ĐẠI HỌC HUẾ
TRƯỜNG ĐẠI HỌC KHOA HỌC

NGUYỄN HOÀNG HÀ, NGUYỄN VĂN TRUNG
NGUYỄN DŨNG, NGUYỄN MẬU HÂN

GIÁO TRÌNH
JAVA CƠ BẢN

NHÀ XUẤT BẢN ĐẠI HỌC HUẾ
Huế, 2020

Biên mục trên xuất bản phẩm của Thư viện Quốc gia Việt Nam

ĐTTS ghi: Đại học Huế. Trường Đại học Khoa học. - Thư mục: tr. 211

006.740711 - dc23

DUF0253p - CIP

Mã số sách: GT/38-2019

LỜI NÓI ĐẦU

Giới thiệu về NNLT Java: ....

Giáo trình "Java cơ bản" sẽ hệ thống toàn bộ những nội dung kiến thức cơ bản liên quan đến kiến thức về ngôn ngữ lập trình Java. Giáo trình được sử dụng cho sinh viên chuyên ngành Công nghệ Thông tin, là môn cơ sở để lập trình di động, lập trình web JSP, ... Giáo trình được chia làm 3 chương. Cuối mỗi chương sẽ có phần tổng kết chương, giúp người học nắm được các nội dung và ý nghĩa của chương. Phần câu hỏi và bài tập ở cuối mỗi chương cũng giúp người học nâng cao kỹ năng thực hành trên cơ sở phần lý thuyết được trang bị.

Các chương của giáo trình được tóm tắt như sau:

Chương 1: *Lập trình với Java*,....

Chương 2: *Lập trình hướng đối tượng trong Java*, ...

Chương 3: *Lập trình ứng dụng csdl*. ...

Mặc dù đã rất cố gắng để hoàn thiện giáo trình này với mong muốn đến tay người đọc nhưng giáo trình không thể tránh khỏi những thiếu sót về cách diễn đạt, bố cục, nội dung và các lỗi cú pháp. Rất mong được bạn đọc góp ý.

Để hoàn tất giáo trình này chúng tôi xin cám ơn các Thầy Cô giáo của Khoa Công nghệ Thông tin, Trường Đại học Khoa học, Đại học Huế đã góp ý, chỉnh sửa để giáo trình sớm được ra mắt bạn đọc.

Huế, tháng 02 năm 2019

Nhóm tác giả

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

CHƯƠNG 1

TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH JAVA

1.1. Giới thiệu về ngôn ngữ lập trình Java

Java là ngôn ngữ lập trình bậc cao, hướng đối tượng (tựa C/C++) do hãng Sun Microsystem phát triển và phát hành vào năm 1995. Mục tiêu chung của ngôn ngữ lập trình Java là cho phép các nhà phát triển ứng dụng "Viết một lần chạy mọi nơi" (WORA – Write once run anywhere), nghĩa là mã Java đã được biên dịch có thể chạy trên tất cả nền tảng hỗ trợ Java mà không cần biên dịch lại.

Java được James Gosling và các cộng sự của Công ty Sun Microsystem (Sau được công ty Oracle mua lại) khởi xướng và phát triển vào đầu những năm 90. Ban đầu ngôn ngữ này có tên là Oak theo tên một cây sồi bên ngoài văn phòng của Gosling. Sau đó dự án có tên là Green và cuối cùng là Java. Java là tên gọi của một hòn đảo ở Indonexia. Đây là nơi nhóm nghiên cứu đã chọn để đặt tên cho ngôn ngữ lập trình của mình trong một chuyến đi tham quan và làm việc trên hòn đảo này. Hòn đảo Java này là nơi rất nổi tiếng với nhiều khu vườn trồng café, đó chính là lý do chúng ta thường thấy biểu tượng ly café trong nhiều sản phẩm phần mềm, công cụ lập trình Java của Sun.

Một số đặc trưng của ngôn ngữ Java là:

- Hướng đối tượng: Trong Java, mọi thứ đều là Đối tượng. Do đó chương trình viết bằng Java có thể dễ dàng được mở rộng vì nó dựa trên mô hình đối tượng.
- Độc lập nền tảng: Không giống như nhiều ngôn ngữ lập trình khác bao gồm C và C++, khi Java được biên dịch, nó không được biên dịch trực tiếp thành mã máy cụ thể, thay vào đó là mã bytecode độc lập với nền tảng. Mã bytecode này được Máy ảo Java (JVM – Java Virtual Machine) thông dịch và chạy trên bất

kỳ nền tảng nào mà máy ảo này đang được chạy như: Window, Linux, Sun Solaris, Mac/OS,... Với cách thiết kế này mà Java đã đáp ứng được phương châm mà họ đã đưa ra: "Viết một lần, Chạy mọi nơi".

- Đơn giản: Java được thiết kế để dễ học. Cú pháp của Java dựa trên C/C++, trong đó đã loại bỏ nhiều tính năng phức tạp như: Con trỏ, quá tải toán tử,... Đặc biệt trong Java có tính năng Bộ thu gom rác tự động (Garbage Collection), do đó chúng ta không cần phải loại bỏ các đối tượng khi không dùng nữa như C++.
- Bảo mật: Java bảo mật bởi vì:
  - Không có con trỏ tường minh.
  - Chương trình chạy bên trong máy ảo.
  - Classloader: Nó bổ sung tính bảo mật bằng cách tách gói cho các lớp của hệ thống tệp cục bộ từ các gói được nhập từ các nguồn mạng.
  - Bytecode Vertifier: Kiểm tra các đoạn mã để tìm ra các phần mã truy cập trái phép tới các đối tượng.
  - Security Manager: Nó xác định những tài nguyên mà một lớp có thể truy cập được như đọc và ghi đĩa cục bộ.
  - Những tính năng bảo mật này được cung cấp bởi Ngôn ngữ Java. Ngoài ra, một vài tính năng bảo mật khác được cung cấp thông qua nhà phát triển như SSL, JAAS, cryptography, ...
- Mạnh mẽ: Java quản lý bộ nhớ rất tốt, có bộ thu gom rác tự động; không sử dụng con trỏ để tránh các vấn đề về bảo mật; Có cơ chế xử lý ngoại và kiểm tra kiểu dữ liệu tốt.
- Kiến trúc trung lập: Java là kiến trúc trung lập vì không có các tính năng phụ thuộc vào môi trường phát triển và thực thi; ví dụ, kích thước của các kiểu dữ liệu cơ bản là cố định. Như trong ngôn ngữ lập trình C, kiểu dữ liệu int chiếm 2byte trong bộ nhớ cho kiến trúc 32bit và 4byte bộ nhớ cho kiến trúc 64bit. Tuy nhiên, nó chiếm 4byte bộ nhớ cho cả kiến trúc 32 và 64bit trong

Java.

- Linh động: Vì Java biển dịch mã thành bytecode, do đó chương trình có thể chạy trên bất kỳ nền tảng nào có máy ảo Java mà không cần phải viết lại.
- Đa luồng: Với tính năng đa luồng của Java, có thể viết các chương trình có thể thực hiện đồng thời nhiều tác vụ. Tính năng thiết kế này cho phép các nhà phát triển xây dựng các ứng dụng tương tác có thể chạy trơn tru.
- Thông dịch: Mã bytecode của Java được dịch một cách nhanh chóng sang tập lệnh của máy thực thi chương trình và không được lưu trữ ở bất cứ đâu.
- Hiệu suất cao: Với việc sử dụng các trình biên dịch, thông dịch đúng lúc, Java cho phép tạo ra các ứng dụng có hiệu năng cao. Java nhanh hơn các ngôn ngữ lập trình thông dịch truyền thống khác vì mã bytecode của Java "gần" với mã mãy. Nó vẫn chậm hơn một chút so với ngôn ngữ biên dịch như C++. Java là một ngôn ngữ thông dịch, đó là lý do tại sao nó chậm hơn các ngôn ngữ biên dịch như C/C ++.
- Phân tán: Java được thiết kế cho môi trường phân tán của internet. RMI và EJB là hai bộ thư viện được sử dụng để tạo các ứng dụng phân tán. Tính năng này của Java làm cho chúng ta có thể gọi các phương thức từ bất kỳ máy nào trên internet.
- Động: Java là một ngôn ngữ động. Nó hỗ trợ tải động các lớp, tức là các lớp được tải theo yêu cầu. Nó cũng hỗ trợ gọi các chức năng từ các ngôn ngữ khác như C/C++. Java hỗ trợ biên dịch động và quản lý bộ nhớ tự động.

Một số loại ứng dụng được viết bởi Java:

- Ứng dụng Desktop
- Ứng dụng Web
- Ứng dụng di động
- Hệ thống nhúng

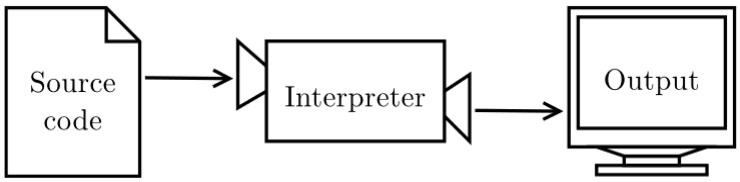
- Thẻ thông minh
- Hệ thống Robotic
- Game, ...

1.2. Mô hình biên dịch và thông dịch của Java

Các chương trình trong các ngôn ngữ lập trình bậc cao trước khi có thể chạy, chúng phải được dịch sang ngôn ngữ cấp thấp, còn được gọi là ngôn ngữ máy. Việc chuyển đổi này đôi khi là một nhược điểm nhỏ của các ngôn ngữ bậc cao. Tuy nhiên các ngôn ngữ bậc cao có hai ưu điểm:

- Dễ lập trình: Các chương trình được viết ra tốn ít thời gian lập trình, mã lệnh ngắn gọn, dễ đọc và có tính chính xác cao.
- Có thể chạy được trên nhiều nền tảng khác nhau mà chỉ cần một ít sửa đổi nhỏ hoặc không cần sửa đổi.

Có hai loại chương trình dịch để dịch ngôn ngữ bậc cao sang ngôn ngữ bậc thấp: trình thông dịch (interpreter) và trình biên dịch (compiler). Trình thông dịch đọc một chương trình bậc cao và thực hiện nó. Nó thực hiện chương trình theo từng lệnh một mà nó đã đọc, thay vì đọc toàn bộ các dòng và thực hiện tính toán. Hình 1.1 cho thấy cấu trúc của một trình thông dịch.



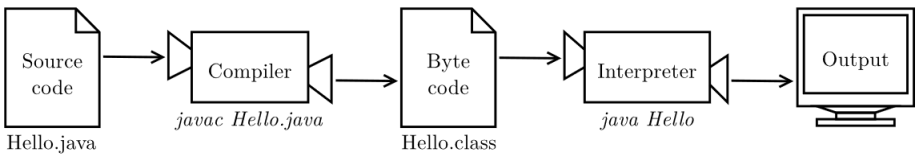
Hình 1.1. *Cấu trúc một chương trình thông dịch*

Ngược lại, một trình biên dịch đọc toàn bộ chương trình và dịch nó hoàn toàn trước khi chương trình bắt đầu chạy. Trong ngữ cảnh này, chương trình bậc cao được gọi là mã nguồn và chương trình đã dịch được gọi là mã đối tượng (object code) hoặc tệp thực thi. Khi một chương trình được biên dịch, chúng ta có thể chạy nó nhiều lần mà

không cần dịch lại. Kết quả là, các chương trình được biên dịch thường chạy nhanh hơn các chương trình thông dịch.

Java là ngôn ngữ vừa biên dịch và thông dịch. Thay vì dịch các chương trình trực tiếp sang mã máy, trình biên dịch Java tạo ra mã bytecode. Tương tự như mã máy, mã bytecode dễ dàng được thông dịch thành mã máy một cách nhanh chóng và dễ dàng. Và nhờ vào mã bytecode này, mà một chương trình có thể được biên dịch trên một máy và sau đó có thể được thông dịch và chạy trên máy khác. Trình thông dịch này được gọi là Máy ảo Java (JVM – Java Virtual Machine).

Hình 1.2 cho thấy các bước của quy trình này. Mặc dù có vẻ phức tạp, các bước này được tự động hóa cho chúng ta trong hầu hết các môi trường phát triển tích hợp (IDE – Intergrated Development Enviroment). Thông thường, chúng ta chỉ phải nhấn một nút hoặc gõ một lệnh duy nhất để biên dịch chương trình. Mặt khác, điều quan trọng là chúng ta cần phải biết những bước nào đang diễn ra, để nếu có sự cố xảy ra, chúng ta có thể tìm ra nó là gì và khắc phục nó.



Hình 1.2. *Mô hình biên dịch và thông dịch của chương trình Java*

1.3. Cấu trúc chương trình Java

Thông thường, chương trình đầu tiên chúng ta viết khi học một ngôn ngữ lập trình mới được gọi là chương trình `Hello World`. Tất cả những gì nó làm là hiển thị dòng chữ `Hello, World` trên màn hình. Trong Java, nó trông như thế này:

```
1 public class Hello {
2    public static void main(String[] args) {
3    //Xuất kết quả ra màn hình
4    System.out.println("Hello, World");
5    }
```

```
6 }
```

Khi chương trình này chạy, nó cho kết quả:

```
Hello, World
```

Các chương trình Java được tạo thành từ các định nghĩa lớp và phương thức, và các phương thức được tạo thành từ các câu lệnh. Một câu lệnh là một dòng mã thực hiện một hoạt động cơ bản. Trong chương trình `Hello World`, dòng sau là một câu lệnh nhằm hiển thị thông báo trên màn hình:

```
System.out.println("Hello, World");
```

`System.out.println` hiển thị kết quả trên màn hình và xuống hàng; tên `println` là viết tắt của dòng chữ `"print line"`. `"print"` có nghĩa là hiển thị trên màn hình hoặc gửi đến máy in. Trong giáo trình này, khi chúng tôi nói hiển thị, nghĩa là chúng tôi muốn xuất kết quả ra màn hình. Giống như hầu hết các câu lệnh, câu lệnh in kết thúc bằng dấu chấm phẩy (;).

Java là phân biệt chữ hoa chữ thường, có nghĩa là chữ hoa và chữ thường không giống nhau. Trong ví dụ này, `System` phải bắt đầu bằng một chữ cái viết hoa; trường hợp chúng ta viết `system` hoặc `SYSTEM` thì chương trình sẽ không hoạt động được.

Một phương thức là một chuỗi các câu lệnh được đặt tên. Chương trình này định nghĩa một phương thức có tên là main:

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

Tên và định dạng của phương thức main rất đặc biệt: khi chương trình chạy, nó bắt đầu ở câu lệnh đầu tiên trong main và kết thúc khi nó kết thúc câu lệnh cuối cùng.

Một lớp là một tập hợp các phương thức và thuộc tính. Chương trình này định nghĩa một lớp có tên `Hello`. Chúng ta có thể đặt cho một lớp bất kỳ tên nào (nhưng phải phù hợp với quy tắc đặt tên, sẽ được đề cập ở bên dưới), nhưng thông thường bắt đầu bằng chữ in hoa. Tên của lớp phải khớp với tên của tệp mà nó nằm trong, vì vậy lớp này phải nằm trong một tệp có tên `Hello.java`.

Java sử dụng cặp dấu ngoặc nhọn () để nhóm các câu lệnh lại với nhau. Trong `Hello.java`, cặp dấu ngoặc ngoài cùng chứa định nghĩa lớp `Hello` và cặp dấu ngoặc bên trong định nghĩa phương thức `main`.

Dòng bắt đầu bằng hai dấu gạch chéo (//) là một ghi chú, đó là một đoạn giải thích ngắn ý nghĩa của mã lệnh. Khi trình biên dịch nhìn thấy dấu //, nó bỏ qua mọi thứ từ đó cho đến hết dòng. Ghi chú không có tác dụng trong việc thực hiện chương trình, nhưng chúng giúp các lập trình viên khác (và bản thân của bạn trong tương lai) dễ hiểu hơn những gì bạn muốn làm.

1.4. Bộ từ khóa

Từ khóa là những từ có một ý nghĩa nhất định trong các ngôn ngữ lập trình. Ngôn ngữ lập trình Java có khoảng 50 từ khóa khác nhau và được viết bằng chữ thường. Chúng ta cần nhớ những từ khóa này vì chúng sẽ được dùng rất nhiều trong quá trình lập trình sau này.

- abstract: Khai báo lớp, phương thức, interface trừu tượng không có thể hiện (instance) cụ thể
- assert: Kiểm tra điều kiện đúng hay sai (thường dùng trong Unit Test)
- boolean: Kiểu dữ liệu logic với 2 giá trị: true, false.
- break: Thoát ra khỏi vòng lặp hoặc lệnh switch-case.
- byte: Kiểu byte với các giá trị nguyên chiếm 8bit (1byte).
- case: Trường hợp được tuyển chọn theo switch (chỉ được dùng khi đi kèm switch)
- catch: Được sử dụng để bắt ngoại lệ, được sử dụng cùng với try để xử lý các ngoại lệ xảy ra trong chương trình
- char: Kiểu ký tự Unicode, mỗi ký tự chiếm 16bit (2byte).
- class: Được sử dụng để định nghĩa lớp
- const: Chưa được sử dụng vì vậy chúng ta không thể dùng nó trong ngôn ngữ Java
- continue: Dừng chu trình lặp hiện tại và bắt đầu chu trình tiếp theo
- default: Mặc định được thực thi khi không có case nào trả về giá trị true (dùng trong switch-case)

- do: Dùng trong vòng lặp do... while
- double: Kiểu số thực với các giá trị biểu diễn theo dạng dấu phẩy động 64bit (8byte)
- else: Rẽ nhánh theo điều kiện ngược lại của if.
- enum: Định nghĩa kiểu dữ liệu enum
- extends: Được sử dụng để định nghĩa lớp con kế thừa các thuộc tính và phương thức từ lớp cha.
- final: Chỉ ra các biến, phương thức không được thay đổi sau khi đã được định nghĩa. Các phương thức final không thể được kế thừa và override
- finally: Thực hiện một khối lệnh đến cùng bất chấp các ngoại lệ có thể xảy ra. Được sử dụng trong try-catch
- float: Kiểu số thực với các giá trị biểu diễn theo dạng dấu phẩy động 32bit.
- for: Sử dụng cho vòng lặp for với bước lặp được xác định trước
- goto: Chưa được sử dụng
- if: Lệnh chọn theo điều kiện logic
- implements: Xây dựng một lớp mới cài đặt những phương thức từ interface xác định trước.
- import: Yêu cầu một hay một số lớp ở các gói chỉ định cần nhập vào để sử dụng trong ứng dụng hiện thời.
- instanceof: Kiểm tra xem một đối tượng nào đó có phải là một thể hiện của một lớp được định nghĩa trước hay không
- int: Kiểu số nguyên với các giá trị chiếm 32bit (4byte).
- interface: Được sử dụng để định nghĩa interface
- long: Kiểu số nguyên lớn với các giá trị chiếm 64bit (8byte).
- native: Giúp lập trình viên có thể sử dụng code được viết bằng các ngôn ngữ khác

- new: Khởi tạo đối tượng
- package: Xác định một gói sẽ chứa một số lớp ở trong file mã nguồn.
- private: Khai báo biến dữ liệu, phương thức riêng trong từng lớp và chỉ cho phép truy cập trong lớp đó.
- protected: Khai báo biến dữ liệu, phương thức chỉ được truy cập ở lớp cha và các lớp con của lớp đó.
- public: Khai báo lớp, biến dữ liệu, phương thức công khai có thể truy cập ở mọi nơi trong hệ thống.
- return: Kết thúc phương thức và trả về giá trị cho phương thức
- short: Kiểu số nguyên ngắn với các giá trị chiếm 16 bit (2 byte).
- static: Định nghĩa biến, phương thức của một lớp có thể được truy cập trực tiếp từ lớp mà không thông qua khởi tạo đôi tượng của lớp
- super: Biến chỉ tới đối tượng ở lớp cha
- switch: Sử dụng trong câu lệnh điều khiển switch case
- synchronized: Chỉ ra là ở mỗi thời điểm chỉ có một đối tượng hoặc một lớp có thể truy nhập đến biến dữ liệu, hoặc phương thức loại đó, thường được sử dụng trong lập trình đa luồng (multithreading)
- this: Biến chỉ tới đối tượng hiện thời.
- throw: Tạo một đối tượng exception để chỉ định một trường hợp ngoại lệ xảy ra
- throws: Chỉ định cho qua ngoại lệ khi exception xảy ra
- transient: Chỉ định rằng nếu một đối tượng được serialized, giá trị của biến sẽ không cần được lưu trữ
- try: Thử thực hiện cho đến khi gặp một ngoại lệ.
- void: Chỉ định một phương thức không trả về giá trị

- volatile: Báo cho chương trình dịch biết là biến khai báo volatile có thể thay đổi tùy ý trong các luồng (thread).
- while: Được sử dụng trong lệnh điều khiển while

1.5. Quy tắc đặt tên

Quy tắc đặt tên được áp dụng cho tên gói, tên lớp, tên phương thức, tên thuộc tính, tên biến, tên hằng, ... Quy tắc này được mô tả như sau:

- Phân biệt hoa thường
- Chỉ chứa các ký tự là chữ cái, số,
- Không chứa các ký tự đặc biệt, không chứa khoảng trắng, ngoài dấu "_" và "$".
- Không được bắt đầu bằng số
- Không được trùng với từ khóa

1.6. Ghi chú

Các ghi chú trong Java được sử dụng để cung cấp thêm thông tin hoặc giải thích về biến, phương thức, lớp, hoặc bất kỳ câu lệnh nào. Nó cũng có thể được sử dụng để tạm thời vô hiệu hóa một đoạn mã nào đó trong Java. Các ghi chú không được thực thi bởi các trình biên dịch và trình thông dịch.

Trong Java, để ghi chú, chúng ta có 3 cách:

1.6.1. Ghi chú trên một dòng

Cú pháp:

```
//Nội dung ghi chú
```

Ví dụ:

```
1 int i = 10; //Đây là một biến kiểu int
```

1.6.2. Ghi chú trên nhiều dòng

Cú pháp:

```
/*
Nội dung ghi chú thứ nhất
Nội dung ghi chú thứ hai
...
*/
```

Ví dụ:

```
1 /*
2 - Khai báo biến
3 - In giá trị của nó ra màn hình
4 */
5 int i = 10;
6 System.out.println(i);
```

1.6.3. Ghi chú tài liệu

Ghi chú dạng tài liệu là ghi chú được sử dụng để tạo ra tài liệu cho các API. Được dùng chủ yếu để giải thích ý nghĩa, tham số đầu vào, kết quả đầu ra của các API mà tác giả đã xây dựng. Sau đó, chúng ta có thể sử dụng công cụ javadoc để kết xuất tài liệu này ra dạng tài liệu HTML.

Cú pháp:

```
/**
Đây là ghi chú dạng tài liệu
*/
```

1.7. Biến

Biến là vùng nhớ dùng để lưu trữ dữ liệu khi chương trình chạy. Dữ liệu lưu trong một biến được gọi là giá trị của biến đó. Chúng ta có thể truy cập, gán hay thay đổi giá trị của các biến, khi biến được gán một giá trị mới, giá trị cũ sẽ bị ghi đè. Mỗi biến gắn liền với một kiểu dữ liệu và một định danh duy nhất gọi là tên biến. Trong java, biến có

thể được khai báo ở bất kỳ nơi đâu trong chương trình.

Cách khai báo:

<kiểu_dữ_liệu> <tên_biến>;

<kiểu_dữ_liệu> <tên_biến> = <giá_trị>;

Gán giá trị cho biến:

<tên_biến> = <giá_trị>;

Chẳng hạn:

```
1 int x, y = 10;
2 x = 20;
```

Trong Java có 3 loại kiểu biến bao gồm:

- Biến local
- Biến instance
- Biến static

1.7.1. Biến local (biến địa phương)

Biến local được khai báo trong các phương thức, hàm tạo hoặc trong các khối lệnh . Đặc điểm của biến local:

- Biến local sẽ bị phá hủy khi kết thúc các phương thức, hàm tạo và khối lệnh.
- Không được sử dụng từ khóa chỉ mức độ truy cập (access modifier) khi khai báo biến local.
- Các biến local được lưu trên vùng nhớ stack của bộ nhớ.
- Cần khởi tạo giá trị mặc định cho biến local trước khi có thể sử dụng.

Ví dụ: Biến sum bên dưới là một biến local

```
1 int cong2So(int n1, int n2){
2    int sum = 0;
3    sum = n1 + n2;
```

```
4    return sum;
5 }
```

1.7.2. Biến instance (biến toàn cục)

Biến instance được khai báo trong một lớp, bên ngoài các phương thức, hàm tạo và các khối lệnh (). Đặc điểm của loại biến này:

- Biến instance được lưu trong bộ nhớ heap.
- Biến instance được tạo khi một đối tượng được tạo bằng việc 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, hàm tạo, khối lệnh, ... Nhưng nó phải được sử dụng thông qua một đối tượng cụ thể.
- Chúng ta được phép sử dụng từ khóa chỉ mức độ truy cập khi khai báo biến instance, mặc định là "private".
- 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, chúng ta sẽ không cần khởi tạo giá trị cho biến instance trước khi sử dụng.
- Bên trong lớp mà chúng ta khai báo biến instance, chúng ta có thể gọi nó trực tiếp bằng tên khi sử dụng ở khắp nới bên trong lớp đó.

Cú pháp:

```
1 class <Tên_Lớp>
2 {
3    <mức_độ_truy_cập> <kiểu_dữ_liệu> <biến_instanse_1>;
4    <mức_độ_truy_cập> <kiểu_dữ_liệu> <biến_instanse_2>;
5 }
```

1.7.3. Biến static

Biến static được khai báo trong một lớp với từ khóa `"static"`, phía bên ngoài các phương thức, hàm tạo và khối lệnh. Đặc điểm của loại biến này:

- Sẽ chỉ có duy nhất một bản sao của các biến static được tạo ra, dù bạn tạo bao nhiêu đối tượng từ lớp tương ứng.
- Biến static được lưu trữ trong bộ nhớ static riêng.
- Biến static được tạo khi chương trình bắt đầu chạy và chỉ bị phá hủy khi chương trình dừng.
- Giá trị mặc định của biến static phụ thuộc vào kiểu dữ liệu chúng ta khai báo tương tự biến instance.
- Biến static được truy cập thông qua tên của class chứa nó, với cú pháp: TenClass.tenBien
- Trong lớp, 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"`.

Ví dụ:

```
class <Tên_Lớp>
{
   <mức_độ_truy_cập> static <kiểu_dữ_liệu> <biến_static_1>;
   <mức_độ_truy_cập> static <kiểu_dữ_liệu> <biến_static_2>;
}
```

1.8. Kiểu dữ liệu

Kiểu dữ liệu trong Java chia làm 2 nhóm:

- Kiểu dữ liệu cơ bản (hay còn gọi là nguyên thủy): `byte, short, int, long, float, double, boolean và char`
- Kiểu dữ liệu tham chiếu: `String, Array, Class`

Một kiểu dữ liệu cơ bản có kích thước cố định và không có phương thức bổ sung. Kiểu dữ liệu dạng số chia làm 2 nhóm:

- Kiểu số nguyên:`byte, short, int và long`
- Kiểu số thực: `float và double`.

Có tám kiểu dữ liệu nguyên thủy trong Java:

Kiểu	Kích thước	Vùng
byte	1 byte	-128 ⇒ 127
short	2 bytes	-32,768 ⇒ 32,767
int	4 bytes	-2,147,483,648 ⇒ 2,147,483,647
long	8 bytes	-9,223,372,036,854,775,808 ⇒ 9,223,372,036,854,775,8
float	4 bytes	Độ chính xác đơn. Lưu 6 to 7 chữ số thập phân
double	8 bytes	Độ chính xác kép. Lưu 15 chữ số thập phân
boolean	1 bit	True/False
char	2 bytes	Lưu 1 ký tự đơn trong bảng mã Unicode

1.9. Hằng

Hằng là một giá trị bất biến trong chương trình, nghĩa là chúng ta không thể dùng phép gán để gán lại giá trị cho hằng sau khi đã định nghĩa. Một số đặc điểm của hằng trong Java:

- Tên hằng được đặt theo qui ước giống như tên biến.
- Hằng số nguyên: trường hợp giá trị hằng ở dạng long ta thêm vào cuối chuỗi số chữ "l" hay "L". Chẳng hạn: 1L
- Hằng số thực: trường hợp giá trị hằng có kiểu float ta thêm tiếp vị ngữ "f" hay "F", còn kiểu số double thì ta thêm tiếp vị ngữ "d" hay "D". Chẳng hạn: 2.3f
- Hằng Boolean: java có 2 hằng boolean là `true, false`.
- Hằng ký tự: là một ký tự đơn nằm giữa nằm giữa 2 dấu ngoặc đơn (chẳng hạn: 'a'). Một số hằng ký tự đặc biệt:

  `\b: Xóa lùi (BackSpace)`
  `\t: Tab`

```
\n: Xuống hàng
\r: Dấu enter
\": Nháy kép
\': Nháy đơn
\\: Số ngược
\f: Đẩy trang
\uxxxx: Ký tự unicode
```

- Hằng chuỗi: là tập hợp các ký tự được đặt giữa hai dấu nháy kép "". Một hằng chuỗi không có ký tự nào là một hằng chuỗi rỗng. Chẳng hạn: "Hello, world!".

Cú pháp:

```
final <kiểu_dữ_liệu> <tên_hằng> = <giá_trị_hằng>;
```

Ví dụ:

```
final double PI = 3.1415926535897;
```

1.10. Các phép toán cơ bản

1.10.1. Phép toán số học

Java hỗ trợ năm phép toán số học sau:
(bảng)

Các phép toán này chỉ áp dụng được cho các biến kiểu cơ bản và không áp dụng được cho các kiểu tham chiếu. Lưu ý với phép chia được thực hiện cho hai giá trị kiểu nguyên sẽ cho kết quả là thương nguyên.

Chẳng hạn: biểu thức `10/3` cho kết quả bằng `1`, còn `3/5` cho kết quả bằng `0`.

1.10.2. Phép toán tăng giảm

(bảng)

Các phép toán ++ và – có đặt trước hoặc sau biến và sự khác nhau ở chỗ:

- Nếu đặt trước biến a thì tăng/giảm giá trị biến a rồi mới sử dụng giá trị của a
- Nếu đặt sau biến a thì sử dụng giá trị của biến a rồi mới tiến hành tăng/giảm giá trị của a.

Chẳng hạn:

```
int a = 5;
int x = ++a; // thì x = 6, a = 6
//còn
int x = a++; // thì x = 5, a = 6
```

1.10.3. Toán tử trên bit

Toán tử	Ý nghĩa	Dạng thức
&	AND	a & b
|	OR	a | b
$<<$	Dịch trái	a $<<$ b
$>>$	Dịch phải	a $>>$ b
$>>>$	Dịch phải và điền 0 vào bit trống	a $>>>$ b

Các toán tử xử lý bit là các toán tử chỉ thực hiện trên các toán hạng có kiểu dữ liệu là số nguyên. Bảng sau cho kết quả của các phép toán:

a	b	a	a&b	a|b	a$\hat{}$b
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

1.10.4. Phép toán quan hệ & logic

Toán tử	Ý nghĩa	Dạng thức
==	So sánh bằng	a == b
!=	So sánh khác	a != b
>	So sánh lớn hơn	a > b
<	So sánh nhỏ hơn	a < b
>=	So sánh lớn hơn hay bằng	a >= b
<=	So sánh nhỏ hơn hay bằng	a <= b
||	Phép hoặc	a || b
&&	Phép giao	a && b
!	Phủ định	!a

Chú ý: Kết quả của phép toán quan hệ và logic là True/False.

Ý nghĩa của phép toán logic được cho ở bảng sau:

a	b	!a	a&&b	a||b
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

1.10.5. Phép toán điều kiện

Cú pháp:

`<điều_kiện> ? <biểu_thức_1> : < biểu_thức_2>`

Nếu điều kiện đúng thì có giá trị, hay thực hiện `<biểu_thức_1>`, còn ngược lại là `<biểu_thức_2>`. Trong đó:

- `<điều_kiện>`: là một biểu thức logic
- `<biểu_thức_1>, <biểu_thức_2>`: có thể là hai giá trị, hai biểu thức hoặc hai hành động.

Chẳng hạn: `z = ((x < y) ? (x) : (y));` sẽ gán giá trị nhỏ nhất của `x và y cho z`.

1.10.6. Phép gán

Phép gán là cách gắn một giá trị cho một biến hoặc thay đổi giá trị của một biến. Lệnh gán trong Java có công thức: `biến = biểu_thức;`

Trong đó, dấu bằng (=) được gọi là dấu gán hay toán tử gán, biểu thức ở vế phải dấu gán được tính rồi lấy kết quả gán cho biến nằm ở vế trái.

Biểu thức tại vế phải có thể là một giá trị trực tiếp, một biến, hoặc một biểu thức phức tạp. Ngoài ra chúng ta có thể kết hợp phép gán với các phép toán khác theo cách gộp như ví dụ sau:

- a += b; tương đương với a = a + b;
- a -= b; tương đương với a = a - b;
- a *= b; tương đương với a = a * b;
- a /= b; tương đương với a = a / b;
- a %= b; tương đương với a = a % b;

1.10.7. Chuyển đổi kiểu

Trong một biểu thức, các toán hạng có thể có các kiểu dữ liệu khác nhau, để việc tính toán được chính xác, đôi lúc cần phải chuyển đổi kiểu dữ liệu cho phù hợp. Có 2 cách chuyển đổi kiểu dữ liệu:

- Chuyển kiểu tự động:
  Khi 2 toán hạng trong một phép toán có kiểu dữ liệu khác nhau thì toán hạng có kiểu dữ liệu bé hơn sẽ tự động chuyển thành kiểu dữ liệu lớn hơn của toán hạng kia trước khi thực hiện phép toán. Đối với lệnh gán thì sự chuyển kiểu căn cứ vào kiểu dữ liệu của biến ở vế trái.
- Chuyển kiểu ép buộc:
  Thực hiện theo cú pháp sau: `(kiểu) biểu_thức`. Nếu chuyển từ kiểu dữ liệu nhỏ sang lớn thì sẽ không làm mất mát thông tin. Ngược lại nó sẽ mất mát một phần thông tin.

Chẳng hạn:

```
float a = 10.3f;
int b = (int)a;
```

Lúc này b sẽ có giá trị là 10.

1.11. Các cấu trúc điều khiển

1.11.1. Cấu trúc if

Cú pháp:

```
if (<điều_kiện_exp>) {
   <khối_lệnh_1>;
} [else {
   <khối_lệnh_2>;
}]
```

Sơ đồ thực hiện:

Ý nghĩa: Nếu biểu thức điều kiện exp là True thì chương trình sẽ thực hiện `khối_lệnh_1`, ngược lại thì thực hiện `khối_lệnh_2` (nếu có phần else). Các câu lệnh sau if hoặc else có thể là một khối lệnh. Nếu các câu lệnh if lồng nhau trong phạm vi một khối thì else sẽ đi với if gần nó nhất.

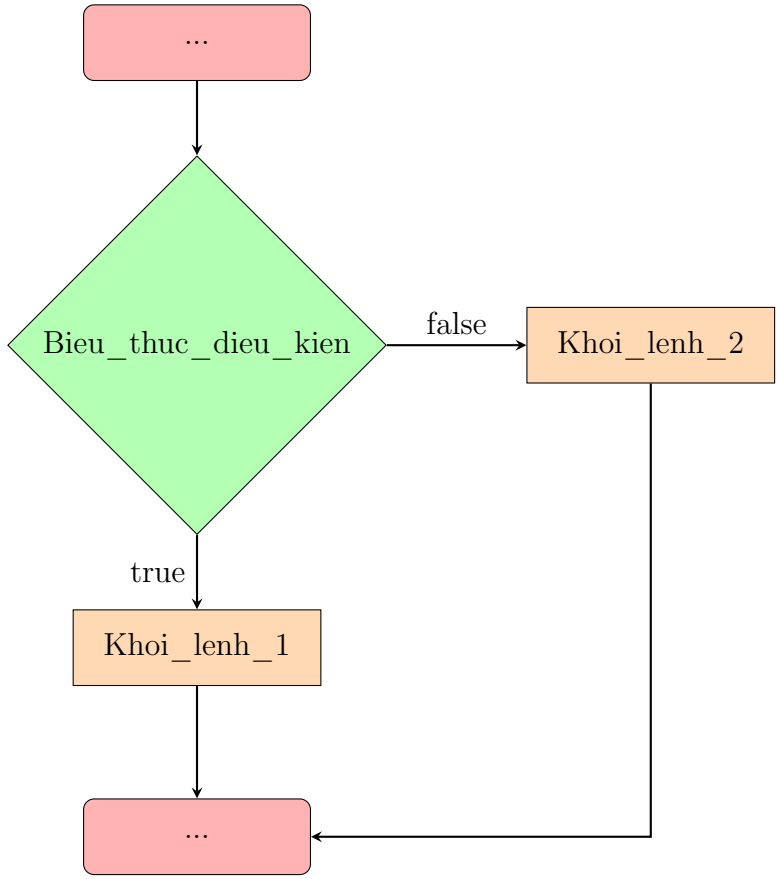
Ví dụ 1:

```
int a = 20, b = 10, c = 0;
c = a;
System.out.print("So lon nhat la: ");
if (a < b) {
   c = b;
}
System.out.print(c);
```

Kết quả:

```
So lon nhat la 20
```

Ví dụ 2:



Hình 1.3. *Sơ đồ thực hiện cấu trúc if*

```
int number = 13;
if (number % 2 == 0) {
   System.out.println("So " + number + " la so chan");
} else {
   System.out.println("So " + number + " la so le.");
}
```

Kết quả:

```
So 13 la so le
```

1.11.2. Cấu trúc switch

Cú pháp:

```
switch (<biểu_thức>) {
   case <giá_trị_1>:
      <khối_lệnh_1>;
      break;
      ...
   case <giá_trị_n>:
      <khối_lệnh_n>;
      break;
   default:
      <khối_lệnh_default>;
}
```

Ý nghĩa: Nếu `<biểu_thức>` có giá trị bằng `giá_trị_k` nào đó thì chương trình sẽ chuyển điều khiển đến `case giá_trị_k` và thực hiện `<khối_lệnh_k>` cho đến khi gặp lệnh `break` mới thoát khỏi cấu trúc `switch`. Ngược lại nếu giá trị của `<biểu_thức>` không có giá trị trong các giá trị được liệt kê sau `case` thì chương trình sẽ thực hiện `<khối_lệnh_default>` (nếu có).

Trong đó:

`<biểu_thức>, <giá_trị_1>, ..., <giá_trị_n>`: phải có giá trị nguyên

Ví dụ:

```
1  int day = 4;
2  switch (day) {
3     case 1:
4        System.out.println("Thu hai");
5        break;
6     case 2:
7        System.out.println("Thu ba");
8        break;
9     case 3:
10       System.out.println("Thu tu");
11       break;
```

```
12    case 4:
13       System.out.println("Thu nam");
14       break;
15    case 5:
16       System.out.println("Thu sau");
17       break;
18    case 6:
19       System.out.println("Thu bay");
20       break;
21    case 7:
22       System.out.println("Chu nhat");
23       break;
24 }
```

1.11.3. Cấu trúc for

Cú pháp:

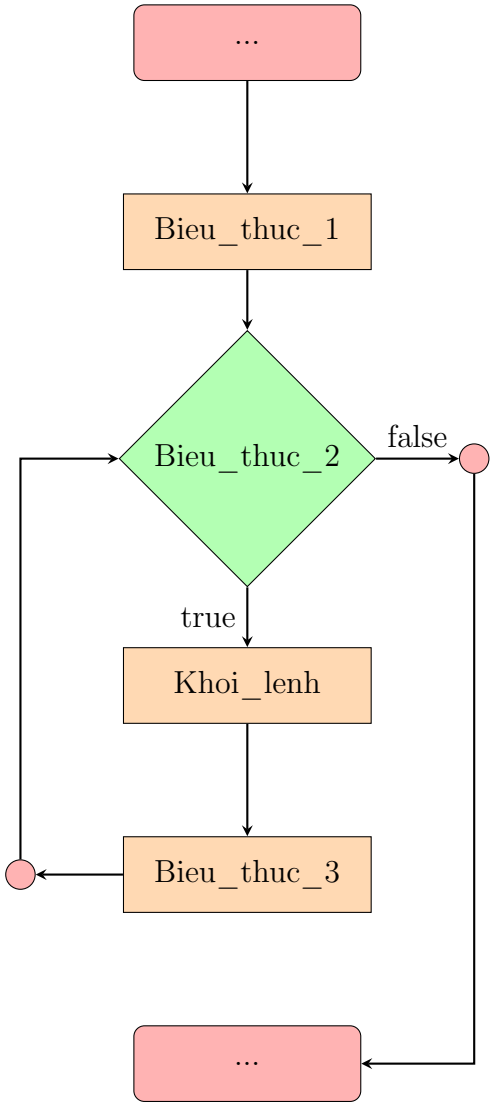
```
for (<biểu_thức_1>;<biểu_thức_2>;<biểu_thức_3>)
   <khối_lệnh>;
```

Sơ đồ thực hiện (hình 1.4):

Ý nghĩa:

- Đầu tiên `biểu_thức_1` được thực hiện. Đây thường là phép khởi gán để khởi động một hay nhiều biến. `Biểu_thức_1` chỉ được thực hiện 1 lần duy nhất khi bắt đầu thực hiện cấu trúc.
- Tiếp theo, `biểu_thức_2` được thực hiện. Đây là phần điều kiện thực hiện `khối_lệnh` của cấu trúc for. Nếu `biểu_thức_2` sai thì chương trình sẽ thoát khỏi cấu trúc for. Nếu `biểu_thức_2` đúng thì chương trình sẽ thực hiện `khối_lệnh`.
- Sau khi thực hiện `khối_lệnh` thi các biến tăng/giảm ở `biểu_thức_3` mới được thực hiện và cấu trúc sẽ lặp lại việc kiểm tra `biểu_thức_2`.

Ví dụ:



Hình 1.4. *Sơ đồ thực hiện cấu trúc for*

```
for (int i = 0; i <= 10; i = i + 2) {
  System.out.println(i);
}
```

Kết quả:

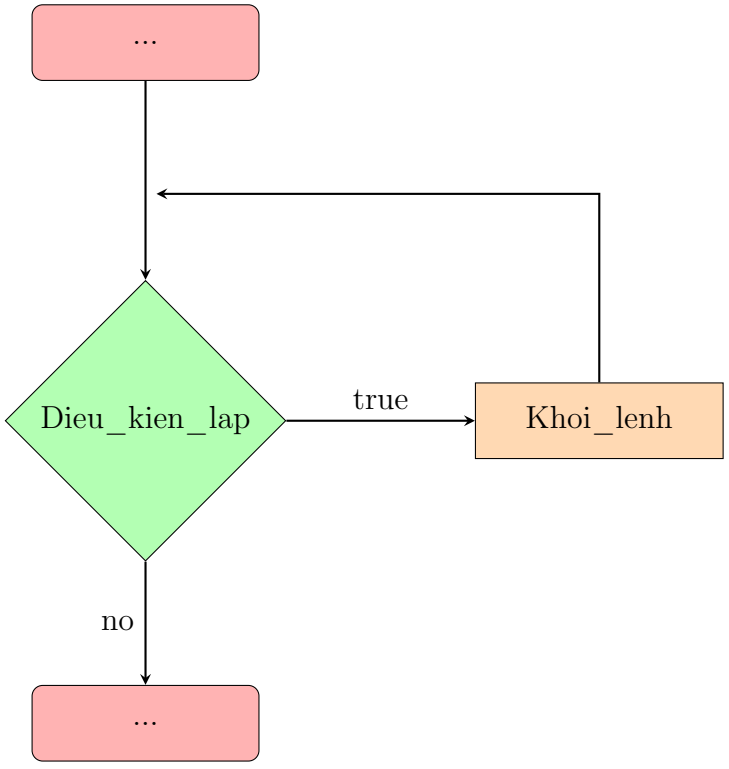
```
0
2
4
6
8
10
```

1.11.4. Cấu trúc while

Cú pháp:

```
while (<điều_kiện_lặp>)
    <khối_lệnh>;
```

Sơ đồ thực hiện (hình 1.5):



Hình 1.5. *Sơ đồ thực hiện cấu trúc while*

Ý nghĩa: Trong khi `điều_kiện_lặp` vẫn còn đúng thì thực hiện `khối_lệnh`.

Ví dụ:

```
int i = 0;
while (i < 5) {
   System.out.println(i);
   i++;
}
```

Kết quả:

```
0
1
2
3
4
```

1.11.5. Cấu trúc do...while

Cú pháp: `do`
`<khối_lệnh>;`
`while (<điều_kiện_lặp>);`

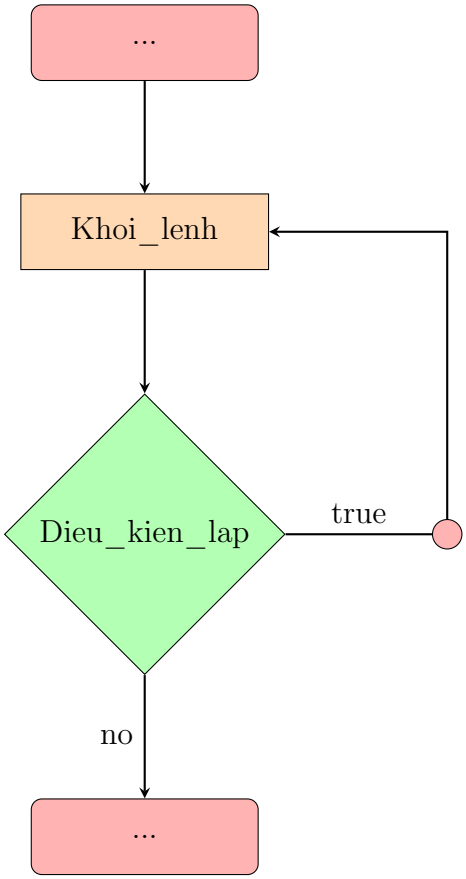
Sơ đồ thực hiện (hình 1.6):

Ý nghĩa: Cấu trúc `do...while` sẽ thực hiện `khối_lệnh` cho đến khi nào `điều_kiện_lặp` là sai.

Ví dụ:

```
int i = 0;
do {
   System.out.println(i);
   i++;
} while (i < 5);
```

Kết quả:



Hình 1.6. *Sơ đồ thực hiện cấu trúc do... while*

```
0
1
2
3
4
```

1.11.6. Cấu trúc lệnh nhảy

Lệnh break: trong cấu trúc switch chúng ta dùng câu lệnh break để thoát thỏi cấu trúc switch trong cùng chứa nó. Tương tự như vậy, trong cấu trúc lặp, câu lệnh break dùng để thoát khỏi cấu trúc lặp trong cùng chứa nó.

Lệnh continue: dùng để ngắt vòng lặp hiện tại và tiếp tục vòng lặp tiếp theo.

1.12. Các lớp bao kiểu dữ liệu cơ bản

Đôi khi, ta muốn đối xử với một giá trị kiểu cơ bản như là một đối tượng. Trong trường hợp như vậy, ta có các lớp bọc ngoài mỗi kiểu cơ bản (wrapper class). Các lớp bọc ngoài này có tên gần trùng với tên kiểu cơ bản tương ứng: `Boolean, Character, Byte, Short, Integer, Long, Float, Double`. Mỗi đối tượng thuộc các lớp trên bao bọc một giá trị kiểu cơ bản tương ứng, kèm theo các phương thức để thao tác với giá trị đó. Ví dụ:

```
1 //Tạo một đối tượng bọc ngoài 1 giá trị
2 int i = 10;
3 Integer iWrap = new Integer(i);
4 //Lấy giá trị bên trong đối tượng
5 int iUnwrap = iWrap.intValue();
6 //Biến chuỗi số thành số
7 Int i = Integer.parseInt("10");
8 //Biến số thành chuỗi số
9 String iString = iWrap.toString();
```

Các lớp bọc ngoài khác cũng có cách sử dụng và phương thức tương tự như `Integer`. Lúc này việc sử dụng kiểu dữ liệu cơ bản và đối tượng của lớp bọc kiểu cơ bản sẽ có một vài trường hợp sử dụng như sau:

- Đối số của phương thức: dù một phương thức khai báo tham số kiểu cơ bản hay kiểu lớp bọc ngoài thì nó vẫn chấp nhận đối số ở cả dạng cơ bản cũng như kiểu lớp bọc ngoài.
- Giá trị trả về: dù một phương thức khai báo kiểu trả về kiểu cơ bản hay bọc ngoài thì lệnh return trong phương thức dùng giá trị ở cả dạng cơ bản cũng như bọc ngoài đều được.
- Biểu thức boolean: ở những vị trí yêu cầu một biểu thức boolean, ta có thể dùng biểu thức cho giá trị boolean (chẳng hạn $2 < a$), hoặc một biến boolean, hoặc một tham chiếu kiểu Boolean đều được. Phép toán số học: ta có thể dùng tham chiếu kiểu bọc ngoài làm toán hạng của các phép toán số học, kể cả phép ++.

- Phép gán: ta có thể dùng một tham chiếu kiểu bọc ngoài để gán trị cho một biến kiểu cơ bản và ngược lại. Ví dụ: `Double d = 10.0;`

1.13. Mảng

1.13.1. Khái niệm

Mảng là tập hợp nhiều phần tử có cùng tên, cùng kiểu dữ liệu và mỗi phần tử trong mảng được truy xuất thông qua chỉ số của nó trong mảng.

1.13.2. Khai báo mảng

Java cũng như các ngôn ngữ khác có 2 loại mảng: Mảng 1 chiều và mảng nhiều chiều.

Cú pháp khai báo mảng 1 chiều:

`<kiểu dữ liệu> <tên mảng>[];` hoặc

`<kiểu dữ liệu>[] <tên mảng>;`

Ví dụ:

`int arrInt[];` hoặc

`int[] arrInt;`

Cú pháp khai báo mảng nhiều chiều:

`<Kiểu dữ liệu>[][]...[] <Tên mảng>;` hoặc

`<Kiểu dữ liệu> <Tên mảng> [][]...[];`

Ví dụ:

```
1 int a[][];
2 int[][] a;
```

1.13.3. Cấp phát bộ nhớ cho mảng

Không giống như trong C/C++ kích thước của mảng được xác định khi khai báo. Chẳng hạn như: `int arrInt[100];`. Để cấp phát bộ nhớ cho mảng trong Java ta cần dùng từ khóa `new`. Chẳng hạn để cấp phát vùng nhớ cho mảng trong Java ta làm như sau:

```
int arrInt = new int[100];
int arrInt2Dim[][]= new int[2][3];
```

1.13.4. Khởi tạo mảng

Chúng ta có thể khởi tạo giá trị ban đầu cho các phần tử của mảng khi nó được khai báo.

Ví dụ:

```
int arrInt[] = {1, 2, 3};
char arrChar[] = {'a', 'b','c'};
String arrString[] = {"ABC", "EFG", "GHI"};
int a2Dim[][]={{3,4},{2,8}};
```

1.13.5. Truy cập mảng

Chỉ số mảng trong Java bắt đầu tư 0. Vì vậy phần tử đầu tiên có chỉ số là 0, và phần tử thứ n có chỉ số là n-1. Các phần tử của mảng được truy xuất thông qua chỉ số của nó đặt giữa cặp dấu ngoặc vuông ([]).

Ví dụ:

```
int arrInt[] = {1, 2, 3};
int x = arrInt[0]; // x sẽ có giá trị là 1.
int y = arrInt[1]; // y sẽ có giá trị là 2.
int z = arrInt[2]; // z sẽ có giá trị là 3.
```

1.13.6. Duyệt mảng

Để duyệt mảng, ngoài việc sử dụng các cấu trúc lặp cổ điển như `for, while hay do...while` thì trong Java còn hỗ trợ cấu trúc `for-each` cho phép duyệt mảng rất tiện lợi.

Cú pháp:

```
for (<kiểu_dữ_liệu> <tên_biến>: <tên_mảng>)
    <khối_lệnh>;
```

Ví dụ:

```
int arr[] = { 10, 20, 30, 40 };
for (int i : arr) {
   System.out.println(i);
}
```

Kết quả:

```
10
20
30
40
```

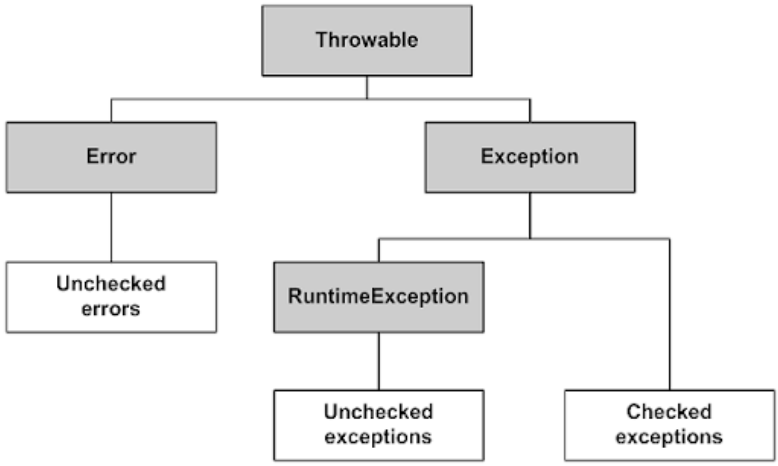
1.14. Xử lý ngoại lệ

Một ngoại lệ (Exception) trong Java là một vấn đề phát sinh trong quá trình thực thi chương trình. Khi xảy ra ngoại lệ, luồng xử lý (flow) bị gián đoạn, chương trình/ứng dụng dừng bất thường. Nó là một đối tượng được ném ra tại Runtime. Ngoại lệ trong Java có thể xảy ra vì nhiều lý do khác nhau:

- Nhập dữ liệu không hợp lệ.
- Không tìm thấy file cần mở.
- Kết nối mạng bị ngắt trong quá trình thực hiện giao tác.
- JVM hết bộ nhớ.
- Truy cập vượt ngoài chỉ số của mảng, ...

Ngoại lệ xảy ra có thể do người dùng, lập trình viên hoặc số khác do tài nguyên bị lỗi. Java Exeption được triển khai bằng cách sử dụng các lớp như `Throwable, Exception, RuntimeException` và các từ khóa như `throw, throws, try, catch và finally`.

Dựa vào tính chất các ngoại lệ, người ta chia ngoại lệ thành ba loại như hình 1.7:



Hình 1.7. *Cấu trúc của các ngoại lệ*

- Ngoại lệ được kiểm tra (Checked Exceptions): Là một ngoại lệ được kiểm tra và thông báo bởi trình biên dịch tại thời điểm biên dịch, chúng cũng có thể được gọi là ngoại lệ thời gian biên dịch (Compile-time Exceptions).
- Ngoại lệ không được kiểm tra (Unchecked Exceptions): Là một ngoại lệ không được kiểm tra trong quá trình biên dịch. Chúng cũng được gọi là ngoại lệ thời gian chạy (Runtime Exceptions)
- Lỗi (Error): Error là những vấn đề nghiêm trọng liên quan đến môi trường thực thi của ứng dụng hoặc hệ thống mà lập trình viên không thể kiểm soát. Nó thường làm chết chương trình. Ví dụ: OutOfMemoryError, VirtualMachineError, and StackOverflowError, ...

Xử lý ngoại lệ trong Java là một cơ chế mạnh mẽ để xử lý các lỗi runtime (Runtime exeption) để có thể duy trì luồng bình thường của ứng dụng. Khối lệnh try trong java được sử dụng để chứa một đoạn code có thế xảy ra một ngoại lệ. Nó phải được khai báo trong phương thức. Sau một khối lệnh try bạn phải khai báo khối lệnh catch hoặc finally hoặc cả hai.

1.14.1. Khối lệnh try-catch trong Java

Cú pháp:

```
try {
     // code có thể ném ra ngoại lệ
} catch (Exception_class_Name ref) {
     // code xử lý ngoại lệ
}
```

Khối catch trong Java được sử dụng để xử lý các Exception. Nó phải được sử dụng sau khối try. Chúng ta có thể sử dụng nhiều khối catch với một khối try duy nhất.

1.14.2. Khối lệnh try-finally trong Java

Cú pháp:

```
try {
     // code có thể ném ra ngoại lệ
} finally {
     // code trong khối này luôn được thực thi
}
```

Khối lệnh finally trong Java luôn được thực thi cho dù có ngoại lệ xảy ra hay không hoặc gặp lệnh return trong khối try. Khối lệnh finally trong Java được khai báo sau khối lệnh try hoặc sau khối lệnh catch.

1.15. Hướng dẫn Cài đặt JDK và Eclipse

1.15.1. Hướng dẫn Cài đặt JDK

- Bước 1. Tải JDK
  Truy cập vào địa chỉ: https://www.oracle.com/java/technologies/javase-downloads.html và chọn phiên bản JDK mà bạn muốn và nhấn chọn nút JDK Download (Giả sử chúng ta chọn Java SE 11 (LTS))
- Bước 2. Cài đặt JDK
  Sau khi tải JDK về, chúng ta tiến hành cài đặt bằng cách double-click vào chương trình mà chúng ta đã tải về và làm theo hướng dẫn trên màn hình cài đặt

- Bước 3. Thiết lập biến môi trường
  Giả sử JDK chúng ta cài đặt tại thư mục: C:/Java/JDK11, thì lúc này ta có thể thiết lập biến môi trường như sau:
  - Vào Control panel và chọn mục System
  - Nhấn Advanced System settings và chọn Environment Variables
  - Thêm thư mục bin trong thư mục JDK mà chúng ta đã cài đặt (C:/Java/JDK11/bin) vào biến PATH trong System Variables

  Lưu ý các đường dẫn thư mục trong PATH phải cách nhau bởi dấu ; và không phân biệt HOA thường.

- Bước 4. Kiểm tra cài đặt
  Để chắc chắn việc cài đặt ở trên là đúng đắn, chúng ta mở cửa sở Terminal của Window và gõ: `java - version`. Nếu kết quả trả về là phiên bản của JDK mà chúng ta đã cài thì việc cài đặt JDK đã hoàn tất.

1.15.2. Hướng dẫn Cài đặt Eclipse

- Bước 0. Để lập trình bằng ngôn ngữ Java thì chúng ta phải chắc chắn là đã cài JDK ở bước trước.

- Bước 1. Truy cập vào địa chỉ: https://www.eclipse.org/downloads/ và chọn nút "Download Packages" thay vì chọn nút "Download x86_64"

- Bước 2. Ở trang tiếp theo, chúng ta chọn "Windows x86_64" ở mục Eclipse IDE for Java Developers" và tiến hành Download

- Bước 3. Để chạy Eclipse, đơn giản là chúng ta tiến hành giải nén tập tin vừa mới download về và đặt ở thư mục phù hợp, chẳng hạn: C:
  Eclipse. Vì chương trình Eclipse mà chúng ta vừa tải chỉ là thư mục nén và không hề chạy chương trình để cài đặt lên máy tính, do đó chúng ta có thể đổi tên, di chuyển sang thư mục khác mà không ảnh hưởng gì.

1.15.3. Tạo chương trình đầu tiên, chương trình "Hello world"

- Khởi động Eclipse bằng cách nhấn vào file Eclipse.exe trong thư mục mà chúng ta vừa giải nén
- Chọn thư mục làm Workspace, nơi lưu trữ các chương trình của chúng ta
- Nếu xuất hiện màn hình Welcome thì chúng ta có thể đóng nó bằng cách nhấn nút Close ở thanh tiêu đề
- Để tạo dự án, chúng ta chọn menu "File" ⇒ "New" ⇒ "Java project" (Hoặc "File" ⇒ "New" ⇒ "Project" ⇒ "Java project").
- Cửa sổ "New Java Project" xuất hiện:
  - Trong "Project name", chúng ta gõ HelloWorld
  - Tích vào tùy chọn: Use default location
  - Trong JRE, chúng ta chọn "Use default JRE (currently 'JDK11.0.x')"
  - Trong "Project Layout" chúng ta nhấn vào tùy chọn "Use project folder as root for sources and class files"
  - Nhấn nút Finish để hoàn thành tạo mới dự án
  - Trong cửa sổ "Create module-info.java", chúng ta chọn "Don't"
- Trong khung "Package Explorer" nằm bên trái, nhấn chuột phải lên HelloWorld (hoặc trong menu File) chọn ⇒ New ⇒ Class. Cửa sổ "New Java Class" xuất hiện:
  - Trong "Source folder", vẫn để "HelloWorld".
  - Trong mục "Package" thì chúng ta tạm thời để rỗng
  - Trong mục "Name", chúng ta gõ Hello
  - Tích chọn "public static void main(String[] args)" để hệ thống sinh ra hàm main trong lớp Hello
  - Nhấn nút Finish để hoàn thành tạo lớp
  - File "Hello.java" trong khung soạn thảo sẽ trông như sau:

```
public class Hello {
   public static void main(String[] args) {
       System.out.println("Hello, world!");
```

```
    }
}
```

- Biên dịch và thực thi chương trình:
  - Để chạy chương trình, chúng ta nhấn chuột phải vào file Hello.java (Hoặc chọn menu Run) ⇒ Runas ⇒ Java Application
  - Trong khung Console sẽ xuất hiện Hello, World!
  - Như vậy việc cài đặt JDK và IDE Eclipse đã thành công.

1.16. Bài tập

Bài 1: Viết chương trình tìm ước số chung lớn nhất, bội số chung nhỏ nhất của hai số tự nhiên a và b.

Bài 2: Viết chương trình chuyển đổi một số tự nhiên ở hệ cơ số 10 thành số ở hệ cơ số b bất kì ($1 < b \leq 36$).

Bài 3: Hãy viết chương trình tính tổng các chữ số của một số nguyên bất kỳ. Ví dụ: Số 8545604 có tổng các chữ số là: $8+5+4+5+6+0+4=32$.

Bài 4: Viết chương trình phân tích một số nguyên thành các thừa số nguyên tố. Ví dụ: Số 28 được phân tích thành 2 x 2 x 7

Bài 5: Viết chương trình phân tích một số nguyên thành các thừa số nguyên tố. Ví dụ: Số 28 được phân tích thành 2 x 2 x 7

Bài 6: Viết chương trình liệt kê tất cả các số nguyên tố nhỏ hơn n cho trước.

Bài 7: Viết chương trình liệt kê n số nguyên tố đầu tiên.

Bài 8: Dãy số Fibonacci được định nghĩa như sau: $F_0 = 1, F_1 = 1; F_n = F_{n-1} + F_{n-2}$ với $n \geq 2$. Hãy viết chương trình tìm số Fibonacci thứ n.

Bài 9: Một số được gọi là số thuận nghịch độc nếu ta đọc từ trái sang phải hay từ phải sang trái số đó ta vẫn nhận được một số giống nhau. Hãy liệt kê tất cả các số thuận nghịch độc có sáu chữ số (Ví dụ số: 558855).

Bài 10: Viết chương trình liệt kê tất cả các xâu nhị phân độ dài n.

Bài 11: Viết chương trình liệt kê tất cả các tập con k phần tử của $1, 2, ..., n (k \leq n)$.