VIETNAM AUSTRALIA Vocational College

Slide-1.1: Tổng quan Collection, List, ArrayList, LinkedList

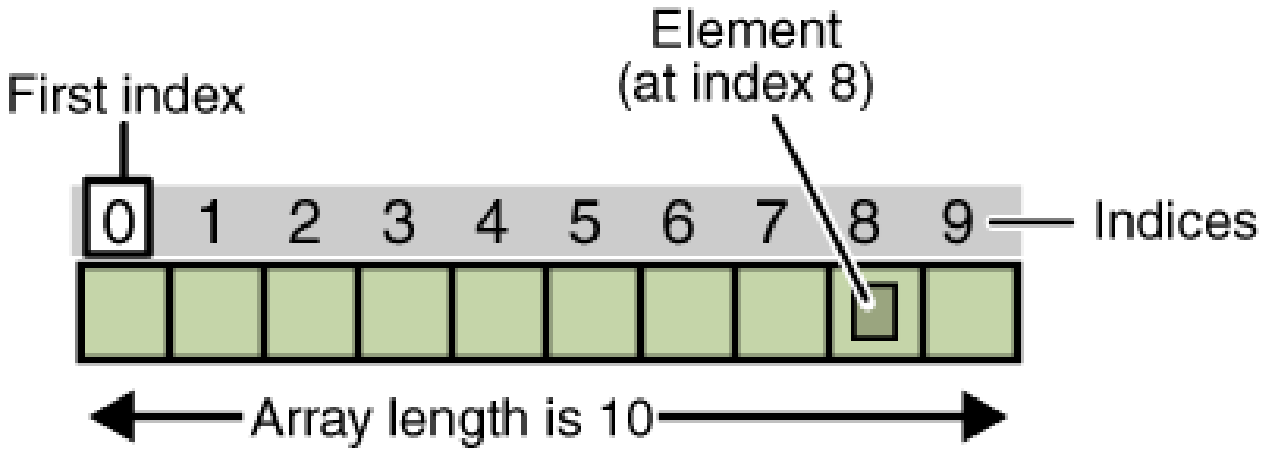
Giảng viên: Nguyễn Bá Minh Đạo

Nội dung

1. Tại sao nên sử dụng Collection
2. Hệ thống phân cấp của Collection
3. List, Iterator, ListIterator Interfaces
4. Giới thiệu lớp ArrayList trong Java
5. So sánh Array và ArrayList
6. Chuyển đổi Array sang ArrayList
7. Chuyển đổi ArrayList sang Array
8. Loại bỏ phần tử trùng lặp trong ArrayList
9. Giới thiệu lớp LinkedList trong Java
10. So sánh ArrayList và LinkedList

Tại sao nên sử dụng Java Collection?

Giới thiệu



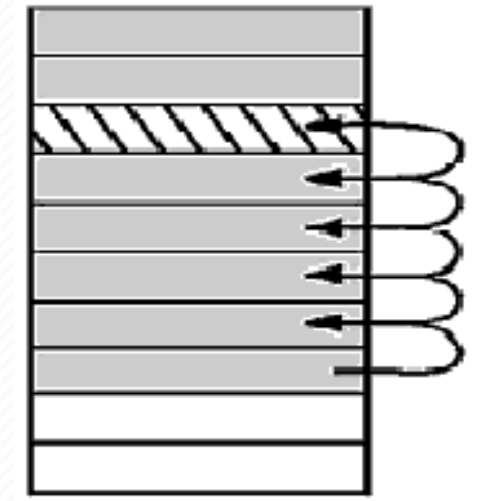
- Như các ngôn ngữ lập trình khác, **Java** hỗ trợ mảng (array) như một tập hợp cơ bản nhất, nhưng làm việc với mảng không thuận tiên trong nhiều trường hợp vì:
  - Thời gian sống của mảng việc tăng thêm phần tử hoặc xóa các phần tử của mảng rất khó khăn
  - Phải trả giá đắt về hiệu năng chương trình nếu cố tình làm điều đó

Tại sao nên sử dụng Java Collection?

Các giới hạn của việc sử dụng mảng (Array)

- **Mảng** rất cơ bản và quen thuộc:
  - Lưu trữ các kiểu tham chiếu (object), các kiểu nguyên thủy (primitivate type)
  - int[] myArray = new int[]{1,4,3}
  - Object[] myArrayObj = new Object[] {“Object”, new Integer(100)};
- **Mảng** có kích cỡ và số chiều cố định
  - Khó khăn cho việc mở rộng mảng ra khỏi mảng
- **Các phần tử** được đặt và tham chiếu liên tiếp nhau trong bộ nhớ
  - Khó khăn cho việc xóa một phần tử ra khỏi mảng

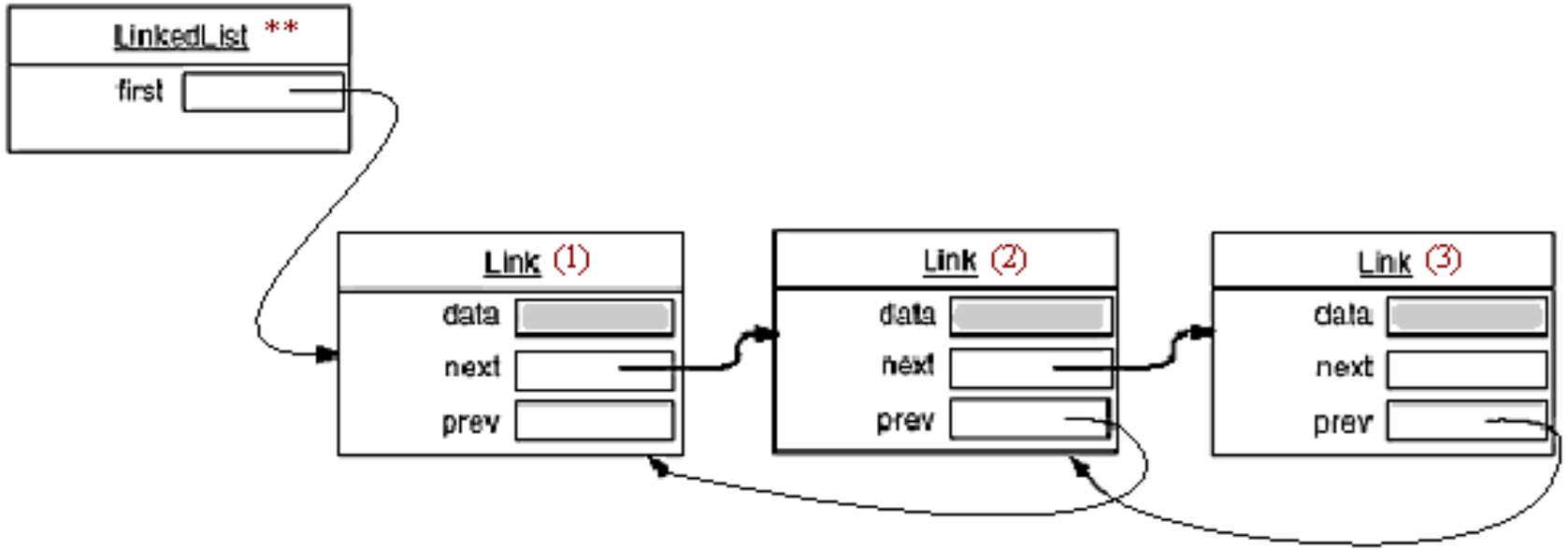
=> **Mảng** không phải là một cách tốt nhất cho nhiều trường hợp



Tại sao nên sử dụng Java Collection?

Danh sách có kết nối (Linked List)

- Danh sách có kết nối (**Linked List**) là một trong các cách quản lý danh sách dữ liệu khắc phục được các nhược điểm của mảng.
- Tất nhiên để quản lý danh sách trong Java có nhiều cách khác ví dụ **ArrayList**.



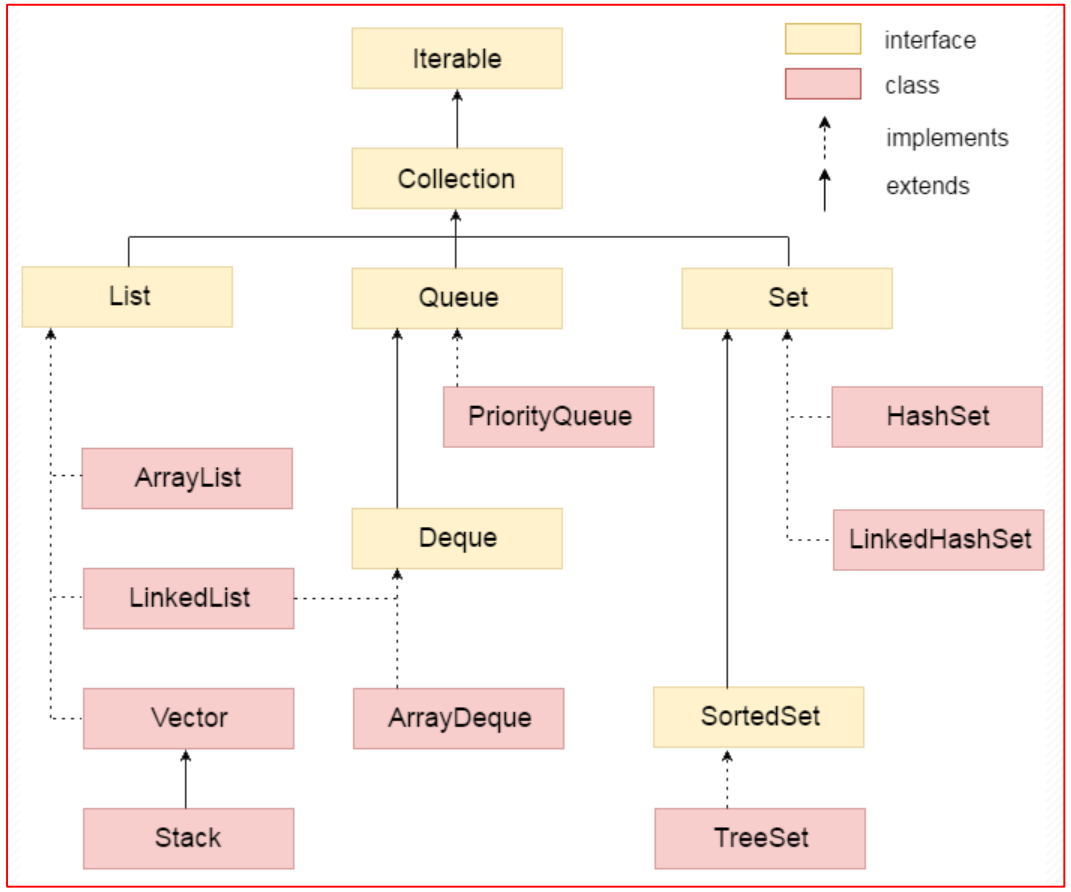
Hệ thống phân cấp của Collection

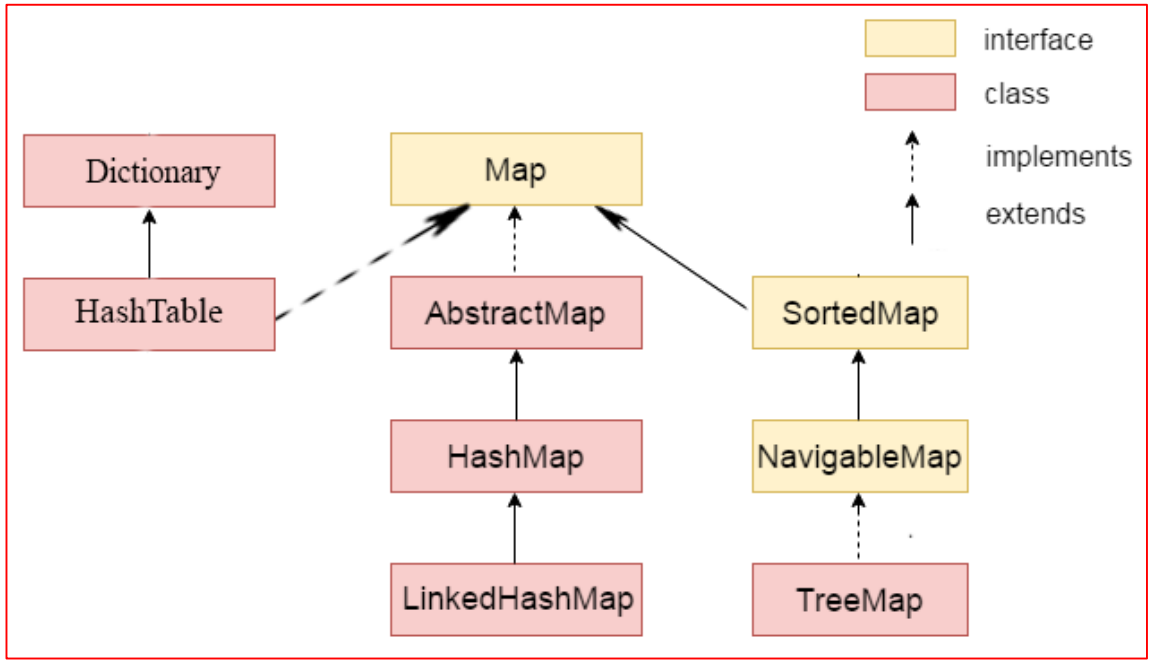
Collection Framework

- Các kiểu tập hợp của **Collection Framework** được xây dựng trên cơ sở một số interface trong package java.util.
- Được phân chia ra làm 2 hệ thống phân cấp dẫn đầu bởi 2 interface:
  - Interface java.util.Collection chứa danh sách các đối tượng
  - Interface java.util.Map chứa các cặp key/value

Hệ thống phân cấp của Collection

Collection Framework



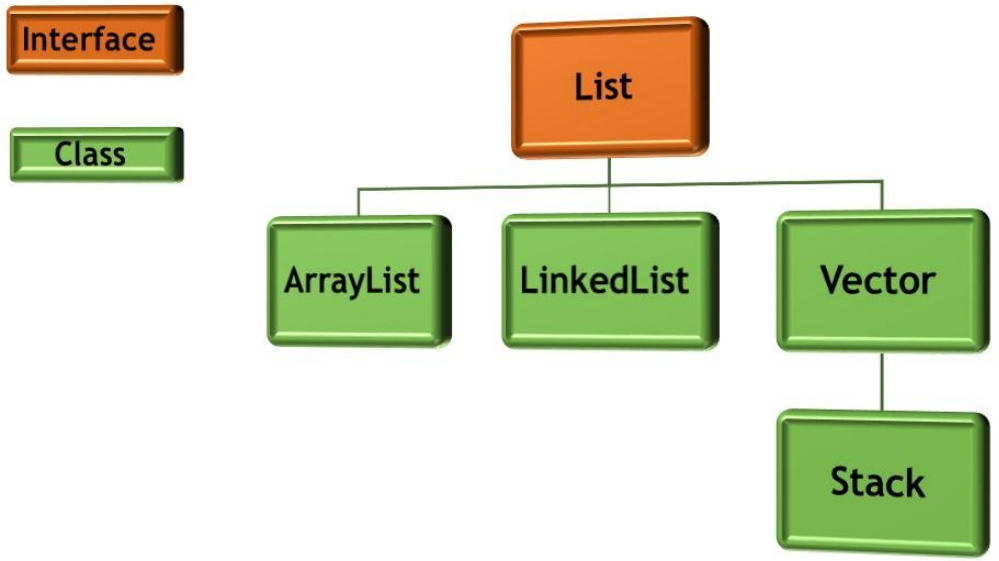


List, Iterator, ListIterator Interfaces

List Interface

- **List** Interface trong Java kế thừa **Collection** và nó cung cấp các phương thức để thao tác với các phần tử trong danh sách.
- Interface **java.util.List** được định nghĩa như sau:

```
public interface List<E> extends Collection<E>
```



List, Iterator, ListIterator Interfaces

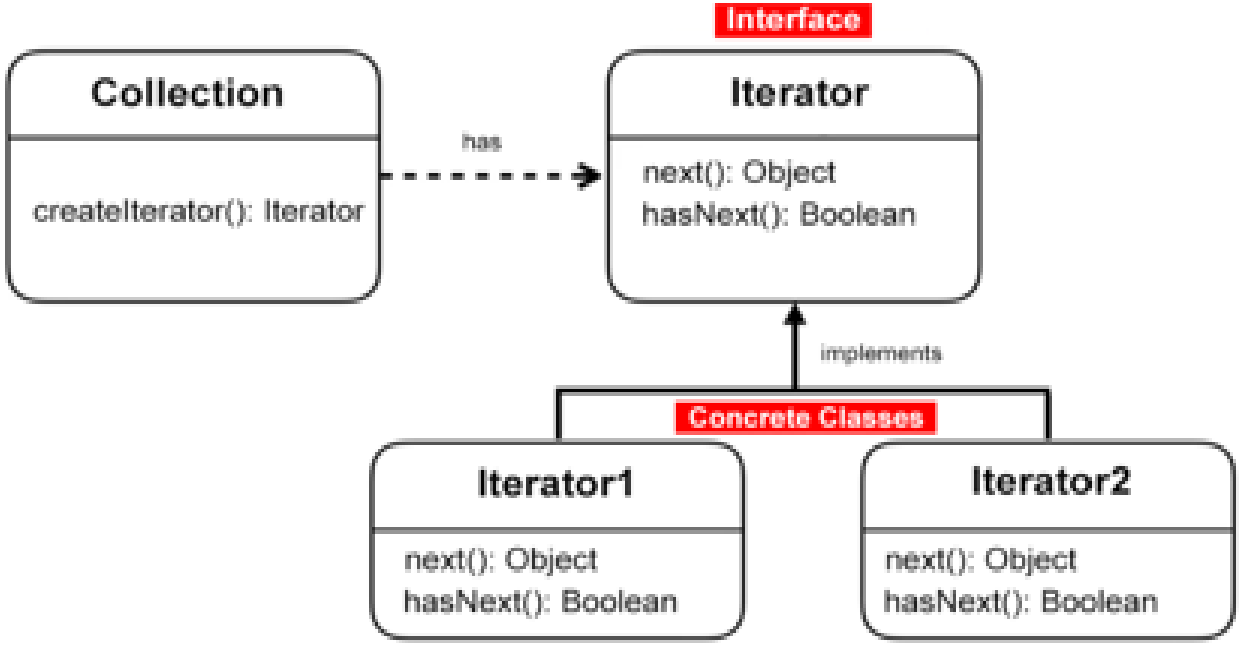
Ví dụ khai báo List Interface và lớp hiện thực ArrayList:

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

public class ListExample {
    public static void main(String[] args) {
        // Create list with no parameter
        List<String> list = new ArrayList<>();
        for (int i = 1; i <= 5; i++) {
            // Add element to list
            list.add("0" + i);
        }

        // Show list through for-each
        for (String item : list) {
            System.out.print(item + " ");
        }
    }
}
```

List, Iterator, ListIterator Interfaces

Iterator Interface

- **Iterator** Interface trong Java kế thừa **Collection** và nó cung cấp các phương thức để duyệt qua các phần tử của bất kỳ tập hợp nào.
- Mỗi Interface **Collection** trong Java đều chứa một phương thức **iterator** để trả về một thực thể của interface **Iterator**.
- **Iterator** có khả năng xóa những phần tử từ tập hợp trong quá trình lặp. Sử dụng for hoặc for-each, bạn không thể xóa phần tử trong khi duyệt các phần tử.



List, Iterator, ListIterator Interfaces

Iterator Interface

- Interface **java.util.Iterator** được định nghĩa như sau: **public interface** Iterator<E>
- Ví dụ không thể duyệt và xóa phần tử với **for**:

```
public class IteratorExample01 {
    public static void main(String[] args) {
        // Create list with no parameter
        List<String> list = new ArrayList<>();
        for (int i = 1; i <= 5; i++) {
            // Add element to list
            list.add("0" + i);
        }

        // traversing the list in the
        // forward direction
        for (int i = 0; i < 5; i++) {
            list.remove(i);
        }
    }
}
```

Thực thi chương trình trên, bạn sẽ gặp lỗi **IndexOutOfBounds Exception** như sau:

```
Exception in thread "main" java.lang.IndexOutOfBoundsException:
Index 3 out of bounds for length 2
at ...java.base/java.util.Objects.checkIndex(Objects.java:359)
at java.base/java.util.ArrayList.remove(ArrayList.java:504)
at com.vu.IteratorExample01.main(IteratorExample01.java:17)
```

List, Iterator, ListIterator Interfaces

Iterator Interface

- Interface **java.util.Iterator** được định nghĩa như sau: **public interface** Iterator<E>
- Ví dụ không thể duyệt và xóa phần tử với **for**:

```
public class IteratorExample02 {
    public static void main(String[] args) {
        // Create list with no parameter
        List<String> list = new ArrayList<>();
        for (int i = 1; i <= 5; i++) {
            // Add element to list
            list.add("0" + i);
        }

        // traversing the list in
        // the forward direction
        for (String item : list) {
            list.remove(item);
        }
    }
}
```

Thực thi chương trình trên, bạn sẽ gặp lỗi **ConcurrentModificationException** như sau:

```
Exception in thread "main" java.util.ConcurrentModificationException
at java.base/java.util.ArrayList$Itr.checkForComodification(ArrayList.java:1013)
at java.base/java.util.ArrayList$Itr.next(ArrayList.java:967)
at com.vu.IteratorExample02.main(IteratorExample02.java:16)
```

List, Iterator, ListIterator Interfaces

Iterator Interface

- Interface **java.util.Iterator** được định nghĩa như sau: **public interface** Iterator<E>
- Ví dụ có thể duyệt từ cuối về đầu và xóa phần tử với **for**:

```
public class IteratorExample03 {
    public static void main(String[] args) {
        // Create list with no parameter
        List<String> list = new ArrayList<>();
        for (int i = 1; i <= 5; i++) {
            // Add element to list list.add("0" + i); }

            // traversing the list in the backward direction
            // for (int i = 4; i >= 0; i--) {
            list.remove(i);
        }
        System.out.println("list.isEmpty() = " + list.isEmpty());
    }
}
```

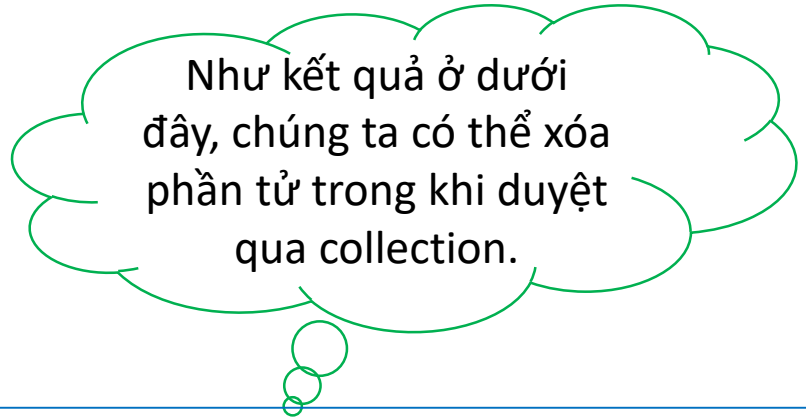
Thực thi chương trình trên, bạn sẽ ta kết quả:
list.isEmpty() = **true**

List, Iterator, ListIterator Interfaces

Iterator Interface

- Interface **java.util.Iterator** được định nghĩa như sau: **public interface** Iterator<E>
- Ví dụ có thể sử dụng Iterator để duyệt và xóa phần tử khi duyệt:

```
public class IteratorExample04 {
    public static void main(String[] args) {
        // Create list with no parameter
        List<String> list = new ArrayList<>();
        for (int i = 1; i <= 5; i++) {
            // Add element to list
            list.add("0" + i);
        }

        Iterator<String> iterator = list.iterator();
        System.out.println("Using Iterator: Only Traversing the list "
                + "in the forward direction ");
        while (iterator.hasNext()) {
            System.out.print(iterator.next() + " ");
            iterator.remove();
        }
        System.out.println();
        System.out.println("list.isEmpty() = " + list.isEmpty());
    }
}
```



Using Iterator: Only Traversing the list in the forward direction

01 02 03 04 05

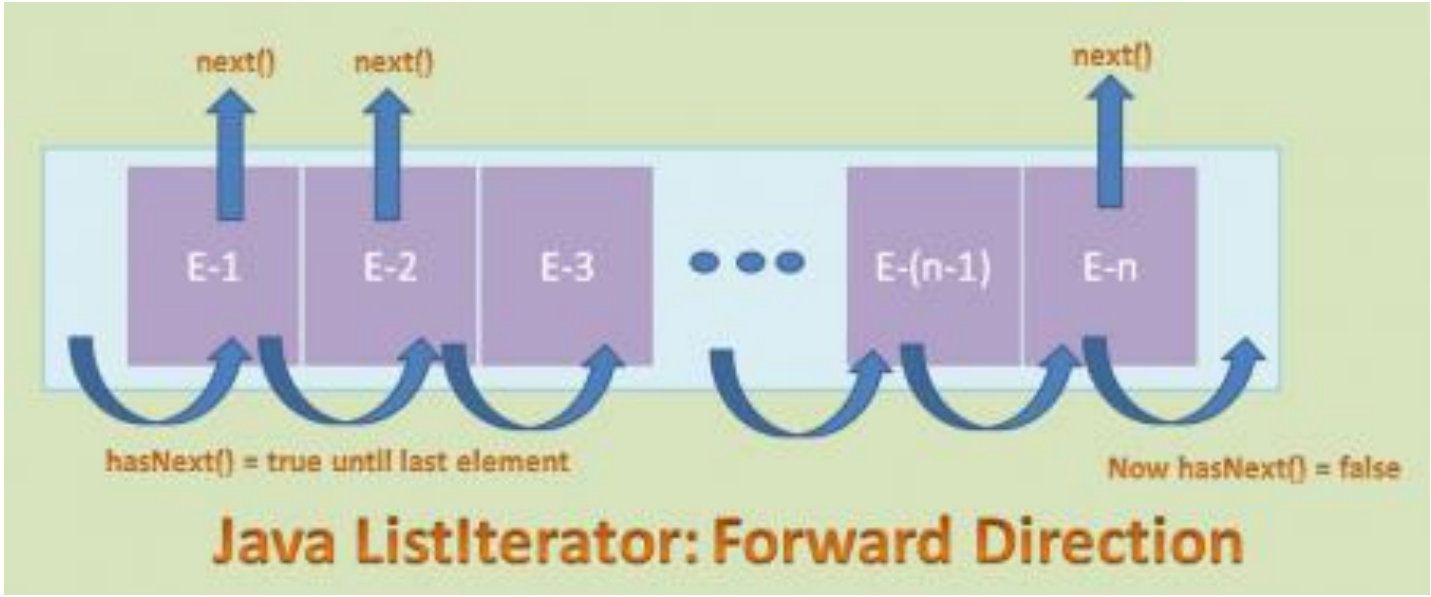
list.isEmpty() = **true**

List, Iterator, ListIterator Interfaces

ListIterator Interface

- **ListIterator** Interface là một interface kế thừa từ lớp **Iterator**, nó cung cấp phương thức để duyệt các phần tử của List trong java theo cả 2 chiều.
- Interface **java.util.Iterator** được định nghĩa như sau:

```
public interface ListIterator<E> extends Iterator<E>
```



List, Iterator, ListIterator Interfaces

ListIterator Interface

❑ Các phương thức của **ListIterator** Interface:

Phương thức	Mô tả
boolean hasNext()	Trả về true nếu list interator có tồn tại phần tử kế tiếp phần tử hiện tại.
Object next()	Trả về phần tử kế tiếp trong danh sách và vị trí con trỏ tăng lên 1.
boolean hasPrevious()	Trả về true nếu list interator có tồn tại phần tử kế sau phần tử hiện tại.
Object previous()	Trả về phần tử kế sau trong danh sách và vị trí con trỏ giảm đi 1

List, Iterator, ListIterator Interfaces

Ví dụ khai báo ListIterator Interface trong Java:

```
public static void main(String[] args) {
    // Create list with no parameter
    List<String> list = new ArrayList<>();
    for (int i = 1; i <= 5; i++) {
        // Add element to list
        list.add("0" + i);
    }

    Iterator<String> iterator = list.iterator();
    System.out.println("Using Iterator: Only Traversing the list in the forward direction ");
    while (iterator.hasNext()) {
        System.out.print(iterator.next() + " ");
    }

    System.out.println();
    ListIterator<String> itr = list.listIterator();
    System.out.println("Using ListIterator: Traversing the list in the forward direction ");
    while (itr.hasNext()) {
        System.out.print(itr.next() + " ");
    }

    System.out.println();
    System.out.println("Using ListIterator: Traversing the list in the reverse direction ");
    while (itr.hasPrevious()) {
        System.out.print(itr.previous() + " ");
    }
}
```

List, Iterator, ListIterator Interfaces

So sánh Iterator Interface và ListIterator Interface

Iterator	ListIterator
Iterator được sử dụng để duyệt qua các interface Set và List.	ListIterator chỉ được sử dụng để duyệt qua List.
Iterator có thể duyệt qua các phần tử trong tập hợp chỉ theo một hướng.	ListIterator có thể duyệt qua List tất cả các hướng (2 hướng).
Không thể thêm, thay thể phần tử khi sử dụng Iterator.	ListIterator cài đặt interface Iterator và chứa thêm một số chức năng mới, như thêm mới một phần tử, thay thế phần từ, lấy chỉ mục (index) cho phần từ kế trước hay sau nó, etc.

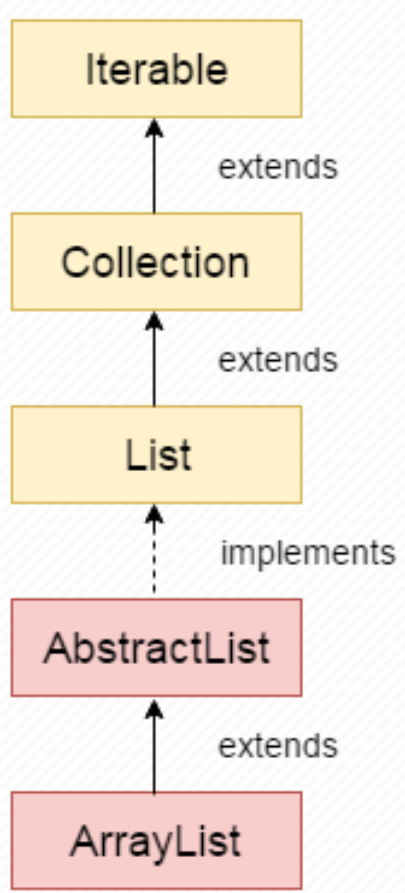
Giới thiệu lớp ArrayList trong Java

Giới thiệu ArrayList

- **Lớp ArrayList** trong Java được sử dụng như một mảng động để lưu trữ các phần tử. **ArrayList** extends từ lớp **AbstractList** và implements từ **List Interface**.

- Các **Array** chuẩn có độ dài cố định. Sau khi các **Array** được tạo, chúng không thể tăng hoặc giảm kích cỡ, nghĩa là bạn phải có bao nhiêu phần tử mà một mảng sẽ giữ.

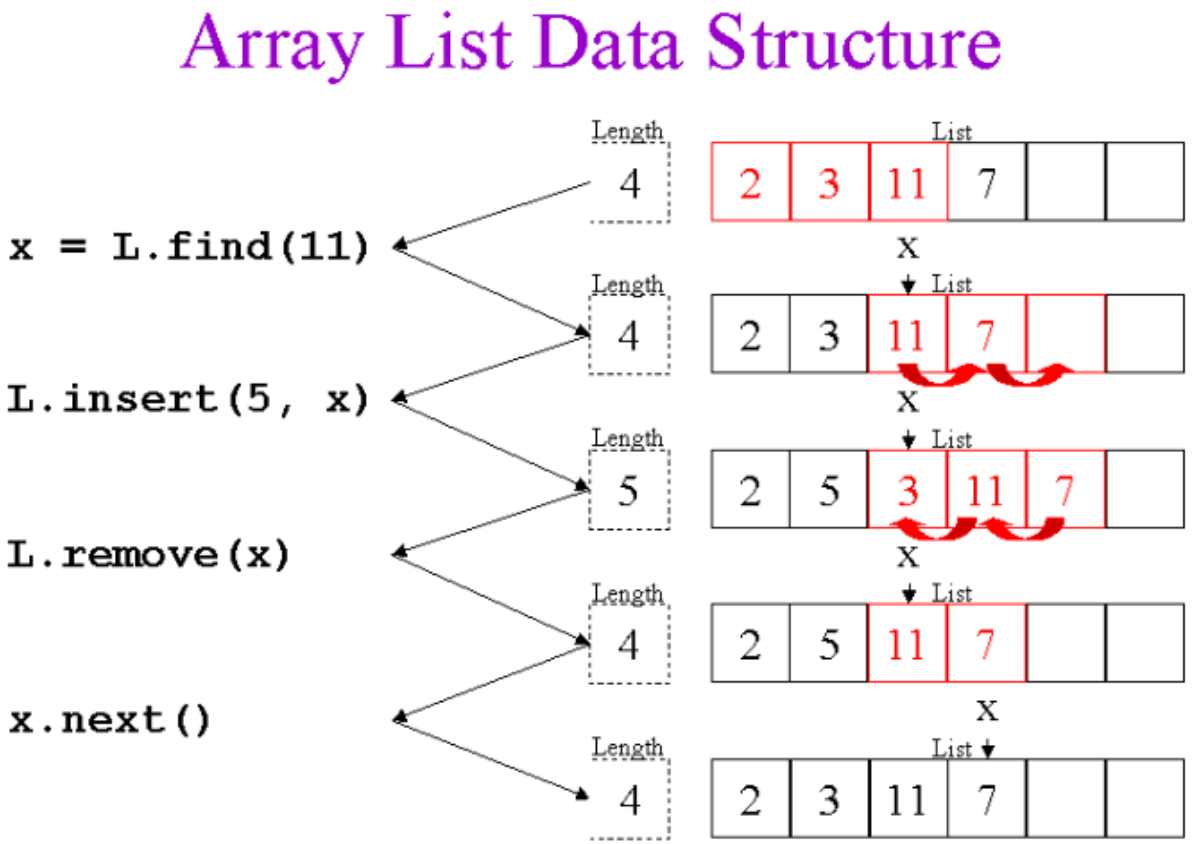
- **ArrayList** được tạo với một kích cỡ ban đầu. Khi kích cỡ này bị vượt, **Collection** tự động được tăng. Khi các đối tượng bị gỡ bỏ, **ArrayList** có thể bị giảm kích cỡ.



Giới thiệu lớp ArrayList trong Java

Các đặc điểm của lớp ArrayList

- **Có thể** chứa các phần tử trùng lặp.
- **Duy trì** thứ tự của phần tử được thêm vào.
- **Không đồng bộ** (non-synchronized)
- **Cho phép** truy cập ngẫu nhiên, tốc độ truy xuất (get) phần tử nhanh vì nó lưu trữ dữ liệu theo chỉ mục.
- **Thao tác** thêm/xóa (add/remove) phần tử chậm vì cần nhiều sự dịch chuyển nếu bất kỳ phần tử nào thêm/xóa khỏi danh sách



Giới thiệu lớp ArrayList trong Java

Các đặc điểm của lớp ArrayList

❑ Lớp **java.util.ArrayList** được định nghĩa như sau:

```
public class ArrayList<E> extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable, java.io.Serializable
```

Các phương thức khởi tạo (constructor) của lớp ArrayList

Phương thức	Mô tả
ArrayList()	khởi tạo một danh sách mảng trống
ArrayList(Collection c)	khởi tạo một danh sách mảng với các phần tử của collection c
ArrayList(int capacity)	khởi tạo một danh sách mảng mà có sức chứa (compacity) ban đầu được chỉ định. Nếu không chỉ định, mặc định là **10**. Mỗi lần thêm một phần tử vào danh sách, nếu vượt quá sức chứa cho phép thì danh sách sẽ tự động tăng thêm **50%** kích thước hiện có.

Giới thiệu lớp ArrayList trong Java

Ví dụ sử dụng ArrayList với kiểu dữ liệu cơ bản (Wrapper) trong Java:

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

public class ArrayListExample01 {

    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {

    public static void printData(List<String> list) {
        // Show list through for-each
        for (String item : list) {
            System.out.print(item + " ");
        }
        System.out.println();
    }

}
```

Giới thiệu lớp ArrayList trong Java

Ví dụ sử dụng ArrayList với kiểu dữ liệu cơ bản (trong main):

```
// Create list with no parameter
List<String> list1 = new ArrayList<>();
for (int i = 1; i <= NUM_OF_ELEMENT; i++) {
    // Add element to list
    list1.add("0" + i);
}
System.out.print("Init list1 = ");
printData(list1);

// addAll()
List<String> list2 = new ArrayList<>();
list2.addAll(list1);
list2.add("0" + 3);
System.out.print("After list1.addAll(03): list2 = ");
printData(list2);

// IndexOf()
System.out.println("list2.indexOf(03) = " + list2.indexOf("03"));
System.out.println("list2.indexOf(06) = " + list2.indexOf("06"));

// lastIndexOf()
System.out.println("list2.lastIndexOf(03) = " + list2.lastIndexOf("03"));

// Remove
list2.remove("01");
System.out.print("After list2.remove(01): list2 = ");
printData(list2);
```

```
// retainAll()
List<String> list4 = new ArrayList<>(list1);
ArrayList<String> list3 = new ArrayList<String>();
list3.add("0" + 3);
list3.add("0" + 2);
System.out.print("Init list3 = ");
printData(list3);
list4.retainAll(list3);
System.out.print("After list1.retainAll(list3): list4 = ");
printData(list4);

// removeAll()
List<String> list5 = new ArrayList<>(list1);
list5.removeAll(list3);
System.out.print("After list1.removeAll(list3): list5 = ");
printData(list5);
```

```
Init list1 = 01 02 03 04 05
After list1.addAll(03): list2 = 01 02 03 04 05 03
list2.indexOf(03) = 2
list2.indexOf(06) = -1
list2.lastIndexOf(03) = 5
After list2.remove(01): list2 = 02 03 04 05 03
Init list3 = 03 02
After list1.retainAll(list3): list4 = 02 03
After list1.removeAll(list3): list5 = 01 04 05
```

Giới thiệu lớp ArrayList trong Java

Ví dụ sử dụng ArrayList với kiểu dữ liệu người dùng tự định nghĩa (Object):

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

class Student {
    private int id;
    private String name;

    public Student(int id, String name) {

    public String toString() {
}

public class ArrayListExample02 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
}
```

Giới thiệu lớp ArrayList trong Java

Ví dụ sử dụng ArrayList với kiểu dữ liệu người dùng tự định nghĩa (Object):

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

class Student {
    private int id;
    private String name;

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

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + "]";
    }
}
```

```java
public class ArrayListExample02 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create list with compacity = 3
        List<Student> students = new ArrayList<>(3);
        for (int i = 1; i <= NUM_OF_ELEMENT; i++) {
            // Add element to list
            Student student = new Student(i, "myname" + i);
            students.add(student);
        }

        // Show list student
        for (Student student : students) {
            System.out.println(student);
        }
    }
}
```

```
Student [id=1, name=myname1]
Student [id=2, name=myname2]
Student [id=3, name=myname3]
Student [id=4, name=myname4]
Student [id=5, name=myname5]
```

So sánh Array và ArrayList

Khác nhau giữa Array và ArrayList

Array	ArrayList
Array có kích thước cố định. Không thể thay đổi kích thước nếu nó đã được tạo.	Kích thước của ArrayList có thể thay đổi được. Kích thước của nó tự tăng (50% kích thước hiện có) nếu thêm một phần tử vượt quá sức chứa (compacity) hiện có của nó.
Có thể lưu trữ dữ liệu kiểu nguyên thủy và đối tượng.	Chỉ có thể lưu trữ dữ liệu kiểu đối tượng (Object). Kể từ Java 5, kiểu nguyên thủy được tự động chuyển đổi trong các đối tượng được gọi là auto-boxing. Ví dụ: không thể khai báo **ArrayList<int>**
Sử dụng for hoặc for-each để duyệt qua các phần tử của mảng.	Tương tự như Array, có thể duyệt qua các phần tử bằng lệnh for hoặc for-each. Tuy nhiên, nó có thể duyệt qua Iterator hoặc ListIterator.

So sánh Array và ArrayList

Khác nhau giữa Array và ArrayList

Array	ArrayList
Kích thước của Array có thể được kiểm tra bằng thuộc tính length.	Kích thước của ArrayList được kiểm tra bằng phương thức size().
Không hỗ trợ kiểu Generic	Hỗ trợ kiểu Generic
Các phần tử được thêm vào danh sách bằng lệnh gán.	Các phần tử được thêm vào danh sách bằng phương thức add().

So sánh Array và ArrayList

Khác nhau giữa Array và ArrayList

- **Ví dụ:** Khởi tạo một mảng có 5 phần tử: bạn không thể thêm phần tử thứ 6 vào mảng hoặc thay đổi kích thước của nó.

```
int []arr = new int[5];
arr[5] = 6;
```

- Khi thực thi chương trình trên, bạn sẽ gặp lỗi **java.lang.ArrayIndexOutOfBoundsException.**

So sánh Array và ArrayList

Khác nhau giữa Array và ArrayList

- Tuy nhiên, đối với **ArrayList** không bị giới hạn về số lượng phần tử. Khi vượt quá kích thước khởi tạo, nó tự động tăng lên 50% để lưu thêm phần tử mới.

```
public class ArrayVsArrayList {
    public static void main(String[] args) throws Exception {
        ArrayList<String> list = new ArrayList<>(5);
        list.add("Item01");
        list.add("Item02");
        list.add("Item03");
        list.add("Item04");
        list.add("Item05");

        int initSize = list.size();
        list.add("Item06");

        int currentSize = list.size();
        System.out.format("Init Size: %d, Current Size: %d%n", initSize, currentSize);
    }
}
```

Chuyển đổi Array sang ArrayList

Ví dụ chuyển đổi Array sang ArrayList sử dụng Arrays.asList()

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

public class ArrayToArrayListExample01 {

    public static void main(String[] args) {
        // Array Declaration and initialization
        String[] arr = { "JAVA", "J2EE", "JSP", "SERVLETS" };

        // Array to ArrayList conversion
        ArrayList<String> list = new ArrayList<String>(Arrays.asList(arr));

        // Adding new elements to the converted List
        list.add("C#");
        list.add("PHP");

        // Print list data
        System.out.println(list); // [JAVA, J2EE, JSP, SERVLETS, C#, PHP]
    }
}
```

Chuyển đổi Array sang ArrayList

Ví dụ chuyển đổi Array sang ArrayList sử dụng Collections.addAll()

```java
public class ArrayToArrayListExample02 {

    public static void main(String[] args) {
        // Array Declaration and initialization
        String[] arr = { "JAVA", "J2EE", "JSP", "SERVLETS" };

        // ArrayList declaration
        ArrayList<String> list = new ArrayList<String>();

        // Array to ArrayList conversion
        Collections.addAll(list, arr);

        // Adding new elements to the converted List
        list.add("C#");
        list.add("PHP");

        // Print list data
        System.out.println(list); // [JAVA, J2EE, JSP, SERVLETS, C#, PHP]
    }
}
```

Chuyển đổi Array sang ArrayList

Ví dụ chuyển đổi Array sang ArrayList sử dụng vòng lặp

```java
public class ArrayToArrayListExample03 {

    public static void main(String[] args) {
        // Array Declaration and initialization
        String[] arr = { "JAVA", "J2EE", "JSP", "SERVLETS" };

        // ArrayList declaration
        ArrayList<String> list = new ArrayList<String>();

        // Array to ArrayList conversion
        for (String item : arr) {
            list.add(item);
        }

        // Adding new elements to the converted List
        list.add("C#");
        list.add("PHP");

        // Print list data
        System.out.println(list); // [JAVA, J2EE, JSP, SERVLETS, C#, PHP]
    }
}
```

Chuyển đổi ArrayList sang Array

Ví dụ chuyển đổi ArrayList sang Array sử dụng phương thức toArray()

```
public class ArrayListToArrayExample01 {

    public static void main(String[] args) {
        // Create list
        List<Integer> list = new ArrayList<Integer>();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);

        // Convert ArrayList to Array
        Integer[] arr = new Integer[list.size()];
        arr = list.toArray(arr);

        // Print data
        System.out.println(Arrays.toString(arr)); // [10, 20, 30, 40]
    }
}
```

Chuyển đổi ArrayList sang Array

Ví dụ chuyển đổi ArrayList sang Array sử dụng vòng lặp

```java
public class ArrayListToArrayExample02 {

    public static void main(String[] args) {
        // Create list
        List<Integer> list = new ArrayList<Integer>();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);

        // Convert ArrayList to Array
        Integer[] arr = new Integer[list.size()];
        for (int i = 0; i < list.size(); i++) {
            arr[i] = list.get(i);
        }

        // Print data
        System.out.println(Arrays.toString(arr)); // [10, 20, 30, 40]
    }
}
```

Loại bỏ phần tử trùng lặp trong ArrayList

Ví dụ loại bỏ phần tử trùng lặp trong ArrayList sử dụng vòng lặp

```
public class ArrayListExample03 {

    public static void main(String[] args) {
        // Constructing An ArrayList
        List<String> listWithDuplicateElements = new ArrayList<String>();
        listWithDuplicateElements.add("JAVA");
        listWithDuplicateElements.add("J2EE");
        listWithDuplicateElements.add("JSP");
        listWithDuplicateElements.add("SERVLETS");
        listWithDuplicateElements.add("JAVA");
        listWithDuplicateElements.add("STRUTS");
        listWithDuplicateElements.add("JSP");

        // Printing listWithDuplicateElements
        System.out.println("ArrayList With Duplicate Elements :");
        System.out.println(listWithDuplicateElements);

        // Constructing Another ArrayList
        List<String> listWithoutDuplicateElements = new ArrayList<String>();
        for (String element : listWithDuplicateElements) {
            // Check if element not exist in list, perform add element to list
            if (!listWithoutDuplicateElements.contains(element)) {
                listWithoutDuplicateElements.add(element);
            }
        }

        // Printing listWithoutDuplicateElements
        System.out.println("\nArrayList After Removing Duplicate Elements :");
        System.out.println(listWithoutDuplicateElements);
    }
}
```

Loại bỏ phần tử trùng lặp trong ArrayList

Ví dụ loại bỏ phần tử trùng lặp trong ArrayList sử dụng HashSet

```
public class ArrayListExample04 {

    public static void main(String[] args) {
        // Constructing An ArrayList
        List<String> listWithDuplicateElements = new ArrayList<String>();
        listWithDuplicateElements.add("JAVA");
        listWithDuplicateElements.add("J2EE");
        listWithDuplicateElements.add("JSP");
        listWithDuplicateElements.add("SERVLETS");
        listWithDuplicateElements.add("JAVA");
        listWithDuplicateElements.add("STRUTS");
        listWithDuplicateElements.add("JSP");

        // Printing listWithDuplicateElements
        System.out.println("ArrayList With Duplicate Elements :");
        System.out.println(listWithDuplicateElements);

        // Constructing HashSet using listWithDuplicateElements
        Set<String> set = new HashSet<String>(listWithDuplicateElements);

        // Constructing listWithoutDuplicateElements using set
        List<String> listWithoutDuplicateElements = new ArrayList<String>(set);

        // Printing listWithoutDuplicateElements
        System.out.println("\nArrayList After Removing Duplicate Elements :");
        System.out.println(listWithoutDuplicateElements);
    }
}
```

Loại bỏ phần tử trùng lặp trong ArrayList

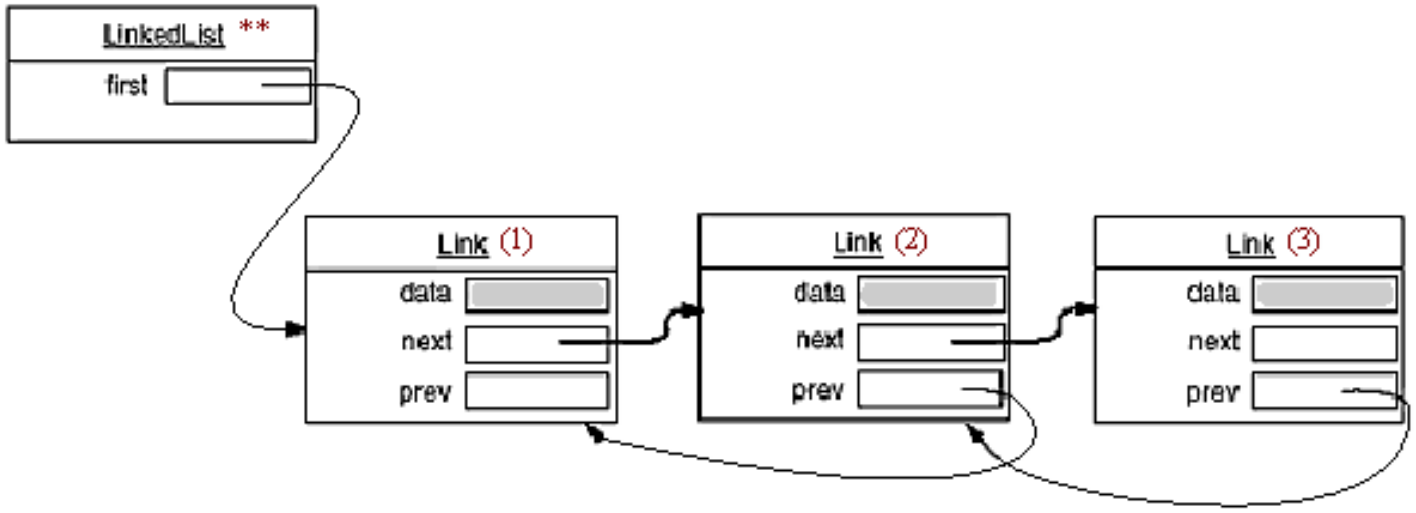
Ví dụ loại bỏ phần tử trùng lặp trong ArrayList sử dụng LinkedHashSet

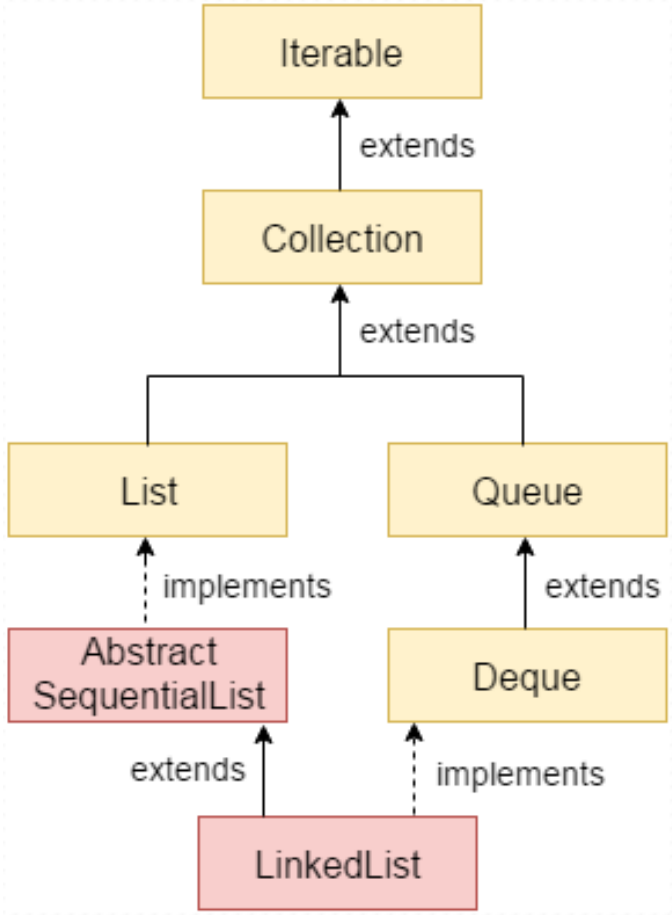
```
public class ArrayListExample05 {

    public static void main(String[] args) {
        // Constructing An ArrayList
        List<String> listWithDuplicateElements = new ArrayList<String>();
        listWithDuplicateElements.add("JAVA");
        listWithDuplicateElements.add("J2EE");
        listWithDuplicateElements.add("JSP");
        listWithDuplicateElements.add("SERVLETS");
        listWithDuplicateElements.add("JAVA");
        listWithDuplicateElements.add("STRUTS");
        listWithDuplicateElements.add("JSP");

        // Printing listWithDuplicateElements
        System.out.println("ArrayList With Duplicate Elements :");
        System.out.println(listWithDuplicateElements);

        // Constructing LinkedHashSet using listWithDuplicateElements
        Set<String> set = new LinkedHashSet<String>(listWithDuplicateElements);

        // Constructing listWithoutDuplicateElements using set
        List<String> listWithoutDuplicateElements = new ArrayList<String>(set);

        // Printing listWithoutDuplicateElements
        System.out.println("\nArrayList After Removing Duplicate Elements :");
        System.out.println(listWithoutDuplicateElements);
    }
}
```

Giới thiệu lớp LinkedList trong Java

Giới thiệu LinkedList

- **Danh sách kết nối LinkedList** là một trong các cách quản lý danh sách dữ liệu khắc phục được các nhược điểm của **Array**.
- Lớp **LinkedList** trong java sử dụng cấu trúc danh sách liên kết **Doubly Linked List** để lưu trữ các phần tử.

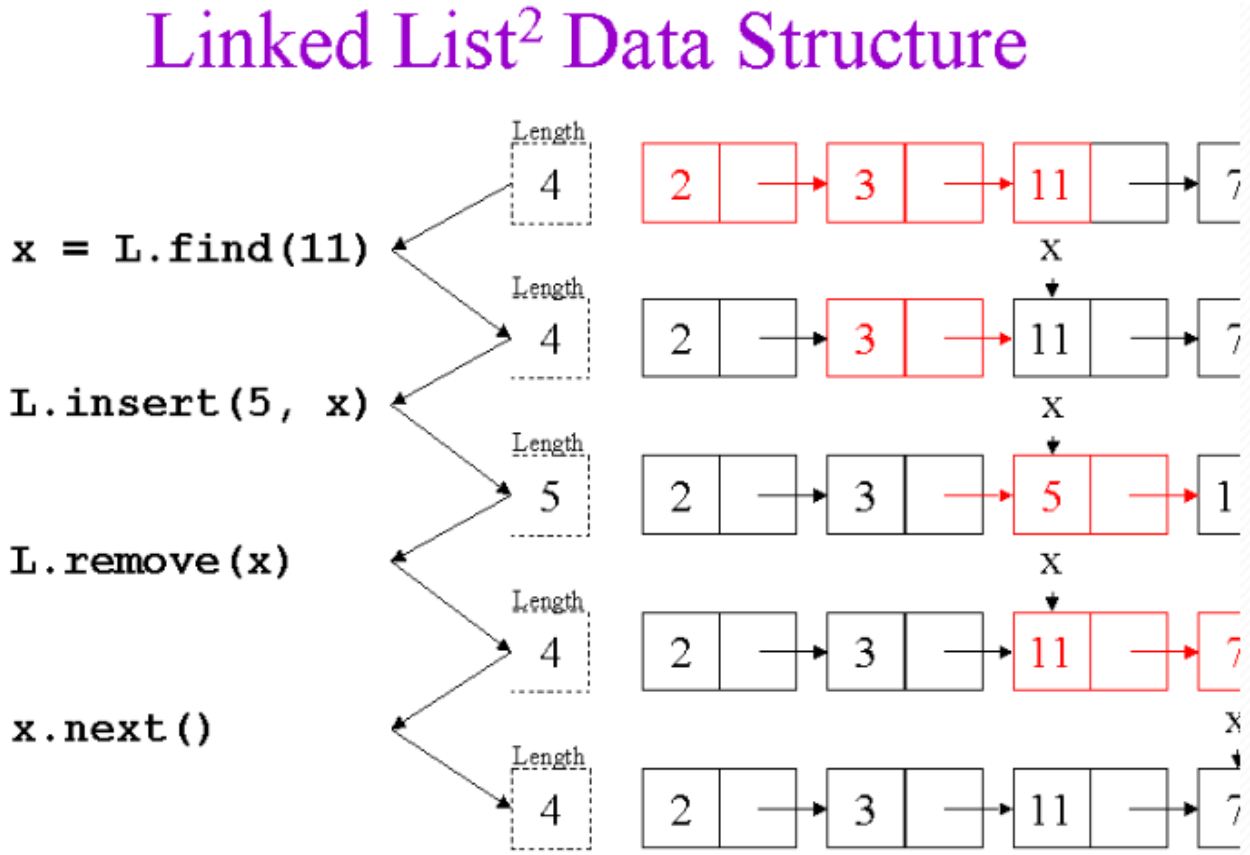




Giới thiệu lớp LinkedList trong Java

Các đặc điểm của lớp LinkedList

- **Có thể** chứa các phần tử trùng lặp.
- **Duy trì** thứ tự của phần tử được thêm vào.
- **Không đồng bộ** (non-synchronized
- **Thao tác** thêm/xóa (add/remove) phần tử nhanh vì không cần phải dịch chuyển nếu bất kỳ phần tử nào thêm/xóa khỏi danh sách.
- **LinkedList** có thể được sử dụng như danh sách (List), ngăn xếp (Stack), hoặc hàng đợi (Queue).



Giới thiệu lớp LinkedList trong Java

Các đặc điểm của lớp LinkedList

❑ Lớp **java.util.LinkedList** được định nghĩa như sau:

```
public class LinkedList<E> extends AbstractSequentialList<E>
    implements List<E>, Deque<E>, Cloneable, java.io.Serializable
```

Các phương thức khởi tạo (constructor) của lớp ArrayList

Phương thức	Mô tả
LinkedList()	khởi tạo một danh sách mảng trống
LinkedList(Collection c)	khởi tạo một danh sách mảng với các phần tử của collection c

Giới thiệu lớp LinkedList trong Java

Ví dụ sử dụng *LinkedList* với kiểu dữ liệu cơ bản (Wrapper) trong Java:

```
import java.util.LinkedList;
import java.util.List;

public class LinkedListExample01 {

    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {

    public static void printData(List<String> list) {
        // Show list through for-each
        for (String item : list) {
            System.out.print(item + " ");
        }
        System.out.println();
    }

}
```

Giới thiệu lớp LinkedList trong Java

Ví dụ sử dụng *LinkedList* với kiểu dữ liệu cơ bản (Wrapper) trong Java:

```
// Create list with no parameter
List<String> list1 = new LinkedList<>();
for (int i = 1; i <= NUM_OF_ELEMENT; i++) {
    // Add element to list
    list1.add("0" + i);
}
System.out.print("Init list1 = ");
printData(list1);

// addAll()
List<String> list2 = new LinkedList<>();
list2.addAll(list1);
list2.add("0" + 3);
System.out.print("After list1.addAll(03): list2 = ");
printData(list2);

// IndexOf()
System.out.println("list2.indexOf(03) = " + list2.indexOf("03"));
System.out.println("list2.indexOf(06) = " + list2.indexOf("06"));

// lastIndexOf()
System.out.println("list2.lastIndexOf(03) = " + list2.lastIndexOf("03"));
```

```
// Remove
list2.remove("01");
System.out.print("After list2.remove(01): list2 = ");
printData(list2);

// retainAll()
List<String> list4 = new LinkedList<>(list1);
LinkedList<String> list3 = new LinkedList<String>();
list3.add("0" + 3);
list3.add("0" + 2);
System.out.print("Init list3 = ");
printData(list3);
list4.retainAll(list3);
System.out.print("After list1.retainAll(list3): list4 = ");
printData(list4);

// removeAll()
List<String> list5 = new LinkedList<>(list1);
list5.removeAll(list3);
System.out.print("After list1.removeAll(list3): list5 = ");
printData(list5);
```

```
After list2.remove(01): list2 = 02 03 04 05 03
Init list3 = 03 02
After list1.retainAll(list3): list4 = 02 03
After list1.removeAll(list3): list5 = 01 04 05
```

Giới thiệu lớp LinkedList trong Java

Ví dụ sử dụng *LinkedList* với kiểu dữ liệu người dùng tự định nghĩa (Object):

```java
import java.util.LinkedList;
import java.util.List;

class Student02 {
    private int id;
    private String name;

    public Student02(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + "]";
    }
}
```

```java
public class LinkedListExample02 {
    public static final int NUM_OF_ELEMENT = 5;

    public static void main(String[] args) {
        // Create list
        List<Student02> students = new LinkedList<>();
        for (int i = 1; i <= NUM_OF_ELEMENT; i++) {
            // Add element to list
            Student02 student = new Student02(i, "myname" + i);
            students.add(student);
        }

        // Show list student
        for (Student02 student : students) {
            System.out.println(student);
        }
    }
}
```

```
Student [id=1, name=myname1]
Student [id=2, name=myname2]
Student [id=3, name=myname3]
Student [id=4, name=myname4]
Student [id=5, name=myname5]
```

So sánh ArrayList và LinkedList

Giống nhau giữa ArrayList và LinkedList

- Cả hai lớp **ArrayList** và **LinkedList** đều được implements từ **List Interface** và duy trì thứ tự của phần tử được thêm vào.
- **Cả hai lớp này** đều là lớp không đồng bộ (non-synchronized).

0	1	2	3	4
23	3	17	9	42

So sánh ArrayList và LinkedList

Khác nhau giữa ArrayList và LinkedList

	ArrayList	LinkedList
Cấu trúc dữ liệu (Structure)	ArrayList sử dụng **mảng động** để lưu trữ các phần tử.	LinkedList sử dụng **danh sách liên kết (Doubly Linked List)** để lưu trữ các phần tử.
Thao tác thêm và xóa (Insertion And Removal)	Thao tác thêm và xóa phần tử với ArrayList là chậm bởi vì nó sử dụng nội bộ mảng. Bởi vì sau khi thêm hoặc xóa các phần tử cần sắp xếp lại. Độ phức tạp: **O(n)**	Thao tác thêm và xóa phần tử với LinkedList nhanh hơn ArrayList. Bởi vì nó không cần sắp xếp lại các phần tử sau khi thêm hoặc xóa. Nó chỉ cần cập nhật lại tham chiếu tới phần tử phía trước và sau nó. Độ phức tạp: **O(1)**

So sánh ArrayList và LinkedList

Khác nhau giữa ArrayList và LinkedList

	ArrayList	LinkedList
Thao tác tìm kiếm hoặc truy xuất phần tử (Retrieval)	Truy xuất phần tử trong ArrayList nhanh hơn LinkedList. Bởi vì các phần tử trong ArrayList được lưu trữ dựa trên chỉ mục (index). Độ phức tạp: **O(1)**	Thao tác truy xuất phần tử trong LinkedList chậm hơn nhiều so với ArrayList. Bởi vì, nó phải duyệt qua lần lượt các phần tử từ đầu tiên cho đến cuối cùng. Độ phức tạp: **O(n)**
Truy xuất ngẫu nhiên (Random Access)	ArrayList có thể truy xuất ngẫu nhiên phần tử.	LinkedList không thể truy xuất ngẫu nhiên. Nó phải duyệt qua tất cả các phần tử từ đầu tiên đến phần tử cuối cùng để tìm phần tử.

So sánh ArrayList và LinkedList

Khác nhau giữa ArrayList và LinkedList

	ArrayList	LinkedList
Trường hợp sử dụng (Usage)	ArrayList chỉ có thể hoạt động như một list vì nó chỉ implements giao tiếp **List**.	LinkedList có thể hoạt động như một **ArrayList**, stack (hàng đợi), queue (hàng đợi), **Singly Linked List and Doubly Linked List** vì nó implements các giao tiếp List và Deque.
Sử dụng bộ nhớ	ArrayList yêu cầu **ít bộ nhớ hơn** so với LinkedList. Bởi vì ArrayList chỉ lưu trữ dữ liệu (data) của nó và chỉ mục (index).	LinkedList yêu cầu **nhiều bộ nhớ hơn** so với ArrayList. Bởi vì LinkedList lưu giữ thông tin của nó và tham chiếu tới phần tử trước và sau nó.
Khi nào sử dụng?	ArrayList **tốt hơn trong việc lưu trữ và truy xuất dữ liệu (get).**	LinkedList **tốt hơn trong việc thao tác dữ liệu (thêm/ xóa).**

Tổng kết nội dung bài học

- Tại sao nên sử dụng Collection
- Hệ thống phân cấp của Collection
- List, Iterator, ListIterator Interfaces
- Giới thiệu lớp ArrayList trong Java
- So sánh Array và ArrayList
- Chuyển đổi Array sang ArrayList
- Chuyển đổi ArrayList sang Array
- Loại bỏ phần tử trùng lặp trong ArrayList
- Giới thiệu lớp LinkedList trong Java
- So sánh ArrayList và LinkedList