List Interface

1. Nêu ra các đặc điểm **List Interface**

**- Đặc điểm của List Interface**

**List** là một trong những interface quan trọng nhất trong **Java Collections Framework**. Nó kế thừa từ interface Collection và đại diện cho một danh sách có thứ tự (ordered collection). List cho phép kiểm soát chính xác vị trí mà mỗi phần tử được chèn vào.

Các đặc điểm chính của List Interface bao gồm:

* **Có thứ tự**: Các phần tử được lưu trữ và truy cập theo một thứ tự cụ thể, dựa trên chỉ mục (index) của chúng.
* **Cho phép trùng lặp**: List có thể chứa nhiều phần tử có giá trị giống nhau.
* **Truy cập ngẫu nhiên**: Có thể truy cập bất kỳ phần tử nào trong danh sách một cách nhanh chóng thông qua chỉ mục của nó (ví dụ: list.get(2)).

*List<Type> list = new ArrayList<Type>();*

2. Kể ra các **class**triển khai từ **List Interface**

**List Interface** được triển khai bởi một số lớp cụ thể, mỗi lớp có những ưu và nhược điểm riêng. Ba lớp phổ biến nhất là:

**ArrayList**: Lớp này sử dụng một mảng động (dynamic array) để lưu trữ các phần tử.

*ArrayList<Integer> arr = new ArrayList<>();*

*ArrayList<String> arr = new ArrayList<>(collection);*

*ArrayList<Double> arr = new ArrayList<>(20);*

**LinkedList**: Lớp này sử dụng một cấu trúc dữ liệu danh sách liên kết kép (doubly linked list).

*LinkedList list = new LinkedList();*

*LinkedList list = new LinkedList(C);*

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| Mảng động | Danh sách liên kết kép |
| O(1) – truy cập ngẫu nhiên nhanh | O(n) – truy cập ngẫu nhiên chậm |
| Bộ nhớ thấp | Bộ nhớ cao |
| Tốc độ duyệt nhanh hơn | Tốc độ duyệt chậm hơn |
| Chèn và xóa chậm hơn | Chèn và xóa nhanh hơn |

**Vector**: Tương tự như ArrayList, nhưng là phiên bản cũ hơn và có tính đồng bộ (synchronized).

3. Phân biệt rõ trường hợp sử dụng của từng **class**đó

**ArrayList** là lựa chọn phổ biến nhất và được sử dụng rộng rãi nhất. Không đồng bộ: ArrayList sử dụng Collections.synchronizedList() để đảm bảo an toàn cho luồng.

* **Cấu trúc dữ liệu**: Dựa trên mảng động.
* **Ưu điểm**:
* **Truy cập nhanh**: Truy cập ngẫu nhiên một phần tử thông qua chỉ mục rất nhanh (độ phức tạp O(1)) vì các phần tử được lưu trữ liên tiếp trong bộ nhớ.
* **Sử dụng hiệu quả bộ nhớ**: Tận dụng tốt bộ nhớ cache của CPU.
* ArrayList bảo toàn thứ tự của các phần tử, cho phép bạn truy cập các phần tử theo thứ tự chúng được thêm vào.
* **Hỗ trợ giá trị null:** ArrayList có thể lưu trữ giá trị null, rất hữu ích trong trường hợp cần biểu diễn sự vắng mặt của một giá trị.
* **Nhược điểm**:
* **Thêm/xóa ở giữa danh sách chậm**: Khi thêm hoặc xóa một phần tử ở giữa, các phần tử còn lại phải được di chuyển, gây tốn kém thời gian (độ phức tạp O(n)).

**Trường hợp sử dụng**: Dùng khi cần truy cập ngẫu nhiên các phần tử thường xuyên hơn là thêm hoặc xóa chúng. Ví dụ: lưu trữ danh sách các sản phẩm trong một cửa hàng trực tuyến, nơi bạn cần hiển thị danh sách đó nhưng không thay đổi nó thường xuyên.

**LinkedList** là một lựa chọn tốt khi bạn cần thường xuyên thay đổi danh sách.

* **Cấu trúc dữ liệu**: Dựa trên danh sách liên kết kép. Mỗi phần tử (node) chứa dữ liệu và con trỏ đến phần tử đứng trước và sau nó.
* **Ưu điểm**:
* **Thêm/xóa nhanh**: Thêm hoặc xóa một phần tử ở bất kỳ vị trí nào, đặc biệt là ở đầu hoặc cuối danh sách, rất nhanh (độ phức tạp O(1)) vì chỉ cần thay đổi con trỏ.
* Danh sách liên kết kép cho phép duyệt theo cả hướng thuận và hướng ngược.
* **Nhược điểm**:
* **Truy cập chậm**: Để truy cập một phần tử ở giữa, bạn phải duyệt qua danh sách từ đầu hoặc cuối cho đến khi tìm thấy phần tử đó (độ phức tạp O(n)).
* **Tốn bộ nhớ hơn**: Mỗi node phải lưu thêm con trỏ, làm tăng chi phí bộ nhớ.

**Trường hợp sử dụng**: Dùng khi bạn thường xuyên thêm, xóa các phần tử, đặc biệt là ở đầu và cuối danh sách. Ví dụ: hệ thống quản lý danh sách nhạc, nơi người dùng có thể thêm bài hát vào đầu hoặc cuối danh sách phát.

**Vector** là một lớp cũ hơn, có một điểm khác biệt lớn so với ArrayList.

* **Cấu trúc dữ liệu**: Tương tự như ArrayList, dựa trên mảng động.
* **Ưu điểm**:
* **Đồng bộ (Synchronized)**: Các phương thức của Vector được đồng bộ hóa. Điều này có nghĩa là nó an toàn cho môi trường đa luồng (multi-threaded), nơi nhiều luồng có thể truy cập và thay đổi danh sách cùng lúc mà không gây ra lỗi.
* **Nhược điểm**:
* **Hiệu suất thấp**: Tính đồng bộ hóa làm giảm hiệu suất vì mỗi thao tác phải chờ luồng khác hoàn thành.

**Trường hợp sử dụng**: Rất ít khi được dùng trong các ứng dụng hiện đại. Nếu làm việc trong môi trường đa luồng, nên sử dụng các lớp không đồng bộ (như ArrayList) và tự quản lý việc đồng bộ hóa bên ngoài bằng cách sử dụng các lớp như Collections.synchronizedList() hoặc CopyOnWriteArrayList để có hiệu suất tốt hơn.