Session 15:

STACK VÀ QUEUE TRONG JAVA

Module – Java Fudamental

Phiên bản: 3.0

NỘI DUNG

Rikkei Academy
where the dream begins

01 **Stack (Ngăn Xếp)**

LIFO - Last In, First Out

- Định nghĩa & Nguyên lý
- Các thao tác cơ bản
- Cài đặt trong Java
- Ứng dụng thực tế

02 **Queue (Hàng Đợi)**

FIFO - First In, First Out

- Định nghĩa & Nguyên lý
- Các thao tác cơ bản
- Các loại Queue
- Cài đặt trong Java

03 **So Sánh & Ứng Dụng**

Stack vs Queue

- Bảng so sánh chi tiết
- Khi nào dùng Stack?
- Khi nào dùng Queue?

04 **Thực Hành**

Bài tập & Ví dụ

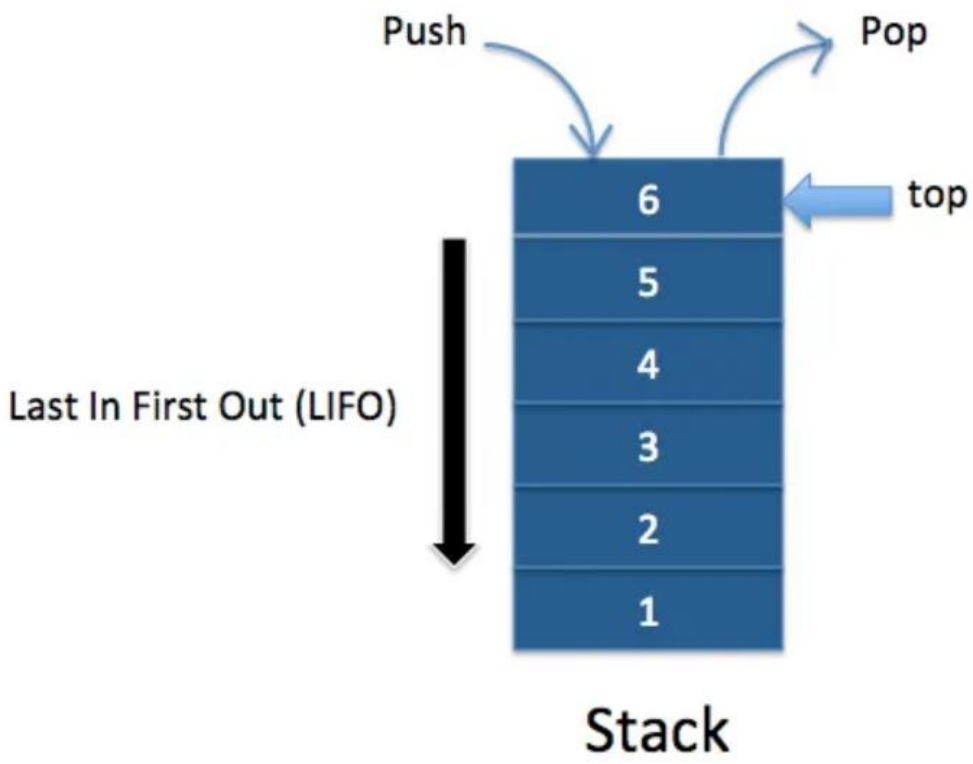
- Kiểm tra dấu ngoặc
- Hàng đợi in tài liệu
- Chuyển đổi nhị phân

Stack

Ngăn Xếp

Cấu trúc dữ liệu hoạt động theo nguyên lý **LIFO**

(Last In, First Out) - Vào sau, ra trước



Push Pop Peek

Stack là gì?

Định nghĩa

Stack (Ngăn xếp) là cấu trúc dữ liệu tuyến tính, trong đó các phần tử chỉ có thể được thêm vào và lấy ra từ một đầu duy nhất (đỉnh stack - top).

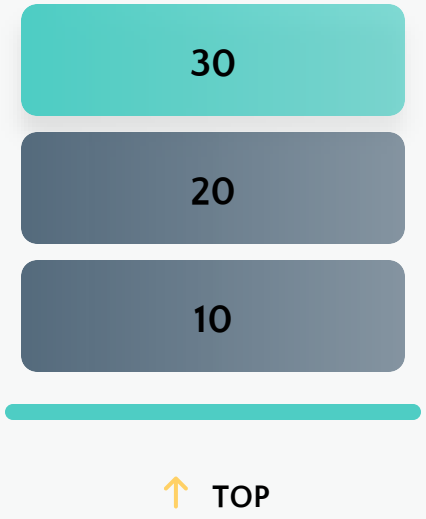
Đặc điểm chính

- Chỉ thao tác ở **một đầu** (top)
- Tuân theo nguyên lý **LIFO**
- Không thể truy cập phần tử ở giữa

Ví dụ thực tế

- Chồng đĩa
- Chồng sách
- Thùng hàng
- Undo/Redo

Minh họa Stack

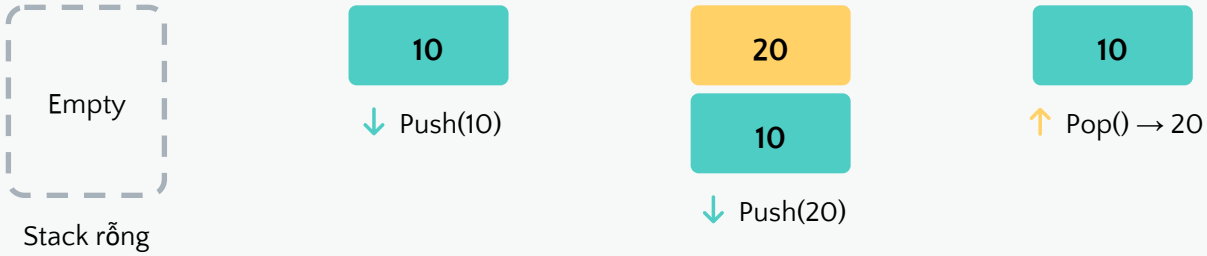


Phần tử **30** vào sau cùng nên sẽ được lấy ra **đầu tiên**

Nguyên Lý LIFO

Last In, First Out – Vào sau, ra trước

Quá trình Push & Pop



Kết quả: Phần tử 20 vào sau cùng nên được lấy ra đầu tiên

Các thao tác

Push
Thêm vào đỉnh

Pop
Lấy từ đỉnh

Lưu ý quan trọng

- Không thể lấy phần tử ở **giữa** hoặc **đáy**
- Thứ tự ra vào **ngược nhau**

Các Thao Tác Cơ Bản

Rikkei Academy
where the dream begins

01 **Push**

Thêm một phần tử vào đỉnh stack

```
stack.push(element)
```

Độ phức tạp:
O(1)

02 **Pop**

Lấy và xóa phần tử ở đỉnh stack

```
element = stack.pop()
```

Độ phức tạp:
O(1)

03 **Peek**

Xem phần tử đỉnh không xóa

```
element = stack.peek()
```

Độ phức tạp:
O(1)

04 **IsEmpty**

Kiểm tra stack có rỗng không

O(1)

```
boolean empty = stack.isEmpty()
```

05 **Size**

Trả về số phần tử trong stack

O(1)

```
int size = stack.size()
```

Cài Đặt Stack - Stack Class

Sử dụng lớp Stack có sẵn trong java.util

```
import java.util.Stack;
// Tạo Stack chứa số nguyên
Stack stack = new Stack<>();
// Thêm phần tử (Push)
stack.push(10);
stack.push(20);
stack.push(30);
// Xem phần tử đỉnh (Peek)
System.out.println("Top: " + stack.peek());
// Output: Top: 30// Lấy phần tử (Pop)int top = stack.pop();
System.out.println("Popped: " + top);
// Output: Popped: 30// Kiểm tra rỗngboolean isEmpty = stack.isEmpty();
// Kích thướcint size = stack.size();
```

Ưu điểm

- Dễ sử dụng, không cần cài đặt thủ công
- Được tối ưu bởi Java
- Tự động mở rộng kích thước

Lưu ý

- Stack là class, không phải interface
- Có thể dùng Deque (ArrayDeque) thay thế

Import thư viện

```
import java.util.Stack;
```

Cài Đặt Stack - Sử Dụng Mảng

Tự cài đặt Stack thủ công để hiểu cơ chế

```java
public class ArrayStack {
    private int[] stackArray;
    private int top;
    private int capacity;
    public ArrayStack(int size) {
        capacity = size;
        stackArray = new int[capacity];
        top = -1;
    }
    public void push(int value) {
        if (top == capacity - 1) {
            System.out.println("Stack đầy!");
            return;
        }
        stackArray[++top] = value;
    }
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack rỗng!");
            return -1;
        }
        return stackArray[top--];
    }
    public int peek() {
        if (isEmpty()) return -1;
        return stackArray[top];
    }
    public boolean isEmpty() {
        return (top == -1);
    }
}
```

Cơ chế hoạt động

1. **Mảng stackArray**
   Lưu trữ phần tử
2. **Biến top**
   Chỉ vị trí đỉnh
3. **Biến capacity**
   Kích thước tối đa

Lưu ý

- Cần kiểm tra tràn stack
- Cần kiểm tra stack rỗng

So Sánh: Stack Class vs Mảng

Tiêu chí	Stack Class	Mảng
Sử dụng tài nguyên	Dùng thư viện chuẩn Java	Tự quản lý mảng và kích thước
Độ dễ sử dụng	Dễ dàng	Phức tạp
Hiệu suất	Tốt, có thể chậm hơn	Nhanh hơn
Tính mở rộng	Tự động mở rộng	Cần tự xử lý tràn mảng
Kiểm soát	Hạn chế	Toàn quyền kiểm soát
Phù hợp khi	Cần triển khai nhanh	Cần hiểu sâu cơ chế

Khuyến nghị: Dùng Stack Class cho dự án thực tế

Học tập: Cài đặt bằng mảng để hiểu sâu

Ứng Dụng Thực Tế của Stack

Undo/Redo

Lưu lại các thao tác để người dùng có thể quay lại trạng thái trước đó .

Ví dụ:
Ctrl+Z trong Word, Photoshop

Biểu thức toán học

Đánh giá biểu thức hậu tố (postfix) , chuyển đổi từ trung tố sang hậu tố.

Ví dụ:
Máy tính, trình biên dịch

Quản lý đệ quy

Stack được sử dụng tự động khi thực hiện các lời gọi đệ quy (call stack).

Ví dụ:
Giai thừa, Fibonacci, DFS

Duyệt đồ thị DFS

Sử dụng trong thuật toán DFS (Depth-First Search) để duyệt đồ thị hoặc cây.

Ví dụ:
Tìm đường, phân tích cú pháp

Ứng dụng khác: Kiểm tra dấu ngoặc đúng/sai, chuyển đổi số nhị phân, quản lý lịch sử trình duyệt web

Stack vs ArrayList

Khi nào nên dùng Stack thay vì ArrayList?

S Stack

Đặc điểm

- Chỉ thao tác ở **đầu** (top)
- Tuân theo nguyên lý **LIFO**
- Không truy cập ngẫu nhiên

Dùng khi

- Cần **undo/redo**
- Xử lý **đệ quy**
- Đánh giá **biểu thức**

A ArrayList

Đặc điểm

- Truy cập **bất kỳ vị trí** nào
- Thêm/xóa ở **bất kỳ đâu**
- Không có nguyên lý cố định

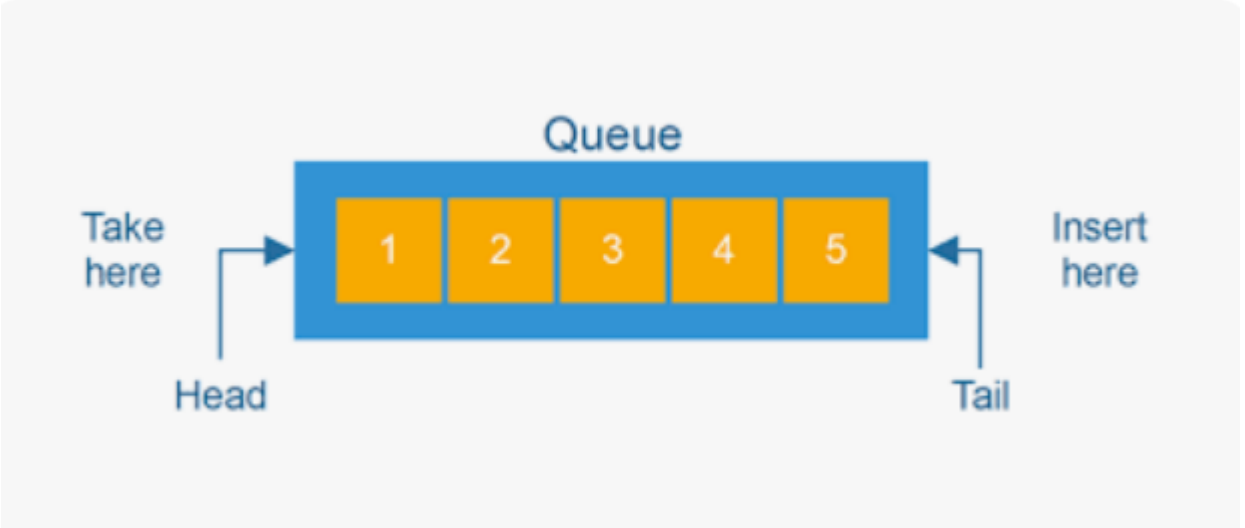
Dùng khi

- Cần **truy cập ngẫu nhiên**
- Thao tác ở **nhiều vị trí**
- Lưu trữ **danh sách động**

Quy tắc vàng: Nếu bài toán có tính chất "vào sau ra trước" → Dùng **Stack**. Nếu cần truy cập linh hoạt → Dùng **ArrayList**.

Queue

Hàng Đợi



Cấu trúc dữ liệu hoạt động theo nguyên lý **FIFO**

(First In, First Out) - Vào trước, ra trước

Enqueue

Dequeue

Peek

Queue là gì?

Định nghĩa

Queue (Hàng đợi) là cấu trúc dữ liệu tuyến tính, trong đó phần tử được thêm vào cuối và lấy ra từ đầu.

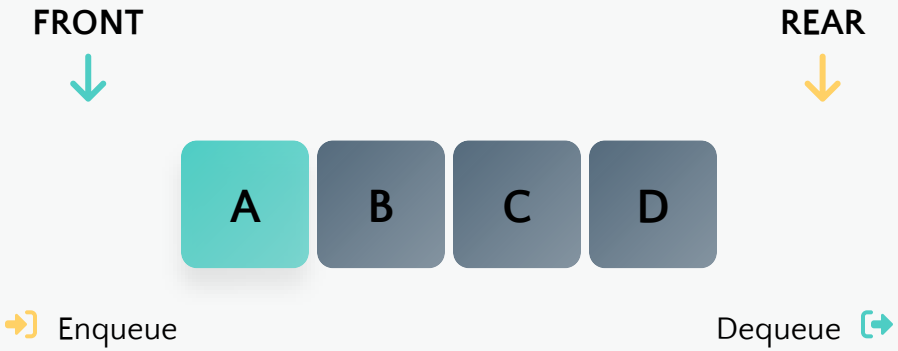
Đặc điểm chính

- Thêm ở **cuối** (rear)
- Lấy ở **đầu** (front)
- Tuân theo nguyên lý **FIFO**

Ví dụ thực tế

- Xếp hàng ATM
- Hàng đợi in
- Đơn hàng
- Xử lý tiến trình

Minh họa Queue

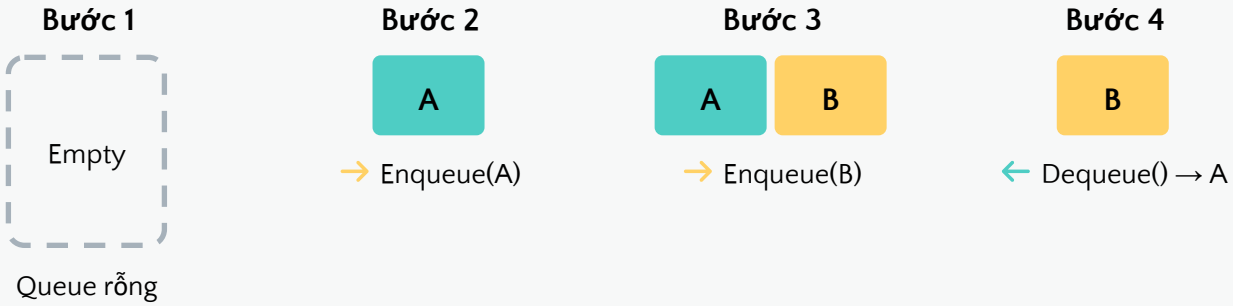


Phần tử **A** vào trước nên được lấy ra **trước**

Nguyên Lý FIFO

First In, First Out - Vào trước, ra trước

Quá trình Enqueue & Dequeue



Kết quả: Phần tử A vào trước nên được lấy ra trước

Các thao tác

Enqueue
Thêm vào cuối

Dequeue
Lấy từ đầu

Lưu ý quan trọng

- Thứ tự ra vào **giống nhau**
- Công bằng cho phần tử **vào trước**

Các Thao Tác Cơ Bản

Rikkei Academy
where the dream begins

01 Enqueue

Thêm một phần tử vào cuối queue

```
queue.offer(element)
```

Độ phức tạp: O(1)

02 Dequeue

Lấy và xóa phần tử ở đầu queue

```
element = queue.poll()
```

Độ phức tạp: O(1)

03 Peek

Xem phần tử đầu không xóa

```
element = queue.peek()
```

Độ phức tạp: O(1)

04 IsEmpty

Kiểm tra queue có rỗng không

```
boolean empty = queue.isEmpty()
```

O(1)

05 IsFull

Kiểm tra queue có đầy không

```
boolean full = queue.isFull()
```

O(1)

Lưu ý: Trong Java Collections, Queue là interface , nên ta dùng các lớp implement như LinkedList hoặc ArrayDeque .

Các Loại Queue

1 Simple Queue

Queue cơ bản theo nguyên tắc FIFO tiêu chuẩn.

Đặc điểm:

- Enqueue ở rear
- Dequeue ở front

2 Circular Queue

Hàng đợi vòng , tận dụng lại vị trí trống.

Ưu điểm:

- Không lãng phí bộ nhớ
- Hiệu quả hơn mảng thường

3 Priority Queue

Hàng đợi ưu tiên , phần tử có độ ưu tiên cao được xử lý trước.

Ứng dụng:

- Xử lý tác vụ theo mức độ ưu tiên
- Thuật toán Dijkstra

4 Deque

Double-ended Queue , thêm/xóa ở cả 2 đầu.

Đặc điểm:

- Thêm/xóa ở front và rear
- Kết hợp Stack và Queue

Lựa chọn: Dùng Simple Queue cho hàng đợi cơ bản, PriorityQueue khi cần ưu tiên, Deque khi cần linh hoạt 2 đầu.

Cài Đặt Queue - LinkedList

Sử dụng Queue interface với LinkedList

```java
import java.util.Queue;
import java.util.LinkedList;
// Tạo Queue sử dụng LinkedList
Queue queue = new LinkedList<>();
// Thêm phần tử (Enqueue)
queue.offer("A");
queue.offer("B");
queue.offer("C");
// Xem phần tử đầu (Peek)
System.out.println("Front: " + queue.peek());
// Output: Front: A// Lấy phần tử (Dequeue)
String front = queue.poll();
System.out.println("Dequeued: " + front);
// Output: Dequeued: A// Kiểm tra rỗngboolean isEmpty = queue.isEmpty();
// Kích thướcint size = queue.size();
```

Ưu điểm của LinkedList

- Thêm/xóa ở 2 đầu **O(1)**
- Không cần dịch chuyển phần tử
- Kích thước động, không giới hạn

Các phương thức chính

offer()	Thêm vào cuối
poll()	Lấy và xóa đầu
peek()	Xem không xóa

Tại sao dùng LinkedList?

ArrayList có độ phức tạp O(n) khi thêm/xóa ở đầu, LinkedList chỉ O(1).

Cài Đặt Queue - Sử Dụng Mảng

Tự cài đặt Queue thủ công

```java
public class ArrayQueue {
    private int[] queueArray;
    private int front, rear, size;
    private int capacity;
    public ArrayQueue(int capacity) {
        this.capacity = capacity;
        queueArray = new int[capacity];
        front = rear = -1;
        size = 0;
    }
    public void enqueue(int value) {
        if (isFull()) {
            System.out.println("Queue đầy!");
            return;
        }
        if (front == -1) front = 0;
        queueArray[++rear] = value;
        size++;
    }
    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue rỗng!");
            return -1;
        }
        int value = queueArray[front++];
        size--;
        return value;
    }
}
```

Cơ chế hoạt động

- **F** **Biến front**
  Chỉ vị trí đầu
- **R** **Biến rear**
  Chỉ vị trí cuối
- **S** **Biến size**
  Số phần tử hiện tại

Nhược điểm

- Lãng phí bộ nhớ
- Không tận dụng vị trí trống

Giải pháp: Dùng Circular Queue hoặc Linked List

PriorityQueue trong Java

Hàng đợi ưu tiên - Phần tử quan trọng được xử lý trước

Đặc điểm

- Phần tử được **sắp xếp** theo độ ưu tiên
- Phần tử **nhỏ nhất** (hoặc lớn nhất) ở đầu
- Không đảm bảo **FIFO** thuần túy

Ứng dụng

- Xử lý tác vụ
- Thuật toán Dijkstra
- Sắp xếp dữ liệu
- Top K elements

Cách sắp xếp

· **Tự nhiên:** implements Comparable

· **Tùy chỉnh:** dùng Comparator

```java
import java.util.PriorityQueue;
// Tạo PriorityQueue cho số nguyên// (Sắp xếp tự nhiên: tăng dần)
PriorityQueue pq = new PriorityQueue<>();
// Thêm phần tử
pq.offer(30);
pq.offer(10);
pq.offer(50);
pq.offer(20);
// Lấy phần tử (ưu tiên nhỏ nhất)
while (!pq.isEmpty()) {
    System.out.print(pq.poll() + " ");
}
// Output: 10 20 30 50// PriorityQueue giảm dần
PriorityQueue pqDesc =
    new PriorityQueue<>((a, b) -> b - a);
```

Lưu ý: peek() và poll() luôn trả về phần tử có độ ưu tiên cao nhất (nhỏ nhất theo mặc định).

Ứng Dụng Thực Tế của Queue

Quản lý tiến trình

Hệ điều hành sử dụng Queue để quản lý các tiến trình chờ xử lý.

Ví dụ:
CPU scheduling, process queue

Hàng đợi in

Các tài liệu được xếp hàng chờ in theo thứ tự gửi đến.

Ví dụ:
Print spooler trong Windows

Xử lý yêu cầu mạng

Server xử lý các request từ client theo thứ tự đến .

Ví dụ:
Web server, message queue

Thuật toán BFS

Sử dụng trong thuật toán BFS (Breadth-First Search) để duyệt đồ thị.

Ví dụ:
Tìm đường đi ngắn nhất, duyệt cây

Ứng dụng khác: Hàng đợi đơn hàng, chat messaging, cache implementation, load balancing

So Sánh Stack vs Queue

Tiêu chí	Stack	Queue
Nguyên lý	LIFO	FIFO
Thêm phần tử	Push (đỉnh)	Enqueue (cuối)
Lấy phần tử	Pop (đỉnh)	Dequeue (đầu)
Thứ tự xử lý	Ngược thứ tự vào	Giống thứ tự vào
Ví dụ thực tế	Chồng đĩa, Undo/Redo	Xếp hàng, In tài liệu
Ứng dụng chính	Đệ quy, DFS, Biểu thức	BFS, Scheduling, Print
Java Class	java.util.Stack	java.util.Queue (interface)

Stack: Dùng khi cần "vào sau ra trước"

Queue: Dùng khi cần "vào trước ra trước"

TỔNG KẾT

Tổng Kết Kiến Thức

Rikkei Academy where the dream begins

S Stack

Nguyên lý LIFO
Last In, First Out – Vào sau, ra trước

Thao tác chính
Push, Pop, Peek tại đỉnh stack

Cài đặt Java
java.util.Stack hoặc mảng/LinkedList

Ứng dụng
Undo/Redo, đệ quy, DFS, biểu thức

Q Queue

Nguyên lý FIFO
First In, First Out – Vào trước, ra trước

Thao tác chính
Enqueue (cuối), Dequeue (đầu), Peek

Cài đặt Java
Queue interface + LinkedList/ArrayDeque

Ứng dụng
Scheduling, print queue, BFS, messaging

Quy tắc quyết định: Nếu bài toán có tính chất "vào sau ra trước" → Dùng **Stack**. Nếu có tính chất "vào trước ra trước" → Dùng **Queue**.

Bài Tập Thực Hành

RIKKEI Academy
where the dream begins

01 Kiểm tra dấu ngoặc

Viết chương trình kiểm tra biểu thức có dấu ngoặc đúng hay không.

Ví dụ:
"(())" ✓ | "(()" ✗ | "{[()]}" ✓

Gợi ý: Dùng Stack

02 Hàng đợi in tài liệu

Mô phỏng hàng đợi in tài liệu, xử lý theo thứ tự đến .

Yêu cầu:
Thêm tài liệu, in tài liệu đầu tiên

Gợi ý: Dùng Queue

03 Chuyển đổi nhị phân

Chuyển đổi số nguyên sang hệ nhị phân sử dụng Stack.

Ví dụ:
10 → 1010 | 5 → 101

Gợi ý: Dùng Stack

04 Xử lý tiến trình

Mô phỏng hệ điều hành xử lý tiến trình theo thứ tự.

Yêu cầu:
Thêm tiến trình, xử lý tiến trình đầu

Gợi ý: Dùng Queue

Mục tiêu: Thông qua các bài tập này, sinh viên sẽ nắm vững cách cài đặt và sử dụng Stack & Queue trong Java.

RIKEI Academy®
where the dream begins

KẾT THÚC

HỌC VIỆN ĐÀO TẠO LẬP TRÌNH CHẤT LƯỢNG NHẬT BẢN