**CHƯƠNG I: GIỚI THIỆU NGÔN NGỮ LẬP TRÌNH JAVA**

**I. Đặc điểm ngôn ngữ lập trình Java:**

1 Đơn giản:

* Không có con trỏ.
* Không có nạp chồng toán tử.
* Không có Goto
* Không có Struct
* Không có Union //
* Không có đa kế thừa (dùng interface)
* Không có từ khoá friend

2. Thuần hướng đối tượng:

- Chương trình có ít nhất một lớp, hàm main cũng phải nằm trong một lớp.

- Hỗ trợ các tính chất hướng đối tượng: Đóng gói (Các class chứa thuộc tính và phương thức, tính đóng gói còn thể hiển qua các package), trừu tượng, kế thừa (không hỗ trợ đa kế thừa mà thay vào đó là các interface), đa hình.

- Lớp nội, interface

3. Độc lập nền tảng:

- Vừa thông dịch, vừa biên dịch (Javac biên dịch file .java thành bytescode có đuôi .class, JVM thông dịch sang mã máy)

4. Đăc điểm mới:

- Chặt chẽ kiểu dữ liệu: tường minh, phải cấp phát.

- Kiểm tra lúc biên dịch và cả trong thời gian thông dịch vì vậy Java loại bỏ một một số loại lỗi lập trình nhất định

- Kiểm tra tất cả các truy nhập đến mảng, chuỗi khi thực thi

- Kiểm tra sự chuyển đổi kiểu dữ liệu từ dạng này sang dạng khác lúc thực thi

- Giải phóng vùng nhớ tự động (Garbage Collection)

- Quản lý ngoại lệ (Exception)

5. Đa luồng:

- Cung cấp giải pháp đồng bộ giữa các luồng

- Cho phép xây dựng các ứng dụng trên mạng chạy hiệu quả

6. Phân tán:

- Xây dựng các ứng dụng có thể làm việc trên nhiều phần cứng, hệ điều hành và giao diện đồ họa

- Hỗ trợ cho các ứng dụng chạy trên Internet

7. An toàn:

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

- Dữ liệu và các phương thức được đóng gói bên trong lớp (đóng gói), không cho phép truy xuất bộ nhớ trực tiếp, không cho truy xuất thông tin bên ngoài kích thước của mảng, GC

- Trình biên dịch: kiểm soát để đảm bảo mã là an toàn, và tuân theo các nguyên tắc của Java

- Trình thông dịch: kiểm tra xem bytecode có đảm bảo các qui tắc an toàn trước khi thực thi

- 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

**II. Ngôn ngữ Java:**

1. Phân loại biến:

- Member:

+ Không cần khởi tạo giá trị (được tự động gán giá trị mặc định)

+ Được khai báo là thành phần của lớp.

- Local:

+ Bắt buộc phải khởi tạo giá trị trước khi sử dụng (nếu không sẽ tạo ra lỗi khi biên dịch)

+ Được khai báo trong một phương thức.

\* Từ khoá final

\* Kiểu dữ liệu cơ sở: primitive data type.

\* Kiểu dữ liệu tham chiếu: reference data type.

\* Một số lưu ý với kiểu dữ liệu cơ sở:

+ lớn => nhỏ thì cần ép kiểu, nhỏ => lớn thì tự động chuyển.

+ Không thể chuyển kiểu giữa int và boolean

+ Nếu một toán hạng kiểu double thì toán hạng kia chuyển thành double, nếu một toán hạng là float thì toán hạng kia chuyển thành float, nếu một toán hạng là long thì toán hạng kia chuyển về long, ngược lại thì tất cả chuyển thành int (cần ép kiểu đề không lỗi)

Ví dụ cho ý thứ 3:

byte a = 1;

byte b = 2;

byte c = a + b; // lỗi vì không ép kiểu nên a + b được tính là kiểu dữ liệu int, do byte nhỏ hơn int nên phải có từ khoá (byte) phía trước nếu không chương trình sẽ lỗi.

=> fix: byte c = (byte) a + b;

\*Một số lưu ý với kiểu dữ liệu tham chiếu:

+ Khai báo: <Kiểu đối tượng> <Tên đối tượng>;

+ Khởi tạo: <Kiểu đối tượng> <Tên đối tượng> = new <Kiểu đối tượng>;

2. Access control modifier:

- defaul:

- private:

- public:

- protected:

3. Non access modifier:

- abstract:

- final:

- static:

- synchronized:

4. Wrapper class:

- Là lớp bao kiểu dữ liệu primitive (viết hoa chữ cái đầu)

- Tác dụng:

+ Hỗ trợ truyền tham chiếu cho kiểu số.

+ Trong java hầu hết là lớp nên như vậy sẽ tạo sự tương thích.

+ Các cấu trúc trong Collection chỉ hỗ trợ các biến là đối tượng.

+ Chỉ các đối tượng mới được hỗ trợ đa luồng.

**CHƯƠNG II: HƯỚNG ĐỐI TƯỢNG TRONG JAVA**

**I. Sự khác nhau giữa hướng đối tượng trong Java và C++:**

|  |  |  |
| --- | --- | --- |
| Đặc điểm | C++ | Java |
| Ngôn ngữ thuần OOP | Không phải là ngôn ngữ thuần OOP, được phát triển từ C với tên là C++  hay được gọi là “C với các lớp”. | Java là ngôn ngữ thuần OOP. Nghĩa là thành phần nào nó cũng thuộc về lớp, nằm trong lớp. |
| Phạm vi truy cập | C++ có 3 phạm vi truy của cập của thuộc tính và phương thức: private, protected, public | Java mở rộng hơn C++ có 4 phạm vi truy cập: private, default, protected, public. |
| Nạp chồng toán tử | C++ có phép chúng ta nạp chồng các toán tử như toán tử vào ra << >>, + – \* / hay -= += /= \*/. | Java không cho phép nạp chồng toán tử. Mà chỉ có toán tử + cho phép nối hai chuổi. |
| Hỗ trợ đa kế thừa | C++ hổ trợ đa kế thừa, nghĩa là một class có kể kế thừa nhiều lớp dẫn xuất. | Java không cho phép bạn đa kế thừa từ nhiều lớp, mà chỉ kế thừa 1 lớp và được implements nhiều interface |
| Hỗ trợ interface | C++ không tồn tại khái niệm interface | Java hỗ trợ interface |
| Lớp vô sinh | C++ không có lớp vô sinh, nghĩa là lớp dẫn xuất được tạo ra thì các lớp con có thể kế thừa | Java có lớp vô sinh khi chúng ta đánh dấu final cho lớp. Khi đó lớp này sẽ không có lớp nào kế thừa từ nó. |
| Hàm bạn, lớp bạn | C++ có hổ trợ hàm bạn, lớp bạn | Java không hổ trợ hàm bạn, lớp bạn |
| Lớp cha của mọi lớp | Một lớp trong C++ được tạo ra và chưa kế thừa từ lớp nào thì nó không có lớp cha. | Mặc định trong Java khi một lớp được tạo ra thì cha của nó chính là lớp Object. Lớp Object này là cha của mọi lớp trong Java. |
| Tổ chức trong class | C++ chúng ta có thể gom nhóm những thuộc tính hay phương thức có cùng phạm vi truy cập với nhau. Ví dụ public: int a; int b; C++ định nghĩa các prototype trong file .h và định nghĩa thân các method trong file .cpp | Trước mỗi method hay thuộc tính chúng ta phải chỉ định phạm vi truy cập. Nếu không chỉ định thì Java hiểu đó là defaut.  Java khai báo và định nghĩa method trong một file .java |
| Phạm vi truy cập mặc định | Trong C++ nếu chúng ta không chỉ rõ phạm vi truy cập thì là private. | Trong Java nếu không chỉ rõ tường minh thì mặc định thuộc tính hay phương thức đó sẽ có phạm vi truy cập là default. |
| Phạm vi truy cập protected | Phạm vi truy cập là protected trong C++ chỉ truy cập được ở trong class định nghĩa và ở lớp con | Phạm vi truy cập protected trong Java được truy cập ở Java là trong class định nghĩa, trong cùng package và lớp con trong cùng package. |
| Gọi lại phương thức của lớp cha | C++ sử dụng cú pháp <class\_name>::method(); | Java sử dụng từ khóa super với cú pháp: Super.method(); |
| Gọi lại constructor của lớp | C++ không cho phép gọi lại contructor khác trong một contructor khác của lớp. | Java cho phép gọi lại contructor của lớp trong một contructor khác với cú pháp: this(parameter) |
| Kiểu kế thừa | Trong C++ có các kiểu kế thừa lớp là: Private, protected, public | Trong Java chỉ có kiểu kế thừa là public. |
| Phương thức vô sinh | Trong C++ không tồn tại phương thức vô sinh. Nghĩa là có thể override lại ở lớp con. | Trong Java nếu phương thức được đánh dấu là final thì ở lớp con không thể override lại được. |
| Phương thức thuần túy ảo | C++ có khái niệm phương thức thuần túy ảo. Với cú pháp: return type  name\_method()= 0; | Java không có khái niệm phương thức thuần túy ảo. |
| Lớp ảo | C++ nhận biết lớp ảo khi lớp đó có các phương thức đánh dấu với từ khóa virtual hoặc phương thức thuần ảo | Java nhận biết lớp ảo khi khai báo lớp có từ khóa abtract và có những phương thức được đánh dấu là abstract. |
| Phương thức ảo | Phương thức ảo trong C++ được đánh dấu bởi từ khóa virtual và có định nghĩa bên trong thân hàm. | Phương thức ảo trong Java được đánh dấu bởi từ khóa abstract. Những phương thức này không có định nghĩa thân hàm, kết thúc bằng dấu ‘;’ |
| Tổ chức class như thư mục | Trong C++ chúng ta nhắc đến namespace để định nghĩa các lớp trong này. | Java định nghĩa các class trong các package. |
| Lập trình đồng bộ | C++ không hổ trợ lập trình đồng bộ cho class. | Java hổ trợ lập trình đồng bộ với từ khóa synchronized. |
| Hàm main | Hàm main của C++ không thuộc về lớp nào. Là một hàm cục bộ | Hàm main trong java cũng nằm trong một class vì java là ngôn ngữ thuần OOP. |
| Destructor | C++ có destructor. Người lập trình phải chủ động thu dọn vùng nhớ đã cấp phát. | Java không có khái niệm destructor mà có bộ thu gom rác gọi là GC (garbage collection). Khi một đối tượng không còn tham chiếu nữa thì bộ thu gom rác sẽ hoạt động. |

**III. Xử lý ngoại lệ trong Java:**

**- Có hai loại ngoại lệ trong Java:**

**+ Unchecked Exception.**

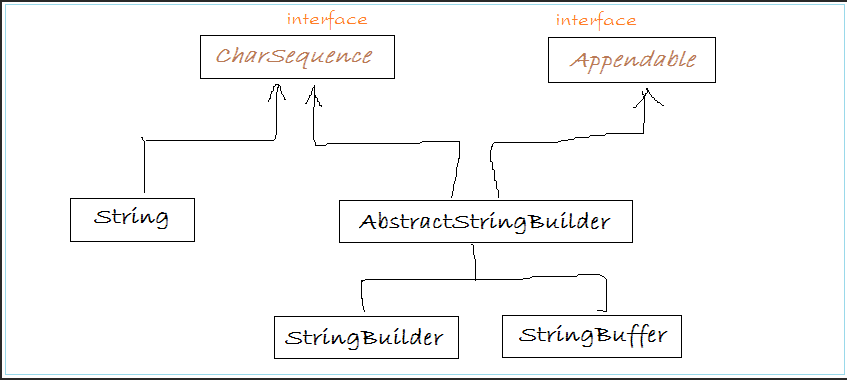
**Xử lý ngoại lệ trong Java được thực hiện theo mô hình hướng đối tượng:**

**+ Tất cả các ngoại lệ đều là thể hiện của môt lớp kế thừa từ lớp Throwable hoặc các lớp con của nó.**

**+ Các đối tượng này có nhiệm vụ chuyển thông tin về ngoại lệ (loại va trạng thai của chương trinh) từ vị trí xảy ra ngoại lệ đến nơi quản lý/xử lý nó+ Checked Exception.**

**IV. Các Class thông dụng trong Java:**

1. String, StringBuilder, StringBuffer:



- **String** không thay đổi (immutable) và không cho phép có class con.

- **StringBuffer**, **StringBuilder** có thể thay đổi (mutable).

**- StringBuilder** và **StringBuffer** là giống nhau, nó chỉ khác biệt tình huống sử dụng có liên quan tới đa luồng (Multi Thread).

* Nếu xử lý văn bản sử dụng nhiều luồng (Thread) bạn nên sử dụng **StringBuffer** để tránh tranh chấp giữa các luồng.
* Nếu xử lý văn bản sử dụng 1 luồng (Thread) nên sử dụng **StringBuilder**.

- Nếu so sánh về tốc độ xử lý **StringBuilder** là tốt nhất, sau đó **StringBuffer** và cuối cùng mới là **String**.

a) String:

\*String là class vừa có tính nguyên thuỷ vừa có tính đối tượng:

- Tính nguyên thuỷ:

+ Có thể tạo một **string literal** (chuỗi chữ), **string literal** được lưu trữ trong ngăn sếp (stack), đòi hỏi không gian lưu trữ ít, và rẻ hơn khi thao tác.

* String literal = "Hello World"; //không thông qua toán tử new.

+ Có thể sử dụng toán tử + để nối 2 string, toán tử này vốn quen thuộc và sử dụng cho các kiểu dữ + liệu nguyên thủy **int, float, double**.   
+ Các **string literal** được chứa trong một bể chứa (common pool). Như vậy hai string literal có nội dung giống nhau sử dụng chung một vùng bộ nhớ trên stack, điều này giúp tiết kiệm bộ nhớ.

- Tính đối tượng:

+ Có thể tạo ra một đối tượng bằng toán tử new.

+ Các đối tượng **String** được lưu trữ trên Heap, yêu cầu quản lý bộ nhớ phức tạp và tốn không gian lưu trữ. Hai đối tượng **String** có nội dung giống nhau lưu trữ trên 2 vùng bộ nhớ khác nhau của Heap.

\* Sử dụng String literal làm tăng tốc chương trình so với toán tử new.

b) StringBuilder và StringBuffer:  
- **StringBuilder** và **StringBuffer** là rất giống nhau, điều khác biệt là tất cả các phương thức của **StringBuffer** đã được đồng bộ, nó thích hợp khi bạn làm việc với ứng dụng đa luồng, nhiều luồng có thể truy cập vào một đối tượng **StringBuffer** cùng lúc. Trong khi đó **StringBuilder** có các phương thức tương tự nhưng không được đồng bộ, nhưng vì vậy mà hiệu suất của nó cao hơn, bạn nên sử dụng **StringBuilder** trong ứng dụng đơn luồng, hoặc sử dụng như một biến địa phương trong một phương thức.

**IV Các thông tin khác:**

**1. Immutable và mutable class:**

- Immutable là class không có các hàm set giá trị (chỉ có thể khởi tạo giá trị thông qua khởi tạo đối tượng, muốn thay đổi **2. equal() vs “==”**

thì phải tạo đối tượng mới).

- Mutable là class mà sau khi tạo đối tượng thì có thể thay đổi giá trị của nó.

**2. equal() vs “==”**

- equal(): dùng để so sánh hai đối tượng (với String là so sánh hai nội dung).

- “==”: với các kiểu tham chiếu thì “==” dùng để so sánh hai địa chỉ.

3. Java không cho phép gán giá trị mặc nhiên.