Map Interface

**1. Đặc điểm của Map Interface:**

* Map là một interface trong Java, đại diện cho một cấu trúc dữ liệu lưu trữ các cặp khóa-giá trị (key-value), trong đó mỗi khóa phải là duy nhất.
* Mỗi khóa chỉ ánh xạ tới một giá trị duy nhất, nhưng một giá trị có thể được ánh xạ tới bởi nhiều khóa khác nhau.
* Map không kế thừa từ Collection mà là một phần riêng của Java Collections Framework.

**Các phương thức phổ biến:**

* put(K key, V value): Thêm hoặc thay thế giá trị gắn với một khóa.
* get(Object key): Trả về giá trị tương ứng với khóa hoặc null nếu khóa không tồn tại.
* remove(Object key): Xóa khóa và giá trị tương ứng khỏi map.
* containsKey(Object key): Kiểm tra xem khóa có tồn tại trong map hay không.
* containsValue(Object value): Kiểm tra xem giá trị có tồn tại trong map hay không.
* keySet(): Trả về tập hợp các khóa trong map.
* values(): Trả về tập hợp các giá trị trong map.
* entrySet(): Trả về tập hợp các cặp khóa-giá trị.

**2. Các lớp triển khai từ Map Interface:**

**a. HashMap:**

* **Đặc điểm**:
  + Sử dụng cơ chế băm (hashing) để lưu trữ các cặp khóa-giá trị.
  + Không đảm bảo thứ tự của các khóa và giá trị.
  + Hiệu suất cao, đặc biệt là cho các thao tác như tìm kiếm, thêm và xóa (trung bình là O(1)).
  + Cho phép một khóa có giá trị là null và nhiều giá trị có thể là null.
* **Trường hợp sử dụng**:
  + Sử dụng khi cần lưu trữ các cặp khóa-giá trị mà không quan tâm đến thứ tự.
  + Ví dụ: Lưu trữ thông tin cấu hình ứng dụng, trong đó khóa là tên cấu hình và giá trị là các giá trị cấu hình.

**b. LinkedHashMap:**

* **Đặc điểm**:
  + Triển khai dựa trên HashMap nhưng duy trì thứ tự chèn của các cặp khóa-giá trị.
  + Hiệu suất chậm hơn HashMap một chút do phải duy trì thứ tự chèn.
* **Trường hợp sử dụng**:
  + Sử dụng khi cần lưu trữ các cặp khóa-giá trị và cần duy trì thứ tự chèn.
  + Ví dụ: Lưu trữ lịch sử duyệt web hoặc danh sách người dùng gần đây theo thứ tự thời gian.

**c. TreeMap:**

* **Đặc điểm**:
  + Triển khai dựa trên NavigableMap, sắp xếp các khóa theo thứ tự tự nhiên hoặc theo một comparator tùy chỉnh.
  + Việc tìm kiếm, thêm và xóa phần tử có độ phức tạp O(log n) vì sử dụng cấu trúc cây nhị phân cân bằng (Red-Black Tree).
* **Trường hợp sử dụng**:
  + Sử dụng khi cần lưu trữ các cặp khóa-giá trị và sắp xếp chúng theo thứ tự khóa.
  + Ví dụ: Lưu trữ danh sách nhân viên theo mã số nhân viên theo thứ tự tăng dần hoặc giảm dần.

**d. Hashtable:**

* **Đặc điểm**:
  + Là một phiên bản cũ của HashMap, nhưng tất cả các phương thức của nó đều được đồng bộ hóa (synchronized).
  + Không cho phép khóa hoặc giá trị null.
  + Hiệu suất thấp hơn so với HashMap vì cần đồng bộ hóa.
* **Trường hợp sử dụng**:
  + Sử dụng khi cần một Map thread-safe trong môi trường đa luồng.
  + Ví dụ: Lưu trữ thông tin trạng thái của người dùng trong ứng dụng đa luồng.

**e. EnumMap:**

* **Đặc điểm**:
  + Sử dụng cho các khóa là kiểu enum (kiểu liệt kê).
  + Hiệu suất cao và tiết kiệm bộ nhớ vì sử dụng các enum làm khóa.
  + Duy trì thứ tự của các khóa theo thứ tự khai báo trong enum.
* **Trường hợp sử dụng**:
  + Sử dụng khi cần lưu trữ các cặp khóa-giá trị mà khóa là kiểu enum.
  + Ví dụ: Lưu trữ các cài đặt theo các chế độ khác nhau (mỗi chế độ là một giá trị enum).