Session 13:

Tổng quan về Collection, List & Generic

NỘI DUNG

1. Biết, hiểu được về Collection Framework

2. Hiểu, áp dụng được List, ArrayList, LinkedList

3. Biết, hiểu được về Type Safety & Generic

4. Áp dụng được các kỹ thuật duyệt danh sách

1. Hạn chế của Array

Array là tập hợp cơ bản nhất trong Java, nhưng Array còn hạn chế gì khi sử dụng?

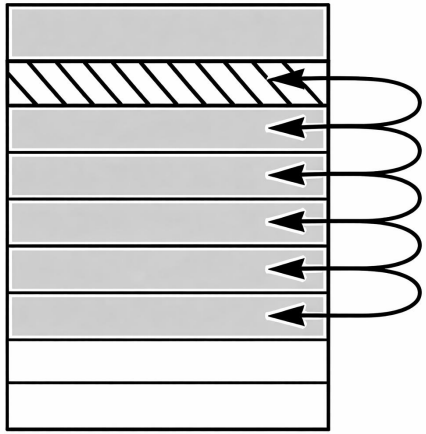
- **Kích thước cố định**, phải khai báo số lượng phần từ ngay từ đầu

  -> **khó khăn khi thêm phần tử mảng**

- **Các phần tử** được **đặt** và **tham chiếu liên tiếp** nhau **trong bộ nhớ**

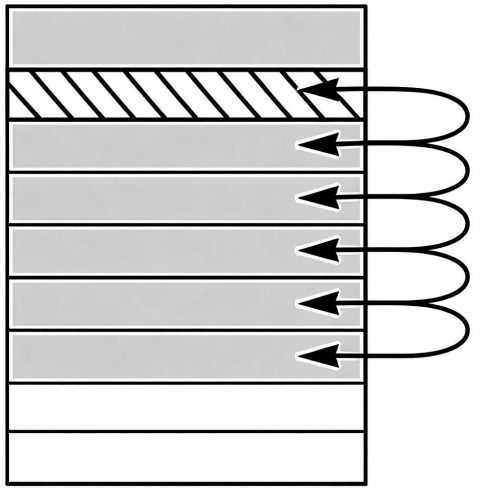
  -> **khó khăn khi xóa phần tử mảng**

- **Hiệu suất KHÔNG TỐT** khi sử dụng để **thêm / xóa phần tử mảng**



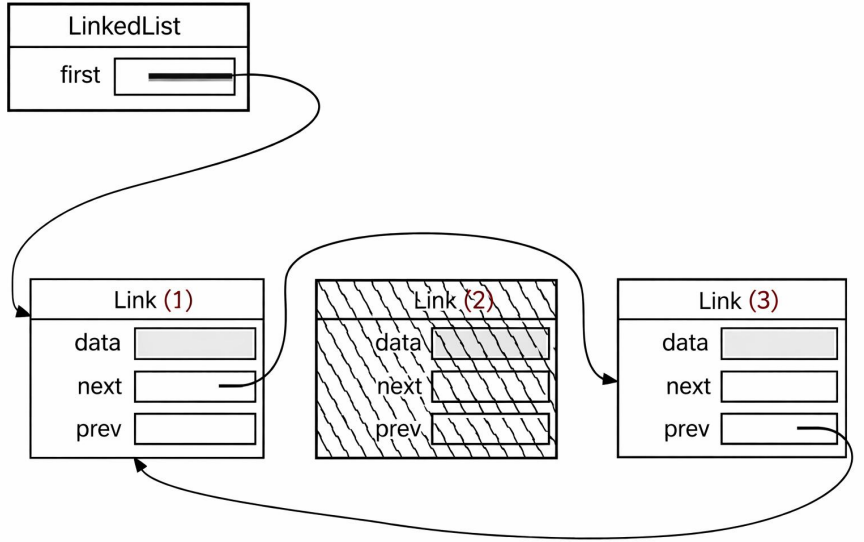
Khắc phục giới hạn kích thước với mảng

- Sử dụng **danh sách mảng động** (**ArrayList**) có thể **khắc phục nhược điểm** của **mảng về việc kích thước mảng cố định** -> **THÊM phần tử vào ArrayList mà không bị giới hạn kích thước**



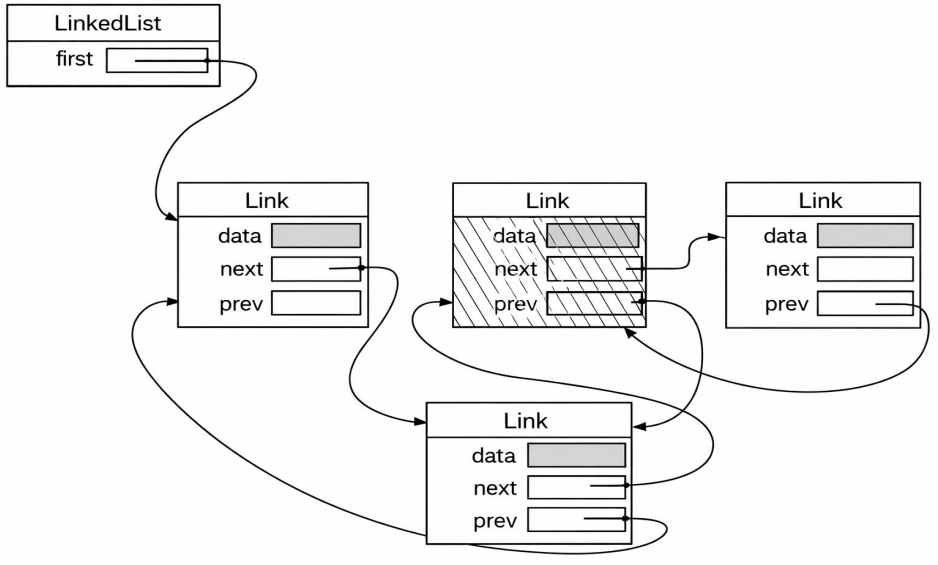
Khắc phục hạn chế xoá phần tử khỏi mảng

- Sử dụng **danh sách liên kết** (**LinkedList**) có thể **khắc phục nhược điểm** của **mảng khi XÓA phần tử** ra khỏi mảng -> **XÓA phần tử ra khỏi LinkedList nhanh hơn**



Khắc phục hạn chế thêm phần tử vào mảng

- Sử dụng **danh sách liên kết** (**LinkedList**) có thể **khắc phục nhược điểm** của **mảng khi THÊM phần tử** ra khỏi mảng -> **THÊM phần tử vào LinkedList nhanh hơn**



Sử dụng Collection Framework

Collection Framework trong Java sẽ khắc phục được các nhược điểm của Array

- **Danh sách mảng động** (**ArrayList**) là một cấu trúc dữ liệu thuộc **Collection Framework**
- **Danh sách liên kết** (**LinkedList**) là một cấu trúc dữ liệu thuộc **Collection Framework**
- **Java Collection Framework** còn **có nhiều cấu trúc dữ liệu thông dụng, hữu ích khác**

2. Khái niệm về Collection Framework

- **Collection Framework** (bộ khung làm việc với tập hợp của Java) là một kiến trúc thống nhất được cung cấp sẵn trong gói **java.util**
- **Collection** cung cấp các **Interface**, **Class** để lưu trữ và thao tác với một nhóm các đối tượng

So sánh giữa Array và Collection

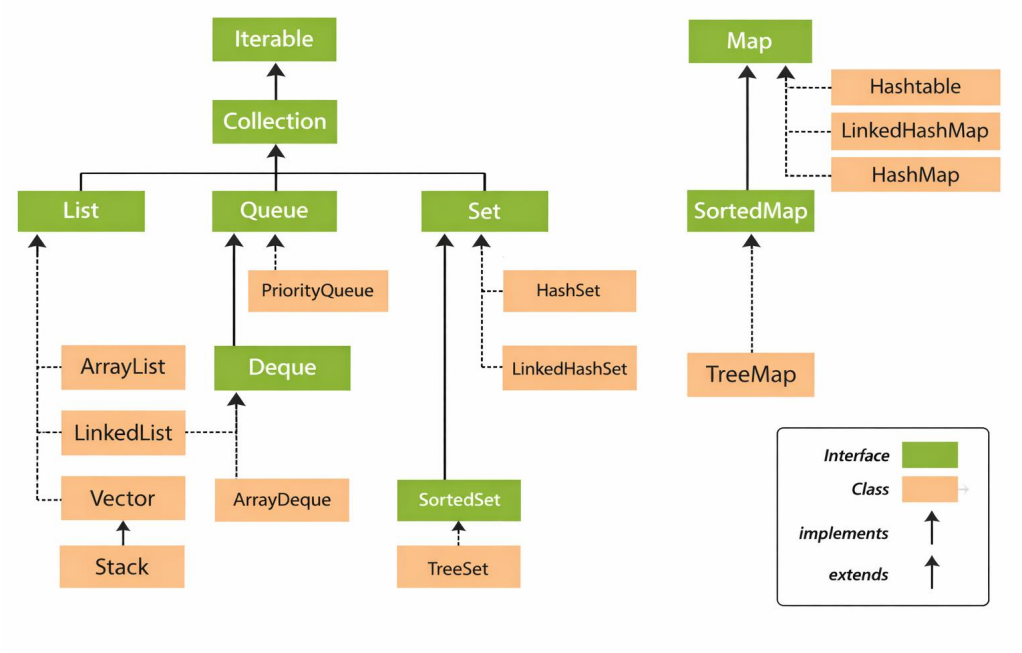
Đặc điểm	Mảng (Array)	Tập hợp (Collection)
Kích thước	Cố định (**Fixed Size**) Phải khai báo số lượng phần tử ngay từ đầu	Động (**Dynamic Size**). Tự động mở rộng hoặc thu hẹp khi thêm/xóa phần tử
Kiểu dữ liệu	Chứa được cả dữ liệu nguyên thủy (**int, double**) và đối tượng (**object**)	Chỉ chứa các đối tượng (**Objects**) Các kiểu nguyên thủy phải dùng qua **Wrapper Class** (Integer, Double)
Hiệu suất	Tốc độ truy xuất nhanh nhất	Phụ thuộc vào cấu trúc dữ liệu cụ thể (**List, Set, Map**,...)

Đặc điểm chính của Collection Framework

- **Lưu trữ đối tượng**: Java hỗ trợ cơ chế **Auto-Boxing**, các số nguyên thủy khi thêm vào Collection sẽ tự động chuyển thành đối tượng tương ứng.
- **Cung cấp sẵn các thuật toán**: Framework này tích hợp sẵn các thuật toán hiệu suất cao cho các tác vụ phổ biến như tìm kiếm (**Searching**), sắp xếp (**Sorting**), xáo trộn (**Shuffling**), đảo ngược (**Reversing**)
- **Hỗ trợ tính đa hình thông qua Interface**: Collection hỗ trợ việc khai báo biến thông qua Interface (**List, Set, Map**,...) nhưng khởi tạo bằng các lớp triển khai cụ thể (**ArrayList, HashSet, HashMap**,...) -> mã nguồn linh hoạt, dễ bảo trì hơn.

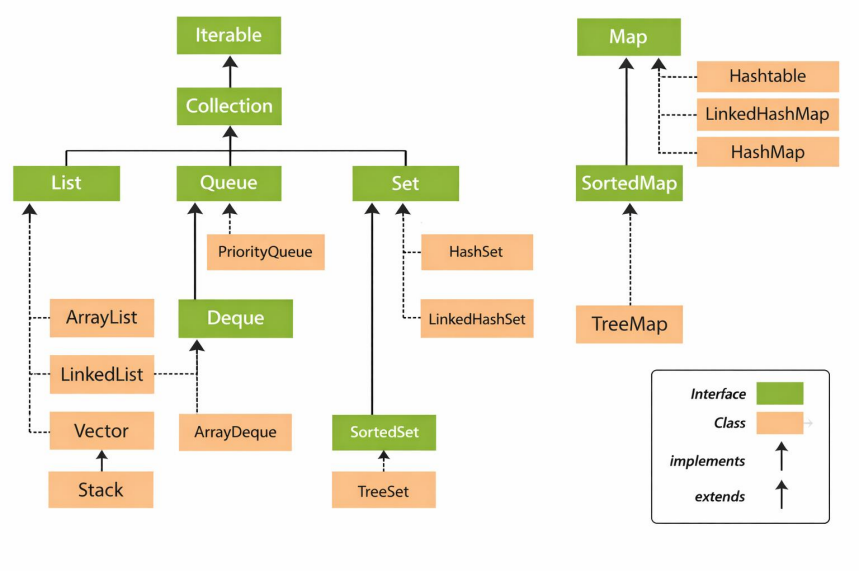
Phân loại và kiến trúc của Collection

Collection bao gồm (2 nhánh chính): Collection, Map



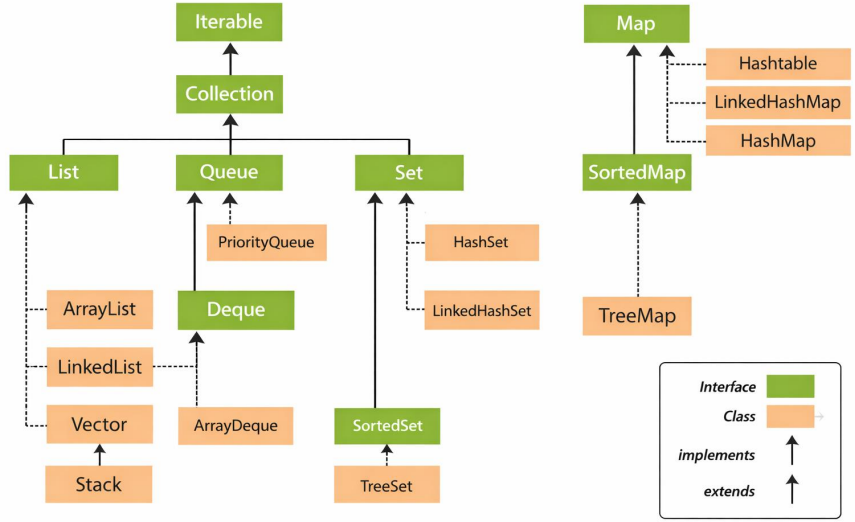
Nhánh Collection

- Nhánh **Collection** (java.util.Collection):
  - Là giao diện gốc (Root Interface) cho các tập hợp chứa các phần tử đơn lẻ.
  - Có 3 Interface con phổ biến:
    - **List** (Danh sách)
    - **Set** (Tập hợp)
    - **Queue** (Hàng đợi)



Nhánh Map

- Nhánh **Map** (java.util.Map):
  - **Map KHÔNG kế thừa** từ **interface Collection**
  - **Đặc điểm:**
    - Map lưu trữ dạng **Key - Value**
    - **Key** (**Khóa**) lưu duy nhất, không trùng lặp, **Value** (**Giá trị**) có thể trùng lặp
    - Mỗi **Key** ánh xạ đúng một **Value**
    - Đại diện: **HashMap, TreeMap, LinkedHashMap**



Lựa chọn cấu trúc dữ liệu phù hợp

- Việc hiểu rõ đặc điểm, phân loại **Collection** rất quan trọng để lựa chọn dùng cấu trúc dữ liệu phù hợp
- Cách lựa chọn, sử dụng các **Collection** phù hợp:
  - Cần quản lý **danh sách có thứ tự, truy xuất nhanh**

    -> Sử dụng **List**
  - Cần **lọc bỏ các phần tử trùng lặp trong danh sách**

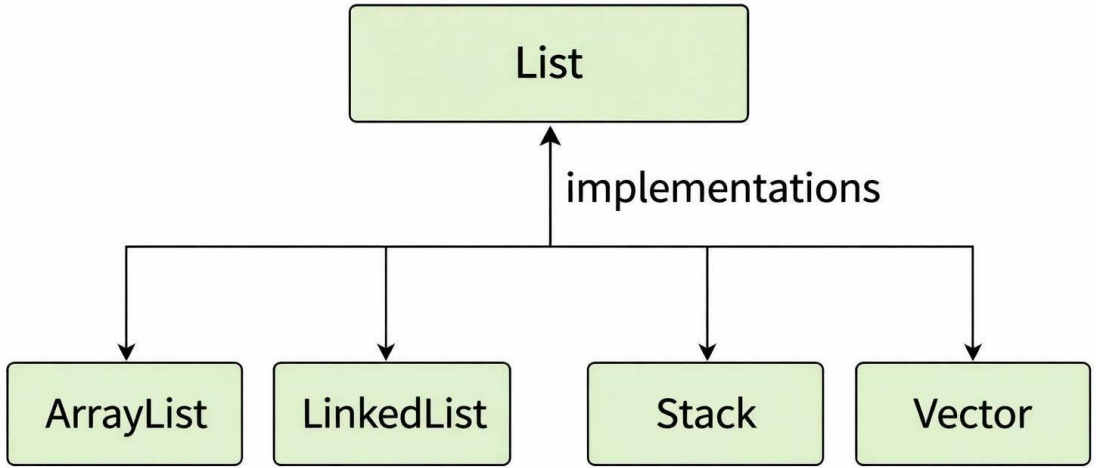
    -> Sử dụng **Set**
  - **Cần tra cứu dữ liệu theo từ khóa trong danh sách**

    -> Sử dụng **Map**

3. List, ArrayList & LinkedList

Khái niệm, đặc điểm:

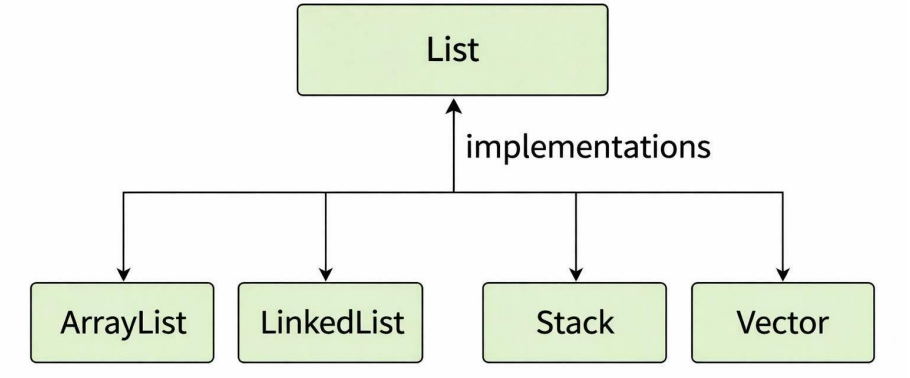
- **List** là một giao diện con (**Sub-interface**) kế thừa từ **interface Collection**.
- **List** đại diện cho một danh sách các phần tử được sắp xếp theo thứ tự (**Ordered-sequence**)



Giao diện List

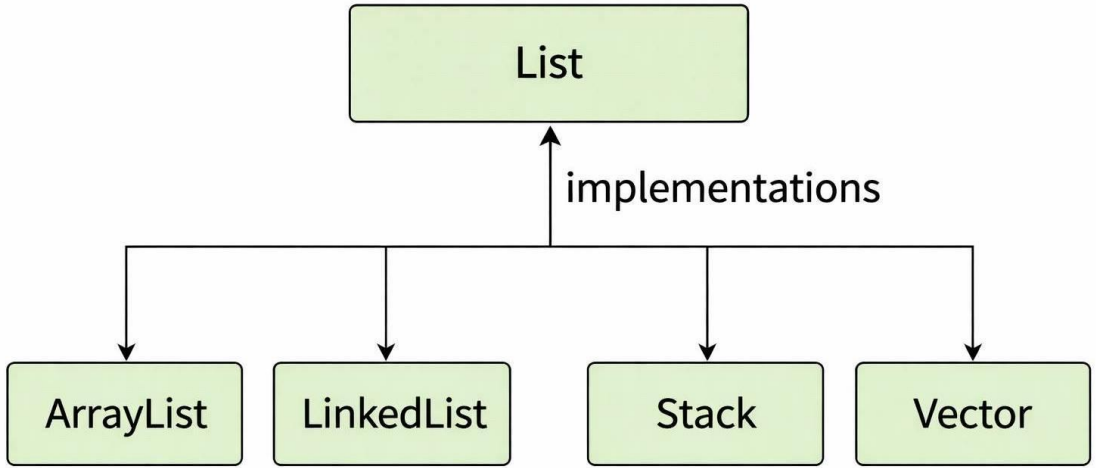
Khái niệm, đặc điểm:

- Ba đặc điểm cốt lõi của **List**:
  - Duy trì thứ tự chèn (**Insertion Order**)
  - Quản lý theo chỉ số (**Index-based**)
  - Chấp nhận các phần tử trùng lặp (**Duplicates Allowed**)



Các lớp triển khai phổ biến của List

- **List** là một **interface**, chúng ta **KHÔNG** thể khởi tạo đối tượng trực tiếp từ **List**
- **PHẢI sử dụng** các lớp triển khai cụ thể của List. 2 lớp phổ biến nhất là: **ArrayList, LinkedList**



So sánh giữa ArrayList và LinkedList

Đặc điểm	Mảng động (ArrayList)	Danh sách liên kết (LinkedList)
Cơ chế	Sử dụng mảng động (Dynamic Array) để lưu trữ phần tử	• Sử dụng cấu trúc danh sách liên kết đôi (Doubly Linked List) • Mỗi phần tử nắm địa chỉ của phần tử trước và sau nó
Ưu điểm	Tốc độ truy xuất phần tử (get) rất nhanh vì dựa trên chỉ số	Tốc độ thêm/xóa phần tử ở bất kỳ vị trí nào rất nhanh (chỉ cần thay đổi liên kết)
Nhược điểm	Tốc độ thêm/xóa phần tử ở giữa danh sách chậm (do phải dịch chuyển các phần tử còn lại)	Tốc độ truy xuất chậm hơn ArrayList (phải duyệt từ đầu/cuối danh sách)
Ứng dụng	Dùng khi nhu cầu chính là lưu trữ và tìm kiếm dữ liệu	Dùng khi ứng dụng cần thêm/xóa dữ liệu liên tục

Các phương thức quan trọng của List

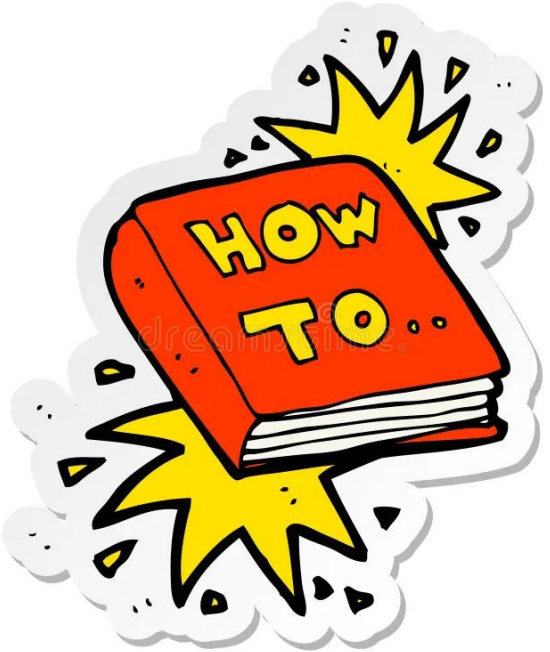
Nhóm chức năng	Phương thức	Mô tả
Thêm (Add)	boolean add(E e)	Thêm phần tử e vào cuối danh sách
	void add(int index, E e)	Chèn phần tử e vào vị trí index chỉ định
Truy xuất (Get)	E get(int index)	Lấy giá trị của phần tử tại vị trí index
Sửa (Update)	E set(int index, E element)	Thay thế phần tử tại index bằng giá trị element mới
Xóa (Remove)	E remove(int index)	Xóa phần tử tại vị trí index
	boolean remove(Object o)	Xóa phần tử đầu tiên có giá trị o tìm thấy trong danh sách
Kiểm tra	Boolean isEmpty()	Kiểm tra danh sách có rỗng hay không

Lựa chọn các lớp triển khai phù hợp cho List

- **Interface List** là công cụ cơ bản nhưng mạnh mẽ nhất để lưu trữ dữ liệu dạng tuyến tính.
- Lựa chọn, sử dụng các triển khai của **List** phù hợp:
  - Các trường hợp thông thường, **truy xuất nhiều**

    -> Sử dụng **ArrayList**
  - Khi cần thao tác theo tác **thêm/xóa phần tử ở đầu** hoặc **giữa danh sách** với **tần suất lớn**

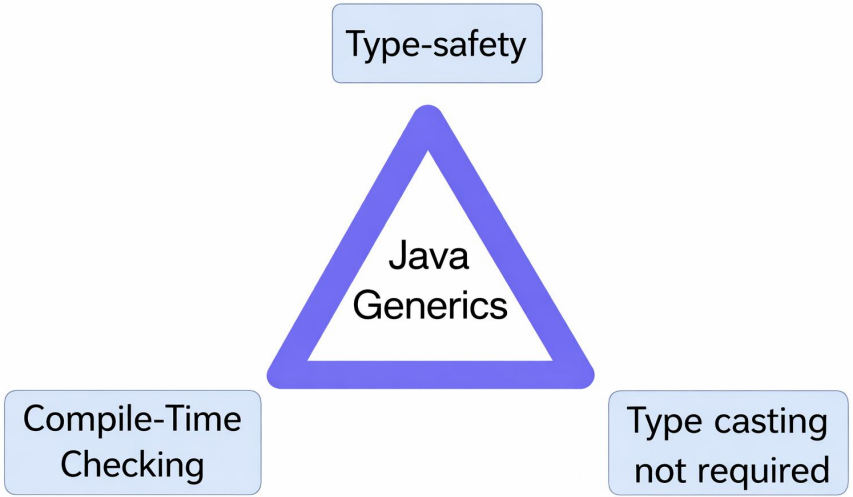
    -> Sử dụng **LinkedList**



4. Type Safety & Generic

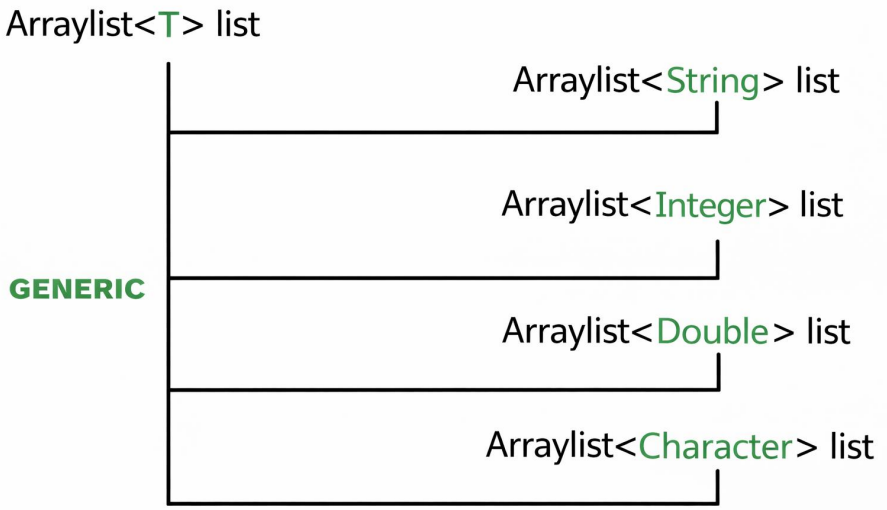
Type Safety trong Java

- Đảm bảo đúng kiểu dữ liệu trong quá trình biên dịch (**compile-time**)
- Nhằm tránh lỗi khi chương trình thực thi (**runtime**)
- Thể hiện mạnh nhất của **Type Safety** là **Generic + Collection**



Tổng quan về Generic

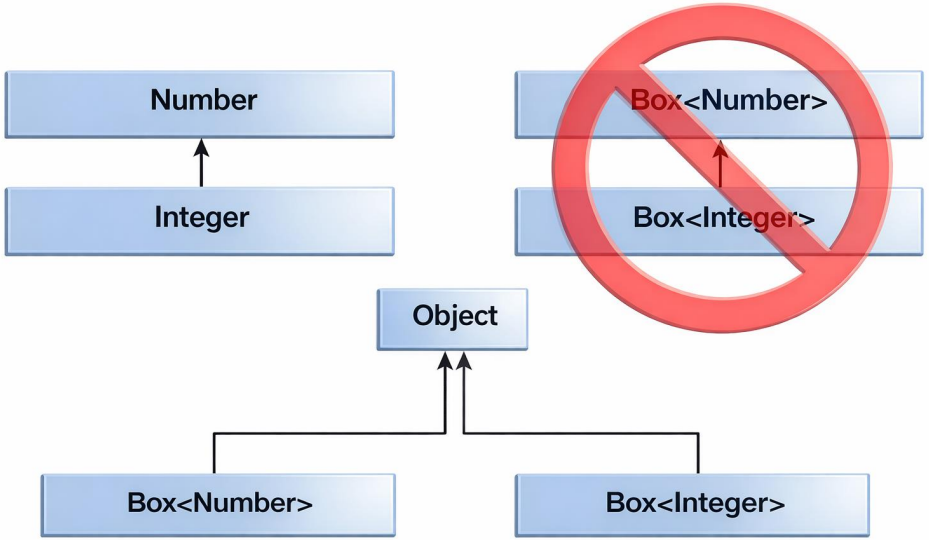
- **Generic** (**Tham số hóa kiểu dữ liệu**) là một tính năng quan trọng giới thiệu từ **Java 5**
- **Cơ chế cho phép chỉ định rõ kiểu dữ liệu** mà một **class**, **interface** hoặc **method sẽ thao tác** tại **thời điểm khai báo / khởi tạo**.
- Nâng cao tính an toàn của kiểu dữ liệu (**Type Safety**), tái sử dụng mã nguồn.



Vấn đề của Non-Generic (trước Java 5)

- Thiếu an toàn về kiểu dữ liệu
- Phải ép kiểu thủ công
- Dễ dẫn đến lỗi **ClassCastException**

=> Giải pháp: sử dụng Generic



Đặc điểm của giải pháp Generic

- **Generic** giải quyết triệt để các vấn đề của lập trình **Non-Generic**
- **Generic** yêu cầu xác định kiểu dữ liệu cụ thể thông qua cặp dấu ngoặc nhọn **<>**
- **Cú pháp**:

```
ClassName<Type> objectName = new ClassName<();
```

Quy ước đặt tên tham số trong Generic

- **Các ký tự in hoa**, **đơn** thường được **sử dụng để đại diện cho các tham số kiểu**
- **Đây là quy ước chuẩn** để **phân biệt** giữa **tham số kiểu** và **tên lớp thông thường**:
  - **E** (Element)
  - **K** (key)
  - **V** (Value)

Xây dựng lớp Generic tùy chỉnh

- Lập trình viên có thể tự định nghĩa một lớp **Generic** để xử lý dữ liệu linh hoạt
- **Cú pháp**:

```
public class ClassName<T>  {
        private T item;
        public T getItemInfo() {
          return  this.item;
        }
}
```

5. Các kỹ thuật duyệt danh sách

- Là **hành động đi qua lần lượt từng phần tử trong danh sách** để **thực hiện một tác vụ nào đó**
- Có **4 kỹ thuật** phổ biến nhất để **duyệt danh sách**:
  - Vòng lặp for dựa trên chỉ số (**Classic For Loop**)
  - Vòng lặp For-each (**Enhanced For Loop**)
  - Sử dụng Iterator (**Interface con trỏ duyệt**)
  - Sử dụng ListIterator (**Con trỏ duyệt danh sách 2 chiều**)

So sánh các kỹ thuật duyệt danh sách

Tiêu chí	Classic For (For-i)	For-each	Iterator	ListIterator
Cần truy cập Index	Có	Không	Không	Có thể lấy index
Khả năng thay đổi	Chỉ sửa (set), hạn chế xóa	Chỉ đọc (Read-only)	Có thể xóa (remove)	Thêm, sửa, xóa
Hướng duyệt	Xuôi & ngược	Chỉ duyệt xuôi	Chỉ duyệt xuôi	Xuôi & ngược
An toàn với Thread	Thấp (dễ lỗi index)	Trung bình	Cao (An toàn khi xóa)	Cao
Độ phức tạp code	Trung bình	Thấp (Đơn giản nhất)	Cao	Cao

Cách sử dụng các kỹ thuật duyệt mảng

- Sử dụng **For-each** cho **90% các trường hợp thông thường** (chỉ cần hiển thị hoặc tính toán)
- Sử dụng **Iterator** nếu bạn **cần lọc bỏ (xóa) phần tử khi đang duyệt**
- Sử dụng **Classic For** nếu bạn **cần thao tác dựa trên chỉ số** (ví dụ: get(i))
- Sử dụng **ListIterator** nếu bạn **cần duyệt ngược hoặc chèn/sửa dữ liệu phức tạp**

TỔNG KẾT

- Biết được hạn chế của Array và hiểu tại sao nên sử dụng Collection
- Nắm được các đặc điểm chính của Collection Framework
- Phân loại được Collection Framework có 2 nhánh chính là Collection và Map
- Biết được cách sử dụng các loại Collection phù hợp cho từng bài toán
- Nắm được các đặc điểm chính của List, cách thức hoạt động của List
- So sánh được điểm khác nhau giữa ArrayList và LinkedList
- Biết được lúc nào nên sử dụng ArrayList, khi nào nên sử dụng LinkedList
- Biết được về khái niệm Type Safety và hiểu được về giải pháp Generic
- Áp dụng được các kỹ thuật duyệt danh sách thông dụng trong Java

RIKEI Academy®
where the dream begins

HỌC VIỆN ĐÀO TẠO LẬP TRÌNH CHẤT LƯỢNG NHẬT BẢN