1. **OOP**

Abstraction

Encapsulation

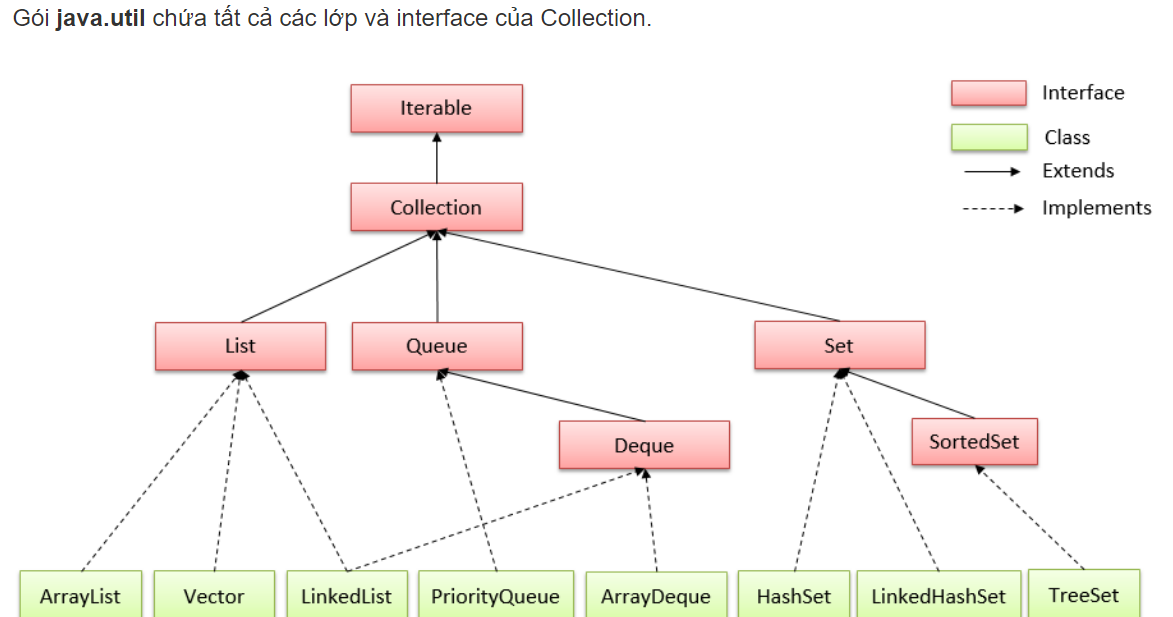
Inheritance

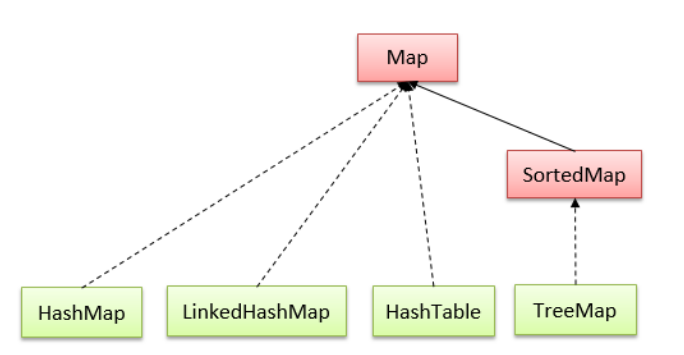
Polymorphism

1. **Collections vs Collection**

**Collections trong java** là một khuôn khổ cung cấp một kiến trúc để lưu trữ và thao tác tới nhóm các đối tượng. Tất cả các hoạt động mà bạn thực hiện trên một dữ liệu như tìm kiếm, phân loại, chèn, xóa,... có thể được thực hiện bởi Java Collections.

**Collection trong java** là một root interface trong hệ thống cấp bậc Collection. Java Collection cung cấp nhiều interface (Set, List, Queue, Deque vv) và các lớp (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet vv).





* **Set:**là một collection không thể chứa 2 giá trị trùng lặp. Set được sử dụng để biểu diễn các bộ, chẳng hạn như bộ tú lu khơ, thời khóa biểu của học sinh, các tiến trình đang chạy trên máy tính...
* **List:**là một collection có thứ tự (đôi khi còn được gọi là một chuỗi). List có thể chứa các phần tử trùng lặp. Thường có quyền kiểm soát chính xác vị trí các phần tử được chèn vào và có thể truy cập chúng bằng chỉ số (vị trí của chúng).
* **Queue (hàng đợi):**là một collection được sử dụng để chứa nhiều phần tử trước khi xử lý. Bên cạnh các thao tác cơ bản của collection, Queue cung cấp các thao tác bổ sung như chèn, lấy ra và kiểm tra. Queue có thể được sử dụng như là FIFO (first-in, first-out - vào trước, ra trước)
* **Deque:**là một collection được sử dụng để chứa nhiều phần tử trước khi xử lý. Ngoài các thao tác cơ bản của collection, một Deque cung cấp các thao tác bổ sung như chèn, lấy ra và kiểm tra. Deques có thể được sử dụng như là FIFO (first-in, first-out - vào trước, ra trước) và LIFO (last-in, first-out - vào sau, ra trước). Trong một Deque, tất cả các phần tử mới có thể được chèn vào, lấy ra và lấy ra ở cả hai đầu.
* **Map:**là một đối tượng ánh xạ mỗi key tương úng với một giá trị. Map không thể chứa giá trị trùng lặp. Mỗi key có thể ánh xạ đến nhiều nhất một giá trị.

1. **Static**

Trong Java, từ khóa **static** được sử dụng để quản lý bộ nhớ tốt hơn và nó có thể được truy cập trực tiếp thông qua lớp mà không cần khởi tạo.

Các trường hợp sử dụng static:

1. **Biến static (static variables):** khi bạn khai báo một biến là static, thì biến đó được gọi là biến tĩnh, hay biến static.
2. **Phương thức static (static methods):** khi bạn khai báo một phương thức là static, thì phương thức đó gọi là phương thức static.
3. **Khối static (static blocks):** được sử dụng để khởi tạo thành viên dữ liệu static.
4. **Lớp static (static class)**: một class được có thể được đặt là static chỉ khi nó là một nested class. Một  nested static class có thể được truy cập mà không cần một object của outer class (lớp bên ngoài).
5. **Import static**: từ phiên bản Java 5, cho phép import các thành viên tĩnh (static member) của một class hoặc package vào một class khác bằng cách sử dụng từ khóa import và sau đó sử dụng chúng như là thành viên của lớp đó.

Việc cấp phát bộ nhớ cho biến static chỉ xảy ra một lần khi class được nạp vào bộ nhớ

Nếu một biến vừa khai báo từ khóa final vừa khai báo từ khóa static thì nó được xem như là một hằng số.

1. **Over Loading và Overriding**

|  |  |  |
| --- | --- | --- |
| **No.** | **Nạp chồng phương thức (overloading)** | **Ghi đè phương thức (overriding)** |
| 1) | Nạp chồng phương thức được sử dụng để giúp code của chương trình *dễ đọc hơn*. | Ghi đè phương thức được sử dụng để cung cấp *cài đặt cụ thể* cho phương thức được khai báo ở lớp cha. |
| 2) | Nạp chồng được thực hiện bên *trong một class*. | Ghi đè phương thức xảy ra *trong 2 class* có quan hệ kế thừa. |
| 3) | Nạp chồng phương thức thì *tham số phải khác nhau*. | Ghi đè phương thức thì *tham số phải giống nhau*. |
| 4) | Nạp chồng phương thức là ví dụ về *đa hình lúc biên dịch*. | Ghi đè phương thức là ví dụ về *đa hình lúc runtime*. |
| 5) | Trong java, nạp chồng phương thức không thể được thực hiện khi chỉ thay đổi kiểu giá trị trả về của phương thức. Kiểu giá trị trả về có thể giống hoặc khác. *Giá trị trả về có thể giống hoặc khác*, nhưng tham số phải khác nhau. | Giá trị trả về phải giố |

1. **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.**

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 vấn đề, người ta chia ngoại lệ thành ba loại:

* Ngoại lệ được kiểm tra (Checked Exceptions).
* Ngoại lệ không được kiểm tra (Unchecked Exceptions).
* Lỗi (Error).

**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). Và lập trình viên không thể lường trước.

**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). Là ngoại lệ có thể tránh được bởi lập trình viên.

Về**throw,** nó được sử dụng để ném ra một ngoại lệ cụ thể. Chúng ta có thể ném một trong hai ngoại lệ checked hoặc unchecked trong java bằng từ khóa này. Từ khóa throw chủ yếu được sử dụng để ném ngoại lệ do người dùng tự định nghĩa.

Về **throws**, nó được sử dụng để khai báo một ngoại lệ. Nó thông báo cho lập trình viên rằng một ngoại lệ có thể xảy ra. Ngoại lệ checked được truyền ra ngay cả khi chỉ sử dụng từ khóa throws. Throws được khai báo ngay sau dấu đóng ngoặc đơn của phương thức.

1. **String**

Lớp java.lang.String được implements từ các interface Serializable, Comparable and CharSequence.

CharSequence Interface được sử dụng để biểu diễn chuỗi các ký tự. Nó được cài đặt bởi các lớp String, StringBuffer và StringBuilder. Điều này có nghĩa là chúng ta có thể tạo chuỗi ký tự trong java bằng cách sử dụng 3 lớp này.

String là **bất biến** (immutable) tức là không thể thay đổi. Có nghĩa là khi nào bạn thay đổi giá trị của bất kỳ chuỗi nào thì một instance mới được tạo ra. Đối với chuỗi có thể thay đổi, bạn có thể sử dụng các lớp StringBuffer và StringBuilder.

String literal được tạo ra bằng cách sử dụng 2 dấu nháy kép.

Mỗi khi bạo tạo một biến string literal, đầu tiên JVM sẽ kiểm tra xem giá trị đó đã tồn tại trong Pool chưa. Nếu chuỗi này đã tồn tại trong Pool, thì giá trị của biến sẽ được tham chiếu đến instance đã được tạo ra trong Pool. Nếu chuỗi này không tồn tại trong Pool, một instance mới được tạo ra và đặt vào trong Pool

Theo ví dụ trên, chỉ có một đối tượng chuỗi "Webcome" được tạo ra. Biến s1 và s2 được tham chiếu đến đối tượng chuỗi "Webcome".

String s=**new** String("Welcome"); /

Trong trường hợp này, JVM sẽ tạo ra một đối tượng string mới như một đối tượng trong bộ nhớ HEAP và chữ "Welcome" sẽ được đặt trong Pool. Biến s sẽ tham chiếu tối đối tượng được tạo ra trong HEAP.

1. **Equals and Hashcode**

**Equals:**

Khi so sánh hai đối tượng với nhau, Java gọi phương thức equals() của chúng trả về true nếu hai đối tượng bằng nhau hoặc false nếu hai đối tượng là khác nhau. Lưu ý rằng phép so sánh sử dụng phương thức equals() so với sử dụng toán tử == là khác nhau.

Phương thức equals() được thiết kế để so sánh hai đối tượng về mặt ngữ nghĩa (bằng cách so sánh các thành viên dữ liệu của lớp), trong khi toán tử == so sánh hai đối tượng về mặt kỹ thuật (bằng cách so sánh các tham chiếu của chúng, nghĩa là địa chỉ bộ nhớ).

**Hashcode:**

Số băm này được sử dụng bởi các collection dựa trên bảng băm như Hashtable , HashSet và HashMap để lưu trữ các đối tượng trong các container nhỏ được gọi là "nhóm". Mỗi nhóm được liên kết với mã băm và mỗi nhóm chỉ chứa các đối tượng có mã băm giống hệt nhau.

Nói cách khác, một bảng băm nhóm các phần tử của nó bằng các giá trị mã băm của chúng. Sự sắp xếp này giúp cho bảng băm định vị một phần tử một cách nhanh chóng và hiệu quả bằng cách tìm kiếm trên các phần nhỏ của collection thay vì toàn bộ collection.

Dưới đây là các bước để định vị một phần tử trong một bảng băm:

* Nhận giá trị mã băm của phần tử được chỉ định bằng cách gọi phương thức hashCode().
* Tìm nhóm thích hợp được liên kết với mã băm đó.
* Bên trong nhóm, tìm phần tử chính xác bằng cách so sánh phần tử được chỉ định với tất cả các phần tử trong nhóm. Bằng phương thức equals() của phần tử đã chỉ định được gọi.