# 🎓 HANDOUT BUỔI 13: JAVA COLLECTION FRAMEWORK

## 🔶 MỤC TIÊU BUỔI HỌC

- Phân biệt được Collection và Collections.
- Hiểu hệ thống cấp bậc Collection trong Java.
- Nắm được các interface chính của Collection (Collection, List, Set, Map).
- Hiểu và sử dụng được Iterator.
- Làm việc với List và ArrayList.

## 1. Collection vs. Collections

### 📌 1.1. Collection là gì?

- Collection là một **interface gốc** trong Java, nằm trong gói java.util, đại diện cho **một tập hợp các đối tượng (elements)**.
- Đây là **cha của các interface** như List, Set, và Queue.
- Các lớp như ArrayList, HashSet, LinkedList, PriorityQueue **đều gián tiếp kế thừa từ Collection**.

✅ Collection không thể tạo đối tượng trực tiếp, mà được triển khai (implemented) bởi các lớp cụ thể như ArrayList, HashSet, v.v.

🔧 **Một số phương thức quan trọng của interface Collection**

| Phương thức | Mô tả |
|---|---|
| add(E e) | Thêm phần tử vào collection |
| remove(Object o) | Xóa phần tử khỏi collection |
| clear() | Xóa toàn bộ phần tử |
| contains(Object o) | Kiểm tra phần tử có tồn tại |
| size() | Trả về số lượng phần tử |
| isEmpty() | Kiểm tra collection có rỗng không |
| iterator() | Trả về một iterator để duyệt collection |

📚 **Ví dụ minh họa - Sử dụng Collection thông qua ArrayList**

```
import java.util.*;

public class Example1 {
    public static void main(String[] args) {
        Collection<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");

        System.out.println("Fruits: " + fruits);
        System.out.println("Size: " + fruits.size());
        System.out.println("Contains Banana? " + fruits.contains("Banana"));
    }
}
```

✅ **Kết quả:**

```
Fruits: [Apple, Banana, Mango]
Size: 3
Contains Banana? true
```

## 📌 1.2. Collections là gì?

- Collections là **một lớp tiện ích (utility class)** nằm trong java.util.
- Lớp này chứa **các phương thức tĩnh (static methods)** dùng để **thao tác với các đối tượng Collection** như: sắp xếp, đảo ngược, tìm min/max, trộn ngẫu nhiên,…

⚠️ Đừng nhầm giữa Collection (interface) và Collections (class tiện ích).

🔧 **Một số phương thức hữu ích trong Collections**

| Phương thức | Mô tả |
|---|---|
| sort(List<T> list) | Sắp xếp list theo thứ tự tăng dần |
| reverse(List<T> list) | Đảo ngược thứ tự list |
| shuffle(List<T> list) | Trộn ngẫu nhiên phần tử |
| max(Collection<T> coll) | Trả về phần tử lớn nhất |
| min(Collection<T> coll) | Trả về phần tử nhỏ nhất |
| frequency(Collection<?> c, Object o) | Đếm số lần xuất hiện của phần tử |

📚 **Ví dụ minh họa - Dùng lớp Collections**

```
import java.util.*;

public class Example2 {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<>(Arrays.asList(5, 3, 8, 1, 4));

        System.out.println("Original: " + numbers);

        Collections.sort(numbers);
        System.out.println("Sorted: " + numbers);

        Collections.reverse(numbers);
        System.out.println("Reversed: " + numbers);

        Collections.shuffle(numbers);
        System.out.println("Shuffled: " + numbers);

        int max = Collections.max(numbers);
        int min = Collections.min(numbers);
        System.out.println("Max: " + max + ", Min: " + min);
    }
}
```

✅ **Kết quả mẫu:**

```
Original: [5, 3, 8, 1, 4]
Sorted: [1, 3, 4, 5, 8]
Reversed: [8, 5, 4, 3, 1]
Shuffled: [4, 1, 5, 8, 3] (ngẫu nhiên)
Max: 8, Min: 1
```

🎯 **Tóm tắt phân biệt**

| Tiêu chí | Collection (interface) | Collections (class) |
|---|---|---|
| Loại | Interface | Class tiện ích (utility class) |

| Tiêu chí | Collection (interface) | Collections (class) |
|---|---|---|
| Chức năng | Xác định cấu trúc dữ liệu | Cung cấp các phương thức xử lý collection |
| Có thể tạo đối tượng? | ❌ Không (chỉ được triển khai) | ❌ Không (chỉ có static methods) |
| Ví dụ dùng | List, Set, Queue | Collections.sort(list), shuffle(list) |

### ☑️ 1.3. Khi nào sử dụng Collection và khi nào dùng Collections?

| Tình huống | Dùng Collection | Dùng Collections |
|---|---|---|
| Khai báo & tổ chức cấu trúc dữ liệu | ✅ (List, Set, Queue…) | ❌ Không dùng để khai báo |
| Cần thêm/xóa/truy xuất phần tử | ✅ (add(), remove(), ...) | ❌ Không dùng trực tiếp cho việc này |
| Muốn **sắp xếp**, **đảo ngược**, **trộn ngẫu nhiên**, **tìm max/min** | ❌ Không hỗ trợ | ✅ Có nhiều phương thức tĩnh hỗ trợ |
| Dùng như kiểu dữ liệu linh hoạt trong hàm | ✅ (vd: void printList(Collection<?> c)) | ❌ Không dùng như kiểu dữ liệu |

### 🧠 1.4. Lý do tại sao Java tách biệt Collection và Collections

- Collection là khái niệm cốt lõi, mô tả **hành vi trừu tượng** của các tập hợp dữ liệu.
- Collections là lớp hỗ trợ thao tác **một cách tiện lợi** lên các cấu trúc đó — ví dụ: sắp xếp một List không cần viết lại hàm sort thủ công.

🧠 Java áp dụng nguyên lý **"chia để trị"**: interface tập trung vào thiết kế, còn lớp tiện ích lo xử lý cụ thể.

### 🔍 1.5. Một số lỗi thường gặp

| Tình huống gây nhầm lẫn | Giải thích |
|---|---|
| Viết new Collection() → ❌ lỗi | Collection là interface, không thể tạo object trực tiếp. |
| Gọi Collections.add(list, "A") → ❌ lỗi | Không có add() trong lớp Collections. Sử dụng list.add("A"). |
| Nhầm Collections là phần mở rộng của Collection | Không phải! Hai thực thể hoàn toàn độc lập. |

### 🖊️ 1.6. Bài tập thực hành gợi ý

📝 **Bài 1: Khai báo và sử dụng Collection**

Viết chương trình cho phép người dùng nhập vào danh sách môn học, lưu vào Collection<String>, sau đó in ra danh sách đó.

```
Collection<String> subjects = new ArrayList<>();
subjects.add("Math");
subjects.add("Physics");
subjects.add("Programming");
```

```
for (String s : subjects) {
    System.out.println(s);
}
```

📝 **Bài 2: Sử dụng Collections.sort() và shuffle()**

```
List<String> students = new ArrayList<>(Arrays.asList("Lan", "Hùng", "Tú", "An"));
Collections.sort(students);
System.out.println("Sắp xếp: " + students);

Collections.shuffle(students);
System.out.println("Trộn ngẫu nhiên: " + students);
```

### 💡 1.7. Gợi ý trình bày slide hoặc handout

- Trang đầu: tiêu đề lớn “Collection vs. Collections – Giống & Khác gì?”
- Trang tiếp theo: bảng so sánh.
- Một slide riêng cho từng phần ví dụ.
- Cuối cùng: bài tập thực hành + quiz mini để ôn tập.

## 2. Interface chính trong Collection

### ✅ 2.1. List<E> – Danh sách có thứ tự, cho phép trùng lặp

- List là interface mô tả **danh sách tuyến tính**, nơi các phần tử **được lưu giữ theo thứ tự thêm vào**.
- **Cho phép phần tử trùng lặp** (e.g. "A", "B", "A" là hợp lệ).
- Hỗ trợ **truy cập theo chỉ số (index)** như mảng.

🔧 **Lớp triển khai phổ biến:** ArrayList, LinkedList, Vector.

📚 **Ví dụ:**

```
List<String> names = new ArrayList<>();
names.add("Lan");
names.add("Hùng");
names.add("Lan"); // Trùng lặp được phép

System.out.println(names.get(0)); // In: Lan
System.out.println(names);        // [Lan, Hùng, Lan]
```

### ✅ 2.2. Set<E> – Tập hợp không trùng lặp, không đảm bảo thứ tự

- Set là interface cho **tập hợp các phần tử duy nhất**.
- **Không cho phép phần tử trùng lặp**.
- Không đảm bảo thứ tự phần tử (tuỳ từng lớp triển khai).

🔧 **Lớp triển khai phổ biến:**

- HashSet: không có thứ tự, tốc độ nhanh.
- TreeSet: có thứ tự tăng dần (sorted).
- LinkedHashSet: duy trì thứ tự thêm vào.

📚 **Ví dụ:**

```
Set<String> fruits = new HashSet<>();
fruits.add("Táo");
fruits.add("Cam");
```

```
fruits.add("Táo"); // Bị bỏ qua

System.out.println(fruits); // Có thể in: [Cam, Táo] (thứ tự không đảm bảo)
```

### ☑ 2.3. Map<K, V> – Ánh xạ khóa-giá trị (key-value)

- Map là interface dùng để **lưu trữ các cặp khóa-giá trị**.
- **Không cho phép trùng lặp khóa**, nhưng **giá trị có thể trùng**.
- Không kế thừa từ Collection.

🔧 **Lớp triển khai phổ biến:**

- HashMap: nhanh, không thứ tự.
- TreeMap: sắp xếp theo khóa.
- LinkedHashMap: giữ thứ tự thêm.

📚 **Ví dụ:**

```
Map<String, Integer> scores = new HashMap<>();
scores.put("Lan", 9);
scores.put("Tú", 8);
scores.put("Lan", 10); // Ghi đè giá trị của "Lan"

System.out.println(scores.get("Lan")); // In: 10
System.out.println(scores); // {Lan=10, Tú=8}
```

### 📊 2.4. Bảng so sánh tổng quan

| Interface | Cho phép trùng lặp? | Truy cập theo chỉ số? | Dữ liệu có thứ tự? | Dạng dữ liệu lưu |
|---|---|---|---|---|
| List | ✅ Có | ✅ Có | ✅ Có | Các phần tử (elements) |
| Set | ❌ Không | ❌ Không | ❌ Không* (trừ TreeSet, LinkedHashSet) | Các phần tử (duy nhất) |
| Map | ❌ (key) / ✅ (value) | ❌ Không | ❌ Không* | Các cặp key-value |

### 💻 2.5. Bài tập thực hành gợi ý

📝 **Bài 1: Dùng List để lưu danh sách sinh viên**

```
List<String> students = new ArrayList<>();
students.add("An");
students.add("Bình");
students.add("An"); // hợp lệ
System.out.println(students); // [An, Bình, An]
```

📝 **Bài 2: Dùng Set để lọc tên không trùng lặp**

```
Set<String> uniqueNames = new HashSet<>();
uniqueNames.add("Lan");
uniqueNames.add("Tú");
uniqueNames.add("Lan"); // bị loại
System.out.println(uniqueNames); // [Lan, Tú]
```

📝 **Bài 3: Dùng Map để quản lý điểm sinh viên**

```
Map<String, Double> grades = new HashMap<>();
grades.put("Nam", 8.5);
grades.put("Mai", 9.0);
grades.put("Nam", 7.5); // cập nhật điểm mới

System.out.println(grades); // {Nam=7.5, Mai=9.0}
```

### 🧠 2.6. Ghi nhớ nhanh

- 🔁 **List** = Danh sách có thứ tự, trùng lặp được.
- ✋ **Set** = Không trùng lặp, không quan tâm thứ tự.
- 🗂️ **Map** = Dùng để lưu thông tin theo dạng khóa – giá trị.

## 4. Iterator Interface

### ✅ 4.1. Mục đích

- Iterator là một **interface trong Java** cho phép **duyệt tuần tự (sequential traversal)** qua các phần tử trong một Collection (List, Set,…).
- Giúp thao tác duyệt, xóa phần tử một cách an toàn **khi đang lặp**.
- Được dùng rộng rãi trong cả **vòng lặp thủ công** và **vòng lặp for-each** (dùng ngầm bên trong).

### 🔗 4.2. Các phương thức quan trọng trong Iterator

| Phương thức | Chức năng |
|---|---|
| hasNext() | Trả về true nếu còn phần tử kế tiếp trong collection |
| next() | Lấy phần tử kế tiếp và di chuyển con trỏ nội bộ tới vị trí tiếp theo |
| remove() | Xóa phần tử cuối cùng được trả về bởi next() khỏi collection |

📝 Ghi nhớ: Phải **gọi next() trước khi remove()**, nếu không sẽ bị lỗi.

### 🛤️ 4.3. Ví dụ sử dụng Iterator

🧪 **Xóa các chuỗi có độ dài chẵn trong danh sách:**

```
List<String> list = new ArrayList<>(Arrays.asList("An", "Bình", "Tú", "Minh"));
Iterator<String> it = list.iterator();

while (it.hasNext()) {
    String value = it.next();
    if (value.length() % 2 == 0) {
        it.remove(); // Xóa phần tử có độ dài chẵn
    }
}

System.out.println(list); // In ra danh sách sau khi xóa
```

✅ **Kết quả mẫu:** [An, Tú] (nếu "Bình" và "Minh" bị xóa)

### 4.4. Vì sao không nên dùng for-each khi cần xóa phần tử?

```
for (String name : list) {
    if (name.length() % 2 == 0) {
        list.remove(name); // ❌ Sai! Gây ConcurrentModificationException
    }
}
```

❌ Không được sửa đổi (add/remove) collection trong vòng lặp for-each.
✅ Hãy dùng Iterator để tránh lỗi.

### 4.5. Cách Iterator hoạt động trong Java

Hình dung Iterator như **ngón tay trỏ** duyệt từng phần tử:

1. hasNext() kiểm tra còn phần tử nào không.
2. next() lấy phần tử hiện tại và di chuyển con trỏ.
3. remove() loại bỏ phần tử vừa duyệt.

### ⚠ 4.6. Lỗi phổ biến khi dùng Iterator

| Lỗi thường gặp | Mô tả | Cách khắc phục |
|---|---|---|
| Gọi remove() trước next() | IllegalStateException | Luôn gọi next() trước |
| Sử dụng vòng for-each để xóa | ConcurrentModificationException | Dùng Iterator thay thế |
| Gọi next() nhiều lần trong một vòng lặp | Bỏ qua phần tử | Gọi next() một lần, gán vào biến tạm |

### 4.7. Iterator vs for-each vs for thường

| Cách duyệt | Có thể xóa? | Truy cập chỉ số? | Hiệu suất trên List | Thường dùng với |
|---|---|---|---|---|
| Iterator | ✅ Có | ❌ Không | Tốt | Set, List |
| for-each | ❌ Không | ❌ Không | Tốt | Mọi Collection |
| for | ✅ Có (cẩn thận) | ✅ Có | Tốt với ArrayList | List |

### 4.8. Bài tập thực hành gợi ý

**Bài 1: Duyệt danh sách và in ra độ dài từng tên**

```
Iterator<String> it = names.iterator();
while (it.hasNext()) {
    String name = it.next();
    System.out.println(name + " có " + name.length() + " ký tự.");
}
```

**Bài 2: Xóa số chẵn khỏi danh sách List<Integer>**

```
List<Integer> numbers = new ArrayList<>(Arrays.asList(1, 4, 7, 8, 10));
Iterator<Integer> it = numbers.iterator();
while (it.hasNext()) {
    if (it.next() % 2 == 0) {
        it.remove();
```

```
    }
}
System.out.println(numbers); // In ra các số lẻ
```

### 📝 4.9. Ghi nhớ nhanh

- ✅ **Iterator giúp xóa phần tử khi đang duyệt an toàn**
- ✅ Nên dùng khi thao tác với Set, LinkedList
- ❌ Không dùng for-each để sửa/xóa phần tử

## 5. List và ArrayList

### ☑️ 5.1. List<E> – Interface mô tả danh sách có thứ tự

- Là một interface trong java.util, dùng để mô tả một **danh sách các phần tử theo thứ tự**.
- Cho phép **phần tử trùng lặp**.
- Cho phép truy cập phần tử theo **chỉ số (index)**.
- Các lớp triển khai List: ArrayList, LinkedList, Vector.

```
List<String> list = new ArrayList<>();
list.add("A");
list.add("B");
```

👉 Thực tế ta thường dùng List như kiểu khai báo, và khởi tạo bằng lớp cụ thể như ArrayList.

### ☑️ 5.2. ArrayList<E> – Lớp danh sách dựa trên mảng động

- ArrayList là một **lớp cụ thể** triển khai List, sử dụng **mảng động bên trong** để lưu trữ phần tử.
- Kích thước của ArrayList sẽ tự động tăng khi cần.
- Truy cập phần tử rất nhanh (O(1)) vì lưu trong mảng.

📦 **Import cần thiết:**

```
import java.util.ArrayList;
```

📚 **Ví dụ khởi tạo và sử dụng:**

```
ArrayList<String> names = new ArrayList<>();
names.add("An");
names.add("Bình");

System.out.println(names); // [An, Bình]
System.out.println("Phần tử đầu: " + names.get(0)); // An
```

### 🔧 5.3. Các thao tác cơ bản với ArrayList

| Phương thức | Mô tả |
|---|---|
| add(E e) | Thêm phần tử vào cuối danh sách |
| add(int index, E e) | Thêm phần tử tại vị trí chỉ định |
| remove(int index) | Xóa phần tử tại chỉ số index |
| get(int index) | Lấy phần tử tại vị trí index |
| set(int index, E e) | Gán giá trị mới tại vị trí index |

| Phương thức | Mô tả |
|---|---|
| contains(E e) | Kiểm tra xem phần tử có tồn tại |
| indexOf(E e) | Trả về vị trí xuất hiện đầu tiên |
| size() | Trả về số lượng phần tử |

**Ví dụ sử dụng:**

```
ArrayList<String> students = new ArrayList<>();
students.add("Lan");
students.add("Hùng");
students.add("Tú");

System.out.println(students.get(1));       // Hùng
System.out.println(students.contains("Tú")); // true

students.remove(0); // Xóa "Lan"
students.set(0, "Minh"); // Gán "Minh" tại vị trí 0

System.out.println(students); // [Minh, Tú]
```

## 5.4. Ưu điểm & nhược điểm của ArrayList

| Ưu điểm | Nhược điểm |
|---|---|
| Truy cập phần tử theo chỉ số nhanh | Thêm/xóa ở giữa danh sách chậm (phải dịch phần tử) |
| Tự động tăng kích thước | Không tối ưu nếu thao tác nhiều với đầu danh sách |
| Hỗ trợ nhiều thao tác tiện lợi | Không đảm bảo hiệu suất như LinkedList trong mọi trường hợp |

## 5.5. Lỗi thường gặp khi dùng ArrayList

| Lỗi | Nguyên nhân | Cách xử lý |
|---|---|---|
| IndexOutOfBoundsException | Truy cập phần tử sai chỉ số | Kiểm tra .size() trước khi .get() |
| Thêm sai kiểu dữ liệu | Không dùng đúng kiểu ArrayList<E> | Dùng ArrayList<String> hoặc có generic cụ thể |
| Sử dụng vòng for-each để xóa phần tử | Gây lỗi runtime | Dùng Iterator để xóa an toàn |

## 5.6. So sánh ArrayList và mảng thường (String[])

| Tiêu chí | ArrayList<String> | String[] |
|---|---|---|
| Kích thước động | ✅ Có | ❌ Cố định |
| Có thể thêm/xóa dễ | ✅ Dễ | ❌ Phải tạo mảng mới |
| Dễ sử dụng, linh hoạt | ✅ | 🔁 Tốt cho hiệu suất cố định |

## 5.7. Bài tập thực hành gợi ý

**Bài 1: Quản lý danh sách tên sinh viên**

```
ArrayList<String> list = new ArrayList<>();
```

```
list.add("An");
list.add("Bình");
list.add("Cường");

System.out.println("Danh sách SV: " + list);
```

📝 **Bài 2: Đếm số tên có độ dài > 3**

```
int count = 0;
for (String name : list) {
    if (name.length() > 3) count++;
}
System.out.println("Số tên dài > 3 ký tự: " + count);
```

📝 **Bài 3: Xóa tên đầu tiên nếu là "An"**

```
if (list.size() > 0 && list.get(0).equals("An")) {
    list.remove(0);
}
System.out.println("Danh sách sau xóa: " + list);
```

### 🖊 5.8. Ghi nhớ nhanh

- 🔢 ArrayList giống như **mảng linh hoạt**: dễ thêm, dễ xóa, dễ duyệt.
- ✅ Dùng add, get, set, remove, contains, size.
- ❌ Tránh dùng chỉ số ngoài phạm vi (index < 0 hoặc ≥ size).
- ✅ Ưu tiên dùng List<String> list = new ArrayList<>(); để linh hoạt triển khai khác (LinkedList).

## 6. So sánh ArrayList và LinkedList

### 🧠 6.1. Điểm giống nhau

| Tiêu chí | ArrayList / LinkedList |
|---|---|
| Đều triển khai interface List | ✅ Có |
| Lưu trữ danh sách phần tử | ✅ Có |
| Hỗ trợ các thao tác như: add, remove, get, set, contains | ✅ Có |

🧩 Tuy giống nhau về giao diện (interface), nhưng khác biệt lớn về **cách hoạt động bên trong (implementation)**.

### 🛠 6.2. Khác biệt về cấu trúc bên trong

- ArrayList: dùng **mảng động** bên trong → truy cập nhanh theo chỉ số.
- LinkedList: dùng **danh sách liên kết kép (doubly linked list)** → mỗi phần tử là một **node** liên kết với phần tử trước và sau.

### 📊 6.3. Bảng so sánh tổng quan

| Đặc điểm | ArrayList | LinkedList |
|---|---|---|
| ⚡ Truy cập phần tử theo chỉ số | ✅ Nhanh (O(1)) | ❌ Chậm (O(n)) – phải duyệt từ đầu hoặc cuối |

| Đặc điểm | ArrayList | LinkedList |
|---|---|---|
| ➕ Thêm vào cuối list | ✅ Nhanh | ✅ Nhanh |
| ➕ Thêm/Xóa ở giữa | ❌ Chậm (phải dời các phần tử sau) | ✅ Nhanh (chỉ cập nhật liên kết node) |
| ➖ Xóa phần tử đầu tiên | ❌ Chậm (phải dời tất cả phần tử còn lại) | ✅ Rất nhanh (O(1)) |
| 💾 Bộ nhớ sử dụng | 🧠 Ít hơn – chỉ lưu phần tử | 🧠 Nhiều hơn – lưu cả phần tử + liên kết |
| 🔁 Duyệt bằng Iterator | ✅ Rất phù hợp | ✅ Rất phù hợp |

## 🔍 6.4. Khi nào dùng ArrayList, khi nào dùng LinkedList?

| Tình huống thực tế | Nên dùng |
|---|---|
| Cần truy cập phần tử theo chỉ số thường xuyên | ✅ ArrayList |
| Cần thêm/xóa ở đầu hoặc giữa thường xuyên | ✅ LinkedList |
| Danh sách ít thay đổi, đọc nhiều hơn ghi | ✅ ArrayList |
| Cần duyệt liên tục và sửa đổi khi duyệt | ✅ LinkedList (kết hợp Iterator) |

## 🗃️ 6.5. Ví dụ minh họa so sánh

📌 **Thêm 100,000 phần tử ở đầu danh sách**

```
List<Integer> arrayList = new ArrayList<>();
List<Integer> linkedList = new LinkedList<>();

// Thêm vào đầu
long start1 = System.currentTimeMillis();
for (int i = 0; i < 100000; i++) {
    arrayList.add(0, i);
}
long end1 = System.currentTimeMillis();

long start2 = System.currentTimeMillis();
for (int i = 0; i < 100000; i++) {
    linkedList.add(0, i);
}
long end2 = System.currentTimeMillis();

System.out.println("ArrayList time: " + (end1 - start1));
System.out.println("LinkedList time: " + (end2 - start2));
```

✅ **Kết quả thường thấy:** LinkedList nhanh hơn đáng kể khi thêm vào đầu.

## ⚠️ 6.6. Lỗi thường gặp khi sử dụng

| Lỗi | Giải thích |
|---|---|
| Truy cập chỉ số lớn trong LinkedList | Chậm do phải duyệt từ đầu |

| Lỗi | Giải thích |
|---|---|
| Dùng sai khi thao tác nhiều ở giữa list bằng ArrayList | Gây chậm chương trình |
| So sánh tốc độ nhưng dùng sai phép đo (System.nanoTime là chính xác hơn) | Đưa ra kết luận sai |

## 🧪 6.7. Bài tập thực hành gợi ý

📝 **Bài 1: Viết chương trình so sánh tốc độ thêm 10,000 phần tử vào cuối ArrayList và LinkedList.**
📝 **Bài 2: Viết chương trình chèn 1 phần tử vào giữa danh sách (vị trí index = size/2) và so sánh thời gian.**
📝 **Bài 3: Duyệt LinkedList bằng Iterator để xóa các phần tử chẵn.**

## 🖊️ 6.8. Ghi nhớ nhanh

- 🚀 ArrayList: tối ưu cho **truy cập**, **đọc nhiều – ghi ít**.
- 🧩 LinkedList: tối ưu cho **thêm/xóa ở đầu/giữa**.
- ✅ Đều có thể dùng thay thế nhau nếu không có yêu cầu đặc biệt, nhưng **hiệu suất sẽ khác nhau rõ rệt** khi số lượng phần tử lớn.

## Ví dụ có hướng dẫn giải

### 🎓 Ví dụ 1: Quản lý danh sách sinh viên – loại bỏ tên trùng và có độ dài chẵn

🎯 **Mục tiêu**

- Sử dụng ArrayList để lưu danh sách tên sinh viên.
- Sử dụng Collections để sắp xếp danh sách.
- Sử dụng Iterator để xóa các tên trùng và có độ dài chẵn.
- Áp dụng kiến thức về Collection, Collections, List, ArrayList, và Iterator.

📘 **Mô tả bài toán**

Giả sử bạn là một trợ giảng và cần quản lý danh sách tên sinh viên trong một lớp học. Do sinh viên tự nhập tên nên có thể:

- Có **tên trùng**.
- Có **tên có độ dài chẵn** (cần loại bỏ vì lý do kiểm tra thuật toán).
- Cuối cùng cần **sắp xếp danh sách còn lại theo thứ tự từ điển**.

🧠 **Phân tích**

| Thành phần | Áp dụng kiến thức nào? |
|---|---|
| ArrayList<String> | Áp dụng interface List, thao tác động |
| contains() | Dùng để loại bỏ trùng |
| Iterator | Xóa phần tử trong khi duyệt (tránh lỗi ConcurrentModification) |
| Collections.sort() | Dùng để sắp xếp danh sách |
| Collection | Dùng như kiểu tổng quát để nhận dữ liệu |

🧪 **Code minh họa**

```
import java.util.*;
```

```
public class StudentListManager {

    public static void main(String[] args) {
        // Bước 1: Khởi tạo danh sách sinh viên với vài tên trùng và độ dài chẵn
        List<String> studentNames = new ArrayList<>(Arrays.asList(
            "An", "Bình", "Chi", "An", "Dương", "Hà", "Lâm", "Chi", "Minh", "Hoa"
        ));

        System.out.println("Danh sách ban đầu:");
        System.out.println(studentNames);

        // Bước 2: Loại bỏ tên trùng
        List<String> uniqueNames = new ArrayList<>();
        for (String name : studentNames) {
            if (!uniqueNames.contains(name)) {
                uniqueNames.add(name);
            }
        }

        System.out.println("\nSau khi loại bỏ tên trùng:");
        System.out.println(uniqueNames);

        // Bước 3: Dùng Iterator để xóa tên có độ dài chẵn
        Iterator<String> it = uniqueNames.iterator();
        while (it.hasNext()) {
            String name = it.next();
            if (name.length() % 2 == 0) {
                it.remove();
            }
        }

        System.out.println("\nSau khi loại bỏ tên có độ dài chẵn:");
        System.out.println(uniqueNames);

        // Bước 4: Sắp xếp danh sách còn lại
        Collections.sort(uniqueNames);

        System.out.println("\nDanh sách sau khi sắp xếp:");
        System.out.println(uniqueNames);
    }
}
```

✅ **Kết quả minh họa (có thể thay đổi tùy input):**

```
Danh sách ban đầu:
[An, Bình, Chi, An, Dương, Hà, Lâm, Chi, Minh, Hoa]

Sau khi loại bỏ tên trùng:
[An, Bình, Chi, Dương, Hà, Lâm, Minh, Hoa]
```

```
Sau khi loại bỏ tên có độ dài chẵn:
[Bình, Chi, Dương, Hà, Minh]

Danh sách sau khi sắp xếp:
[Bình, Chi, Dương, Hà, Minh]
```

### 🔍 Giải thích chi tiết

- **ArrayList vs List vs Collection**: studentNames và uniqueNames là các ArrayList, nhưng được khai báo kiểu List<String> để tận dụng tính đa hình (polymorphism) – áp dụng kiến thức về interface Collection và List.
- **Dùng contains()** để kiểm tra tên trùng – thể hiện thao tác tìm kiếm trong danh sách.
- **Duyệt bằng Iterator và xóa trực tiếp** giúp tránh lỗi ConcurrentModificationException, chứng minh hiểu rõ cách dùng Iterator.
- **Collections.sort()** là phương thức tĩnh từ lớp Collections, không phải từ interface Collection.

### 💬 Tổng kết ví dụ 1

Ví dụ này đã:

- Bao quát được các điểm chính của Java Collection Framework.
- Thể hiện cách sử dụng ArrayList, Collections, Iterator, và kiến thức hệ thống cấp bậc Collection trong Java.
- Có khả năng mở rộng cho bài tập nâng cao như lọc theo tiêu chí khác, lưu file, hoặc chuyển sang LinkedList, HashSet, v.v.

## 🎓 Ví dụ 2: Thống kê tần suất xuất hiện của từ trong danh sách và lưu vào Map

### 🎯 Mục tiêu học tập

- Áp dụng List (ArrayList) để lưu chuỗi từ.
- Sử dụng Iterator để duyệt danh sách và xử lý chuỗi.
- Dùng Map<String, Integer> để đếm tần suất xuất hiện.
- Dùng Collections để tìm từ có tần suất lớn nhất.
- Làm rõ sự khác biệt giữa Collection và Collections.

### 📘 Mô tả bài toán

Giả sử bạn đang viết một ứng dụng nhỏ giúp thống kê từ xuất hiện trong bài phát biểu (hoặc đoạn văn), bao gồm:

- Bước 1: Lưu danh sách các từ trong một ArrayList.
- Bước 2: Sử dụng Iterator để duyệt qua từng từ.
- Bước 3: Lưu tần suất xuất hiện của mỗi từ vào Map<String, Integer>.
- Bước 4: Tìm từ có tần suất cao nhất bằng Collections.max().

### 🧠 Phân tích kiến thức áp dụng

| Thành phần | Kiến thức liên quan |
|---|---|
| ArrayList<String> | Dùng để lưu chuỗi từ – minh họa List, ArrayList |
| Map<String, Integer> | Sử dụng ánh xạ key-value – hiểu về Map |
| Iterator<String> | Duyệt danh sách để xử lý từ |
| Collections.max() | Tìm phần tử có giá trị lớn nhất từ Map.entrySet() |

| Thành phần | Kiến thức liên quan |
|---|---|
| Collection vs Collections | Làm rõ qua khai báo và phương thức tiện ích |

### 🧪 Code minh họa

```
import java.util.*;

public class WordFrequencyAnalyzer {
    public static void main(String[] args) {
        // Bước 1: Lưu danh sách từ vào ArrayList
        List<String> words = new ArrayList<>(Arrays.asList(
            "java", "collection", "framework", "java", "iterator",
            "list", "map", "collection", "list", "arraylist", "map", "map"
        ));

        // Bước 2: Duyệt bằng Iterator và thống kê tần suất
        Map<String, Integer> frequencyMap = new HashMap<>();
        Iterator<String> it = words.iterator();

        while (it.hasNext()) {
            String word = it.next();
            frequencyMap.put(word, frequencyMap.getOrDefault(word, 0) + 1);
        }

        // In ra toàn bộ từ và số lần xuất hiện
        System.out.println("Tần suất xuất hiện của từng từ:");
        for (Map.Entry<String, Integer> entry : frequencyMap.entrySet()) {
            System.out.println(entry.getKey() + " → " + entry.getValue());
        }

        // Bước 3: Tìm từ có tần suất lớn nhất
        Map.Entry<String, Integer> maxEntry = Collections.max(
            frequencyMap.entrySet(),
            Map.Entry.comparingByValue()
        );

        System.out.println("\nTừ xuất hiện nhiều nhất: " + maxEntry.getKey() +
                           " (" + maxEntry.getValue() + " lần)");
    }
}
```

### ✅ Kết quả minh họa

```
Tần suất xuất hiện của từng từ:
java → 2
collection → 2
framework → 1
iterator → 1
list → 2
map → 3
arraylist → 1
```

```
Từ xuất hiện nhiều nhất: map (3 lần)
```

### 🔍 Giải thích chi tiết

- words là một ArrayList<String> được tạo từ List interface, thể hiện rõ tính kế thừa Collection → List → ArrayList.
- Iterator<String> it = words.iterator() dùng để duyệt danh sách một cách an toàn (có thể dùng it.remove() nếu cần).
- Map<String, Integer> dùng để ánh xạ từ → số lần xuất hiện.
- Collections.max() là phương thức **tĩnh** trong lớp Collections, không liên quan tới Collection interface – **phân biệt rõ**.
- getOrDefault() giúp tránh lỗi null khi tăng biến đếm.

### 📌 Tổng kết kiến thức trong ví dụ

| Chủ đề | Minh họa |
|---|---|
| Collection vs Collections | Khai báo vs tiện ích |
| List, ArrayList | Tạo danh sách từ |
| Map | Lưu tần suất xuất hiện |
| Iterator | Duyệt danh sách từ |
| Collections.max() | Tìm phần tử có giá trị lớn nhất |

## ☑ Ví dụ 3: Quản lý lớp học – Danh sách sinh viên, tìm kiếm, cập nhật

- **Mục tiêu**: Xây dựng lớp Student, dùng ArrayList<Student> để quản lý danh sách sinh viên.
- **Chức năng**:
  - Thêm sinh viên mới
  - Tìm sinh viên theo mã số
  - Xóa sinh viên có GPA thấp
  - Sắp xếp theo tên hoặc GPA
- **Áp dụng**:
  - ArrayList + Collections.sort()
  - Comparable, Iterator, List

### ✅ Ví dụ 4: Thư viện sách – Quản lý kho sách theo chủ đề

- **Mục tiêu**: Xây dựng lớp Book, quản lý các sách theo chủ đề (Map<String, List<Book>>)
- **Chức năng**:
  - Thêm sách vào chủ đề
  - Hiển thị toàn bộ sách trong 1 chủ đề
  - Đếm số sách theo chủ đề
- **Áp dụng**:
  - Map, List, Collections, Set
  - Tích hợp Iterator, xử lý theo nhóm (grouping)

### 🎓 VÍ DỤ 3: Quản lý lớp học với ArrayList<Student>

### 🎯 Mục tiêu

- Củng cố tư duy OOP: class, object, encapsulation.

- Sử dụng ArrayList<Student> để lưu trữ.
- Áp dụng Collections.sort(), Iterator, Comparable.

### 📘 Bài toán

Thiết kế chương trình quản lý danh sách sinh viên gồm:

- **Lớp Student** có: mã số sinh viên (id), tên (name), điểm GPA (gpa).
- **Lớp Classroom** chứa ArrayList<Student> và các phương thức:
  - addStudent(Student s)
  - findById(String id)
  - removeLowGPA(double threshold)
  - sortByName(), sortByGPA()

### 🧪 Code minh họa

**1. Lớp Student (OOP cơ bản + Comparable)**

```
public class Student implements Comparable<Student> {
    private String id;
    private String name;
    private double gpa;

    public Student(String id, String name, double gpa) {
        this.id = id;
        this.name = name;
        this.gpa = gpa;
    }

    public String getId() { return id; }
    public String getName() { return name; }
    public double getGpa() { return gpa; }

    @Override
    public String toString() {
        return id + " - " + name + " - GPA: " + gpa;
    }

    @Override
    public int compareTo(Student other) {
        return this.name.compareToIgnoreCase(other.name); // Mặc định: sắp theo tên
    }
}
```

**2. Lớp Classroom – quản lý danh sách**

```
import java.util.*;

public class Classroom {
    private List<Student> students = new ArrayList<>();

    public void addStudent(Student s) {
        students.add(s);
    }
```

```
    public Student findById(String id) {
        for (Student s : students) {
            if (s.getId().equals(id)) return s;
        }
        return null;
    }

    public void removeLowGPA(double threshold) {
        Iterator<Student> it = students.iterator();
        while (it.hasNext()) {
            if (it.next().getGpa() < threshold) {
                it.remove();
            }
        }
    }

    public void sortByName() {
        Collections.sort(students);
    }

    public void sortByGPA() {
        students.sort(Comparator.comparingDouble(Student::getGpa).reversed());
    }

    public void printAll() {
        for (Student s : students) {
            System.out.println(s);
        }
    }
}
```

**3. Hàm main để chạy chương trình**

```
public class Main {
    public static void main(String[] args) {
        Classroom c = new Classroom();

        c.addStudent(new Student("S01", "An", 3.2));
        c.addStudent(new Student("S02", "Bình", 2.5));
        c.addStudent(new Student("S03", "Chi", 3.8));
        c.addStudent(new Student("S04", "Dương", 1.9));

        System.out.println("=== Danh sách ban đầu ===");
        c.printAll();

        System.out.println("\n=== Xoá SV GPA < 2.0 ===");
        c.removeLowGPA(2.0);
        c.printAll();

        System.out.println("\n=== Sắp xếp theo GPA giảm dần ===");
```

```
        c.sortByGPA();
        c.printAll();

        System.out.println("\n=== Tìm theo ID 'S03' ===");
        Student s = c.findById("S03");
        System.out.println(s != null ? s : "Không tìm thấy.");
    }
}
```

✅ **Kết quả mẫu**

```
=== Danh sách ban đầu ===
S01 - An - GPA: 3.2
S02 - Bình - GPA: 2.5
S03 - Chi - GPA: 3.8
S04 - Dương - GPA: 1.9

=== Xoá SV GPA < 2.0 ===
S01 - An - GPA: 3.2
S02 - Bình - GPA: 2.5
S03 - Chi - GPA: 3.8

=== Sắp xếp theo GPA giảm dần ===
S03 - Chi - GPA: 3.8
S01 - An - GPA: 3.2
S02 - Bình - GPA: 2.5

=== Tìm theo ID 'S03' ===
S03 - Chi - GPA: 3.8
```

🧠 **Tổng kết kiến thức**

| Kiến thức | Minh họa |
|---|---|
| Lập trình hướng đối tượng | Class Student, Classroom |
| Kế thừa từ Collection | ArrayList<Student> |
| Giao diện Comparable | Sắp xếp mặc định theo tên |
| Collections.sort(), Comparator | Sắp xếp theo GPA |
| Iterator | Xóa phần tử an toàn khi duyệt |
| Tính đóng gói | Thuộc tính private, getter |

## 🎓 Ví dụ 4: Thư viện sách – Quản lý kho sách theo chủ đề

🎯 **Mục tiêu**

- Xây dựng class Book và class Library.
- Sử dụng Map<String, List<Book>> để lưu sách theo **chủ đề**.
- Áp dụng Set, List, Collections, Iterator, duyệt map.
- Tư duy hướng đối tượng: **đóng gói**, **trách nhiệm rõ ràng**, **xử lý theo nhóm**.

📘 **Mô tả bài toán**

Viết chương trình mô phỏng **quản lý thư viện**, trong đó:

- Mỗi quyển sách thuộc một chủ đề (vd: "Văn học", "Khoa học", "Lập trình").

- Thư viện có thể:
  - ✅ Thêm sách vào chủ đề.
  - ✅ In danh sách sách của một chủ đề.
  - ✅ In toàn bộ thư viện (gồm nhiều chủ đề).
  - ✅ Đếm tổng số sách mỗi chủ đề.
  - ✅ Duyệt toàn bộ thư viện bằng Iterator.

### 📦 Thiết kế OOP

### ✅ Lớp Book

```
public class Book {
    private String title;
    private String author;
    private int year;

    public Book(String title, String author, int year) {
        this.title = title;
        this.author = author;
        this.year = year;
    }

    public String getTitle() { return title; }
    public String getAuthor() { return author; }
    public int getYear() { return year; }

    @Override
    public String toString() {
        return "\"" + title + "\", " + author + " (" + year + ")";
    }
}
```

### ✅ Lớp Library – quản lý Map<String, List<Book>>

```
import java.util.*;

public class Library {
    private Map<String, List<Book>> shelves = new HashMap<>();

    // Thêm sách vào chủ đề
    public void addBook(String topic, Book book) {
        shelves.computeIfAbsent(topic, k -> new ArrayList<>()).add(book);
    }

    // In toàn bộ sách theo 1 chủ đề
    public void printBooksInTopic(String topic) {
        List<Book> books = shelves.get(topic);
        if (books == null || books.isEmpty()) {
            System.out.println("Không có sách trong chủ đề: " + topic);
            return;
        }
```

```
        System.out.println("📚 Chủ đề: " + topic);
        for (Book b : books) {
            System.out.println("- " + b);
        }
    }

    // In toàn bộ thư viện
    public void printAll() {
        for (String topic : shelves.keySet()) {
            printBooksInTopic(topic);
            System.out.println();
        }
    }

    // Đếm số sách theo chủ đề
    public void countBooksByTopic() {
        for (Map.Entry<String, List<Book>> entry : shelves.entrySet()) {
            String topic = entry.getKey();
            int count = entry.getValue().size();
            System.out.println(topic + ": " + count + " quyển");
        }
    }

    // Duyệt toàn bộ sách bằng Iterator (nâng cao)
    public void iterateAllBooks() {
        System.out.println("📖 Toàn bộ sách trong thư viện (duyệt Iterator):");
        for (Map.Entry<String, List<Book>> entry : shelves.entrySet()) {
            String topic = entry.getKey();
            List<Book> books = entry.getValue();
            Iterator<Book> it = books.iterator();

            while (it.hasNext()) {
                System.out.println("[" + topic + "] " + it.next());
            }
        }
    }
}
```

✅ **Hàm main – kiểm tra chương trình**

```
public class Main {
    public static void main(String[] args) {
        Library lib = new Library();

        // Thêm sách vào thư viện
        lib.addBook("Lập trình", new Book("Java Cơ Bản", "Nguyễn Văn A", 2022));
        lib.addBook("Lập trình", new Book("Cấu Trúc Dữ Liệu", "Trần B", 2020));
        lib.addBook("Văn học", new Book("Chí Phèo", "Nam Cao", 1941));
        lib.addBook("Văn học", new Book("Tắt Đèn", "Ngô Tất Tố", 1939));
```

```
        lib.addBook("Khoa học", new Book("Vũ Trụ Trong Hạt Cát", "Stephen Hawking",
2014));

        // In toàn bộ
        System.out.println("=== Thư viện ===");
        lib.printAll();

        // In sách theo chủ đề
        System.out.println("=== Sách thuộc chủ đề 'Văn học' ===");
        lib.printBooksInTopic("Văn học");

        // Đếm số lượng
        System.out.println("\n=== Số sách theo từng chủ đề ===");
        lib.countBooksByTopic();

        // Duyệt bằng Iterator
        System.out.println("\n=== Duyệt toàn bộ bằng Iterator ===");
        lib.iterateAllBooks();
    }
}
```

✅ **Kết quả minh họa**

```
=== Thư viện ===
📚 Chủ đề: Lập trình
- "Java Cơ Bản", Nguyễn Văn A (2022)
- "Cấu Trúc Dữ Liệu", Trần B (2020)

📚 Chủ đề: Văn học
- "Chí Phèo", Nam Cao (1941)
- "Tắt Đèn", Ngô Tất Tố (1939)

📚 Chủ đề: Khoa học
- "Vũ Trụ Trong Hạt Cát", Stephen Hawking (2014)

=== Sách thuộc chủ đề 'Văn học' ===
📚 Chủ đề: Văn học
- "Chí Phèo", Nam Cao (1941)
- "Tắt Đèn", Ngô Tất Tố (1939)

=== Số sách theo từng chủ đề ===
Lập trình: 2 quyển
Văn học: 2 quyển
Khoa học: 1 quyển

=== Duyệt toàn bộ bằng Iterator ===
📖 Toàn bộ sách trong thư viện (duyệt Iterator):
[Lập trình] "Java Cơ Bản", Nguyễn Văn A (2022)
[Lập trình] "Cấu Trúc Dữ Liệu", Trần B (2020)
[Văn học] "Chí Phèo", Nam Cao (1941)
```

[Văn học] "Tắt Đèn", Ngô Tất Tố (1939)
[Khoa học] "Vũ Trụ Trong Hạt Cát", Stephen Hawking (2014)

### 🧠 Tổng kết kiến thức đã dùng

| Kiến thức Java | Minh họa |
|---|---|
| Map<String, List<Book>> | Group theo chủ đề |
| ArrayList<Book> | Lưu sách trong từng nhóm |
| Iterator | Duyệt qua danh sách sách |
| Set | Duyệt qua các chủ đề (keySet) |
| OOP | Class Book, Library, đóng gói, phương thức rõ ràng |
| Collections | Có thể mở rộng với Collections.sort() nếu sắp xếp theo năm xuất bản |

## TRẮC NGHIỆM

### 🏵️ 1. Collection vs. Collections (Câu 1–5)

**Câu 1 (Dễ):**
Collection trong Java là gì?
A. Một lớp tiện ích để thao tác trên danh sách
B. Một lớp dùng để lưu key-value
C. Một interface gốc cho các cấu trúc dữ liệu
D. Một kiểu dữ liệu đặc biệt của Java

**Câu 2 (Dễ):**
Lớp Collections có đặc điểm nào sau đây?
A. Là một interface
B. Là một lớp trừu tượng
C. Chứa các phương thức tiện ích static
D. Dùng để lưu trữ phần tử

**Câu 3 (Trung bình):**
Phát biểu nào **sai** về Collection và Collections?
A. Collection là interface
B. Collections có thể khởi tạo đối tượng bằng new
C. Collections.sort() có thể dùng cho ArrayList
D. Collection không hỗ trợ sắp xếp

**Câu 4 (Khó):**
Giả sử bạn cần kiểm tra xem danh sách có phần tử "Java" hay không, bạn nên dùng?
A. Collection.equals("Java")
B. Collections.contains("Java")
C. list.contains("Java")
D. Collections.has("Java")

**Câu 5 (Khó):**
Kết quả của đoạn mã sau là gì?

```
List<Integer> list = new ArrayList<>(Arrays.asList(4, 3, 2));
Collections.reverse(list);
System.out.println(list);
```

A. [4, 3, 2]
B. [2, 3, 4]
C. [3, 2, 4]
D. [2, 3, 4, 2]

## 2. Interface chính trong Collection (Câu 6–9)

**Câu 6 (Dễ):**
Interface nào cho phép phần tử trùng lặp?
A. Set
B. List
C. Map
D. Collection

**Câu 7 (Dễ):**
Interface Map<K, V> dùng để:
A. Duyệt theo thứ tự
B. Lưu trữ danh sách có sắp xếp
C. Lưu trữ cặp khóa - giá trị
D. Lưu trữ phần tử duy nhất

**Câu 8 (Trung bình):**
Phát biểu đúng về Set là:
A. Không cho phép phần tử trùng lặp
B. Có thể truy cập qua chỉ số
C. Có thứ tự rõ ràng
D. Cho phép lưu key-value

**Câu 9 (Khó):**
Tập nào duy trì thứ tự phần tử như lúc thêm vào?
A. HashSet
B. TreeSet
C. LinkedHashSet
D. UnorderedSet

## 3. Iterator Interface (Câu 10–13)

**Câu 10 (Dễ):**
Phương thức nào không có trong Iterator?
A. hasNext()
B. next()
C. previous()
D. remove()

**Câu 11 (Dễ):**
Phát biểu nào đúng về Iterator?
A. Cho phép xóa phần tử khi duyệt
B. Không thể dùng cho Set
C. Duyệt ngẫu nhiên
D. Phải khởi tạo bằng new

**Câu 12 (Trung bình):**
Đoạn mã nào dùng Iterator đúng cách?

```
Iterator<String> it = list.iterator();
while (it.hasNext()) {
    String val = it.next();
    if (val.equals("Java")) it.remove();
}
```

A. Sai cú pháp
B. Gây lỗi runtime
C. Hoạt động đúng
D. Sai vì không có get()

**Câu 13 (Khó):**
Điều gì xảy ra nếu gọi remove() hai lần liên tiếp không qua next()?
A. Không ảnh hưởng
B. Xóa hai phần tử
C. Gây lỗi IllegalStateException
D. Vòng lặp bỏ qua phần tử

## 4. List & ArrayList (Câu 14–17)

**Câu 14 (Dễ):**
Phương thức nào dùng để lấy số phần tử trong ArrayList?
A. count()
B. length()
C. size()
D. getSize()

**Câu 15 (Trung bình):**
Giá trị của names.get(1) trong đoạn sau là gì?

```
ArrayList<String> names = new ArrayList<>();
names.add("An");
names.add("Bình");
names.add("Tú");
```

A. An
B. Tú
C. Bình
D. null

**Câu 16 (Khó):**
Phát biểu đúng về set() trong ArrayList?
A. Dùng để thêm mới phần tử
B. Dùng để xóa phần tử

C. Dùng để thay thế phần tử tại chỉ số
D. Không tồn tại

**Câu 17 (Khó):**
Đâu là ưu điểm lớn nhất của ArrayList so với LinkedList?
A. Chèn đầu danh sách nhanh hơn
B. Xóa phần tử nhanh hơn
C. Truy cập phần tử theo chỉ số nhanh hơn
D. Không cần import thêm gì

### 💠 5. So sánh ArrayList và LinkedList (Câu 18–20)

**Câu 18 (Dễ):**
LinkedList hoạt động tốt nhất khi:
A. Truy cập phần tử ngẫu nhiên
B. Xóa phần tử ở cuối
C. Thêm/xóa ở đầu hoặc giữa danh sách
D. Sắp xếp mảng

**Câu 19 (Trung bình):**
Cấu trúc lưu trữ của LinkedList là:
A. Mảng động
B. Cây nhị phân
C. Danh sách liên kết đơn
D. Danh sách liên kết kép (doubly linked list)

**Câu 20 (Khó):**
Đâu là nhược điểm chính của LinkedList?
A. Không thêm được phần tử
B. Truy cập phần tử theo chỉ số chậm
C. Không xóa được phần tử
D. Không thể duyệt bằng Iterator

## BÀI TẬP LUYỆN TẬP

### 🖌️ Bài 1: Phân biệt Collection vs. Collections thông qua lớp CourseList

🎯 **Mục tiêu:**
Hiểu và áp dụng sự khác biệt giữa Collection và Collections.

📝 **Đề bài OOP:**
Thiết kế lớp CourseList dùng để lưu danh sách các môn học. Thêm khả năng sắp xếp danh sách môn học theo thứ tự ABC.

✅ **Hướng dẫn OOP:**

1. Tạo lớp CourseList có thuộc tính: List<String> courses.
2. Xây dựng phương thức addCourse(String course) để thêm môn học.
3. Viết phương thức printCourses() để in danh sách hiện tại.
4. Viết phương thức sortCourses() – dùng Collections.sort(...).
5. Trong main, tạo đối tượng CourseList, thêm 5 môn học, in ra danh sách trước và sau khi sắp xếp.

## 🧪 Bài 2: Quản lý sinh viên với lớp StudentList

🎯 **Mục tiêu:**

Thao tác CRUD (thêm - đọc - cập nhật - xóa) với ArrayList.

📝 **Đề bài OOP:**

Xây dựng lớp StudentList để quản lý danh sách sinh viên. Thực hiện truy xuất và cập nhật thông tin sinh viên theo chỉ số.

✅ **Hướng dẫn OOP:**

1. Tạo lớp StudentList với thuộc tính List<String> students.
2. Thêm phương thức addStudent(String name), updateStudent(int index, String newName), removeLastStudent(), getStudent(int index).
3. Tạo lớp Main để thêm 4 sinh viên và thử các thao tác trên.

## 🧪 Bài 3: Lớp NameAnalyzer – Đếm số lần xuất hiện của "An"

🎯 **Mục tiêu:**

Thao tác tìm kiếm trong danh sách với OOP.

📝 **Đề bài OOP:**

Tạo lớp NameAnalyzer có khả năng đếm số lần một tên cụ thể xuất hiện trong danh sách tên được nhập từ người dùng.

✅ **Hướng dẫn OOP:**

1. Tạo lớp NameAnalyzer có thuộc tính List<String> names.
2. Thêm phương thức inputNames() dùng Scanner.
3. Viết phương thức countName(String target) trả về số lần xuất hiện.
4. In kết quả ra màn hình với tên "An".

## 🧪 Bài 4: Lọc tên sản phẩm trùng với lớp ProductCatalog

🎯 **Mục tiêu:**

Lọc dữ liệu trùng lặp bằng Set trong cấu trúc lớp.

📝 **Đề bài OOP:**

Xây dựng lớp ProductCatalog để quản lý tên sản phẩm, loại bỏ trùng lặp bằng Set.

✅ **Hướng dẫn OOP:**

1. Tạo lớp ProductCatalog với thuộc tính List<String> rawProducts và Set<String> uniqueProducts.
2. Thêm phương thức addProduct(String name) và filterDuplicates().
3. Viết phương thức printAll() để in cả danh sách gốc và sau khi lọc trùng.

## 🧪 Bài 5: Quản lý điểm sinh viên bằng lớp Gradebook

🎯 **Mục tiêu:**

Sử dụng Map<String, Double> để quản lý dữ liệu theo cặp khóa-giá trị.

📝 **Đề bài OOP:**

Thiết kế lớp Gradebook để lưu điểm sinh viên và cho phép truy xuất điểm theo tên.

✅ **Hướng dẫn OOP:**

1. Tạo lớp Gradebook có thuộc tính Map<String, Double> grades.
2. Viết phương thức addStudentGrade(String name, double score) và getGrade(String name).
3. Sử dụng containsKey() để kiểm tra tên có tồn tại trước khi truy xuất điểm.

## 🧪 Bài 6: Duyệt danh sách bằng Iterator với lớp WordList

🎯 **Mục tiêu:**
Làm quen với Iterator trong thiết kế lớp.

📝 **Đề bài OOP:**
Tạo lớp WordList chứa danh sách từ. Viết phương thức in ra từ và độ dài tương ứng.

✅ **Hướng dẫn OOP:**

1. Lớp WordList có thuộc tính List<String> words.
2. Viết phương thức addWord(String w) và printWordLengths() – dùng Iterator<String>.
3. Mỗi lần duyệt, in ra word + length.

## 🧪 Bài 7: Xóa từ có độ dài chẵn với lớp SmartWordList

🎯 **Mục tiêu:**
Sử dụng Iterator.remove() trong thao tác xóa an toàn.

📝 **Đề bài OOP:**
Tạo lớp SmartWordList và thêm phương thức xóa các chuỗi có độ dài chẵn.

✅ **Hướng dẫn OOP:**

1. Tạo lớp SmartWordList với List<String> words.
2. Phương thức removeEvenLengthWords() – dùng Iterator + remove().
3. Sau khi xóa, in danh sách kết quả ra màn hình.

## 🧪 Bài 8: So sánh hiệu năng với lớp PerformanceTest

🎯 **Mục tiêu:**
Phân tích hiệu năng thao tác của ArrayList vs LinkedList.

📝 **Đề bài OOP:**
Viết lớp PerformanceTest đo thời gian thêm 10,000 phần tử vào đầu danh sách.

✅ **Hướng dẫn OOP:**

1. Lớp có 2 phương thức: testArrayListInsert() và testLinkedListInsert().
2. Dùng System.currentTimeMillis() đo thời gian.
3. In thời gian cho mỗi cấu trúc list.

## 🧪 Bài 9: Sắp xếp và trộn ngẫu nhiên với lớp NameShuffler

🎯 **Mục tiêu:**
Thành thạo Collections.reverse() và Collections.shuffle() theo kiểu OOP.

📝 **Đề bài OOP:**
Tạo lớp NameShuffler lưu danh sách tên và có thể đảo thứ tự hoặc trộn ngẫu nhiên.

✅ **Hướng dẫn OOP:**

1. Thuộc tính: List<String> names.
2. Phương thức: reverseNames(), shuffleNames(), printNames().
3. Gọi thử các thao tác trong main.

## 🧪 Bài 10: Tìm tên dài nhất với lớp NameAnalyzer

🎯 **Mục tiêu:**

Áp dụng Iterator và logic so sánh trong thiết kế lớp.

📝 **Đề bài OOP:**

Thiết kế lớp NameAnalyzer để tìm và in ra tên dài nhất trong danh sách.

✅ **Hướng dẫn OOP:**

1. Tạo thuộc tính List<String> names.
2. Phương thức findLongestName() dùng Iterator để duyệt.
3. So sánh độ dài, cập nhật biến longest.

🎁 **Gợi ý nâng cao**

- Bạn có thể gộp các lớp thành **1 package studenttools, collectionshelper, libraryutils,...**
- Có thể thêm giao diện NameCollection, GradeStatistic để làm bài khó hơn.
- Nếu cần: mình có thể đóng gói tất cả bài này thành **file .docx, .pdf** hoặc đề thi tự động!

# DỰ ÁN

## 📓 Dự án Nhóm 1 (Phiên bản OOP): Quản lý Từ điển Cá nhân – English ↔ Vietnamese

🎯 **Mục tiêu hướng đối tượng**

- Tổ chức chương trình theo class và method.
- Sử dụng Map<String, String> trong lớp riêng biệt.
- Phân tách rõ **giao diện – xử lý dữ liệu – điều phối logic**.
- Áp dụng kiểm tra dữ liệu hợp lệ và khả năng mở rộng.

🧱 **Tổng quan cấu trúc lớp**

| Lớp | Vai trò | Trách nhiệm |
|---|---|---|
| PersonalDictionary | Model/Service | Lưu trữ và thao tác với từ điển |
| DictionaryUI | View (menu console) | Hiển thị menu và nhận input từ người dùng |
| DictionaryApp | Controller | Điều phối chương trình chính, gọi UI và logic |

🛠 **Chi tiết triển khai từng phần**

🧩 **Lớp 1: PersonalDictionary**

Quản lý dữ liệu từ điển và các thao tác chính.

🔑 **Thuộc tính:**

```
private Map<String, String> dictionary;
```

🧠 **Phương thức:**

- boolean addWord(String english, String vietnamese) – thêm từ nếu chưa có.
- String lookup(String english) – tra từ.
- boolean removeWord(String english) – xóa từ nếu tồn tại.

- Map<String, String> getAllWordsSorted() – trả về từ điển đã sắp xếp.
- int getWordCount() – đếm số từ hiện có.

✅ Gợi ý: sử dụng TreeMap hoặc Collections.sort(dictionary.keySet()) để sắp xếp từ.

### 🧩 Lớp 2: DictionaryUI

Hiển thị giao diện điều hướng (console-based).

🧠 **Phương thức:**

- void showMenu() – hiển thị menu.
- String getInput(String message) – nhận input từ người dùng.
- void showMessage(String msg) – in thông báo nhanh.
- void displayAllWords(Map<String, String> map) – in danh sách từ.
- void showWordMeaning(String word, String meaning) – in kết quả tra cứu.

✅ UI nên nhẹ và không chứa logic xử lý dữ liệu, chỉ nhận và hiển thị.

### 🧩 Lớp 3: DictionaryApp

Điều phối toàn bộ chương trình.

🦴 **Chức năng chính trong main:**

1. Tạo đối tượng PersonalDictionary và DictionaryUI.
2. Vòng lặp menu:
    - Gọi UI để hiển thị menu.
    - Nhận lựa chọn từ người dùng.
    - Gọi logic phù hợp (thêm, tra, xóa, in, thoát).
3. Kiểm tra hợp lệ khi người dùng:
    - Thêm trùng → cảnh báo.
    - Xóa/tra không tồn tại → thông báo lỗi.

🔁 **Menu Gợi Ý (dùng trong DictionaryUI)**

```
========== TỪ ĐIỂN CÁ NHÂN ==========
1. Thêm từ mới
2. Tra từ
3. Xóa từ
4. In toàn bộ từ điển
5. Thống kê số từ
0. Thoát
=====================================
```

🧪 **Gợi ý mở rộng**

- ✅ Nhập hàng loạt: thêm một chức năng cho phép nhập N từ một lúc.
- ✅ Cho phép cập nhật nghĩa mới nếu từ đã tồn tại.
- ✅ Tìm từ gần đúng (gợi ý startsWith, contains…).
- ✅ Lưu file tạm thời qua session (tuỳ nhóm nâng cao).

👨‍👩‍👧‍👦 **Phân chia nhóm công việc**

| Thành viên | Nhiệm vụ |
|---|---|
| 1 bạn | Thiết kế lớp PersonalDictionary, viết các phương thức |
| 1 bạn | Thiết kế UI menu và xử lý tương tác console |
| 1 bạn | Viết lớp DictionaryApp điều phối các luồng chức năng |

| Thành viên | Nhiệm vụ |
|---|---|
| 1 bạn | Kiểm thử (test case), kiểm tra dữ liệu, kiểm tra input, demo |

### 📦 Đầu ra dự kiến

- File .java cho từng class riêng biệt.
- Có thể đóng gói trong package dictionary.
- Có hướng dẫn chạy, giao diện đơn giản dễ sử dụng.

## 📓 Dự án Nhóm 2 (Phiên bản OOP): Quản lý Đơn hàng Online – Sử dụng List & Map

### 🎯 Mục tiêu hướng đối tượng

- Thiết kế hệ thống gồm nhiều lớp theo OOP.
- Kết hợp ArrayList<Order> để lưu trữ danh sách đơn hàng.
- Kết hợp Map<String, Integer> để thống kê theo sản phẩm.
- Làm quen với xử lý truy vấn bằng khóa (orderId).

### 🧱 Cấu trúc lớp gợi ý

| Lớp | Vai trò | Trách nhiệm |
|---|---|---|
| Order | Model (Entity) | Đại diện cho một đơn hàng |
| OrderManager | Service | Xử lý thêm, xóa, tìm kiếm, thống kê đơn hàng |
| OrderUI | View (Console UI) | Hiển thị menu và nhận input |
| OrderApp | Controller | Luồng điều phối chính |

### 🛠 Chi tiết các lớp

### 🧩 Lớp 1: Order (Model)

Lưu trữ thông tin đơn hàng.

**🔑 Thuộc tính:**

```
private String orderId;
private String customerName;
private String productName;
private int quantity;
```

**🧠 Phương thức:**

- Constructor, getter/setter đầy đủ.
- toString() để hiển thị đơn hàng.

### 🧩 Lớp 2: OrderManager

Chịu trách nhiệm xử lý nghiệp vụ quản lý đơn hàng.

**🔑 Thuộc tính:**

```
private List<Order> orderList;
```

**🧠 Phương thức:**

- boolean addOrder(Order o) – thêm đơn hàng nếu chưa trùng orderId.
- List<Order> getAllOrders() – trả về toàn bộ danh sách.
- Order findOrderById(String id) – tìm đơn hàng theo mã.

- boolean removeOrder(String id) – xóa nếu tìm thấy.
- int getTotalOrderCount() – trả về orderList.size().
- int getTotalQuantitySold() – cộng tất cả quantity từ đơn hàng.
- Map<String, Integer> countOrdersByProduct() – thống kê đơn theo sản phẩm (dùng Map<String, Integer>).

✅ Có thể dùng stream() hoặc duyệt thủ công để tổng hợp.

### 🧩 Lớp 3: OrderUI

Menu điều hướng, nhận dữ liệu từ người dùng.

🧠 **Phương thức:**

- void showMenu() – in menu chính.
- String prompt(String msg) – nhận input từ Scanner.
- void showOrder(Order o) – in đơn hàng.
- void showAllOrders(List<Order>) – in toàn bộ đơn hàng.
- void showStatistics(...) – thống kê tổng số đơn hàng/sản phẩm.

### 🧩 Lớp 4: OrderApp (Controller)

Chạy chương trình chính, kết nối UI và logic xử lý.

🔧 **Vòng lặp main:**

1. Tạo OrderManager, OrderUI.
2. Hiển thị menu, nhận lựa chọn từ người dùng.
3. Sử dụng switch-case:
    - Thêm đơn hàng
    - Hiển thị toàn bộ đơn hàng
    - Tìm đơn hàng theo mã
    - Xóa đơn hàng
    - Thống kê (tổng đơn, tổng sản phẩm, thống kê theo sản phẩm)
    - Thoát chương trình

✅ **Kiểm tra hợp lệ:**

- Kiểm tra orderId đã tồn tại khi thêm.
- Cảnh báo nếu không tìm thấy khi tra cứu/xóa.

### 📜 Gợi ý menu console

```
========= QUẢN LÝ ĐƠN HÀNG =========
1. Thêm đơn hàng mới
2. Hiển thị toàn bộ đơn hàng
3. Tìm đơn hàng theo mã
4. Xóa đơn hàng theo mã
5. Thống kê tổng đơn & sản phẩm
6. Thống kê đơn hàng theo sản phẩm
0. Thoát
==================================
```

### 📊 Gợi ý mở rộng

| Tính năng nâng cao | Cách triển khai |
|---|---|
| 🔄 Sắp xếp đơn hàng theo tên khách hoặc mã | Sử dụng Comparator và Collections.sort() |
| 📦 Nhập nhiều đơn hàng liên tiếp | Cho vòng lặp nhập lặp lại N lần |

| Tính năng nâng cao | Cách triển khai |
| --- | --- |
| 📈 Biểu đồ thống kê (nâng cao, có thể dùng file CSV) | Xuất kết quả từ Map<String, Integer> ra file CSV |
| 🔒 Kiểm tra trùng lặp orderId | Kiểm tra tồn tại trước khi thêm |

### 👫 Gợi ý chia nhóm

| Thành viên | Nhiệm vụ |
| --- | --- |
| 1 bạn | Thiết kế lớp Order và viết test constructor/toString |
| 1 bạn | Viết OrderManager, xử lý CRUD và thống kê |
| 1 bạn | Xây menu OrderUI và nhập dữ liệu từ Scanner |
| 1 bạn | Điều phối OrderApp, gắn các lớp với nhau và kiểm thử |

### 📦 Đầu ra mong muốn

- Dự án gồm nhiều file .java chia lớp rõ ràng.
- Có thể chạy bằng IntelliJ, Eclipse hoặc console.
- Mỗi nhóm tạo tài liệu README mô tả:
  - Cấu trúc lớp
  - Menu chức năng
  - Cách chạy chương trình