Stack và Queue

Môn học: Advanced Programming with JavaScript

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 Stack:

```
class Stack<E> {
}
```

  - Sử dụng lớp Stack:

```
const stack = new Stack<string>();
```

  - E đại diện cho một kiểu dữ liệu generic
  - string là 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

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 INITIAL_SIZE: number = 16;
    private elements: T[];
    private count: number = 0;

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

Generic với nhiều kiểu dữ liệu

- Có thể định nghĩa class 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

Sử dụng generic với hàm

```
function identity<T>(value: T): T {
    return value;
}

const result = identity<number>(123);
```

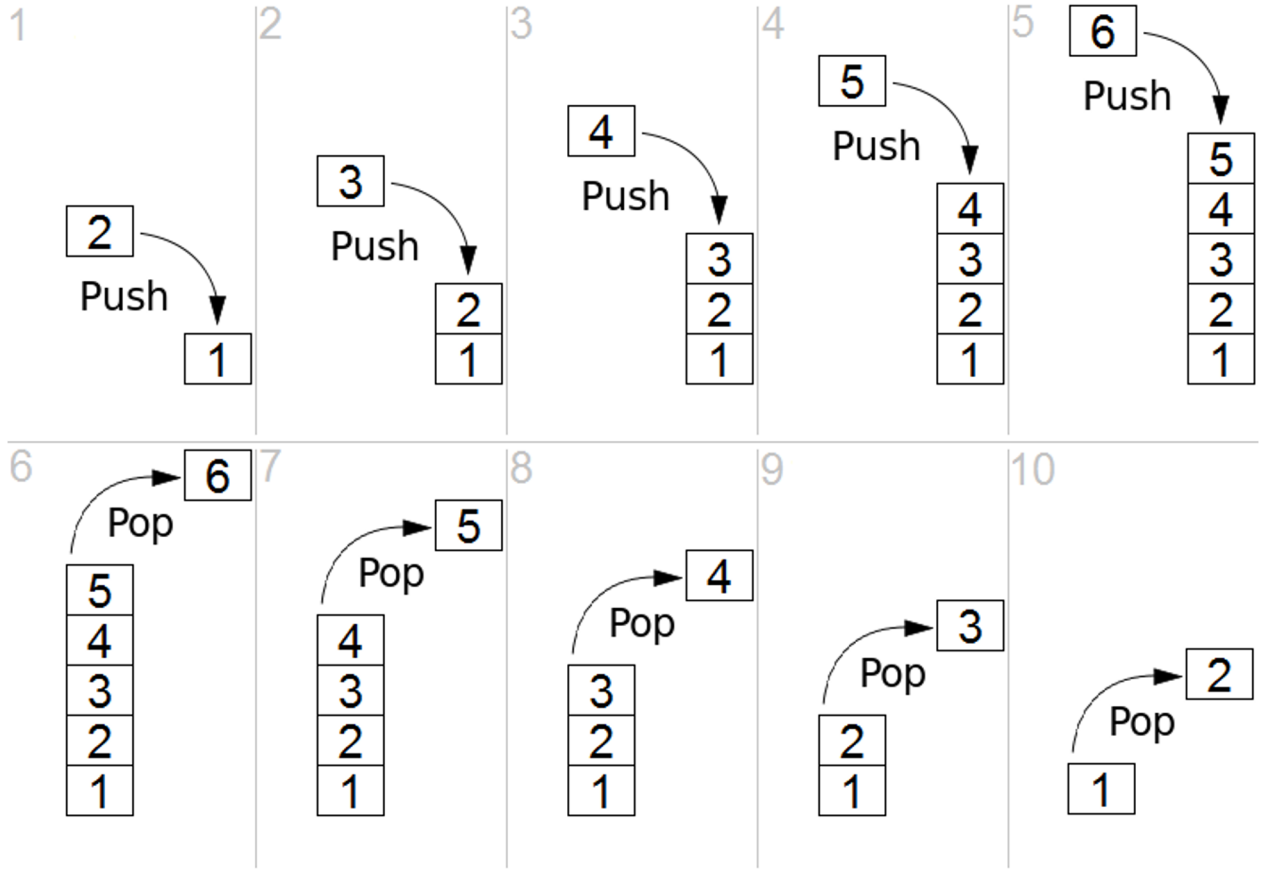
Demo

Demo Generic với class và hàm

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 (First-In/Last-Out)



Triển khai Stack

- Bước 1: Định nghĩa interface IStack
- Bước 2: Triển khai class Stack từ IStack
- Bước 3: Viết mã phương thức push()
- Bước 4: Viết mã phương thức pop()
- Bước 5: Viết mã phương thức peek() - tùy chọn
- Bước 6: Viết mã phương thức size() - tùy chọn
- Bước 7: Sử dụng Stack

Định nghĩa interface IStack

Ngăn xếp thường có các phương thức sau:

- **push** thêm một mục vào ngăn xếp
- **pop** trả về phần tử được thêm cuối cùng và xóa nó khỏi ngăn xếp
- **peek** (tùy chọn) trả về phần tử được thêm cuối cùng mà không xóa nó khỏi ngăn xếp

```
interface IStack<T> {
    push(item: T): void;
    pop(): T | undefined;
    peek(): T | undefined;
    size(): number;
}
```

Triển khai class Stack

```
class Stack<T> implements IStack<T> {
    private storage: T[] = [];

    constructor(private capacity: number = Infinity) { }
}
```

Phương thức push()

```
push(item: T): void {
    if (this.size() === this.capacity) {
        throw Error("Stack has reached max capacity,
you cannot add more items");
    }
    this.storage.push(item);
}
```

Phương thức pop()

```
pop(): T | undefined {
    if(this.size() === 0){
        throw new Error("Stack is empty");
    }
    return this.storage.pop();
}
```

Phương thức size()

```
size(): number {
    return this.storage.length;
}
```

Sử dụng Stack

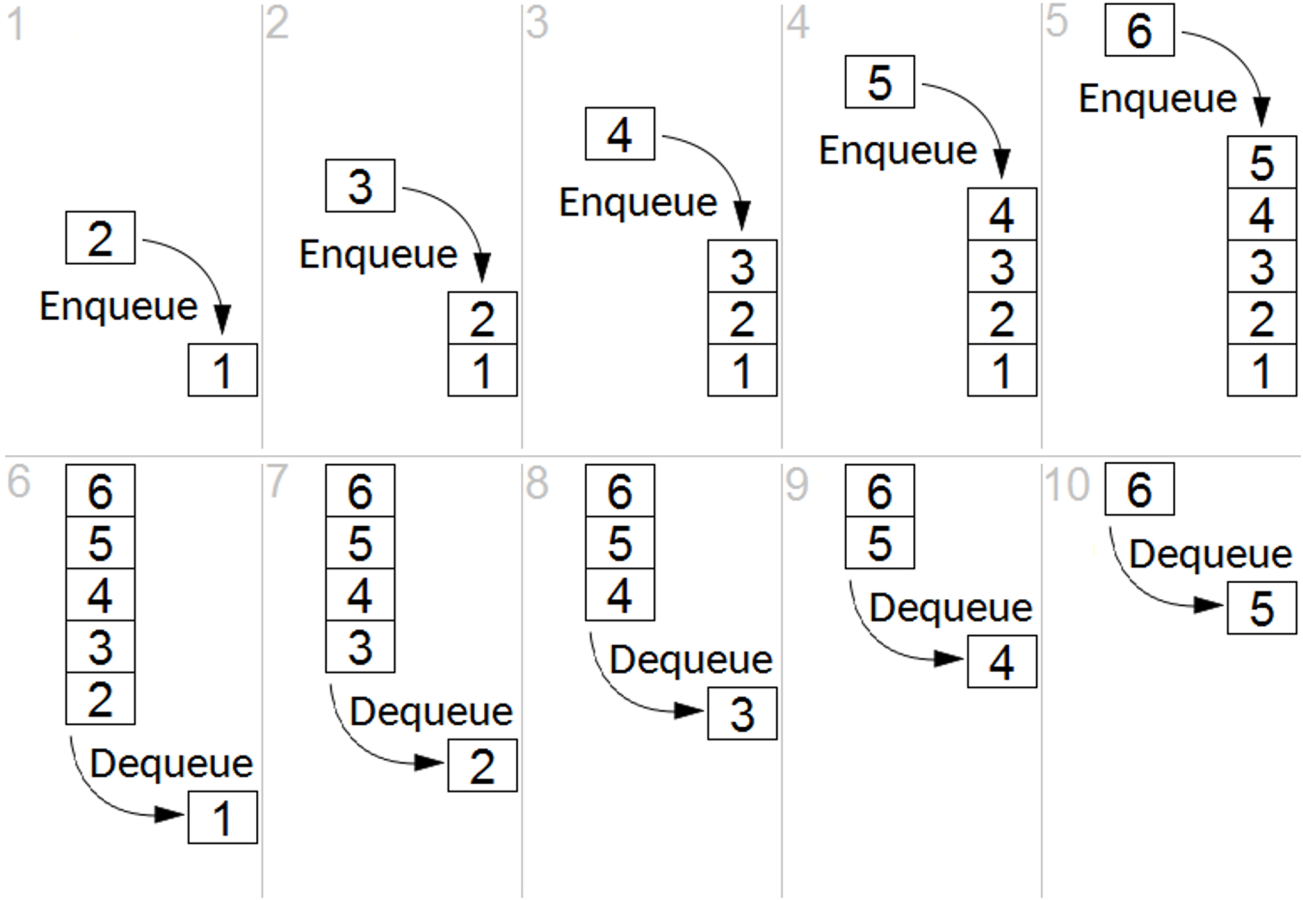
```
const stack = new Stack<string>();
stack.push("A");
stack.push("B");

stack.size(); // Output: 2
stack.peek(); // Output: "B"
stack.size(); // Output: 2
stack.pop();  // Output: "B"
stack.size(); // Output: 1
```

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(First-In/First-Out)



Triển khai Queue

- Bước 1: Định nghĩa interface IQueue
- Bước 2: Triển khai class Queue từ IQueue
- Bước 3: Viết mã phương thức dequeue()
- Bước 4: Viết mã phương thức size()
- Bước 5: Sử dụng Stack

Định nghĩa interface IQueue

Ngăn xếp thường có các phương thức sau:

- **push** thêm một mục vào ngăn xếp
- **pop** trả về phần tử được thêm cuối cùng và xóa nó khỏi ngăn xếp
- **peek** (tùy chọn) trả về phần tử được thêm cuối cùng mà không xóa nó khỏi ngăn xếp

```
interface IQueue<T> {
    enqueue(item: T): void;
    dequeue(): T | undefined;
    size(): number;
}
```

Triển khai class Stack

```
class Queue<T> implements IQueue<T> {
    private storage: T[] = [];

    constructor(private capacity: number = Infinity) { }
}
```

Phương thức enqueue()

```
enqueue(item: T): void {
    if (this.size() === this.capacity) {
        throw Error("Queue has reached max capacity,
you cannot add more items");
    }
    this.storage.push(item);
}
```

Phương thức dequeue()

```
dequeue(): T | undefined {
    return this.storage.shift();
}
```

Phương thức size()

```
size(): number {
    return this.storage.length;
}
```

Sử dụng Queue

```
const queue = new Queue<string>();

queue.enqueue("A");
queue.enqueue("B");

queue.size();     // Output: 2
queue.dequeue();  // Output: "A"
queue.size();     // Output: 1
```

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