**Đặc điểm của Set Interface trong Java:**

Không chứa các phần tử trùng lặp: Nếu thêm phần tử đã tồn tại, nó sẽ bị ghi đè.

Không bảo đảm thứ tự: Một số triển khai của Set (như HashSet) không bảo đảm thứ tự của phần tử.

Cho phép phần tử null: Nhưng không phải tất cả các lớp triển khai đều hỗ trợ nhiều phần tử null.

Các phương thức cơ bản:

add: Thêm phần tử vào tập hợp.

remove: Xóa phần tử khỏi tập hợp.

contains: Kiểm tra sự tồn tại của phần tử.

size: Trả về số lượng phần tử.

**Các class triển khai từ Set Interface:**

HashSet

LinkedHashSet.

TreeSet

**HashSet:**

Cấu trúc: Dựa trên Hash Table.

Đặc điểm nổi bật:

* Không duy trì thứ tự chèn.
* Hiệu năng nhanh nhất cho các thao tác thêm (add), xóa (remove), và kiểm tra tồn tại (contains) vì các phần tử được phân phối vào các bucket theo mã băm (hash code).
* Không bảo đảm thứ tự khi duyệt phần tử.
* Cho phép chứa 1 phần tử null.

Khi nào nên sử dụng HashSet?

* Khi chỉ cần tập hợp các phần tử duy nhất (không trùng lặp) và không quan tâm đến thứ tự.
* Phù hợp trong các tình huống như:
* Lọc dữ liệu trùng lặp (ví dụ: tìm các từ duy nhất trong một văn bản).
* Kiểm tra nhanh xem một phần tử đã tồn tại hay chưa (do thao tác tìm kiếm nhanh).

**LinkedHashSet:**

Cấu trúc: Kết hợp Hash Table với Doubly Linked List (danh sách liên kết kép).

Đặc điểm nổi bật:

* Duy trì thứ tự chèn: Các phần tử được duyệt đúng theo thứ tự đã thêm vào.
* Hiệu năng kém hơn so với HashSet do phải quản lý thêm danh sách liên kết.
* Phù hợp khi cần loại bỏ trùng lặp và bảo toàn thứ tự chèn.

Khi nào nên sử dụng LinkedHashSet?

* Loại bỏ phần tử trùng lặp nhưng vẫn giữ thứ tự chèn.
* Ví dụ: Khi lưu trữ các thao tác người dùng hoặc dữ liệu cần xuất ra theo thứ tự đã thêm.

**TreeSet:**

Cấu trúc: Dựa trên Red-Black Tree (Cây nhị phân tự cân bằng).

Đặc điểm nổi bật:

* Duy trì thứ tự tự nhiên của phần tử (ví dụ: sắp xếp tăng dần đối với số hoặc sắp xếp chữ cái với chuỗi).
* Có thể tùy chỉnh thứ tự bằng cách truyền vào Comparator.
* Hiệu năng thấp hơn so với HashSet và LinkedHashSet do phải cân bằng cây sau mỗi thao tác thêm/xóa.
* Không cho phép phần tử null (vì không thể so sánh null với các phần tử khác).

Khi nào nên sử dụng TreeSet?

* Duy trì thứ tự tự nhiên hoặc sắp xếp tùy chỉnh các phần tử.
* Tìm kiếm phần tử lớn nhất/nhỏ nhất hoặc tìm kiếm theo phạm vi một cách nhanh chóng.

Ví dụ ứng dụng:

* Lưu trữ danh sách số hoặc chuỗi đã được sắp xếp.
* Tìm kiếm và lấy ra phần tử lớn nhất/nhỏ nhất trong tập hợp.

**Phân biệt trường hợp sử dụng từng class:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Thứ tự phần tử | Hiệu năng | Cho phép phần tử null? |
| HashSet | Không bảo đảm thứ tự | Nhanh nhất | 1 phần tử null |
| LinkedHashSet | Duy trì thứ tự chèn | Nhanh, nhưng chậm hơn HashSet | 1 phần tử null |
| TreeSet | Sắp xếp theo thứ tự tăng dần hoặc tùy chỉnh | Chậm nhất (do cân bằng cây) | Không hỗ trợ null |