A screenshot of a computer code

Description automatically generated

Đoạn mã trên định nghĩa một lớp nội danh tĩnh (`static class`) có tên là `WorkerThread`, được triển khai từ giao diện `Runnable`. Lớp này đại diện cho một luồng công việc (worker thread) và chứa các công việc cụ thể cần được thực hiện khi luồng này được khởi động.

Giải thích từng phần của mã:

1. `private static final int NUM\_EXECUTIONS = 4`: Đây là một hằng số (`final`) kiểu `int`, được đặt giá trị là 4. Biến này xác định số lần thực hiện công việc trong vòng lặp.

2. `public void run()`: Đây là phương thức `run()` được triển khai từ giao diện `Runnable` và chứa các công việc cần được thực hiện khi luồng `WorkerThread` được khởi động. Trong trường hợp này, phương thức `run()` sẽ thực hiện vòng lặp `for` để thực hiện công việc một số lần được xác định bởi `NUM\_EXECUTIONS`.

3. Trong vòng lặp `for`, các công việc sau được thực hiện:

- `Thread.sleep((long) (Math.random() \* 5000))`: Luồng sẽ ngừng một khoảng thời gian ngẫu nhiên (từ 0 đến 5000 milliseconds) bằng cách gọi phương thức `sleep()` của lớp `Thread`. Điều này tạo ra hiệu ứng đợi ngẫu nhiên giữa các lần thực hiện công việc.

- `Thread currentThread = Thread.currentThread()`: Lấy tham chiếu đến luồng hiện tại bằng cách gọi phương thức `currentThread()` của lớp `Thread`.

- Xây dựng các chuỗi thông tin về luồng hiện tại, bao gồm ID (`currentThread.getId()`), tên (`currentThread.getName()`), và độ ưu tiên (`currentThread.getPriority()`).

- `tableModel.addRow(new Object[]{name})`, `tableModel.addRow(new Object[]{currentThread\_})`, `tableModel.addRow(new Object[]{ID})`: Thêm các thông tin về luồng hiện tại vào một bảng (`tableModel`), để hiển thị hoặc sử dụng trong giao diện người dùng hoặc mục đích khác.

Vì lớp `WorkerThread` triển khai giao diện `Runnable`, nên khi một đối tượng `WorkerThread` được sử dụng để tạo ra một luồng (`Thread`), phương thức `run()` sẽ được gọi trong luồng đó khi luồng được khởi động.

A screenshot of a computer program

Description automatically generated

Đoạn mã trên định nghĩa một lớp nội danh tĩnh (`static class`) có tên là `DataContainer`. Lớp này đại diện cho một đối tượng chứa dữ liệu và cung cấp các phương thức để đặt và lấy dữ liệu từ đối tượng.

Giải thích từng phần của mã:

1. `private Thai example`: Đây là một biến thành viên (`private`) của kiểu `Thai` (có thể là tên lớp hoặc giao diện khác). Biến này được sử dụng để truy cập các thành phần khác của lớp `Thai`, nếu cần thiết.

2. `public DataContainer(Thai example)`: Đây là hàm khởi tạo của lớp `DataContainer`, nhận vào một đối tượng `Thai` (hoặc tên lớp/giao diện khác). Hàm khởi tạo này được sử dụng để khởi tạo biến thành viên `example` của lớp.

3. `private int data`: Đây là một biến thành viên (`private`) kiểu `int` để lưu trữ dữ liệu.

4. `private List<String> notes`: Đây là một biến thành viên (`private`) kiểu `List<String>` để lưu trữ các ghi chú (notes).

5. `public DataContainer()`: Đây là hàm khởi tạo của lớp `DataContainer` mà không nhận tham số. Hàm khởi tạo này được sử dụng để khởi tạo đối tượng `DataContainer` với một danh sách ghi chú (`notes`) rỗng.

6. `public synchronized void setData(int newData)`: Đây là một phương thức đồng bộ (`synchronized`) để đặt dữ liệu mới (`newData`) vào biến `data`. Trong phương thức này, dữ liệu mới được gán cho `data`, được thêm vào danh sách `notes`, và thông báo (`notify()`) cho các luồng đang chờ biết rằng dữ liệu đã sẵn có.

7. `public synchronized int getData() throws InterruptedException`: Đây là một phương thức đồng bộ (`synchronized`) để lấy dữ liệu từ biến `data`. Trong phương thức này, nếu `data` đang có giá trị là 0, luồng sẽ chờ (`wait()`) cho đến khi dữ liệu sẵn có (khi `data` được đặt giá trị khác 0). Sau đó, dữ liệu được lấy ra (`value`), `data` được đặt lại thành 0, và ghi chú ("Consuming data: value") được thêm vào danh sách `notes`.

Lớp `DataContainer` này sử dụng các phương thức đồng bộ (`synchronized`) và các phương thức `wait()` và `notify()` để đảm bảo rằng các luồng sử dụng đối tượng `DataContainer` có thể an toàn chia sẻ và đồng bộ hóa truy cập vào biến `data`.

A screenshot of a computer code

Description automatically generated

Đoạn mã trên định nghĩa một lớp nội danh tĩnh (`static class`) có tên là `DataProducer`, được triển khai từ giao diện `Runnable`. Lớp này đại diện cho một luồng sản xuất dữ liệu và chứa các công việc cụ thể cần được thực hiện khi luồng này được khởi động.

Giải thích từng phần của mã:

1. `private final DataContainer dataContainer`: Đây là một biến thành viên (`private`) kiểu `DataContainer` (một lớp đã được định nghĩa trước đó) để lưu trữ một đối tượng `DataContainer`. Biến này được sử dụng để truy cập và sử dụng phương thức của `DataContainer`.

2. `private final Thai example`: Đây là một biến thành viên (`private`) kiểu `Thai` (có thể là tên lớp hoặc giao diện khác) để lưu trữ một đối tượng `Thai`. Biến này được sử dụng để truy cập và sử dụng phương thức của `Thai`.

3. `public DataProducer(DataContainer dataContainer, Thai example)`: Đây là hàm khởi tạo của lớp `DataProducer`, nhận vào một đối tượng `DataContainer` và một đối tượng `Thai`. Hàm khởi tạo này được sử dụng để khởi tạo biến thành viên `dataContainer` và `example` của lớp.

4. `public void run()`: Đây là phương thức `run()` được triển khai từ giao diện `Runnable` và chứa các công việc cần được thực hiện khi luồng `DataProducer` được khởi động. Trong trường hợp này, phương thức `run()` sẽ thực hiện vòng lặp `for` để thực hiện công việc một số lần xác định (5 lần trong trường hợp này).

5. Trong vòng lặp `for`, các công việc sau được thực hiện:

- `Thread.sleep(1000)`: Luồng sẽ ngừng thực thi trong 1 giây (1000 milliseconds) để giả lập thời gian xử lý.

- `dataContainer.setData(i)`: Gọi phương thức `setData()` của đối tượng `dataContainer` để đặt giá trị `i` vào `dataContainer`. Điều này sẽ thực hiện việc đặt dữ liệu mới vào `dataContainer` và thông báo cho các luồng khác đang chờ biết rằng dữ liệu đã sẵn có.

Lớp `DataProducer` này được sử dụng để tạo ra một luồng (`Thread`) và sử dụng đối tượng `DataContainer` để sản xuất dữ liệu. Trong phương thức `run()`, một vòng lặp đơn giản được sử dụng để sản xuất dữ liệu thông qua `dataContainer`, với mỗi lần lặp tạo ra một giá trị mới cho `dataContainer`.

A screen shot of a computer code

Description automatically generated

Đoạn mã trên định nghĩa một lớp nội danh tĩnh (`static class`) có tên là `DataConsumer`, được triển khai từ giao diện `Runnable`. Lớp này đại diện cho một luồng tiêu thụ dữ liệu và chứa các công việc cụ thể cần được thực hiện khi luồng này được khởi động.

Giải thích từng phần của mã:

1. `private final DataContainer dataContainer`: Đây là một biến thành viên (`private`) kiểu `DataContainer` (một lớp đã được định nghĩa trước đó) để lưu trữ một đối tượng `DataContainer`. Biến này được sử dụng để truy cập và sử dụng phương thức của `DataContainer`.

2. `private final Thai example`: Đây là một biến thành viên (`private`) kiểu `Thai` (có thể là tên lớp hoặc giao diện khác) để lưu trữ một đối tượng `Thai`. Biến này được sử dụng để truy cập và sử dụng phương thức của `Thai`.

3. `public DataConsumer(DataContainer dataContainer, Thai example)`: Đây là hàm khởi tạo của lớp `DataConsumer`, nhận vào một đối tượng `DataContainer` và một đối tượng `Thai`. Hàm khởi tạo này được sử dụng để khởi tạo biến thành viên `dataContainer` và `example` của lớp.

4. `public void run()`: Đây là phương thức `run()` được triển khai từ giao diện `Runnable` và chứa các công việc cần được thực hiện khi luồng `DataConsumer` được khởi động. Trong trường hợp này, phương thức `run()` sẽ thực hiện vòng lặp `for` để thực hiện công việc một số lần xác định (5 lần trong trường hợp này).

5. Trong vòng lặp `for`, các công việc sau được thực hiện:

- `Thread.sleep(2000)`: Luồng sẽ ngừng thực thi trong 2 giây (2000 milliseconds) để giả lập thời gian xử lý.

- `int data = dataContainer.getData()`: Gọi phương thức `getData()` của đối tượng `dataContainer` để lấy dữ liệu từ `dataContainer`. Điều này sẽ thực hiện việc lấy dữ liệu từ `dataContainer`, đặt `data` bằng giá trị trả về từ `getData()`. Ngoài ra, phương thức `getData()` cũng sẽ thay đổi giá trị của `dataContainer` và thông báo cho các luồng khác đang chờ biết rằng dữ liệu đã được tiêu thụ.

Lớp `DataConsumer` này được sử dụng để tạo ra một luồng (`Thread`) và sử dụng đối tượng `DataContainer` để tiêu thụ dữ liệu. Trong phương thức `run()`, một vòng lặp đơn giản được sử dụng để tiêu thụ dữ liệu thông qua `dataContainer`, với mỗi lần lặp lấy dữ liệu từ `dataContainer` bằng cách gọi phương thức `getData()`.