Session 14:

SET VÀ MAP TRONG JAVA

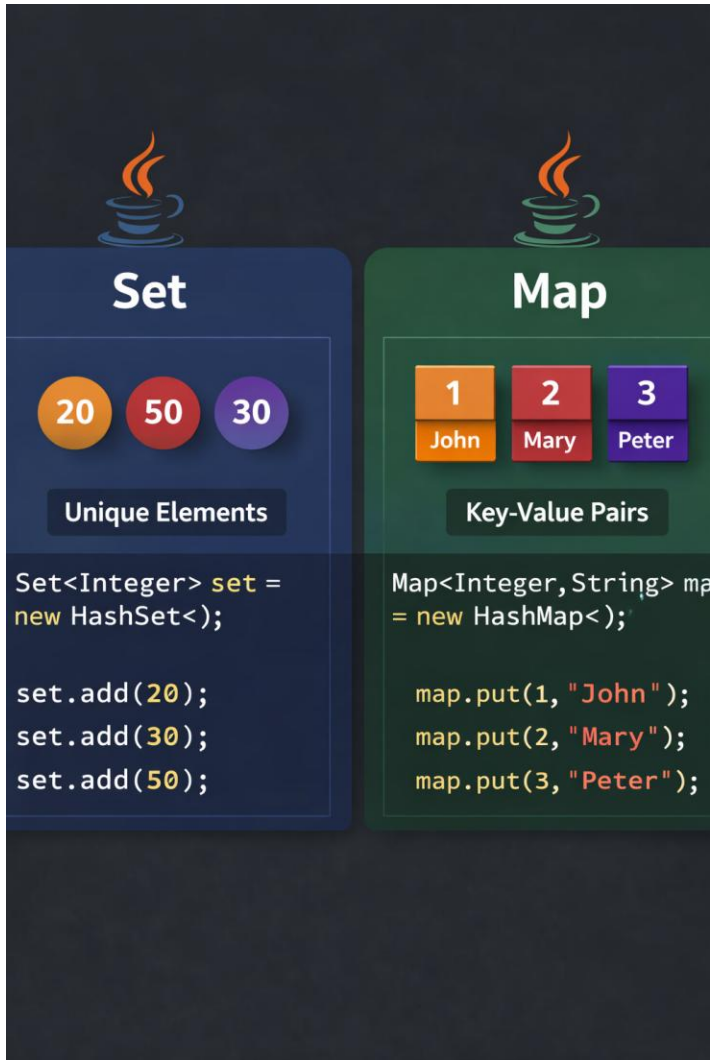
Module – Java Fudamental

Phiên bản: 3.0

NỘI DUNG

1. Giới thiệu Set và các lớp triển khai thông dụng của Set
2. Các phương thức thông dụng để làm việc với Set
3. Giới thiệu Map và các lớp triển khai thông dụng của Map
4. Các phương thức thông dụng để làm việc với Map
5. Comparable và Comparator trong sắp xếp collection

Set và Map trong Java Collection Framework



Set Interface - Quản Lý Dữ Liệu Duy Nhất

Set là một interface quan trọng trong Java Collection Framework, được thiết kế đặc biệt để lưu trữ các phần tử duy nhất. Khác với List cho phép các phần tử trùng lặp, Set đảm bảo rằng mỗi giá trị chỉ xuất hiện một lần duy nhất trong tập hợp.

Đây là lựa chọn lý tưởng khi bạn cần đảm bảo tính duy nhất của dữ liệu, như danh sách email người dùng, ID sản phẩm, hoặc các mã định danh khác trong hệ thống.

Đặc Điểm Nổi Bật Của Set

Không Có Index

Set không hỗ trợ truy cập phần tử theo chỉ số như List. Bạn không thể gọi set.get(0) để lấy phần tử đầu tiên.

Không Đảm Bảo Thứ Tự

Hầu hết các implementation của Set không duy trì thứ tự các phần tử được thêm vào (trừ LinkedHashSet).

Dữ Liệu Duy Nhất

Mỗi phần tử chỉ xuất hiện một lần. Nếu thêm phần tử trùng, Set sẽ tự động bỏ qua.

Ba Lớp Triển Khai Chính Của Set

HashSet

Implementation phổ biến nhất của Set

HashSet là lựa chọn mặc định khi làm việc với Set. Nó sử dụng bảng băm (hash table) để lưu trữ dữ liệu, mang lại hiệu suất truy cập cực nhanh với độ phức tạp O(1) cho các thao tác cơ bản.

Ngoài ra còn có LinkedHashSet giữ thứ tự thêm vào, và TreeSet tự động sắp xếp các phần tử.

So Sánh Các Implementation Của Set

HashSet

Hiệu suất cao nhất với O(1). Không đảm bảo thứ tự. Phù hợp khi chỉ cần kiểm tra sự tồn tại nhanh chóng.

LinkedHashSet

Duy trì thứ tự thêm vào. Hiệu suất tốt O(1) nhưng tốn bộ nhớ hơn HashSet một chút.

TreeSet

Tự động sắp xếp các phần tử. Hiệu suất O(log n). Yêu cầu phần tử phải comparable.

Ứng Dụng Thực Tế Của Set

01

Lọc Dữ Liệu Trùng Lặp

Loại bỏ các bản ghi trùng trong danh sách import từ file Excel hoặc CSV

02

Quản Lý Email Người Dùng

Đảm bảo mỗi email chỉ đăng ký một lần trong hệ thống

03

Kiểm Tra Tồn Tại Nhanh

Tra cứu xem một giá trị có trong danh sách hay không với tốc độ cực nhanh

Khai Báo và Sử Dụng Set

Set được thiết kế để quản lý dữ liệu không trùng lặp một cách hiệu quả. Trong thực tế, chúng ta thường khai báo Set theo interface để linh hoạt trong việc thay đổi implementation.

```
Set<String> emails = new HashSet<>();
Set<Integer> productIds = new LinkedHashSet<>();
Set<String> sortedNames = new TreeSet<>();
```

Cách khai báo này cho phép bạn dễ dàng chuyển đổi giữa HashSet, LinkedHashSet, hoặc TreeSet mà không cần thay đổi phần code sử dụng.

Các Phương Thức Cơ Bản Của Set

add()

Thêm phần tử vào Set. Trả về true nếu thêm thành công, false nếu phần tử đã tồn tại.

remove()

Xóa phần tử khỏi Set. Trả về true nếu phần tử tồn tại và được xóa.

```
Set<String> set = new HashSet<>();
set.add("Java"); // true
set.add("Python"); // true
set.add("Java"); // false (đã tồn tại)

set.remove("Python"); // true
set.clear(); // Set rỗng
```

clear()

Xóa tất cả phần tử trong Set, làm rỗng hoàn toàn tập hợp.

Kiểm Tra Dữ Liệu Trong Set

contains(element)

Kiểm tra xem một phần tử có tồn tại trong Set hay không. Trả về boolean. Với HashSet, thao tác này có độ phức tạp O(1).

size()

Trả về số lượng phần tử hiện có trong Set. Hữu ích để kiểm tra kích thước tập hợp trước khi xử lý.

isEmpty()

Kiểm tra Set có rỗng hay không. Trả về true nếu không có phần tử nào, thường dùng để validate trước khi duyệt.

Hai Cách Duyệt Set Phổ Biến

Sử Dụng For-Each Loop

```
Set<String> languages = new HashSet<>();
languages.add("Java");
languages.add("Python");
languages.add("JavaScript");

for (String lang : languages) {
    System.out.println(lang);
}
```

Cách này đơn giản, dễ đọc và được khuyến nghị cho hầu hết trường hợp.

Sử Dụng Iterator

```
Iterator<String> it = languages.iterator();
while (it.hasNext()) {
    String lang = it.next();
    System.out.println(lang);

    // Có thể xóa phần tử an toàn
    if (lang.equals("Python")) {
        it.remove();
    }
}
```

Iterator cho phép xóa phần tử trong khi duyệt mà không gây lỗi ConcurrentModificationException.

Phép Toán Tập Hợp Với Set

Set hỗ trợ các phép toán tập hợp trong toán học, giúp xử lý dữ liệu một cách linh hoạt và mạnh mẽ.

addAll() - Hợp

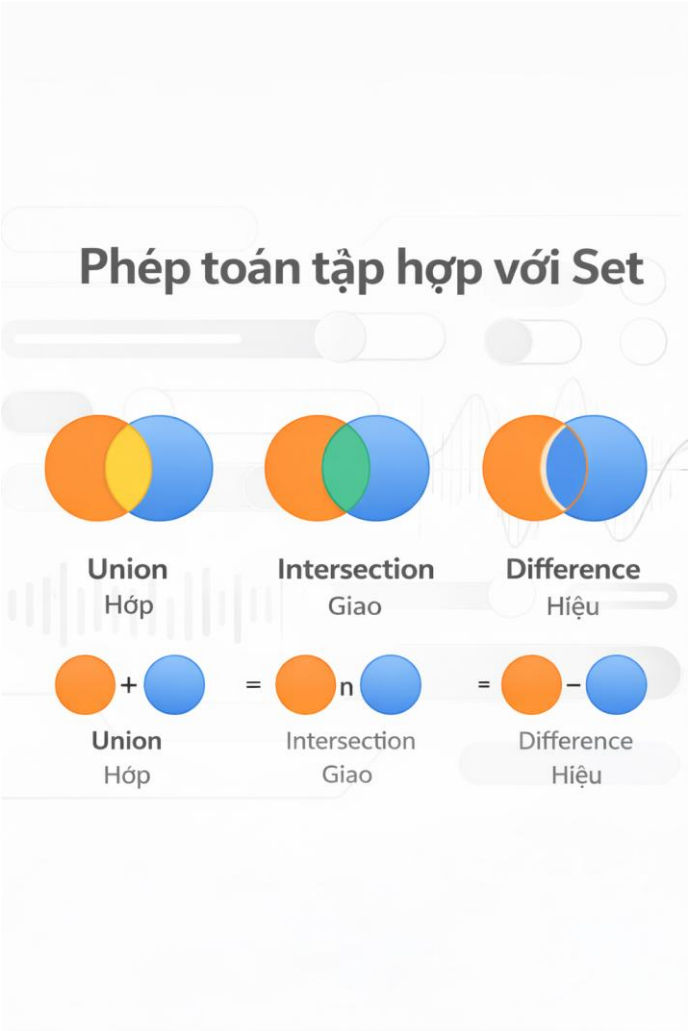
Kết hợp tất cả phần tử từ tập hợp khác vào

retainAll() - Giao

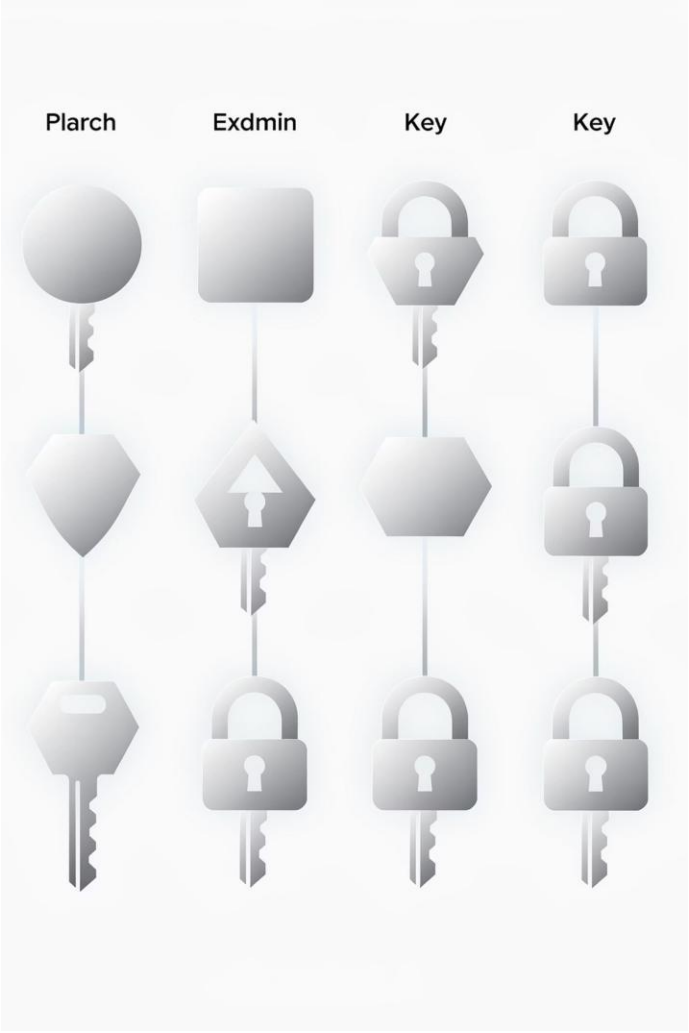
Chỉ giữ lại phần tử xuất hiện ở cả hai tập

removeAll() - Hiệu

Loại bỏ phần tử có trong tập hợp kia



Map Interface - Lưu Trữ Cặp Key-Value



Map là cấu trúc dữ liệu lưu trữ dữ liệu dưới dạng cặp key-value, trong đó mỗi key là duy nhất và ánh xạ đến một giá trị cụ thể. Đây là giải pháp hoàn hảo khi bạn cần tra cứu dữ liệu nhanh chóng thông qua một khóa định danh.

Map không phải là Collection, nhưng là một phần quan trọng của Java Collection Framework, được sử dụng rộng rãi trong các ứng dụng thực tế.

Ba Lớp Thực Thi Của Map

HashMap

Implementation phổ biến nhất, sử dụng hash table. Không đảm bảo thứ tự, nhưng hiệu suất cực cao O(1).

TreeMap

Tự động sắp xếp theo key. Hiệu suất O(log n). Phù hợp khi cần duyệt theo thứ tự.

LinkedHashMap

Duy trì thứ tự chèn vào. Hiệu suất gần bằng HashMap nhưng tốn bộ nhớ hơn.

HashMap - Lựa Chọn Mặc Định

HashMap là lựa chọn phổ biến nhất khi làm việc với Map nhờ hiệu suất vượt trội. Nó sử dụng cơ chế hash để lưu trữ và truy xuất dữ liệu với độ phức tạp O(1) cho các thao tác cơ bản như put, get, remove.

Tuy nhiên, HashMap không đảm bảo thứ tự của các phần tử. Mỗi lần duyệt qua HashMap, thứ tự các entry có thể khác nhau. Nếu thứ tự quan trọng, hãy cân nhắc LinkedHashMap hoặc TreeMap.

O(1)

Hiệu suất truy cập

Thời gian hằng số

TreeMap - Sắp Xếp Tự Động

TreeMap sử dụng cấu trúc Red-Black Tree để lưu trữ dữ liệu, đảm bảo các entry luôn được sắp xếp theo key. Điều này rất hữu ích khi bạn cần duyệt Map theo thứ tự tăng dần hoặc giảm dần.

Độ phức tạp O(log n) cho các thao tác cơ bản. Chậm hơn HashMap nhưng đổi lại có thứ tự.

Yêu cầu key phải comparable hoặc cung cấp Comparator để xác định thứ tự sắp xếp.

LinkedHashMap

Tại Sao Chọn LinkedHashMap?

Khi bạn cần cả hiệu suất và thứ tự

LinkedHashMap kết hợp ưu điểm của HashMap (hiệu suất cao) và khả năng duy trì thứ tự chèn vào. Nó sử dụng doubly-linked list để theo dõi thứ tự, đảm bảo khi duyệt qua Map, bạn sẽ nhận được các entry theo đúng thứ tự đã thêm vào.

Chi phí là tốn bộ nhớ hơn HashMap một chút do phải duy trì cấu trúc linked list. Tuy nhiên, hiệu suất vẫn rất tốt với O(1) cho các thao tác cơ bản.

Quản Lý Dữ Liệu Với Map

Map cung cấp các phương thức mạnh mẽ để quản lý cặp key-value, cho phép tra cứu và cập nhật dữ liệu cực kỳ nhanh chóng dựa trên key.

```
Map<String, Integer> scores = new HashMap<>();
Map<Integer, String> users = new TreeMap<>();
Map<String, Product> catalog = new LinkedHashMap<>();
```

Khai báo theo interface Map giúp code linh hoạt, dễ bảo trì và có thể thay đổi implementation khi cần.

Phương Thức Cơ Bản Của Map

put(key, value)

Thêm hoặc cập nhật cặp key-value. Nếu key đã tồn tại, value cũ sẽ bị thay thế và được trả về.

get(key)

Lấy giá trị tương ứng với key. Trả về null nếu key không tồn tại trong Map.

remove(key)

Xóa cặp key-value khỏi Map. Trả về giá trị đã xóa hoặc null nếu key không tồn tại.

Kiểm Tra Dữ Liệu Trong Map

containsKey(key)

Kiểm tra xem Map có chứa key cụ thể hay không. Trả về boolean.

```
if (map.containsKey("Java")) {
    System.out.println("Tìm thấy!");
}
```

containsValue(value)

Kiểm tra xem Map có chứa giá trị cụ thể hay không. Chậm hơn containsKey do phải duyệt toàn bộ.

```
if (map.containsValue(100)) {
    System.out.println("Có điểm 100!");
}
```

Lưu ý: containsKey() có độ phức tạp O(1) với HashMap, trong khi containsValue() là O(n) vì phải duyệt tất cả các giá trị.

Lấy Tập Dữ Liệu Từ Map

01

keySet()

Trả về Set chứa tất cả các key trong Map. Hữu ích khi chỉ cần duyệt qua các key.

02

values()

Trả về Collection chứa tất cả các value. Sử dụng khi chỉ quan tâm đến giá trị, không cần key.

03

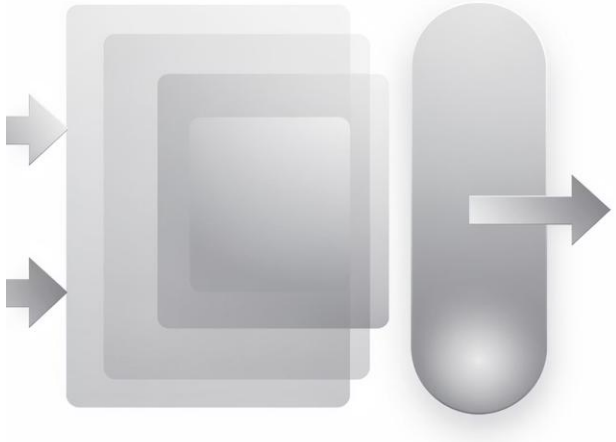
entrySet()

Trả về Set các entry (cặp key-value). Đây là cách hiệu quả nhất để duyệt toàn bộ Map.

```
for (Map.Entry<String, Integer> entry : map.entrySet()) {
 System.out.println(entry.getKey() + ": " + entry.getValue());
}
```

Phương Thức Mở Rộng Từ Java 8

Java 8 giới thiệu nhiều phương thức mới cho Map, giúp xử lý dữ liệu ngắn gọn và an toàn hơn.



putIfAbsent()	computeIfAbsent()	merge()
Chỉ thêm nếu key chưa tồn tại, tránh ghi đè value cũ	Tính toán và thêm value nếu key chưa có	Kết hợp value mới với value cũ theo hàm cho trước

Comparable vs Comparator

Khi làm việc với Collection, đặc biệt là TreeSet và TreeMap, chúng ta thường cần sắp xếp dữ liệu theo một tiêu chí nhất định. Java cung cấp hai cách chính để định nghĩa logic sắp xếp: Comparable và Comparator.

Cả hai đều là interface, nhưng được sử dụng trong các tình huống khác nhau tùy thuộc vào yêu cầu của bạn.

Comparable - Thứ Tự Tự Nhiên

Cài Đặt Trong Class

Implement trực tiếp vào class cần sắp xếp

Override compareTo()

Định nghĩa cách so sánh đối tượng hiện tại với đối tượng khác

Một Tiêu Chí

Chỉ có một cách sắp xếp "tự nhiên"

```
public class Student
    implements Comparable<Student> {

    private String name;
    private int age;

    @Override
    public int compareTo(Student other) {
        return this.name.compareTo(other.name);
    }
}
```

Ví dụ này sắp xếp sinh viên theo tên một cách tự nhiên.

Comparator - Sắp Xếp Linh Hoạt

Comparator cho phép định nghĩa nhiều cách sắp xếp khác nhau cho cùng một class mà không cần sửa đổi code của class đó. Đây là giải pháp linh hoạt hơn Comparable.

Triển Khai Truyền Thống

```
Comparator<Student> ageComparator =
    new Comparator<Student>() {

    @Override
    public int compare(Student s1, Student s2) {
        return Integer.compare(s1.getAge(), s2.getAge());
    }
};
```

Với Lambda (Java 8+)

```
Comparator<Student> ageComparator =
 (s1, s2) -> Integer.compare(s1.getAge(), s2.getAge());

// Hoặc ngắn gọn hơn
Comparator<Student> ageComparator =
 Comparator.comparingInt(Student::getAge);
```

TỔNG KẾT

Chọn Công Cụ Phù Hợp

Set - Dữ Liệu Duy Nhất

Sử dụng khi cần đảm bảo không trùng lặp. HashSet cho hiệu suất, TreeSet cho thứ tự, LinkedHashSet cho cả hai.

Map - Tra Cứu Nhanh

Lựa chọn tốt nhất cho tra cứu theo key. HashMap cho tốc độ, TreeMap cho sắp xếp, LinkedHashMap cho thứ tự chèn.

So Sánh và Sắp Xếp

Comparable cho thứ tự tự nhiên, Comparator cho nhiều tiêu chí. Java 8 lambda giúp code ngắn gọn và dễ đọc hơn.

Hiểu rõ đặc điểm và ứng dụng của từng cấu trúc dữ liệu sẽ giúp bạn viết code Java hiệu quả và chuyên nghiệp hơn!

RIKEI Academy®
where the dream begins

KẾT THÚC

HỌC VIỆN ĐÀO TẠO LẬP TRÌNH CHẤT LƯỢNG NHẬT BẢN