Bài 19
Generic, Stack và Queue

Môn học: PF-JAVA

Mục tiêu

- Trình bày được khái niệm Generic
- Sử dụng được cơ chế Generic
- 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

Generic

Generic

- Generic là cơ chế cho phép sử dụng Kiểu dữ liệu như là tham số
- Có thể định nghĩa Lớp và Phương thức với một kiểu dữ liệu generic, sau đó, compiler sẽ thay thế kiểu dữ liệu generic với một kiểu dữ liệu cụ thể
- Ví dụ:
  - Khai báo lớp ArrayList:

```
class ArrayList<E> {

}
```

  - Sử dụng lớp ArrayList:

```
ArrayList<String> strings = new ArrayList<>();
ArrayList<Customer> customers = new ArrayList<>();
```

  - E đại diện cho một kiểu dữ liệu generic
  - String và Customer là các kiểu dữ liệu cụ thể

Lợi ích của generic

- Giúp phát hiện lỗi ngay tại thời điểm biên dịch, thay vì tại thời điểm thực thi nếu không dùng generic
- Generic cho phép quy định các kiểu dữ liệu được phép sử dụng ở trong một lớp hoặc phương thức
- Nếu kiểu dữ liệu không phù hợp được sử dụng thì sẽ được phát hiện

Lợi ích của generic: Ví dụ

Không sử dụng Generic

```
ArrayList numbers = new ArrayList();
numbers.add(1);
numbers.add("a");

int total = 0;
for (int i = 0; i < numbers.size(); i++){
    total += (int)numbers.get(i);
}
System.out.println("Total: " + total);
```

Cần ép kiểu

Có lỗi xảy ra tại thời điểm thực thi bởi vì không thể ép kiểu từ String sang int

Cósử dụng Generic

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(1);
numbers.add("a");

int total = 0;
for (int i = 0; i < numbers.size(); i++){
    total += numbers.get(i);
}
System.out.println("Total: " + total);
```

Không cần ép kiểu

Thông báo lỗi được hiển thị ngay tại thời điểm compile. String không được add vào ArrayList Integer

So sánh lớp ArrayList có generic và không

java.util.ArrayList
+ArrayList() +add(o: Object): void +add(index: int, o: Object): void +clear(): void +contains(o: Object): boolean +get(index:int): Object +indexOf(o: Object): int +isEmpty(): boolean +lastIndexOf(o: Object): int +remove(o: Object): boolean +size(): int +remove(index: int): boolean +set(index: int, o: Object): Object

java.util.ArrayList<E>
+ArrayList() +add(o: E): void +add(index: int, o: E): void +clear(): void +contains(o: Object): boolean +get(index:int): E +indexOf(o: Object): int +isEmpty(): boolean +lastIndexOf(o: Object): int +remove(o: Object): boolean +size(): int +remove(index: int): boolean +set(index: int, o: E): E

Khai báo lớp và interface generic

- Cú pháp:

```
class ClassName<T> {

}
interface InterfaceName<T> {

}
```

  - Trong đó:
    - ClassName và InterfaceName là tên của lớp và interface
    - T là kiểu dữ liệu Generic. Có thể dùng bất cứ chữ cái nào.

Khai báo lớp generic: Ví dụ

```
class GenericArrayList<T> {
    private static final int INITIAL_SIZE = 16;
    private  T[] elements;
    private int count = 0;

    public GenericArrayList(){
        this.elements = (T[])new Object[INITIAL_SIZE];
    }

    public void add(T element){
        //TODO: Ensure capacity
        this.elements[count++] = element;
    }
    ....
}
```

Generic với nhiều kiểu dữ liệu

- Có thể định nghĩa lớp và interface với nhiều kiểu dữ liệu generic
- Các kiểu dữ liệu cách nhau bởi dấu phẩy (,)
- Ví dụ:

```
class GenericMap<K, V>{

}
```

Trong đó K và V là các kiểu dữ liệu generic

Phương thức generic

- Có thể sử dụng generic cho các phương thức static
- Cú pháp:

```
static <T> data_type MethodName(){

}
```

Trong đó:

- T là kiểu dữ liệu generic
- data_type là kiểu dữ liệu trả về
- MethodName là tên của phương thức

Phương thức generic: Ví dụ

```
public class GenericMethodDemo {
    public static void main(String[] args ) {
        Integer[] integers = {1, 2, 3, 4, 5};
        String[] strings = {"London", "Paris", "New York", "Austin"};

        GenericMethodDemo.<Integer>print(integers);
        GenericMethodDemo.<String>print(strings);
    }
    public static <E> void print(E[] list) {
        for (int i = 0; i < list.length; i++){
            System.out.print(list[i] + " ");
        }
        System.out.println();
    }
}
```

Có thể gọi rút gọn, không cần chỉ rõ kiểu:

GenericMethodDemo.print(integers);
GenericMethodDemo.print(strings);

Ràng buộc cho kiểu Generic

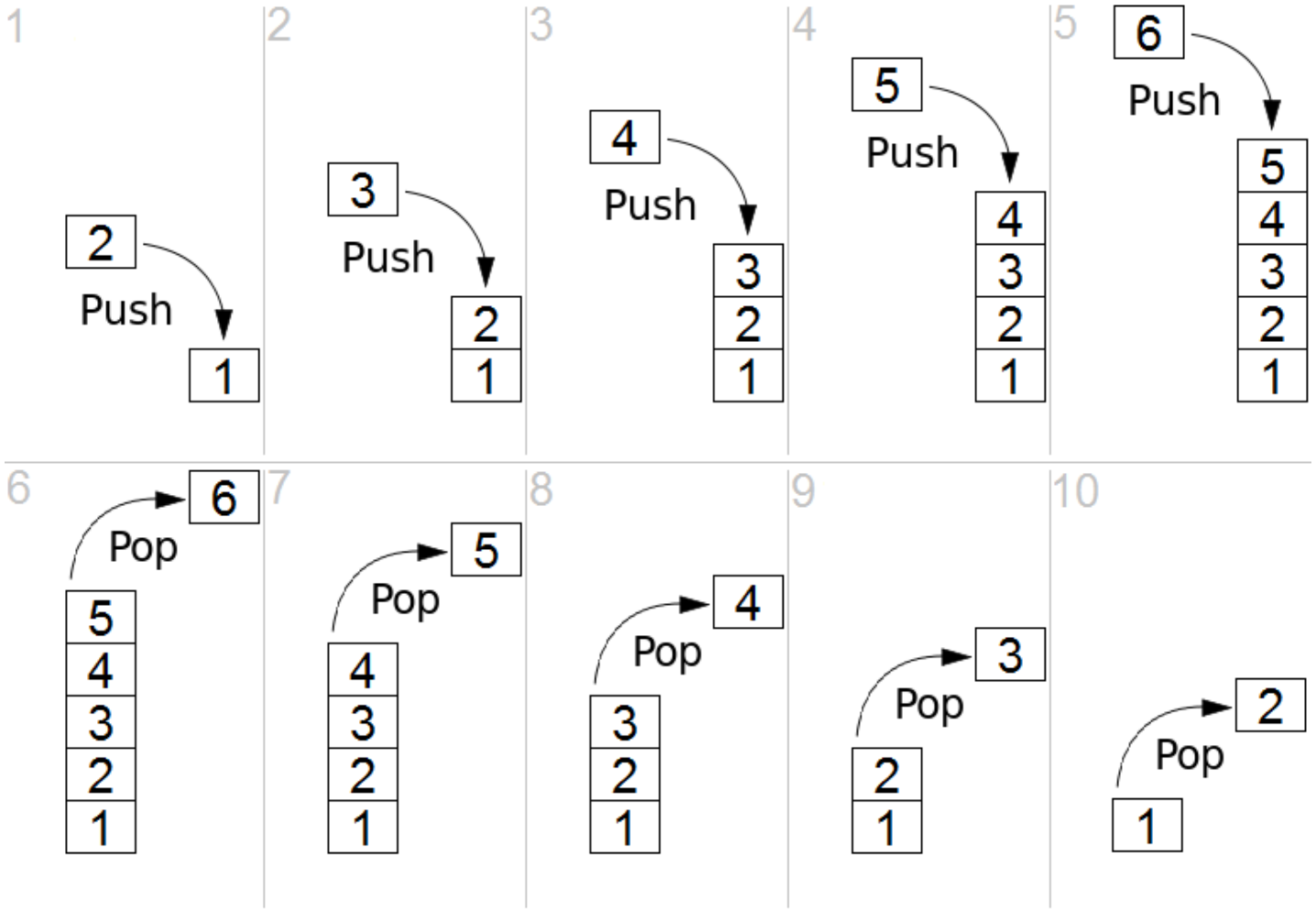
- Có thể quy định kiểu generic là subtype của một kiểu dữ liệu khác
- Ràng buộc này được gọi là Bounded Type
- Ví dụ:

```
public class BoundedTypeDemo {
    public static void main(String[] args ) {
        Rectangle rectangle = new Rectangle(2, 2);
        Circle circle = new Circle(2);

        System.out.println("Same area? " + equalArea(rectangle, circle));
    }

    public static <E extends GeometricObject> boolean equalArea(E object1, E object2)  {
        return object1.getArea() == object2.getArea();
    }
}
```

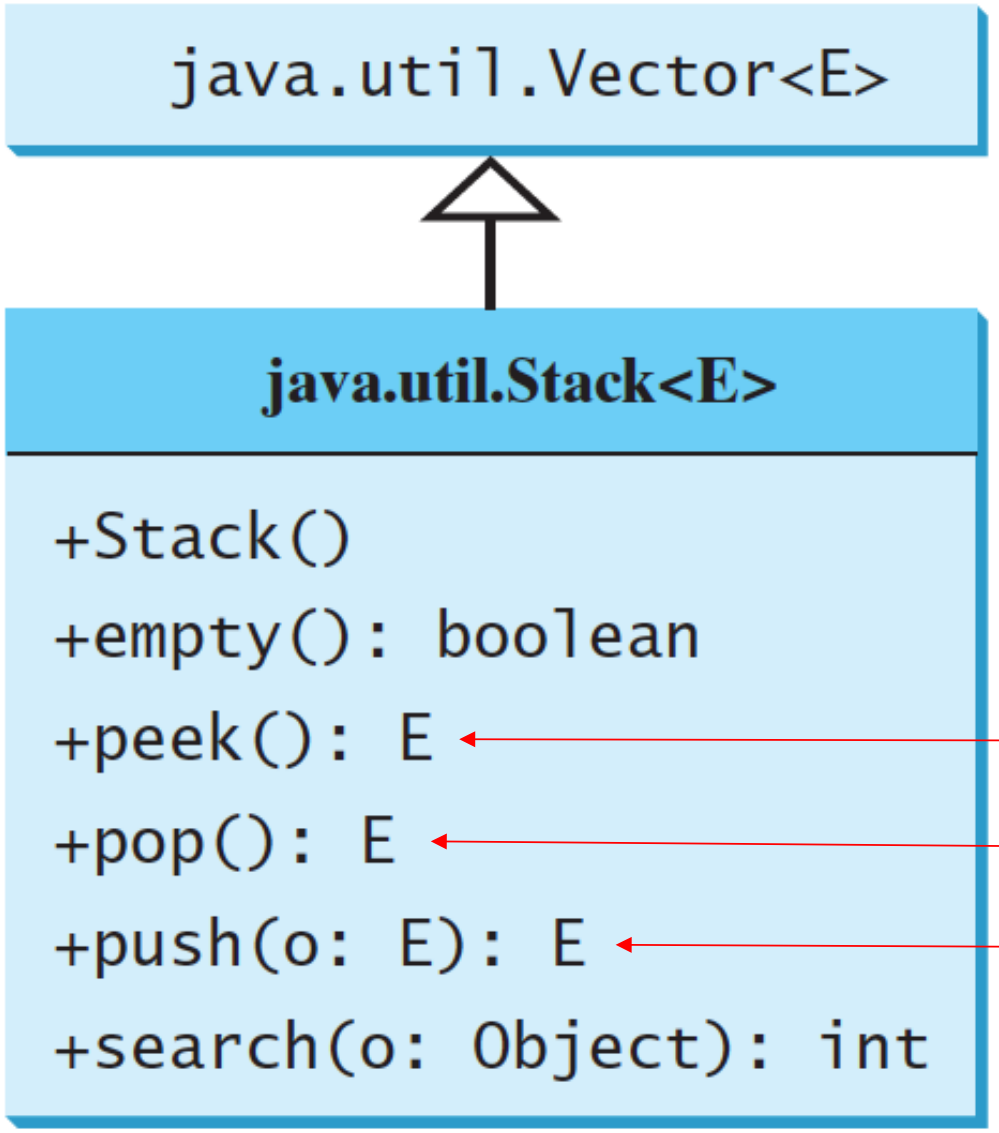
Stack

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;
}
```

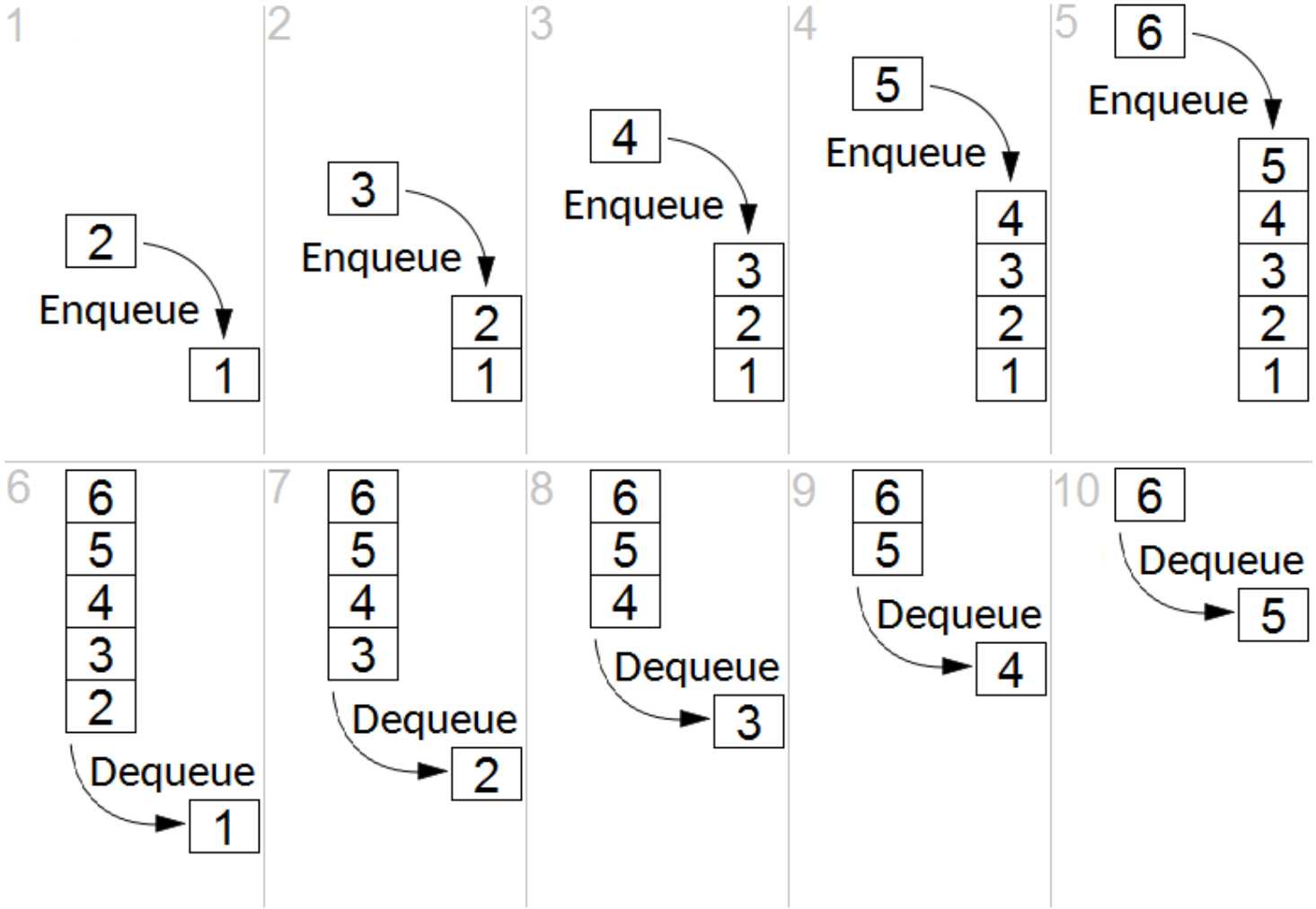
Sử dụng Stack

```
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());
    }
}
```

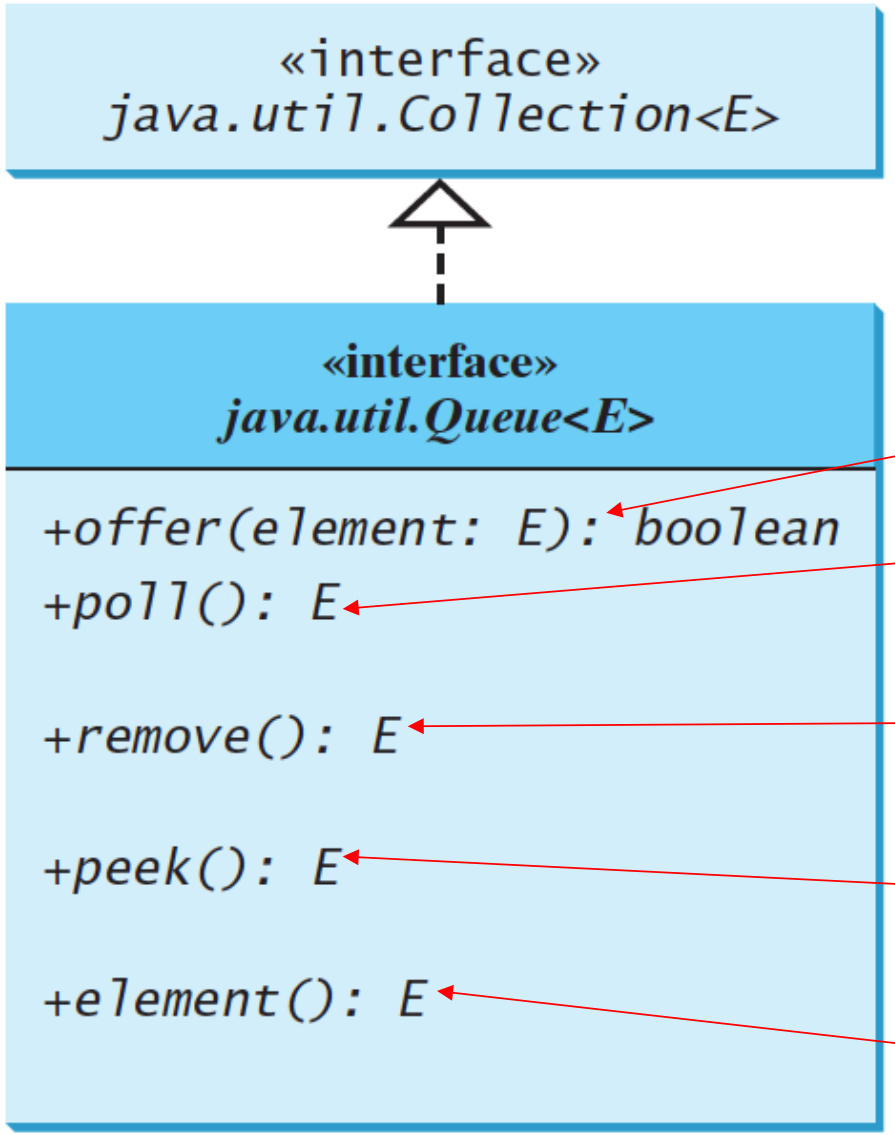
Queue

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