## **1. Biến local trong java**

* Biến local được tạo bên trong các phương thức, contructor, block và sẽ bị phá hủy khi kết thúc các phương thức, contructor và block.
* Không được sử dụng "access modifier" khi khai báo biến local.
* Các biến local được lưu trên vùng nhớ stack của bộ nhớ.
* Bạn cần khởi tạo giá trị mặc định cho biến local trước khi có thể sử dụng.

## **2. Biến biến instance**

* iến 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, 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.

## **3. Biến static trong java**

* Biến static được khai báo trong một class với từ khóa "static", phía bên ngoài các phương thức, constructor và block.
* 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 bạn khai báo tương tự biến instance.

OOP

**Tính đóng gói trong java**

để bảo vệ trạng thái bên trong của một đối tượng. Bởi việc ẩn giấu các biến biểu diễn trạng thái của đối tượng. Việc chỉnh sửa đối tượng được thực hiện, xác nhận thông qua các phương thức. Hơn nữa, việc ẩn giấu các biến thì các lớp sẽ không chia sẻ thông tin với nhau được. Điều này làm giảm số lượng khớp nối có thể có trong một ứng dụng.

**Đa hình trong java (Polymorphism)** là một khái niệm mà chúng ta có thể thực hiện một hành động bằng nhiều cách khác nhau. Có hai kiểu của đa hình trong java, đó là đa hình lúc biên dịch (compile) và đa hình lúc thực thi (runtime).

**Kế thừa trong java** là sự liên quan giữa hai class với nhau, trong đó có class cha (superclass) và class con (subclass). Khi kế thừa class con được hưởng tất cả các phương thức và thuộc tính của class cha. Tuy nhiên, nó chỉ được truy cập các thành viên public và protected của class cha. Nó không được phép truy cập đến thành viên private của class cha.

## **Sự khác nhau giữa overloading và overriding trong java**

Sự khác nhau giữa overloading và overriding phương thức trong java được thể hiện trong bảng sau:

|  |  |  |
| --- | --- | --- |
| **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 giống nhau. |

## **1. Tham chiếu tới biến instance của lớp hiện tại.**

Từ khóa this trong java có thể được dùng để tham chiếu tới biến instance của lớp hiện tại.

Nếu có sự trùng tên nhau giữa biến toàn cục và tham số khiến bạn bị phân vân. Từ khóa this sẽ giúp bạn giải quyết sự phân vân của bạn.

**Từ khóa super trong java** là một biến tham chiếu được sử dụng để tham chiếu trực tiếp đến đối tượng của lớp cha gần nhất.

Bất cứ khi nào bạn tạo ra instance(thể hiển) của lớp con, một instance của lớp cha được tạo ra ngầm định, nghĩa là được tham chiếu bởi biến **super**.

1. **Biến final:** bạn không thể thay đổi giá trị của biến final (nó sẽ là hằng số).
2. **Phương thức final:** bạn không thể ghi đè phương thức final.
3. **Lớp final:** bạn không thể kế thừa lớp final.
4. **Biến static final trống:** Một biến final mà không được khởi tạo tại thời điểm khai báo được gọi là biến final trống.

ảng dưới đây mô tả khả năng truy cập của các Access Modifier trong java:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **Trong lớp** | **Trong package** | **Ngoài package bởi lớp con** | **Ngoài package** |
| Private | Y | N | N | N |
| Default | Y | Y | N | N |
| Protected | Y | Y | Y | N |
| Public | Y | Y | Y | Y |

**Sự khác nhau giữa Array với ArrayList** được tóm tắt lại như trong bảng sau:

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| 1) Kích thước **cố định**. | Kích thước có thể **thay đổi được**. |
| 2) Có thể lưu trữ dữ liệu kiểu **nguyên thủy** và **đối tượng**. | Chỉ có thể lưu trữ dữ liệu kiểu **đối tượng**. Kể từ Java 5, kiểu nguyên thủy được tự động chuyển đổi trong các đối tượng được gọi là **auto-boxing**. |
| 3) Tốc độ lưu trữ và thao tác **nhanh hơn**. | Tốc độ lưu trữ vào thao tác **chậm hơn**. |
| 4) Chỉ có thuộc tính **length**. | Có nhiều phương thức để thao tác với dữ liệu. |

## **Tính trừu tượng trong java**

Tính trừu tượng là một tiến trình ẩn các cài đặt chi tiết và chỉ hiển thị tính năng tới người dùng.

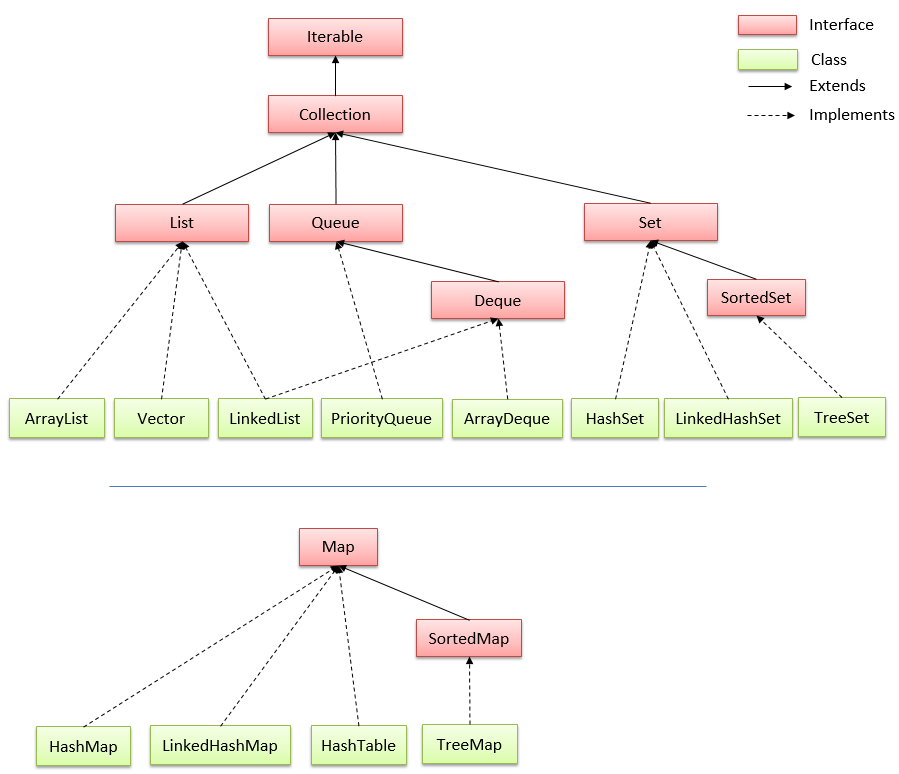
## **Sự khác nhau giữa Abstract class và Interface**

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class có phương thức **abstract**(không có thân hàm) và phương thức **non-abstract** (có thân hàm). | Interface chỉ có phương thức **abstract**. Từ java 8, nó có thêm **các phương thức default và static**. |
| 2) Abstract class **không hỗ trợ đa kế thừa**. | Interface **có hỗ trợ đa kế thừa** |
| 3) Abstract class có các biến **final, non-final, static and non-static**. | Interface chỉ có các biến **static và final**. |
| 4) Abstract class **có thể cung cấp nội dung cài đặt cho phương thức của interface**. | Interface **không thể cung cấp nội dung cài đặt cho phương thức của abstract class**. |
| 5) Từ khóa **abstract** được sử dụng để khai báo abstract class. | Từ khóa **interface** được sử dụng để khai báo interface. |
| 6) Ví dụ: public abstract class Shape { public abstract void draw(); } | Ví dụ: public interface Drawable { void draw(); } |

## **Lớp Wrapper trong java**  **autoboxing** và **unboxing**

|  |  |
| --- | --- |
| **Kiểu nguyên thủy** | **Kiểu Wrapper** |
| boolean | Boolean |
| char | Character |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |

## **Hệ thống cấp bậc Collection trong java**



### **Dưới đây là mô tả những interface chính của Collection**

* **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.

### **Dưới đây là mô tả 2 interface được sắp xếp của Set mà Map**

* **SortedSet:** là một Set chứa các phần tử theo thứ tự tăng dần.
* **SortedMap:** là một Map chứa các phần tử được sắp xếp theo thứ tự tăng dần của key của chúng. Các SortedMap được sử dụng cho các collection theo thứ tự tự nhiên của cặp key/value, chẳng hạn như từ điển và danh bạ điện thoại.

## **Sự khác nhau giữa ArrayList và LinkedList**

Cả hai lớp ArrayList và LinkedList đều được implements từ giao tiếp List và duy trì thứ tự của phần tử được thêm vào. Cả hai lớp này đều là lớp không đồng bộ (non-synchronized).

Có vài sự khác nhau giữa ArrayList và LinkedList được đưa ra như trong bảng dưới đây:

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList nội bộ sử dụng **mảng động**để lưu trữ các phần tử. | LinkedList nội bộ sử dụng **danh sách liên kết doubly**để lưu trữ các phần tử. |
| 2) Thao tác với ArrayList là **chậm**bởi vì nó sử dụng nội bộ mảng. Nếu bất kỳ phần tử nào được xoá khỏi mảng, tất cả các bit được chuyển trong bộ nhớ. | Thao tác với LinkedList là **nhanh hơn**so với ArrayList bởi vì nó sử dụng danh sách liên kết doubly do đó không cần chuyển đổi bit nào trong bộ nhớ. |
| 3) [Lớp ArrayList trong java](https://viettuts.vn/java-collection/arraylist-trong-java) chỉ có thể **hoạt động như một list**vì nó chỉ implements giao tiếp List. | [Lớp LinkedList trong java](https://viettuts.vn/java-collection/linkedlist-trong-java) có thể **hoạt động như một list và queue(hàng đợi)** vì nó implements các giao tiếp List và Deque. |
| 4) ArrayList là **tốt hơn trong việc lưu trữ và truy cập** dữ liệu. | LinkedList là **tốt hơn trong việc thao tác** dữ liệu. |

## **Sự khác nhau giữa String và StringBuffer trong java**

Sự khác nhau giữa String và StringBuffer trong java được thể hiện trong bảng sau:

|  |  |  |
| --- | --- | --- |
| No. | String | StringBuffer |
| 1) | Lớp String là bất biến (immutable). | Lớp StringBuffer là có thể sửa đổi (mutable). |
| 2) | Khi bạn thực hiện nối nhiều chuỗi thì lớp String xử lý chậm và tốn nhiều bộ nhớ hơn, bởi vì mỗi lần nối thêm chuỗi nó tạo ra instance mới. | Khi bạn thực hiện nối nhiều chuỗi thì lớp StringBuffer xử lý nhanh và tốn ít bộ nhớ hơn. |
| 3) | Lớp String ghi đề phương thức equals() của lớp Object. Vì thế bạn có thể so sánh nội dung của 2 chuỗi bằng phương thức equals(). | Lớp StringBuffer không ghi đề phương thức equals() của lớp Object. |

StringBuilder hiệu quả hơn StringBuffer.

|  |  |
| --- | --- |
| 1) Bạn phải implements giao tiếp Comparable cho lớp đối tượng cần được so sánh. | Bạn **KHÔNG** phải implements giao tiếp Comparator cho lớp đối tượng cần được so sánh. |
| 2) Comparable cung cấp **phương thức compareTo()**để sắp xếp các phần tử. | Comparator cung cấp **phương thức compare()**để sắp xếp các phần tử. |
| 3) Comparable thuộc về **java.lang** package. | Comparator thuộc về **java.util** package. |
| 4) Chúng ta có thể sắp xếp các phần tử của kiểu Comparable bởi **phương thức Collections.sort(List)** . | Chúng ta có thể sắp xếp các phần tử của kiểu Comparator bởi **phương thức Collections.sort(List,Comparator)**. |