1. Nêu ra các đặc điểm List Interface ?

* List duy trì thứ tự chèn của các phần tử, tức là các phần tử sẽ được lưu trữ và truy xuất theo đúng thứ tự mà chúng được thêm vào.
* List cho phép lưu trữ các phần tử có giá trị trùng lặp. Nghĩa là một List có thể chứa nhiều phần tử với giá trị giống hệt nhau.
* Các phần tử trong List có thể được truy cập theo chỉ mục (vị trí) của chúng. Chỉ mục bắt đầu từ 0. Ví dụ sử dụng phương thức get(int index) để truy xuất phần tử tại một vị trí cụ thể.
* List cung cấp các phương thức đặc biệt cho việc thêm, xóa, cập nhật phần tử dựa vào index.
* List hỗ trợ cả việc sử dụng Iterator để duyệt qua các phần tử, và cung cấp thêm một ListIterator cho phép duyệt ngược lại danh sách, cũng như thay đổi các phần tử trong khi duyệt.
* List cho phép chứa giá trị null.

1. Kể ra các class triển khai từ List Interface?

2.1. ArrayList

* Là lớp lưu trữ phần tử trong mảng động , chậm hơn mảng thông thường nhưng có kích thước lơn hơn do được cấp phát động.
* Ưu điểm :
  + Kích thước mảng động thuận tiện cho việc thêm phần tử vào mảng .
  + Truy cập đến phần tử trongmangr nhanh do triển khai cấu trúc mảng bên trong.
  + Giữ nguyên phần tử khi được thêm vào , cho phép truy cập theo thứ tự chúng được thêm vào .
  + Cho phép lưu trữ giá trị null.
* Nhược điểm :
  + Truy vấn chậm hơn mảng .
  + Không có thread-safe nên không an toàn trong môi trường đa luồng.
  + Giảm hiệu suất khi tìm kiếm , thêm hoặc xoá phần tử ở giữa mảng.
* Cơ chế hoạt động :
  + Cơ chế khi thêm :
    - Nếu phần tử trong arraylist chưa đầy sẽ tự đọng thêm vào cuối
    - Nếu phần tử đã đầy arraylist sẽ tạo ra mảng mới với kích thước gấp đôi , sau đó copy mảng cũ sang rồi mới thêm phần tử mới vào.
  + Cơ chế khi tim kiếm :
    - Ta có thể tìm kiếm với index với tốc độ cao
    - Nếu tìm theo object ta cần duyệt cả mảng .
  + Cơ chế xoá :
    - Xoá ở vị trí chỉ định rồi các phần tử đằng sau nó sẽ được đánh index lại .
    - Xoá phần tử cuối cùng sẽ rất nhanh vì các phần tử ko cần dịch chuyển .
  + Cơ chế khi duyệt mảng :
    - duyệt qua các vòng lặp vqua từng phần tử .
* Trường hợp sử dụng :
  + Khi cần thực hiện các thao tác **đọc/truy xuất** dữ liệu.

2.2. LinkedList

* LinkedList được triển khai dưới dạng danh sách liên kết kép. Nó hỗ trợ tốt cho việc thêm, xóa các phần tử ở đầu, cuối hoặc giữa danh sách mà không cần di chuyển các phần tử khác.
* Ưu điểm :
  + Bộ nhớ cấp phát động nên không cần lo lắng về kích cỡ khi sử dụng
  + Là cấu trúc dự liệu hiệu quả cho việc thêm hoặc xoá phần tử ở giữa vì nó chỉ việc thay đổi liên kết giữa các phần tử mà không cần dịch chuyển chúng.
  + lặp lại danh sách một cách hiệu quả vì trong linkedlist các phần tử liên kết với nhau .
* Nhược điểm
  + Tốc độ truy cập đến 1 phần tử chậm hơn so với array list vì ta phải duyệt đến phần đó trong mảng trong khi arraylist có thể truy cập bằng index.
  + LinkedList yêu cầu nhiều bộ nhớ hơn ArrayList vì mỗi phần tử yêu cầu bộ nhớ bổ sung cho các liên kết đến các phần tử trước và phần tử sau của nó.
* Cơ chế hoạt động :
  + Cơ chế khi thêm :
    - Khi thêm ta cần thay đổi liên kết giữa các phần tử .
    - thêm vào đầu ta chỉ cần đỏi 1 liên kêt
    - thêm vào vị trí n ta cần duyệt đến vị trí n rồi đổi liên kết
  + Cơ chế khi tim kiếm :
    - Duyệt cả mảng cho đến khi tìm thấy phần tử cần tìm
  + Cơ chế xoá :
    - Xoá ở vị trí chỉ định rồi thay đổi liên kết .
    - Xoá phần tử đầu sẽ rất nhanh vì chỉ cần thay đỏi 1 liên kết.
  + Cơ chế khi duyệt mảng :
    - duyệt qua các vòng lặp vqua từng phần tử .
* Trường hợp sử dụng:
  + Thích hợp khi cần thực hiện nhiều thao tác **thêm/xóa** dữ liệu hơn là truy cập ngẫu nhiên. Hiệu suất tốt khi thao tác với các phần tử đầu hoặc cuối danh sách.

2.3 Vector

* Vector tương tự như arraylist nhưng có cơ chế thread-safe .
* Ưu điểm :
  + Đảm bảo an toàn dữ liệu trong môi trường đa luồng do có sẵn cơ chế thread-safe
* Nhược điểm :
  + Suy giảm hiệu suất của thao tác tìm kiếm , thêm và xoá giữa danh sách.
  + Nếu không dùng trong môi trường đa luồng thì tiêu tốn bộ nhớ không cần thiết.
* Trường hợp sử dụng:
  + Thích hợp trong môi trường đa luồng, khi nhiều luồng truy cập cùng một danh sách và yêu cầu đảm bảo tính toàn vẹn dữ liệu.

2.4 Stack

* Stack là một lớp con của Vector, triển khai cấu trúc dữ liệu ngăn xếp (stack) với nguyên tắc **LIFO** (Last In, First Out).
* Ưu điểm :
  + Stack thường được sử dụng để quản lý bộ nhớ trong các ngôn ngữ lập trình, đặc biệt khi làm việc với các hàm đệ quy.
  + Các ứng dụng cần xử lý ngược dòng như phân tích ngôn ngữ (parser), trình duyệt (quay lại trang trước), hay kiểm tra dấu ngoặc hợp lệ đều được triển khai dễ dàng với stack.
* Nhược điểm :
  + Stack không hỗ trợ truy cập ngẫu nhiên như một danh sách hay mảng. Nếu bạn cần lấy một phần tử ở giữa, bạn phải sử dụng các thao tác lấy phần tử từ đầu đến vị trí đó, gây bất tiện và kém hiệu quả.
  + Chỉ thích hợp các thao tác liên quan đến LIFO , nếu dùng stack ngoài cấu trúc này sẽ gây giảm hiệu năng và có thể lãng phí bộ nhớ trong 1 số trường hợp.
* Trường hợp sử dụng:
  + Thích hợp khi bạn cần thực hiện thao tác theo kiểu LIFO như xử lý undo, redo, hoặc quản lý trình tự quay lại trong trình duyệt web.