1. Các đặc điểm **Set Interface**

* Không cho phép lưu trũ giá trị trùng lặp tức là mỗi phần tử lưu trong set là duy nhất .
* Không lưu trữ thoe thứ tự thaêm vào của phần tử và chũng không có cơ chế đánh index .
* Hiệu suất cao cho cá theo tác tìm kiếm hay thêm xoá phần tử trong set.
* Set cũng được cấp phát đọng bộ nhớ vầ có thể lưu giá trị null nhưng chỉ luuw duy nhất 1 giá trị null.

1. Các class triển khai từ Set Interface và trường hợp sử dụng :

2.1 HashSet:

* Được sử dụng để lưu trữ các phần tử duy nhất . Nó dựa trên HashMap để quản lý dữ liệu và hash code để lưu trữ các phần tử.
* Ưu điểm :
  + Tốc độ tìm kiếm phần tử trong mảng nhanh do lưu trữ bằng hashcode .
  + Giảm bộ nhớ do không cần lưu trữ thứ tụe hoặc liên kết các phần tử từ đó giúp việc thêm xoá nhanh hơn.
* Nhược điểm
  + Có thể xung đột hashcode khi lưu trữ .
  + không an toàn trong môi trường đa lường
* Trường hợp sử dụng : **HashSet** rất phù hợp cho các trường hợp cần lưu trữ các phần tử duy nhất và yêu cầu hiệu suất cao khi thêm/tìm kiếm .

2.2 LinkedHashSet:

* Tương tự như hashSet tuy nhiên có cơ chế duy trì thứ tự của phần tử khhi thêm vào trong mảng .
* Ưu điểm :
  + Ưu điểm lớn nhất của LinkedHashSet là nó duy trì thứ tự các phần tử theo thứ tự chúng được thêm vào. Điều này khác với HashSet, vốn không bảo đảm thứ tự phần tử.
* Nhược điểm
  + Tăng kích thước bộ nhớ .
  + Chậm hơn hashSet do duy trì thứ tụ.
* Trường hợp sử dụng : Khi cần một collection không chứa các phần tử trùng lặp và bạn cần duy trì thứ tự các phần tử được thêm vào.

2.3 TreeSet:

* Ưu điểm:
  + Các phần tử trong TreeSet luôn được sắp xếp theo thứ tự hoặc theo một comparator tùy chỉnh được cung cấp khi khởi tạo.
  + TreeSet không cho phép các phần tử trùng lặp. Điều này đảm bảo rằng mỗi phần tử trong TreeSet là duy nhất.
  + TreeSet dựa trên cấu trúc cân bằng nhị phân. Điều này làm cho các thao tác như tìm kiếm, thêm và xóa phần tử khá hiệu quả cho các tập dữ liệu lớn.
  + TreeSet hỗ trợ các phương thức đặc biệt như first(), last(), higher(), lower() để lấy phần tử lớn hơn, nhỏ hơn, phần tử đầu tiên, phần tử cuối cùng, giúp dễ dàng điều hướng trong tập hợp.
  + TreeSet cho phép lấy các tập hợp con từ một tập hợp lớn thông qua các phương thức như subSet(), headSet(), và tailSet().

### Nhược điểm:

* + Mặc dù TreeSet có thời gian truy xuất tốt hơn nhiều so với các cấu trúc dữ liệu khác như LinkedList hay ArrayList, nhưng nó vẫn chậm hơn HashSet..
  + TreeSet không cho phép phần tử null.
  + Do phải duy trì cấu trúc cây nhị phân cân bằng, TreeSet yêu cầu nhiều bộ nhớ hơn so với HashSet hoặc LinkedHashSet, đặc biệt là khi làm việc với các tập dữ liệu lớn.
  + Để có thể sắp xếp phần tử, các đối tượng trong TreeSet phải triển khai giao diện Comparable hoặc phải cung cấp một Comparator khi khởi tạo TreeSet.
* Trường hợp sử dụng :
  + Khi cần lưu trữ các phần tử không trùng lặp và duy trì thứ tự tự nhiên hoặc tùy chỉnh.

2.4 EnumSet

* Được sử dụng để quản lý các tập hợp (set) của các kiểu enum.
* Ưu điểm của EnumSet:
  + EnumSet được triển khai rất tối ưu về bộ nhớ và thời gian, đặc biệt là so với các tập hợp khác như HashSet. Vì EnumSet sử dụng các bit để đại diện cho các phần tử, các phép toán như thêm, xóa, và kiểm tra phần tử rất nhanh.
  + Tiết kiệm bộ nhớ do lưu trữ dựa vào bit.
  + EnumSet lưu trữ các phần tử theo thứ tự xuất hiện của chúng trong kiểu enum, giúp việc duyệt qua các phần tử của tập hợp theo thứ tự tự nhiên dễ dàng.
  + EnumSet cung cấp nhiều phương thức hữu ích để tạo và thao tác với tập hợp như allOf(), noneOf(), range(), và complementOf(). Những phương thức này giúp dễ dàng làm việc với tập hợp các hằng số enum.

### Nhược điểm của EnumSet:

* + Điểm yếu lớn nhất của EnumSet là chỉ làm việc với các kiểu enum. Điều này có nghĩa là bạn không thể sử dụng EnumSet cho các kiểu dữ liệu khác.
  + EnumSet không phải là thread-safe, vì vậy nếu bạn cần sử dụng nó trong môi trường đa luồng.
  + EnumSet không cho phép chứa giá trị null.
  + Kích thước của EnumSet bị giới hạn bởi số lượng các hằng số của kiểu enum mà nó biểu diễn.
* Trường hợp sử dụng :Khi cần làm việc với các tập hợp của enum vì hiệu suất cao và tính tối ưu về bộ nhớ.