**Task 4: Collection Framework và Generic**

1. **Collection Framework**

- Collection framework: một tập các class và interface hỗ trợ thao tác lưu trữ và quản lý nhóm đối tượng dữ liệu.

- Bao gồm: Interface, Classes và Algorithm.

- Có 2 loại cơ bản là Collection Interface và Map Interface:

* Collection Interface: dùng để đại diện cho các nhóm phần tử và cung cấp các phương thức để quản lý. Các class được triển khai: List, Set, Queue.
* Map Interface: tạo ra ánh xạ giữa các cặp key - value. Hỗ trợ lưu trữ, truy xuất,... theo key. Các class được triển khai: HashMap, TreeMap.

1. **List**

- Là dạng danh sách các phần tử có thể được trùng lặp.

- Không thể tạo đối tượng từ List vì nó là một Interface.

- Các class có thể sử dụng: ArrayList, LinkedList, Vector.

1. **ArrayList**

- Một loại CTDL được dùng để lưu trữ các phần tử tương tự nhau.

- Thao tác dễ dàng với các phần tử trong danh sách thông qua index.

- Khác với mảng tích hợp, ArrayList có khả năng tự động mở rộng kích thước.

- Để sử dụng, nhập vào từ gói java.util: *import java.util.ArrayList;*

- Tạo một đối tượng:

*ArrayList<objectType> name = new ArrayList<>();*

ví dụ: *ArrayList<String> cars = new ArrayList<>();*

- Các phương thức:

|  |  |
| --- | --- |
| **Phương thức** | **Ý nghĩa** |
| add() | - Thêm phần tử vào cuối danh sách: *cars.add(“Vinfast”);*  - Thêm phần tử vào vị trí cụ thể: *cars.add(0, “Honda”);* |
| get() | - Truy cập phần tử ở chỉ mục cụ thể: *cars.get(0); // Vinfast* |
| contains() | - Trả về true nếu phần tử có trong danh sách: *cars.contains(“Honda”); // true* |
| set() | - Thay đổi phần tử ở chỉ mục cụ thể: *cars.set(1, “Volvo”); cars.get(1); // Volvo* |
| remove() | - Xóa phần tử ở chỉ mục được chỉ định: *cars.remove(0);* |
| clear() | - Xóa tất cả các phần tử trong danh sách: *cars.clear();* |
| size() | - Trả về số phần tử trong danh sách: *cars.size();* |
| isEmpty() | - Trả về true nếu danh sách rỗng và ngược lại |

- Có thể sử dụng phương thức sort() của lớp Collections để sắp xếp danh sách, for-each để duyệt danh sách.

- Ưu điểm:

* Dễ dàng duyệt danh sách, tìm kiếm phần tử,...
* Có thể chứa các đối tượng của các class khác nhau (khai báo với kiểu dữ liệu Object).

- Nhược điểm:

* Hiệu suất chương trình tỉ lệ nghịch với kích thước danh sách.
* Thêm phần tử vào giữa danh sách có thể chậm hơn so với cuối danh sách.

1. **Set**

- Lưu trữ các phần tử không trùng lặp và không có thứ tự cụ thể. Không hỗ trợ truy cập các phần tử bằng index.

1. **HashSet**

- HashSet kế thừa các đặc điểm của Set.

- Để sử dụng, nhập từ gói java.util: import java.util.HashSet;

- Khai báo: *HashSet<objectType> name = new HashSet<>();*

ví dụ: *HashSet<String> cars = new HashSet<>();*

- Phương thức:

|  |  |
| --- | --- |
| **Phương thức** | **Ý nghĩa** |
| add() | - Thêm các mục vào set: *cars.add(“Vinfast”);* |
| contains() | - Kiểm tra một mục có tồn tại không: *cars.contains(“BMW”); // false* |
| remove() | - Xóa một mục: *cars.remove(“Vinfast”);* |
| clear() | - Xóa tất cả các mục: *cars.clear();* |
| size() | - Trả về số mục: *cars.size();* |

- Ưu điểm:

* Tốc độ truy xuất nhanh nhờ sử dụng bảng băm để lưu trữ phần tử.
* Hữu ích khi cần lưu danh sách các phần tử duy nhất.

- Nhược điểm:

* Không duy trì thứ tự các phần tử.
* Không khả thi nếu muốn truy cập phần tử bằng index.

1. **Map**

- Biểu diễn một tập các phần tử theo cặp key - value. Các key là duy nhất.

1. **HashMap**

- Để sử dụng, nhập từ gói java.util: import java.util.HashMap;

- Khai báo: *HashMap<objectType1, objectType2> name = new HashMap<>();*

trong đó: *objectType1* là kiểu dữ liệu của key, *objectType2* là kiểu dữ liệu của value.

ví dụ: *HashMap<Integer, String> cars = new HashMap<>();*

- Phương thức:

|  |  |
| --- | --- |
| **Phương thức** | **Ý nghĩa** |
| put() | - Thêm các mục vào HashMap: *cars.put(0, “Vinfast”);* |
| get() | - Truy cập một giá trị thông qua khóa: *cars.get(0); // Vinfast* |
| containsKey() | - Kiểm tra một khóa có tồn tại không: *cars.containsKey(1);*  *// false* |
| remove() | - Xóa một mục thông qua khóa: *cars.remove(0);* |
| clear() | - Xóa tất cả các mục: *cars.clear();* |
| size() | - Trả về số mục: *cars.size();* |
| isEmpty() | - Trả về true nếu HashMap rỗng và ngược lại |
| keySet() | - Trả về một set chứa tất cả các khóa: *cars.keySet();* |
| values() | - Trả về một collection chứa tất cả các giá trị: *cars.keySet();* |

- Ưu điểm:

* Độ phức tạp của các thao tác với phần tử là O(1) trong trường hợp lý tưởng (không có va chạm băm).
* Phù hợp các bài toán cần thao tác nhiều trên dữ liệu.
* Có thể lưu trữ null key và null value

- Nhược điểm:

* Không duy trì thứ tự các phần tử.
* Không cho phép thêm các key trùng lặp. Nếu key đã tồn tại, giá trị cũ sẽ bị ghi đè.

1. **Generics**

- Generics: tham số hóa kiểu dữ liệu

- Ưu điểm:

* Tạo cấu trúc dữ liệu tổng quát
* Tính linh hoạt và khả năng tái sử dụng mã
* Đảm bảo tính an toàn về kiểu trong thời gian biên dịch (phát hiện lỗi về kiểu dữ liệu ngay khi chương trình được biên dịch)
* Không ảnh hưởng đến hiệu suất

1. **Generic class**

- Một lớp có thể được khai báo với một kiểu tham số, ví dụ:

*public class GenericClass<T> { }*

Trong đó: T đại điện cho một kiểu dữ liệu đối tượng bất kỳ. Hoặc có thể là một lý tự khác do người lập trình đặt tên

- Khi sử dụng generic class, cần xác định một kiểu tham số cụ thể, ví dụ:

*GenericClass<String> gc = new GenericClass<>();*

1. **Generic method**

- Các phương thức generics cũng có thể định nghĩa các tham số kiểu riêng biệt, ví dụ:

*public static <T> void printDataType(T inp) {*

*System.out.println(inp.getClass().getSimpleName());*

*}*

- Sử dụng:

*public static void main(String[] args) {*

*printDataType(“123”); //String*

*printDataType(123); // Integer*

*}*

1. **Giới hạn kiểu**

- Giới hạn kiểu hay bounded types được hiểu là giới hạn kiểu dữ liệu mà nó chấp nhận

- Ví dụ:

* *public class BoundedTypeClass<T extends Number> { }*
* *public static <T extends String> void boundedTypeMethod(T String) {}*

1. **Ký tự đại diện**

- Ký tự đại diện ? cho phép làm việc với các kiểu không xác định

- Đại diện cho bất kỳ kiểu nào: *public static void unbounded(List<?> list) {}*

- Đại diện cho kiểu T hoặc kiểu con của T:

*public static void upperBounded(List<? extends Number> list) {}*

- Đại diện cho kiểu T hoặc kiểu cha của T:

*public static void lowerBounded(List<? super Double> list) {}*