List Interface

**1. Đặc điểm của List Interface trong Java**

List là một **interface** trong Java Collections Framework, mở rộng từ interface **Collection**, cung cấp các tính năng để thao tác với **danh sách tuần tự** (sequence of elements) và hỗ trợ các thao tác như thêm, sửa, xóa, và truy cập phần tử.

**Các đặc điểm chính của List Interface:**

* **Duy trì thứ tự chèn**: Mọi phần tử trong List đều có **chỉ số (index)** và thứ tự chèn được bảo toàn. Phần tử đầu tiên có chỉ số là 0, và các phần tử tiếp theo có chỉ số tăng dần.
* **Cho phép phần tử trùng lặp**: List có thể chứa các phần tử trùng lặp (duplicates), nghĩa là có thể thêm nhiều phần tử có giá trị giống nhau.
* **Hỗ trợ truy cập ngẫu nhiên**: Bạn có thể truy cập phần tử tại vị trí bất kỳ trong danh sách thông qua chỉ số của nó.
* **Hỗ trợ chèn/xóa theo vị trí cụ thể**: List cung cấp các phương thức cho phép bạn chèn hoặc xóa phần tử tại một vị trí cụ thể.
* **Các phương thức chính**:
  + add(E element): Thêm phần tử vào danh sách.
  + remove(int index): Xóa phần tử tại chỉ số index.
  + get(int index): Truy cập phần tử tại chỉ số index.
  + set(int index, E element): Thay thế phần tử tại chỉ số index bằng phần tử mới.

**2. Các class triển khai từ List Interface**

Các class phổ biến triển khai từ List Interface bao gồm:

* **ArrayList**
* **LinkedList**
* **Vector**
* **Stack** (lớp con của Vector)

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

**a. ArrayList**

* **Đặc điểm**:
  + Sử dụng **mảng động** làm cấu trúc dữ liệu nền tảng.
  + Cung cấp **truy cập ngẫu nhiên** nhanh (O(1)) vì các phần tử được lưu trữ trong mảng.
  + **Thêm/xóa phần tử ở cuối** nhanh (O(1)), nhưng **thêm/xóa phần tử ở giữa hoặc đầu** chậm (O(n)) vì phải dịch chuyển các phần tử còn lại.
  + Không được đồng bộ hóa (unsynchronized), vì vậy không an toàn cho các luồng (thread-safe).
* **Khi nào sử dụng**:
  + Khi bạn cần **truy cập nhanh** các phần tử thông qua chỉ số.
  + Khi bạn **thao tác chủ yếu ở cuối danh sách** (thêm hoặc xóa phần tử ở cuối).
  + Khi không cần đồng bộ hóa dữ liệu.

**b. LinkedList**

* **Đặc điểm**:
  + Sử dụng **danh sách liên kết kép** (doubly linked list) làm cấu trúc dữ liệu nền tảng.
  + **Thêm/xóa phần tử ở đầu và cuối** nhanh (O(1)), nhưng **truy cập phần tử thông qua chỉ số** chậm (O(n)) vì phải duyệt qua các phần tử từ đầu đến cuối.
  + Hỗ trợ như một **deque** (danh sách hai đầu) và **queue** (hàng đợi), cung cấp các phương thức như addFirst(), addLast(), removeFirst(), removeLast().
  + Không được đồng bộ hóa (unsynchronized).
* **Khi nào sử dụng**:
  + Khi bạn cần thực hiện nhiều **thao tác thêm/xóa ở đầu hoặc cuối danh sách**.
  + Khi bạn **thao tác theo kiểu hàng đợi** (queue) hoặc **deque**.

**c. Vector**

* **Đặc điểm**:
  + Cũng giống như ArrayList, Vector sử dụng **mảng động** làm cấu trúc dữ liệu nền.
  + **Đồng bộ hóa** (synchronized), nghĩa là Vector an toàn cho các luồng (thread-safe), nhưng đồng thời khiến nó **chậm hơn** so với ArrayList trong trường hợp đơn luồng.
  + Hỗ trợ các phương thức tương tự ArrayList, như thêm, xóa và truy cập phần tử.
* **Khi nào sử dụng**:
  + Khi bạn cần một danh sách có tính năng giống ArrayList, nhưng cần **đồng bộ hóa** để sử dụng trong môi trường đa luồng (multi-threaded).

Vd:

List<String> vector = new Vector<>();

vector.add("Java");

vector.add("Python");

vector.add(1, "C++"); // Thêm phần tử tại vị trí thứ 1

System.out.println(vector.get(2)); // Truy cập phần tử thứ 2

**d. Stack (lớp con của Vector)**

* **Đặc điểm**:
  + Stack là một **cấu trúc dữ liệu ngăn xếp** (last-in, first-out - LIFO), được triển khai dựa trên Vector.
  + Hỗ trợ các thao tác tiêu chuẩn của ngăn xếp như push() (thêm phần tử vào đỉnh ngăn xếp), pop() (loại bỏ và trả về phần tử ở đỉnh), và peek() (trả về phần tử ở đỉnh mà không loại bỏ nó).
  + Giống như Vector, Stack cũng **đồng bộ hóa**.
* **Khi nào sử dụng**:
  + Khi bạn cần một **ngăn xếp LIFO** để xử lý dữ liệu trong môi trường đa luồng.