thái và hành vi. Một thực thể sẽ không được gọi là đối tượng nếu thực thể đó không thỏa mãn một trong các điều kiện trên.

- Trạng thái: Những đặc điểm của một đối tượng, chẳng hạn như đối tượng Sinh viên thì có mã, họ tên, ngày tháng năm sinh,...

- Hành vi: Những hành động mà một đối tượng thực hiện, ví dụ: Sinh viên thì có các hành vi như đi học, đi dã ngoại vui chơi giải trí,...

#### 9.1.2. Lớp

Lớp (**Class**) là một tập hợp các đối tượng có cùng trạng thái và hành vi, vì vậy nó định nghĩa các tính chất của một tập hợp các đối tượng cùng kiểu. Ví dụ: Lớp Student sẽ bao gồm một tập hợp các sinh viên của một lớp học, lớp Mammals sẽ bao gồm một tập hợp các động vật có vú trên thế giới,...

Mỗi đối tượng là một thể hiện (instance) của lớp. Thông thường, các đối tượng trong cùng một lớp sẽ có cùng một hành vi (behavior), nghĩa là có cách thức hoạt động tương tự nhau. Cách thức hoạt động của các đối tượng thuộc về một lớp được thiết lập nhờ vào những phương thức (method) của lớp đó.

Cấu trúc của một lớp:

```

quyền truy cập
→ {access_modifier} class TenClass {
 /* khai báo các thuộc tính của Class */
 [access_modifier] [datatype] nameVariable1;
 [access_modifier] [datatype] nameVariable2;
 ...
 [access_modifier] [datatype] nameVariableN;
 /* khai báo các phương thức của Class */
 [access_modifier] [result_type] NameOfMethod1() {
 ...
 }
 [access_modifier] [result_type] NameOfMethod2() {
 ...
 }
 [access_modifier] [result_type] NameOfMethodN() {
 ...
 }
}

```

Tên thuộc tính

Tên phương thức

giá trị trả về

- access\_modifier: phạm vi truy cập của lớp, thuộc tính và phương thức.
- Tên lớp (class name): mỗi lớp có một tên duy nhất để phân biệt với các lớp khác trong cùng một phạm vi.
- Các thuộc tính (attributes): mô tả các trường để lưu dữ liệu cho mỗi đối tượng của lớp đang mô tả hay là lưu các tham chiếu đến các đối tượng của lớp khác. Sau này khi tạo lập một đối tượng của lớp thì mỗi thành phần dữ liệu trong đối tượng sẽ chứa hay liên kết với đối tượng dữ liệu cần thiết.
- Hệ thống các phương thức của lớp (methods): mỗi phương thức của lớp thực chất là một hàm được viết riêng cho các đối tượng của lớp, chỉ được phép gọi để tác động lên chính các đối tượng của lớp này.

Dưới đây là ví dụ minh họa của 1 lớp.

```
public class Student {
 // thuộc tính (instance variable)
 private String id;
 private String name;
 private String information;

 // Phương thức này có tên inputInformation và không có giá trị trả về
 public void inputInformation() {

 }

 // Phương thức này có tên showInformation và có kiểu trả về là String
 public String showInformation() {

 }

 // hàm main
 public static void main(String[] args) {

 }
}
```

### 9.1.3. Phạm vi truy cập (Access Modifier) của lớp, thuộc tính và phương thức

Trong Java, có 4 phạm vi truy cập sau: public, private, protected và default (mặc định):

- public: có thể truy cập ở mọi nơi trong Project.
- private: chỉ có thể truy cập bên trong lớp.
- protected: truy cập được từ trong lớp khai báo, lớp con của lớp khai báo và các lớp cùng gói với lớp khai báo.
- default: truy cập được từ trong lớp khai báo và các lớp cùng gói với lớp khai báo.

### 9.1.4. Thuộc tính và phương thức

Thuộc tính là 1 biến được khai báo bên trong lớp nhưng ở bên ngoài một phương thức, hàm tạo hoặc 1 khối lệnh. Thuộc tính được khởi tạo khi một lớp được khởi tạo và có thể được sử dụng ở bên trong một hàm, hàm tạo hoặc trong một khối lệnh trong lớp đó.

*Phương thức* là một hàm được viết riêng cho các đối tượng của lớp, chỉ được phép gọi để tác động lên chính các đối tượng của lớp này. Một phương thức của lớp bao gồm 3 thành phần chính đó là phạm vi truy cập, giá trị trả về và tên của phương thức đó.

Một phương thức là một tập hợp các câu lệnh được nhóm lại với nhau để thực hiện một chức năng nào đó.

### Tạo phương thức:

Cú pháp tạo phương thức như sau:

```
[quyền_truy_cập] [kiểu_của_phương_thức] tênPhươngThức(Danh_sách_các_tham_số_truyền_vào)
{
 [Khai báo các biến bên trong phương thức];
 [Các câu lệnh bên trong];
 return [giá_trị_trả_về];
}
```

- [quyền\_truy\_cập] là phạm vi truy cập của phương thức (sang bài sau tôi sẽ giới thiệu phần này).

- [kiểu\_của\_phương\_thức] là kiểu dữ liệu trả về của phương thức đó (void, int, long, float,...).

- tênPhươngThức là tên do chúng ta tự định nghĩa và phải tuân theo quy tắc đặt tên phương thức (hàm) trong Java.

- [Danh\_sách\_các\_tham\_số\_truyền\_vào] (nếu có) là các tham số chúng ta truyền vào cho phương thức (thường nằm tại phần đầu của định nghĩa phương thức), với mỗi tham số bao gồm tên tham số và kiểu dữ liệu của tham số. Phương thức có thể có hoặc không có tham số, nếu có nhiều hơn một tham số thì các tham số được viết cách nhau bởi một dấu phẩy.

- [Các câu lệnh bên trong] là các lệnh trong thân phương thức để xử lý các công việc nhằm thực hiện chức năng của phương thức.

- [giá\_trị\_trả\_về] : giá trị trả về của phương thức phải phù hợp với [kiểu\_của\_phương\_thức]. Trong trường hợp phương thức có kiểu void thì chúng ta có thể dùng lệnh return để kết thúc hàm hoặc khi thực hiện xong lệnh cuối cùng (gặp dấu } cuối cùng) thì phương thức cũng sẽ kết thúc. Sau lệnh return có thể trả lại 1 biểu thức để tính giá trị.

### Gọi phương thức:

Một phương thức có thể được gọi thực hiện thông qua tên phương thức. Ví dụ dưới đây sẽ minh họa cách gọi phương thức trong Java:

Ví dụ 9.1:

```
public class Vidu9_1 {
 // tạo 1 phương thức có kiểu trả về là void và không có tham số truyền vào
 // để gọi phương thức này trong main() thì phải khai báo là static
 public static void nhap() {
 System.out.println("Đây là ví dụ minh họa cách gọi phương thức void
không đối trong Java");
 }

 // tạo 1 phương thức có kiểu trả về là void và có tham số truyền vào
 public static void nhapCoDoiSo(String str) {
 System.out.println(str);
 }
}
```

```
}

// tạo 1 phương thức có kiểu trả về là int và có 2 tham số truyền vào có
kiểu int
public static int tinhTong(int a, int b) {
 return a + b;
}

public static void main(String[] args) {
 nhap(); // gọi phương thức nhap()
 nhapCoDoiSo("Đây là ví dụ minh họa cách gọi phương thức void có đối số
trong Java");

 System.out.println("\nĐây là ví dụ minh họa cách gọi phương thức có
kiểu trả về" + " là int và có tham số truyền vào trong Java");
 int c = tinhTong(3, 5); // c = a + b
 System.out.println("c = " + c);
}
}
```

Kết quả khi chạy chương trình:

```
<terminated> Vidu9_1 [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (Sep 21, 2020 9:17:13 AM – 9:17:16 AM)
Đây là ví dụ minh họa cách gọi phương thức void không đối trong Java
Đây là ví dụ minh họa cách gọi phương thức void có đối số trong Java

Đây là ví dụ minh họa cách gọi phương thức có kiểu trả về là int và có tham số truyền vào trong Java
c = 8
```

### 9.1.5. Hàm tạo

Trong lập trình hướng đối tượng, chúng ta có 1 loại hàm đặc biệt đó là hàm tạo. Hàm tạo trong Java là hàm có cùng tên với lớp và không có giá trị trả về. Chúng ta hãy xem đoạn chương trình sau:

```
public class Student {
 // thuộc tính (instance variable)
 private String name;

 // hàm tạo mặc định
 public Student() {

 }

 // hàm tạo có đối số truyền vào có kiểu String và có tên là name
 public Student(String name) {

 }

 // hàm main
 public static void main(String[] args) {

 }
}
```

Một lớp trong Java có thể có một hoặc nhiều hàm tạo. Có 2 loại hàm tạo chúng ta thường gặp đó là: hàm tạo mặc định (không có đối số) và hàm tạo có đối số.

Hàm tạo mặc định là hàm tạo không có tham số đầu vào. Trong trường hợp người lập trình không định nghĩa bất kỳ hàm tạo nào thì trình biên dịch sẽ tự động tạo ra một phương thức tạo lập mặc định cho lớp này.

Hàm tạo có đối số là các hàm tạo do người dùng định nghĩa với các tham số đầu vào khác nhau để khởi tạo dữ liệu cho đối tượng.

Ví dụ trong đoạn chương trình trên thì lớp Student có hàm public Student() và public Student(String name) là hàm tạo. Trong khi viết chương trình, nếu chúng ta không khởi tạo hàm tạo cho lớp thì khi đó trình biên dịch Java sẽ xây dựng 1 hàm tạo mặc định cho lớp đó.

Ý nghĩa của hàm tạo: tự động khởi tạo các thông tin mặc định khi đối tượng được tạo ra.

*Lưu ý:* Khi người lập trình khai báo bất kỳ một hàm tạo nào cho lớp thì hàm tạo mặc định sẽ không được phát sinh bởi trình biên dịch cho lớp này nữa. Trong trường hợp này, nếu muốn sử dụng hàm tạo mặc định, người lập trình phải tự viết thêm vào.

## 9.2. THAO TÁC TRÊN ĐỐI TƯỢNG VÀ PHẠM VI TRUY CẬP

### 9.2.1. Tạo mới một đối tượng

Trong Java, để tạo mới một đối tượng chúng ta sử dụng từ khóa new. Nếu tạo lập thành công, biến lưu trữ đối tượng sẽ nhận một giá trị khác NULL và đối tượng mới sẽ được lưu trữ trong biến đối tượng.

Để khởi tạo một đối tượng, chúng ta có 3 bước như sau:

Bước 1: Khai báo đối tượng với tên và kiểu dữ liệu của đối tượng đó.

Bước 2: Sử dụng từ khóa new để khởi tạo đối tượng.

Bước 3: Theo sau từ khóa new là một lời gọi đến hàm tạo (hàm tạo này là hàm tạo mặc định của lớp). Lời gọi hàm này sẽ khởi tạo giá trị cho đối tượng được khai báo.

Ví dụ 9.2: Tạo lớp **Customer** và tạo đối tượng của lớp đó:

```
public class Customer {
 // hàm tạo có đối số
 public Customer(String name) {
 System.out.println("Your name: " + name);
 }

 public static void main(String[] args) {
 // Khởi tạo đối tượng với giá trị truyền vào là "Freetuts"
 Customer customer = new Customer("Freetuts");
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Customer [Java Application] C:\P
Your name: Freetuts
```

### 9.2.2. Truy cập đến thuộc tính và phương thức của lớp

Chúng ta có thể truy cập đến các thuộc tính và phương thức của lớp thông qua đối tượng đã được tạo thông qua 2 cú pháp sau:



- Cách truy cập đến thuộc tính: [Tên đối tượng].[Tên thuộc tính];
- Cách truy cập đến phương thức của lớp: [Tên đối tượng].[Tên phương thức()];

Ví dụ 9.3: Giả sử chúng ta có đoạn chương trình bao gồm 2 lớp (**ClassOfStudent** , **Student**), 2 class tôi đặt trong package CHUONG9 như sau:

**Lớp ClassOfStudent.java:**

```
package CHUONG9;

import java.util.Scanner;

public class ClassOfStudent {
 private String name;
 private int numberOfPupils;

 public void inputYourClass() {
 Scanner scanner = new Scanner(System.in);
 System.out.println("Nhập tên lớp: ");
 name = scanner.nextLine();
 System.out.println("Nhập số sinh viên của lớp: ");
 numberOfPupils = scanner.nextInt();
 }

 public void showInformation() {
 System.out.println("Tên lớp là " + name + ", số sinh viên = " +
numberOfPupils + ".");
 }
}
```

**Lớp Student.java:**

```
package CHUONG9;

public class Student {
 public static void main(String[] args) {
 // Khởi tạo đối tượng của lớp ClassOfStudent
 ClassOfStudent classOfStudent = new ClassOfStudent();

 // truy cập phương thức inputYourClass() và showInformation() của lớp
 ClassOfStudent

 classOfStudent.inputYourClass();

 // hiển thị thông tin vừa nhập
 classOfStudent.showInformation();
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Student [Java Application] C:\Program Files\
Nhập tên lớp:
01T1
Nhập số sinh viên của lớp:
50
Tên lớp là 01T1, số sinh viên = 50.
```



Chúng ta nhận thấy trong lớp `ClassOfStudent` có 2 phương thức `inputYourClass()` và `showInformation()` được khai báo `public`. Vì vậy, 2 phương thức này có thể được truy cập thông qua đối tượng `classOfStudent` được khai báo trong lớp `Student`.

### 9.3. TÍNH ĐÓNG GÓI (ENCAPSULATION) TRONG JAVA

#### 9.3.1. Khái niệm tính đóng gói

Đóng gói là sự che giấu bên trong dữ liệu riêng của mỗi đối tượng của lớp được khai báo và chỉ được truy xuất thông qua hệ thống các phương thức có sẵn của lớp (chỉ có thể gọi những phương thức có sẵn của lớp). Vì vậy, nó còn được gọi là data hiding (nghĩa là che giấu dữ liệu).

**Tính đóng gói có những đặc điểm như sau:**

- Tạo ra cơ chế để ngăn ngừa việc gọi phương thức của lớp này tác động hay truy xuất dữ liệu của đối tượng thuộc về lớp khác.
- Dữ liệu riêng (khi được khai báo là `private`) của mỗi đối tượng được bảo vệ khỏi sự truy xuất không hợp lệ từ bên ngoài.
- Người lập trình có thể dựa vào cơ chế này để ngăn ngừa việc gán giá trị không hợp lệ vào thành phần dữ liệu của mỗi đối tượng.
- Cho phép thay đổi cấu trúc bên trong của một lớp mà không làm ảnh hưởng đến những lớp bên ngoài có sử dụng lớp đó.

**Để cài đặt tính đóng gói, chúng ta có 2 bước như sau:**

- Khai báo các thuộc tính của đối tượng trong lớp là `private` để các lớp khác không thể truy cập trực tiếp/sửa đổi được.
- Cung cấp các phương thức `getter/setter` có phạm vi truy cập là `public` để truy cập và sửa đổi các giá trị của thuộc tính trong lớp. Phương thức `getter` là phương thức truy cập vào thuộc tính của đối tượng và trả về các thuộc tính của đối tượng, còn phương thức `setter` là phương thức truy cập vào thuộc tính của đối tượng và gán giá trị cho các thuộc tính của đối tượng đó.

#### 9.3.2. Ví dụ về tính đóng gói

Ví dụ 9.4: Tạo lớp **Person** và lớp **TestPerson** để thể hiện tính đóng gói.

**Lớp Person.java:**

```
public class Person {
 // khai báo các thuộc tính của đối tượng là private
 private String cmd;
 private String hoTen;

 // tạo các phương thức getter/setter
 public String getCmd() {
 return cmd;
 }

 // this là từ khóa có ý nghĩa là một tham chiếu đặc biệt
 public void setCmd(String cmd) {
 this.cmd = cmd;
 }

 public String getHoTen() {
 return hoTen;
 }
}
```

```
 }
 public void setHoTen(String hoTen) {
 this.hoTen = hoTen;
 }
}
```

#### Lớp TestPerson.java:

```
public class TestPerson {

 public static void main(String[] args) {
 Person person = new Person();

 person.setHoTen("Trần Văn Bình");
 person.setCmnd("212321678");

 System.out.println("Tên: " + person.getHoTen() + ", số cmd: " +
person.getCmnd());
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> TestPerson [Java Application] C:\Program Files\
Tên: Trần Văn Bình, số cmd: 212321678
```

Trong lớp Person, chúng ta có đoạn code như sau:

```
public void setHoTen(String hoTen) {
 this.hoTen = hoTen;
}
```

↑                      ↑  
biến đối tượng      biến cục bộ (Local  
(Instance variable)      variable)

**this:** là từ khóa có ý nghĩa là một tham chiếu đặc biệt chiếu tới đối tượng chủ của phương thức hiện hành. Trong lớp Person, phương thức setHoTen() lấy một tham số (hay còn gọi là biến cục bộ - Local Variable) hoTen kiểu String trùng tên với thuộc tính hoTen của lớp đó, nếu chúng ta chỉ viết "hoTen" mà không có từ khóa this đằng trước thì trình biên dịch sẽ hiểu là ta đang nói đến tham số hoTen. Vì vậy, để gọi đến thuộc tính (hay còn gọi là biến đối tượng - Instance Variable), cách duy nhất là sử dụng tham chiếu this để gọi một cách tường minh.

## 9.4. TÍNH KẾ THỪA (INHERITANCE) TRONG JAVA

### 9.4.1. Khái niệm tính kế thừa

Khái niệm "kế thừa" thường được dùng trong các ngôn ngữ lập trình hướng đối tượng để chuyển quan hệ tổng quát hóa và đặc biệt hóa của thế giới thực vào các chương trình máy tính. Tính kế thừa cho phép ta xây dựng một lớp mới dựa trên các định nghĩa của một lớp đã có.

Nếu một lớp A là lớp đặc biệt hóa của lớp B (nghĩa là lớp B là lớp tổng quát hóa của lớp A) thì trong lập trình hướng đối tượng lớp A được cài đặt là lớp kế thừa của lớp B (nói ngắn gọn là "lớp A kế thừa lớp B"). Khi lớp A kế thừa lớp B thì lớp B được gọi là "lớp cơ sở" hay "lớp cha" (trong mối quan hệ kế thừa đang xét), còn lớp A thì được gọi là "lớp kế thừa", "lớp con" hay "lớp dẫn xuất".

### Lợi ích của tính kế thừa:

- Lớp con (lớp A) có thể tận dụng lại các thuộc tính và phương thức của lớp cha (lớp B) (nghĩa là các thuộc tính và phương thức của lớp B có thể được tái sử dụng bởi lớp A).
- Lớp A có thể định nghĩa thêm thuộc tính và phương thức mới của riêng nó và có thể định nghĩa lại (hay còn gọi là ghi đè phương thức, overriding) phương thức được kế thừa từ lớp B cho phù hợp với mục đích của nó.

#### 9.4.2. Ví dụ về tính kế thừa

Để cài đặt tính kế thừa, chúng ta sẽ sử dụng từ khóa **extends**.

Ví dụ 9.5: Xây dựng lớp **Calculation.java** để tính toán, tiếp tục xây dựng lớp **MyCalculation.java** kế thừa lớp **Calculation.java**.

#### Lớp **Calculation.java**:

```
public class Calculation {
 protected int c;
 public void phepCong(int a, int b) {
 c = a + b;
 System.out.println("Tổng hai số = " + c);
 }

 public void phepTru(int a, int b) {
 c = a - b;
 System.out.println("Hiệu hai số = " + c);
 }
}
```

#### Lớp **MyCalculation.java**:

```
public class MyCalculation extends Calculation {
 public void phepNhan(int a, int b) {
 c = a * b;
 System.out.println("Tích 2 số = " + c);
 }

 public void phepLuyThua(int a, int b) {
 c = (int) Math.pow(a, b);
 System.out.println(a + "^" + b + " = " + c);
 }

 public static void main(String[] args) {
 int a = 12, b = 2;
 MyCalculation myCalculation = new MyCalculation();
 myCalculation.phepCong(a, b);
 myCalculation.phepTru(a, b);
 myCalculation.phepNhan(a, b);
 myCalculation.phepLuyThua(a, b);
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> MyCalculation [Java Application] (
Tổng hai số = 14
Hiệu hai số = 10
Tích 2 số = 24
12^2 = 144
```

Trong ví dụ này, tôi tạo ra 2 lớp Calculation và MyCalculation kế thừa từ lớp Calculation. Trong lớp Calculation tôi có khai báo 1 thuộc tính c có kiểu int và phạm vi truy cập là protected nhằm mục đích cho phép các lớp con của lớp này có thể dùng lại thuộc tính c (các bạn có thể dùng từ khóa public hoặc không dùng từ khóa nhưng không được khai báo là private) cùng với 2 phương thức là phepCong(int a, int b) để thực hiện phép cộng và phepTru(int a, int b) để thực hiện phép trừ.

Lớp MyCalculation là lớp kế thừa từ lớp Calculation. Lớp này dùng lại thuộc tính c và 2 phương thức phepCong(int a, int b) và phepTru(int a, int b) của lớp Calculation, vì vậy trong lớp MyCalculation không khai báo lại c, phepCong(int a, int b) và phepTru(int a, int b) vì tất cả chúng đều được lấy lại của lớp Calculation. Ngoài ra, lớp MyCalculation còn khai báo 2 phương thức riêng của nó đó là phepNhan(int a, int b) và phepLuyThua(int a, int b) để thực hiện phép nhân và phép lũy thừa hai số.

Hàm main() của lớp MyCalculation minh họa việc khai báo và sử dụng đối tượng myCalculation (kiểu MyCalculation). Trong hàm này, tôi có 2 dòng code myCalculation.phepCong(a, b); và myCalculation.phepTru(a, b); dùng để gọi phương thức phepCong(int a, int b) và phepTru(int a, int b), mã nguồn của 2 phương thức này được kế thừa từ lớp Calculation mà không cần viết lại cho lớp MyCalculation.

## 9.5. TÍNH ĐA HÌNH (POLYMORPHISM) TRONG JAVA

### 9.5.1. Khái niệm tính đa hình

Kỹ thuật đa hình trong các ngôn ngữ lập trình hướng đối tượng tạo điều kiện cho các lập trình viên gia tăng khả năng tái sử dụng những đoạn mã nguồn được viết một cách tổng quát và có thể thay đổi cách ứng xử một cách linh hoạt tùy theo loại đối tượng.

Tính đa hình (Polymorphism) trong Java được hiểu là trong từng trường hợp, hoàn cảnh khác nhau thì đối tượng có hình thái khác nhau tùy thuộc vào từng ngữ cảnh. Đối tượng có tính đa hình được xem như một đối tượng đặc biệt vì có lúc đối tượng này trở thành một đối tượng khác và cũng có lúc đối tượng này trở thành một đối tượng khác nữa (tùy vào từng hoàn cảnh). Sự "nhập vai" vào các đối tượng khác nhau này giúp cho đối tượng đa hình ban đầu có thể thực hiện những hành động khác nhau của từng đối tượng cụ thể.

Ví dụ khi bạn ở trong trường học là sinh viên thì bạn có nhiệm vụ học, nghe giảng,..., nhưng khi bạn ở nhà thì bạn lại đóng vai trò là thành viên trong gia đình và bạn có nhiệm vụ phải làm việc nhà, rồi khi bạn vào siêu thị thì bạn đóng vai trò là khách hàng đi mua hàng. Vì vậy, chúng ta có thể hiểu đa hình của đối tượng là trong từng trường hợp, hoàn cảnh khác nhau thì đối tượng có khả năng thực hiện các công việc khác nhau.

Để thể hiện tính đa hình, chúng ta cần đảm bảo 2 điều kiện sau:

- Các lớp phải có quan hệ kế thừa với 1 lớp cha nào đó.
- Phương thức đa hình phải được ghi đè (override) ở lớp con. Tính đa hình chỉ được thể hiện ghi đã ghi đè lên phương thức của lớp cha.

### 9.5.2. Ví dụ về tính đa hình

Ví dụ 9.6: Xây dựng 3 lớp **Shape.java**, **Rectangle.java**, **Square.java** để thể hiện tính đa hình.

**Lớp Shape.java:**

```
public class Shape {
 public void show() {
 System.out.println("Đây là phương thức show() của lớp Shape");
 }
}
```

#### Lớp Rectangle.java:

```
public class Rectangle extends Shape {
 public void show() {
 System.out.println("Đây là phương thức show() của lớp Rectangle");
 }
}
```

#### Lớp Square.java:

```
public class Square extends Shape {
 public void show() {
 System.out.println("Đây là phương thức show() của lớp Square");
 }
}
```

Xây dựng lớp **Main.java** để chạy chương trình:

```
public class Main {

 public static void main(String[] args) {
 Shape shape = new Shape();
 shape.show(); // hiển thị dòng "Đây là phương thức show() của lớp Shape"

 // bản chất của shape là Shape, nhưng vì khai báo Rectangle nên chúng ta chỉ
 // nhìn thấy những gì mà Rectangle có
 // vì vậy sẽ chạy những hàm của Rectangle
 shape = new Rectangle();
 shape.show(); // hiển thị dòng "Đây là phương thức show() của lớp Rectangle"

 // tương tự lúc này shape sẽ đóng vai trò là 1 Square
 shape = new Square();
 shape.show(); // hiển thị dòng "Đây là phương thức show() của lớp Square"
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> Main [Java Application] C:\Program Files\Java\jre'
Đây là phương thức show() của lớp Shape
Đây là phương thức show() của lớp Rectangle
Đây là phương thức show() của lớp Square
```

Trong chương trình trên, tôi có tạo ra 3 lớp là Shape, Rectangle và Square với Shape là lớp cha và 2 lớp còn lại là lớp con. Cả 3 lớp này đều có chung một phương thức show() nhưng có nội dung phương thức khác nhau.

Trong lớp Main, tôi tiến hành gọi 3 phương thức show() của 3 lớp này. Nếu làm như các bài trước thì để gọi phương thức show() ứng với từng lớp thì chúng ta phải tạo một đối tượng của lớp tương ứng, nhưng đối với tính đa hình thì chúng ta không cần phải tạo ra 3 đối tượng của 3 lớp mà chúng ta sẽ chỉ cần khai báo đối tượng của lớp Shape có khả năng đóng vai trò là lớp

con thông qua 2 dòng code `shape = new Rectangle();` và `shape = new Square();`, lúc này đối tượng `shape` sẽ đóng vai trò là lớp con tương ứng. Đây chính là ý nghĩa của tính đa hình.

## 9.6. TÍNH TRỪU TƯỢNG (ABSTRACTION) TRONG JAVA

### 9.6.1. Khái niệm tính trừu tượng

Tính trừu tượng trong Java là tính chất không thể hiện cụ thể mà chỉ nêu tên vấn đề. Đó là một quá trình che giấu các hoạt động bên trong và chỉ hiển thị những tính năng thiết yếu của đối tượng tới người dùng. Ví dụ: một người sử dụng điện thoại để gửi tin nhắn thì anh ta sẽ nhập nội dung tin nhắn, thông tin người nhận và ấn nút gửi. Khi anh ta bắt đầu gửi tin thì anh ấy không biết những gì diễn ra bên trong quá trình gửi mà chỉ biết được là kết quả của tin nhắn đã được gửi đến người nhận thành công hay chưa. Vì vậy trong ví dụ này, quá trình gửi tin nhắn đã được ẩn đi và chỉ hiển thị những chức năng mà người dùng cần đó là chức năng nhập nội dung tin nhắn, thông tin người nhận, kết quả gửi tin nhắn thành công hay thất bại. Đó chính là tính trừu tượng.

#### Ưu điểm khi sử dụng tính trừu tượng để lập trình:

- Tính trừu tượng cho phép các lập trình viên loại bỏ tính chất phức tạp của đối tượng bằng cách chỉ đưa ra các thuộc tính và phương thức cần thiết của đối tượng trong lập trình, cải thiện khả năng bảo trì của hệ thống.
- Tính trừu tượng giúp chúng ta tập trung vào những cốt lõi cần thiết của đối tượng thay vì quan tâm đến cách nó thực hiện.
- Tính trừu tượng cung cấp nhiều tính năng mở rộng khi sử dụng kết hợp với tính đa hình và kế thừa trong lập trình hướng đối tượng.

Java trừu tượng hóa thông qua các lớp trừu tượng (Abstract class) và các giao diện (Interface).

### 9.6.2. Ví dụ về tính trừu tượng

Ví dụ 9.7: Tạo 3 lớp **Animal.java**, **Dog.java**, **Cat.java** thể hiện tính trừu tượng:

#### Lớp Animal.java:

```
public abstract class Animal {
 private String tiengKeu;

 public abstract void hienThiTiengKeu();
}
```

#### Lớp Dog.java:

```
public class Dog extends Animal {

 @Override
 public void hienThiTiengKeu() {
 System.out.println("Gâu");
 }
}
```

#### Lớp Cat.java:

```
public class Cat extends Animal {
```



```
@Override
public void hienThiTiengKau() {
 System.out.println("Meo");
}
}
```

Xây dựng lớp **MainTT.java** để chạy chương trình:

```
public class MainTT {

 public static void main(String[] args) {
 Dog dog = new Dog();
 dog.hienThiTiengKau();

 Cat cat = new Cat();
 cat.hienThiTiengKau();
 }
}
```

Kết quả khi chạy chương trình:

```
<terminated> MainTT [Java Application]
Gâu
Meo
```

## 9.7. BÀI TẬP CHƯƠNG 9

**Bài tập 9.1:** Tạo 1 lớp Student lưu trữ các thông tin của 1 sinh viên bao gồm họ tên, lớp, điểm 3 môn toán, lý, hóa. Sau đó tính điểm trung bình và xếp loại học lực của sinh viên đó.

```
<terminated> Baitap9_1 [Java Application] C:\Program File
Nhập họ tên sinh viên:
khuong
Nhập lớp:
01t1
Nhập điểm toán:
9
Nhập điểm lý:
4
Nhập điểm hóa:
8
Điểm trung bình 3 môn = 7.0
Khá
```

*Hướng dẫn tham khảo:*

```
package CHUONG9;

import java.util.Scanner;

public class Baitap9_1 {
 private String hoTen, lop;
 private double diemToan, diemLy, diemHoa;

 // tạo 1 phương thức nhập để nhập thông tin của 1 sinh viên
 public void nhap() {
 Scanner scanner = new Scanner(System.in); // scanner là biến cục bộ
 System.out.println("Nhập họ tên sinh viên: ");
 hoTen = scanner.nextLine();
 System.out.println("Nhập lớp: ");
```

```
 lop = scanner.nextLine();
 System.out.println("Nhập điểm toán: ");
 diemToan = scanner.nextDouble();
 System.out.println("Nhập điểm lý: ");
 diemLy = scanner.nextDouble();
 System.out.println("Nhập điểm hóa: ");
 diemHoa = scanner.nextDouble();
 }

 // phương thức tính điểm trung bình
 // phương thức này có 3 tham số truyền vào là a, b, c có kiểu double
 // và có kiểu trả về là double
 // 3 tham số này tượng trưng cho điểm toán, lý, hóa của sinh viên
 public double tinhDiemTrungBinh(double a, double b, double c) {
 return (a + b + c) / 3;
 }

 // phương thức xếp loại học lực
 // phương thức này có tham số truyền vào là điểm trung bình của sinh viên đó
 // và có kiểu trả về là String
 public String xepLoaiHocLuc(double diemTB) {
 if (diemTB >= 8 && diemTB <= 10) {
 return "Giỏi";
 }
 if (diemTB >= 6.5 && diemTB < 8) {
 return "Khá";
 }
 if (diemTB >= 5.0 && diemTB < 6) {
 return "Trung bình";
 }
 if (diemTB >= 0.0 && diemTB < 5) {
 return "Yếu";
 }
 return "Nhập sai!";
 }

 public static void main(String[] args) {
 // tạo 1 đối tượng của Student
 Baitap9_1 student = new Baitap9_1();

 // nhập thông tin sinh viên
 student.nhap();

 // tính điểm trung bình 3 môn
 System.out.println("Điểm trung bình 3 môn = " + student.tinhDiemTrungBinh(
 student.diemToan, student.diemLy, student.diemHoa));

 // hiển thị kết quả xếp loại
 System.out.println(student.xepLoaiHocLuc(
 student.tinhDiemTrungBinh(student.diemToan, student.diemLy,
student.diemHoa)));
 }
}
```

**Bài tập 9.2:** Viết chương trình tạo 1 lớp để lưu trữ chiều dài và chiều rộng của 1 hình chữ nhật và tính chu vi, diện tích hình chữ nhật đó.

**Bài tập 9.3:** Xây dựng chương trình quản lý sách trong thư viện:



Một đối tượng sách trong hệ thống quản lý thư viện có các thuộc tính: tên sách, tổng số quyền sách, số quyền sách đang cho mượn.

Xây dựng lớp Book với các thuộc tính trên và các phương thức sau: phương thức nhập liệu cho đối tượng từ bàn phím (các thông tin cần nhập là tên sách, tổng số sách, số đang cho mượn), phương thức in thông tin tên sách ra màn hình, phương thức tính số sách còn lại trong thư viện (số sách còn lại = tổng số - số đang cho mượn).

Trên cơ sở lớp đã xây dựng, viết chương trình chính thực hiện các công việc: nhập danh sách các quyền sách với số lượng các quyền sách được nhập từ bàn phím, in ra màn hình các quyền sách hiện có trong thư viện.

**Bài tập 9.4:** Viết chương trình quản lý khách đến thuê phòng của khách sạn.

Để quản lý khách hàng đến thuê phòng trọ của một khách sạn, người ta cần quản lý những thông tin sau: Số ngày trọ, loại phòng trọ, giá phòng và các thông tin cá nhân về mỗi khách trọ. Thông tin cá nhân của mỗi khách trọ bao gồm họ và tên, ngày sinh và số chứng minh nhân dân.

Hãy xây dựng lớp Nguoi để quản lý thông tin cá nhân của mỗi cá nhân.

Xây dựng lớp KhachSan để quản lý các thông tin về khách trọ.

Viết các phương thức: nhập, hiển thị thông tin về một khách trọ.

Viết chương trình chính thực hiện các công việc sau: Nhập vào một danh sách gồm n khách trọ (n nhập từ bàn phím), hiển thị ra màn hình thông tin về các cá nhân hiện đang trọ ở khách sạn đó và tính số tiền cần phải trả nếu một khách hàng trả phòng (căn cứ vào số chứng minh để tìm kiếm).

## TÀI LIỆU THAM KHẢO

[1]. Đoàn Văn Ban, *Lập trình hướng đối tượng với Java*, NXB Khoa học kỹ thuật, năm 2003.

[2]. Ngọc Tuấn, *Hướng dẫn lập trình hướng đối tượng*, NXB Thống kê, năm 2004.

[3]. Trần Đình Quế, Nguyễn Mạnh Hùng, *Lập trình hướng đối tượng*, Học viện công nghệ bưu chính viễn thông.

*Website:*

[4]. <https://freetuts.net/>

[5]. <https://viettuts.vn/>

[6]. <https://vncoder.vn/>

...

## MỤC LỤC

|                                                                                  |           |
|----------------------------------------------------------------------------------|-----------|
| <b>CHƯƠNG 1: TỔNG QUAN VỀ KỸ THUẬT LẬP TRÌNH .....</b>                           | <b>1</b>  |
| 1.1. GIỚI THIỆU VỀ CHƯƠNG TRÌNH MÁY TÍNH .....                                   | 1         |
| 1.2. NGÔN NGỮ LẬP TRÌNH.....                                                     | 1         |
| 1.2.1. Giới thiệu về ngôn ngữ lập trình .....                                    | 1         |
| 1.2.2. Một số ngôn ngữ lập trình thông dụng .....                                | 2         |
| 1.3. CÁC PHƯƠNG PHÁP LẬP TRÌNH.....                                              | 5         |
| 1.3.1. Phương pháp lập trình tuần tự (Sequential Programming).....               | 5         |
| 1.3.2. Phương pháp lập trình cấu trúc (Structured Programming) .....             | 5         |
| 1.3.3. Phương pháp lập trình thủ tục (Procedural Programming) .....              | 5         |
| 1.3.4. Phương pháp lập trình hướng đối tượng (Object-Oriented Programming) ..... | 5         |
| 1.4. CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT .....                                        | 6         |
| 1.4.1. Cấu trúc dữ liệu.....                                                     | 6         |
| 1.4.2. Giải thuật.....                                                           | 9         |
| 1.5. BIỂU DIỄN GIẢI THUẬT BẰNG LƯU ĐỒ THUẬT TOÁN .....                           | 11        |
| 1.5.1. Giới thiệu .....                                                          | 11        |
| 1.5.2. Ví dụ về lưu đồ thuật toán.....                                           | 11        |
| 1.6. NHỮNG TÍNH CHẤT CẦN THIẾT ĐỂ TRỞ THÀNH LẬP TRÌNH VIÊN .....                 | 13        |
| 1.7. BÀI TẬP CHƯƠNG 1 .....                                                      | 14        |
| <b>CHƯƠNG 2: GIỚI THIỆU VỀ NGÔN NGỮ LẬP TRÌNH JAVA .....</b>                     | <b>15</b> |
| 2.1. LỊCH SỬ PHÁT TRIỂN NGÔN NGỮ JAVA.....                                       | 15        |
| 2.2. ĐẶC TRƯNG CỦA NGÔN NGỮ JAVA.....                                            | 15        |
| 2.3. CÀI ĐẶT JAVA VÀO MÁY TÍNH .....                                             | 17        |
| 2.4. VIẾT CHƯƠNG TRÌNH JAVA ĐẦU TIÊN .....                                       | 19        |
| 2.5. CHƯƠNG TRÌNH ECLIPSE.....                                                   | 20        |
| 2.5.1. Cài đặt chương trình Eclipse .....                                        | 20        |
| 2.5.2. Viết chương trình Java bằng Eclipse.....                                  | 22        |
| 2.6. CÁC DẠNG CHƯƠNG TRÌNH ỨNG DỤNG CỦA NGÔN NGỮ JAVA .....                      | 23        |
| 2.6.1. Chương trình ứng dụng độc lập.....                                        | 23        |
| 2.6.2. Chương trình ứng dụng nhúng APPLET.....                                   | 23        |
| 2.6.3. Chương trình chạy được cả ứng dụng độc lập và Web Browser.....            | 23        |
| 2.7. BÀI TẬP CHƯƠNG 2 .....                                                      | 23        |
| <b>CHƯƠNG 3: CÁC THÀNH PHẦN CƠ SỞ CỦA JAVA .....</b>                             | <b>24</b> |
| 3.1. CÁC PHẦN TỬ CƠ SỞ CỦA JAVA .....                                            | 24        |
| 3.1.1. Định danh/ tên gọi (Identifier) .....                                     | 24        |
| 3.1.2. Các từ khoá .....                                                         | 24        |
| 3.1.3. Chú thích (Comment).....                                                  | 24        |
| 3.2. CÁC KIỂU DỮ LIỆU NGUYÊN THỦY .....                                          | 25        |

|                                                      |           |
|------------------------------------------------------|-----------|
| 3.2.1. Kiểu nguyên .....                             | 25        |
| 3.2.2. Kiểu số thực .....                            | 25        |
| 3.2.3. Kiểu boolean .....                            | 26        |
| 3.3. KHAI BÁO BIẾN .....                             | 26        |
| 3.3.1. Biến cục bộ.....                              | 26        |
| 3.3.2. Biến toàn cục.....                            | 27        |
| 3.3.3. Biến static.....                              | 28        |
| 3.4. CẤU TRÚC TẬP TIN CHƯƠNG TRÌNH JAVA .....        | 29        |
| 3.5. CÁC PHÉP TOÁN TRONG JAVA (TOÁN TỬ).....         | 30        |
| 3.5.1. Các phép toán số học.....                     | 30        |
| 3.5.2. Các phép toán lô-gic.....                     | 31        |
| 3.5.3. Thứ tự ưu tiên của các phép toán .....        | 32        |
| 3.6. CÁC QUI TẮC CHUYỂN ĐỔI KIỂU .....               | 33        |
| 3.6.1. Chuyển đổi kiểu dữ liệu .....                 | 33        |
| 3.6.2. Ép kiểu .....                                 | 33        |
| 3.7. CÁC HÀM TOÁN HỌC.....                           | 33        |
| 3.8. CÂU LỆNH ĐIỀU KHIỂN Rẽ NHÁNH .....              | 35        |
| 3.8.1. Câu lệnh if / else.....                       | 35        |
| 3.8.2. Câu lệnh switch - case.....                   | 36        |
| 3.9. CÂU LỆNH ĐIỀU KHIỂN LẶP .....                   | 38        |
| 3.9.1. Vòng lặp while .....                          | 38        |
| 3.9.2. Vòng lặp do...while .....                     | 39        |
| 3.9.3. Vòng lặp for .....                            | 40        |
| 3.10. HÀM TRONG JAVA.....                            | 40        |
| 3.10.1. Giới thiệu về hàm.....                       | 40        |
| 3.10.2. Hàm có trả về kết quả.....                   | 41        |
| 3.10.3. Hàm không trả về kết quả .....               | 41        |
| 3.10.4. Truyền tham số khi gọi hàm(method).....      | 42        |
| 3.11. BÀI TẬP CHƯƠNG 3 .....                         | 42        |
| <b>CHƯƠNG 4: MẢNG TRONG JAVA.....</b>                | <b>50</b> |
| 4.1. KHÁI NIỆM VỀ MẢNG .....                         | 50        |
| 4.2. MỘT SỐ THAO THÁC CƠ BẢN TRÊN MẢNG .....         | 51        |
| 4.2.1. Thao tác trên mảng 1 chiều .....              | 51        |
| 4.2.2. Thao tác trên mảng 2 chiều .....              | 53        |
| 4.2.3. Độ dài của mảng.....                          | 62        |
| 4.3. HẠN CHẾ CỦA MẢNG.....                           | 62        |
| 4.4. BÀI TẬP CHƯƠNG 4 .....                          | 62        |
| <b>CHƯƠNG 5: THUẬT TOÁN SẮP XẾP VÀ TÌM KIẾM.....</b> | <b>63</b> |
| 5.1. THUẬT TOÁN SẮP XẾP .....                        | 63        |

|                                                           |           |
|-----------------------------------------------------------|-----------|
| 5.1.1. Giới thiệu .....                                   | 63        |
| 5.1.2. Bubble sort .....                                  | 63        |
| 5.1.3. Simple selection sort .....                        | 65        |
| 5.1.4. Quick Sort .....                                   | 67        |
| 5.2. THUẬT TOÁN TÌM KIẾM.....                             | 70        |
| 5.2.1. Tìm kiếm tuyến tính.....                           | 70        |
| 5.2.2. Tìm kiếm nhị phân .....                            | 71        |
| 5.3. BÀI TẬP CHƯƠNG 5 .....                               | 73        |
| <b>CHƯƠNG 6: KỸ THUẬT ĐỆ QUY VÀ CHUỖI TRONG JAVA.....</b> | <b>74</b> |
| 6.1. KỸ THUẬT ĐỆ QUY.....                                 | 74        |
| 6.1.1. Giới thiệu về đệ quy .....                         | 74        |
| 6.1.2. Ví dụ về đệ quy trong java .....                   | 74        |
| 6.2. CHUỖI TRONG JAVA .....                               | 76        |
| 6.2.1. Khai báo chuỗi ký tự .....                         | 76        |
| 6.2.2. Các hàm xử lý chuỗi ký tự .....                    | 77        |
| 6.2.3. Một số ví dụ về chuỗi.....                         | 82        |
| 6.3. BÀI TẬP CHƯƠNG 6 .....                               | 83        |
| <b>CHƯƠNG 7: NGOẠI LỆ TRONG JAVA .....</b>                | <b>84</b> |
| 7.1. TỔNG QUAN VỀ NGOẠI LỆ TRONG JAVA .....               | 84        |
| 7.1.1. Khái niệm ngoại lệ .....                           | 84        |
| 7.1.2. Các loại Exception trong Java.....                 | 84        |
| 7.1.3. Các cách xử lý Exception.....                      | 85        |
| 7.2. TRY – CATCH XỬ LÝ NGOẠI LỆ .....                     | 85        |
| 7.2.1. Cú pháp try - catch trong Java.....                | 85        |
| 7.2.2. Ví dụ về try catch .....                           | 85        |
| 7.2.3. Nhiều catch block trong Java .....                 | 86        |
| 7.3. FINALLY XỬ LÝ NGOẠI LỆ.....                          | 87        |
| 7.3.1. Cú pháp của finally block.....                     | 87        |
| 7.3.2. Ví dụ về finally block.....                        | 87        |
| 7.3.3. Try-catch-finally block.....                       | 88        |
| 7.3.4. Vài lưu ý về finally block.....                    | 90        |
| 7.4. THROW EXCEPTION TRONG JAVA .....                     | 90        |
| 7.4.1. Cú pháp của từ khóa throw trong Java.....          | 90        |
| 7.4.2. Ví dụ về từ khóa throw trong Java .....            | 90        |
| 7.5. CUSTOM EXCEPTION TRONG JAVA .....                    | 91        |
| 7.5.1. Giới thiệu .....                                   | 91        |
| 7.5.2. Ví dụ về user-defined exception trong Java .....   | 91        |
| 7.6. BÀI TẬP CHƯƠNG 7 .....                               | 92        |
| <b>CHƯƠNG 8: LUỒNG VÀ TẬP TIN TRONG JAVA.....</b>         | <b>94</b> |

|                                                                                   |            |
|-----------------------------------------------------------------------------------|------------|
| 8.1. LƯỠNG TRONG JAVA.....                                                        | 94         |
| 8.1.1. Giới thiệu về luồng.....                                                   | 94         |
| 8.1.2. Tạo và quản lý luồng trong Java .....                                      | 94         |
| 8.1.3. Đồng bộ hóa đa luồng .....                                                 | 99         |
| 8.2. TẬP TIN TRONG JAVA.....                                                      | 101        |
| 8.2.1. Cách tạo một file .....                                                    | 101        |
| 8.2.2. Cách đọc file .....                                                        | 102        |
| 8.2.3. Cách ghi file .....                                                        | 103        |
| 8.2.4. Cách ghi nối thêm nội dung vào file .....                                  | 104        |
| 8.2.5. Cách xóa file .....                                                        | 106        |
| 8.3. BÀI TẬP CHƯƠNG 8 .....                                                       | 106        |
| <b>CHƯƠNG 9: ĐỐI TƯỢNG VÀ LỚP TRONG JAVA.....</b>                                 | <b>107</b> |
| 9.1. TỔNG QUAN VỀ ĐỐI TƯỢNG VÀ LỚP.....                                           | 107        |
| 9.1.1. Đối tượng .....                                                            | 107        |
| 9.1.2. Lớp .....                                                                  | 107        |
| 9.1.3. Phạm vi truy cập (Access Modifier) của lớp, thuộc tính và phương thức..... | 108        |
| 9.1.4. Thuộc tính và phương thức .....                                            | 108        |
| 9.1.5. Hàm tạo .....                                                              | 110        |
| 9.2. THAO TÁC TRÊN ĐỐI TƯỢNG VÀ PHẠM VI TRUY CẬP .....                            | 111        |
| 9.2.1. Tạo mới một đối tượng .....                                                | 111        |
| 9.2.2. Truy cập đến thuộc tính và phương thức của lớp.....                        | 111        |
| 9.3. TÍNH ĐÓNG GÓI (ENCAPSULATION) TRONG JAVA .....                               | 113        |
| 9.3.1. Khái niệm tính đóng gói.....                                               | 113        |
| 9.3.2. Ví dụ về tính đóng gói.....                                                | 113        |
| 9.4. TÍNH KẾ THỪA (INHERITANCE) TRONG JAVA.....                                   | 114        |
| 9.4.1. Khái niệm tính kế thừa.....                                                | 114        |
| 9.4.2. Ví dụ về tính kế thừa.....                                                 | 115        |
| 9.5. TÍNH ĐA HÌNH (POLYMORPHISM) TRONG JAVA .....                                 | 116        |
| 9.5.1. Khái niệm tính đa hình.....                                                | 116        |
| 9.5.2. Ví dụ về tính đa hình.....                                                 | 116        |
| 9.6. TÍNH TRỪU TƯỢNG (ABSTRACTION) TRONG JAVA.....                                | 118        |
| 9.6.1. Khái niệm tính trừu tượng.....                                             | 118        |
| 9.6.2. Ví dụ về tính trừu tượng.....                                              | 118        |
| 9.7. BÀI TẬP CHƯƠNG 9 .....                                                       | 119        |
| <b>TÀI LIỆU THAM KHẢO.....</b>                                                    | <b>122</b> |