Bài 11
Generic, Stack và Queue

Module: ADVANCED PROGRAMMING WITH JAVA

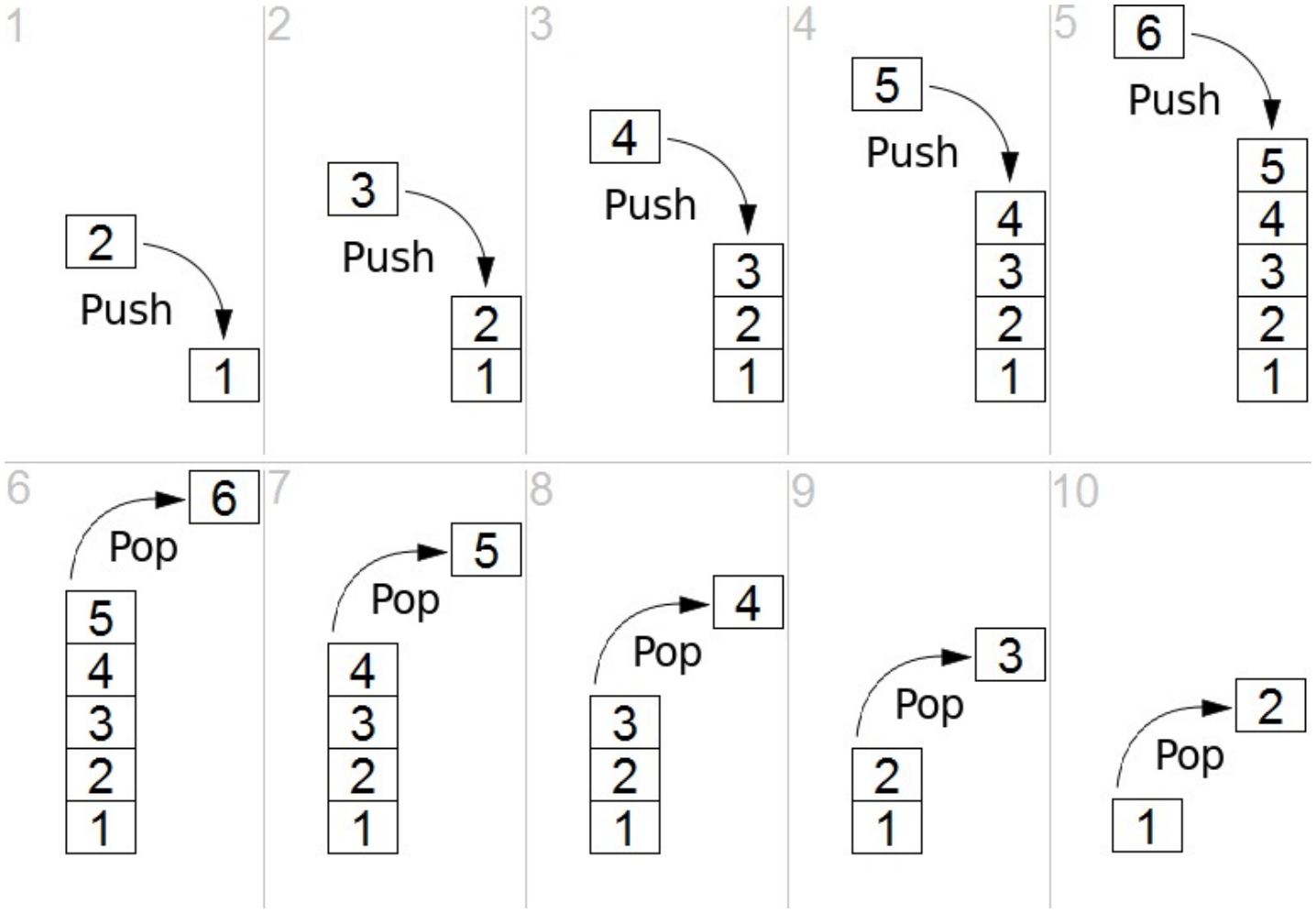
Mục tiêu

- Trình bày được cấu trúc dữ liệu Stack
- Cài đặt được cấu trúc dữ liệu Stack
- Trình bày được cấu trúc dữ liệu Queue
- Cài đặt được cấu trúc dữ liệu Queue

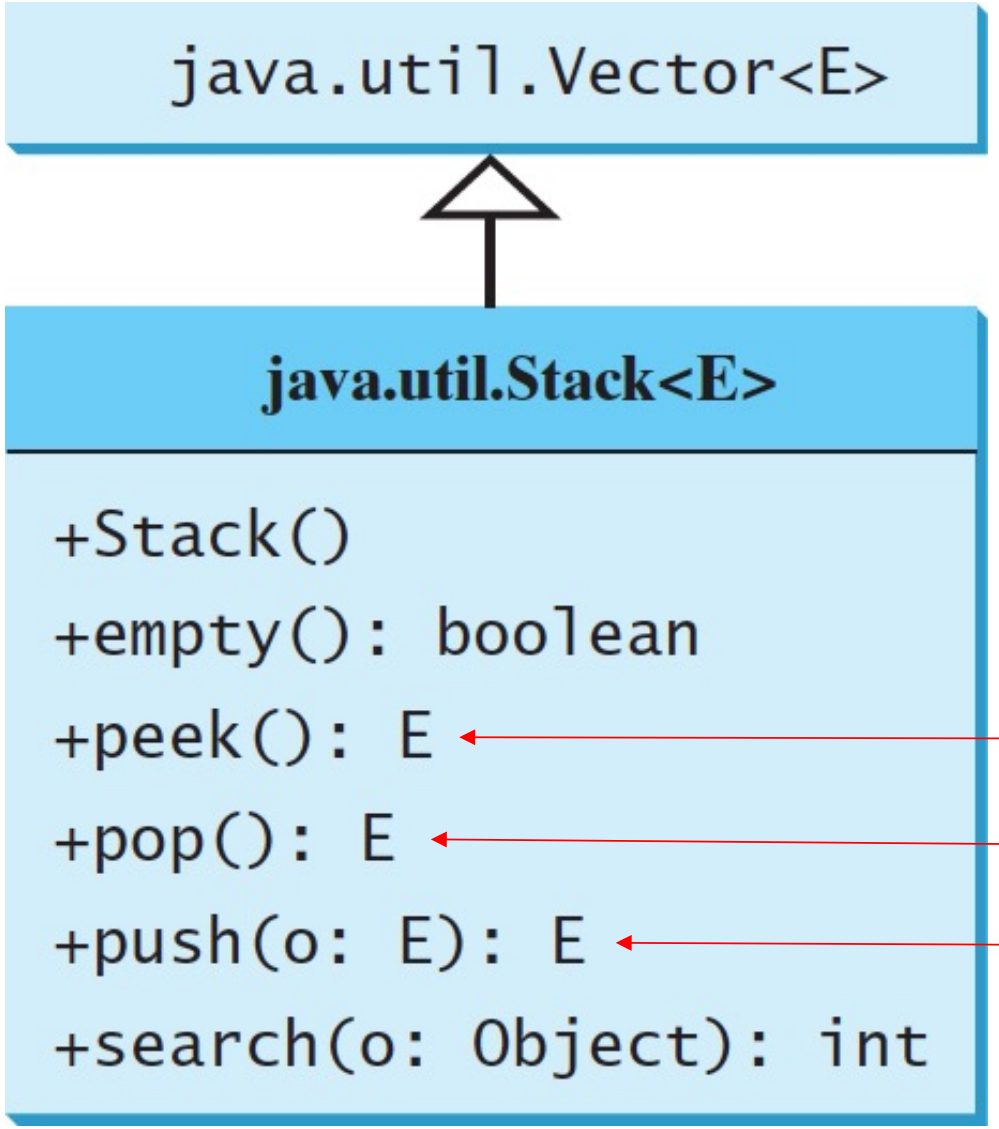
Stack

Stack (Ngăn xếp)

- Stack là một cấu trúc dữ liệu danh sách, trong đó việc thêm và lấy các phần tử được thực hiện theo quy tắc FILO (Fist-In/Last-Out)



Lớp Stack của Java Collection Framework



Trả về phần tử trên cùng của stack

Trả về và xoá phần tử trên cùng của stack

Thêm một phần tử vào trên cùng của stack

Triển khai Stack

```
public class MyStack<E> {
    private static final int INITIAL_SIZE = 16;
    private E[] elements;
    private int count = 0;

    public MyStack() {
        elements = (E[]) new Object[INITIAL_SIZE];
    }
}
```

Có thể sử dụng ArrayList để triển khai Stack thay vì sử dụng mảng

Phương thức push()

```
public void push(E e){
    ensureCapacity();

    elements[count++] = e;
}

private void ensureCapacity() {
    if(count >= elements.length){
        E[] newElements = (E[]) new Object[elements.length * 2 + 1];
        System.arraycopy(elements, 0, newElements, 0, count);  elements =
        newElements;
    }
}
```

Phương thức pop()

```
public E pop(){
    if(count == 0){
        throw new IndexOutOfBoundsException("Stack is empty");
    }
    E e = elements[count - 1];
    elements[count - 1] = null;  count--;
    return e;
}
```

Sử dụng Stack

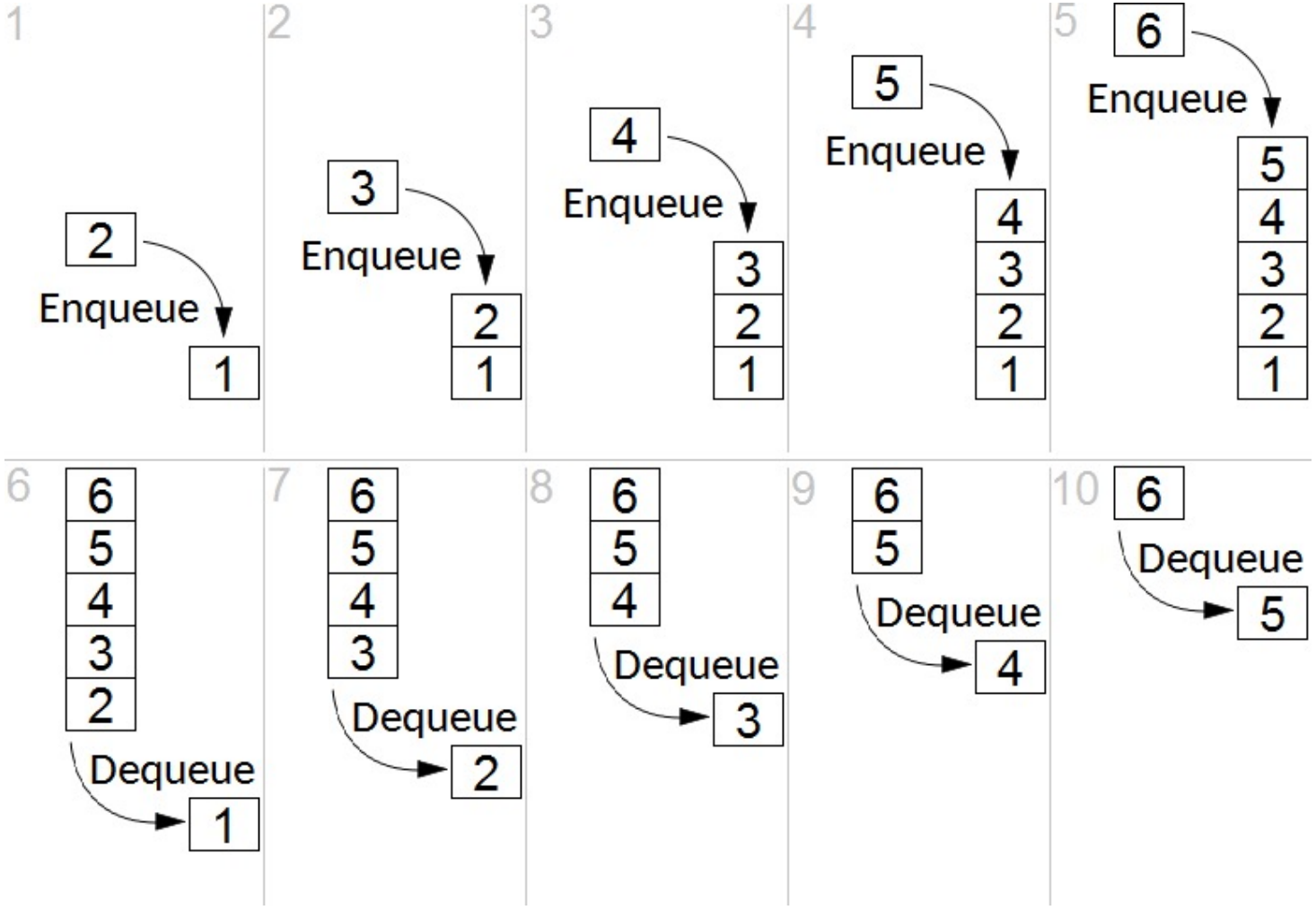
```
public static void main(String[] args) {
    MyStack<String> stack = new MyStack<>();
    stack.push("America");  stack.push("Canada");
    stack.push("France");

    while (!stack.isEmpty()){
        System.out.println(stack.pop());
    }
}
```

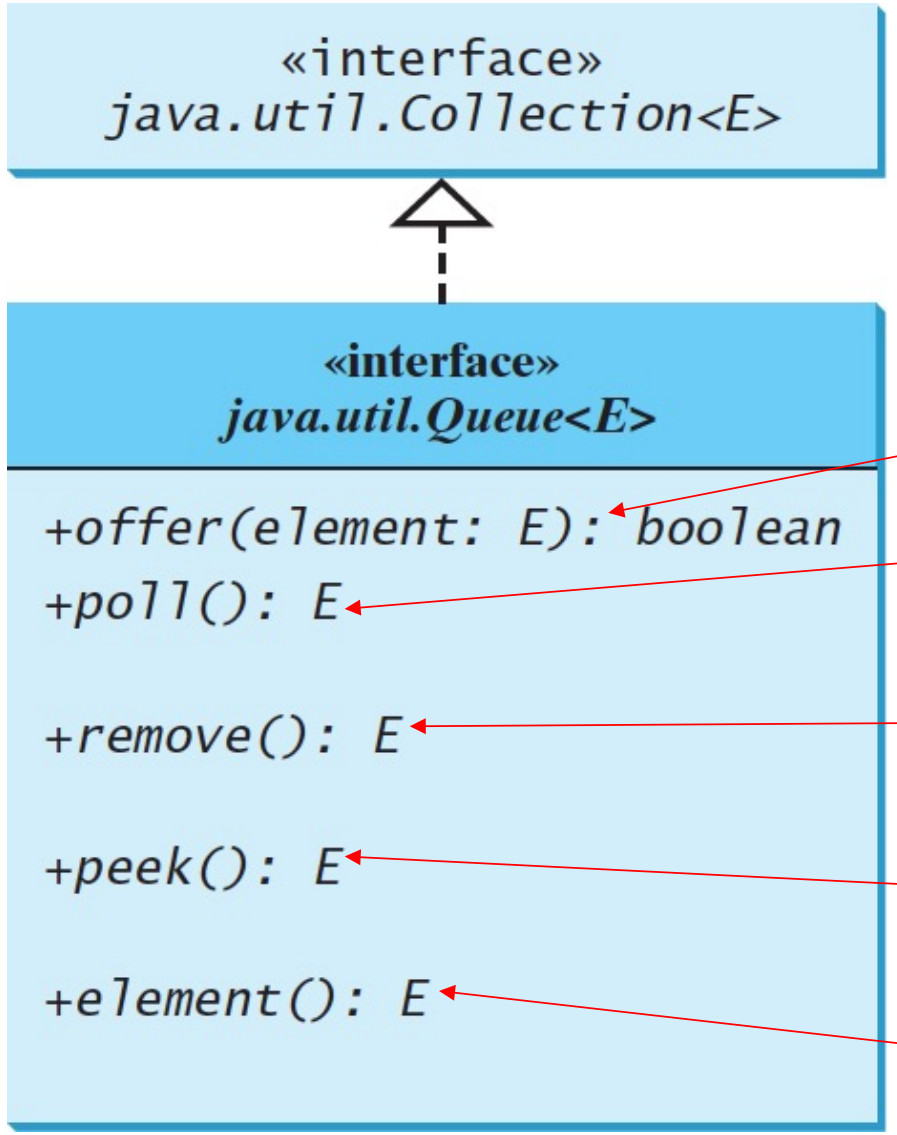
Queue

Queue (Hàng đợi)

- Queue là một cấu trúc dữ liệu danh sách, trong đó việc thêm và lấy các phần tử được thực hiện theo quy tắc FIFO(Fist-In/First-Out)



Lớp PriorityQueue của Java Collection



Thêm phần tử vào queue

Lấy phần tử ở phần đầu của queue hoặc trả về null nếu rỗng

Lấy và xoá phần tử ở phần đầu của queue và tung ngoại lệ nếu rỗng

Lấy phần tử ở phần đầu của queue hoặc trả về null nếu rỗng

Lấy phần tử ở phần đầu của queue và tung ngoại lệ nếu rỗng

Triển khai Queue

```
import java.util.LinkedList;

public class GenericQueue<E> {
    private LinkedList<E> elements;

    public GenericQueue(){
        elements = new LinkedList<>();
    }

        public void enqueue(E e){
            elements.addLast(e);
    }
    ...
    ....
    public E dequeue(){
            return elements.removeFirst();
        }

        public int getSize(){
            return elements.size();
        }

        public boolean isEmpty(){
            return elements.size() == 0;
        }
}
```

Sử dụng Queue

```
public static void main(String[] args) {  GenericQueue<String>
    queue = new GenericQueue<>();

    queue.enqueue("America");
    queue.enqueue("Canada");
    queue.enqueue("France");

    while (!queue.isEmpty()){  System.out.println(queue.dequeue());
    }
}
```

Triển khai Priority Queue

```
public class MyPriorityQueue<E extends Comparable<E>> {
    private Heap<E> heap;

        public MyPriorityQueue(){
            heap = new Heap<>();
        }

    public void enqueue(E e){
        heap.add(e);
    }

    public E dequeue(){
        return heap.remove();
    }

    public boolean isEmpty(){
        return heap.getSize() == 0;
    }
}
```

Comparable và Comparator

Interface Comparable

- Interface Comparable định nghĩa phương thức compareTo() để so sánh giữa các đối tượng
- Khai báo của Comparable:

```
public interface Comparable<E> {
    public int compareTo(E o);
}
```

- Giá trị trả về của phương thức compareTo():
  - 0 nếu hai đối tượng bằng nhau
  - Số nguyên âm nếu đối tượng hiện tại nhỏ hơn *o*
  - Số nguyên dương nếu đối tượng hiện tại lớn hơn *o*

Triển khai Comparable: Ví dụ 1

```
public class Student implements Comparable<Student>{
    private int age;
    private String name;

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

    @Override
    public int compareTo(Student o) {
        return this.age - o.age;
    }
}
```

Triển khai Comparable: Ví dụ 2

```
public class Student implements Comparable<Student>{  private
int age;
private String name;

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

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

Interface Comparator

- Interface Comparator định nghĩa phương thức compare() để so sánh các đối tượng
- Khai báo của Comparator:

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

- Giá trị trả về của phương thức compare ():
  - 0 nếu hai đối tượng bằng nhau
  - Số nguyên âm nếu đối tượng *o1* nhỏ hơn *o2*
  - Số nguyên dương nếu đối tượng *o1* lớn hơn *o2*

Triển khai Comparator: Ví dụ

```
public class CustomerAgeComparator<T extends Customer>
implements Comparator<T> {  @Override
    public int compare(T o1, T o2) {
        return o1.getAge() - o2.getAge();
    }
}
```

```
public class CustomerNameComparator<T extends Customer>   implements
Comparator<T> {  @Override
    public int compare(T o1, T o2) {
        return o1.getName().compareTo(o2.getName());
    }
}
```

[Thực hành] Triển khai Stack sử dụng mảng

[Thực hành] Triển khai Stack sử dụng ArrayList

[Thực hành] Triển khai Queue sử dụng mảng

[Thực hành] Triển khai Queue sử dụng LinkedList

[Bài tập] Đảo ngược số sử dụng Stack

[Bài tập] Chuyển thập phân sang nhị phân

[Bài tập] Kiểm tra dấu ngoặc của biểu thức

[Bài tập] Kiểm tra chuỗi đối xứng

[Bài tập] Tổ chức dữ liệu hợp lý

Tổng kết

- Generic là cơ chế cho phép truyền kiểu dữ liệu vào như là tham số cho các lớp, interface và phương thức
- Stack là cấu trúc dữ liệu với các thao tác tuân theo trật tự First-In/Last-Out
- Sử dụng ArrayList để triển khai Stack hiệu quả hơn là sử dụng LinkedList
- Queue là cấu trúc dữ liệu với các thao tác tuân theo trật tự First-In/last-Out
- Sử dụng LinkedList để triển khai Queue hiệu quả hơn là sử dụng ArrayList

Hướng dẫn

- Hướng dẫn làm bài thực hành và bài tập
- Chuẩn bị bài tiếp: Map Tree