Java Object-Oriented Programming

- Giảng viên : Nguyễn Đức Hiển
  - Email : ndhien@udn.vn
  - Website :

- Thời lượng
  - Lý thuyết : 2 tín chỉ (30 tiết)
  - Thực hành + thảo luận : 1 tín chỉ

Lập trình đa luồng

(Multi-Thread Programming)

Nội dung

- Giới thiệu về luồng (thread)
- Cách tạo luồng trong Java
- Đồng bộ hóa luồng

Giới thiệu

- Một luồng (thread) là gì?
  - Một “dòng điều khiển " trong chương trình
  - Các chương trình thường chỉ có một dòng điều khiển.
  - Với các luồng, bạn có thể có nhiều dòng điều khiển thực hiện cùng lúc trong chương trình
- Ví dụ: Xem xét bộ xử lý từ cơ bản
  - Bạn soạn thảo văn bản và nhấn nút lưu trữ
  - Nó có thể mất một lượng thời gian đáng kể để lưu dữ liệu mới trên đĩa, tất cả điều này được thực hiện với một luồng tách biệt dưới nền (background)
  - Không có các luồng, ứng dụng sẽ bị treo trong khi bạn đang lưu file và không đáp ứng cho đến khi thao tác lưu hoàn thành

Luồng Java

- Khi chương trình Java thực thi hàm main() tức là tạo ra một luồng (luồng main). Trong luồng main:
  - Có thể tạo các luồng con.
  - Chương trình phải đảm bảo main là luồng kết thúc cuối cùng.
  - Khi luồng main ngừng thực thi, chương trình sẽ kết thúc
- Luồng có thể được tạo ra bằng 2 cách:
  - Tạo lớp dẫn xuất từ lớp Thread
  - Tạo lớp hiện thực giao tiếp Runnable.

Tạo luồng

- Trong Java có sẵn lớp Thread. Để tạo một luồng mới ta có thể tạo một lớp thừa kế (extends) lớp Thread và ghi đè phương thức run()
- Ví dụ:

```
public class MyThread extends Thread
{
    private String data;
    public MyThread(String data) {
        this.data = data;
    }
    public void run() {
        System.out.println("I am a thread!");
        System.out.println("The data is : " + data);
    }
}
```

Chạy luồng

- Tạo ra một thể hiện của lớp Thread (hoặc dẫn xuất của nó) và gọi phương thức start()

```
public class ExampleThread
{
    public static void main(String[] args) {
        Thread myThread = new MyThread("my data");
        myThread.start();
        System.out.println("I am the main thread");
    }
}
```

- Khi gọi myThread.start() một luồng mới tạo ra và chạy phương thức run() của myThread.
- myThread.start() trả về gần như ngay lập tức.

Bài tập

- Bài 1. Tạo 2 luồng: luồng 1 hiển thị các số chẵn, luồng 2 hiển thị các số lẻ.
- Bài 2. Tạo 2 luồng: luồng 1 hiển thị các số nguyên tố, luồng 2 hiển thị các số hoàn thiện.

Giao tiếp Runnable

- Ngoài tạo luồng bằng cách thừa kế từ lớp Thread, cũng có một cách khác để tạo luồng trong Java.
- Bạn có thể tạo luồng bằng cách tạo lớp mới hiện thực giao tiếp Runnable và định nghĩa phương thức:
  - public abstract void run()
- Điều này đặc biệt hữu ích nếu bạn muốn để tạo ra một đối tượng Thread nhưng muốn sử dụng một lớp cơ sở khác Thread.

Ví dụ

```
class MyThreadRbl extends JFrame implements Runnable
{
    private String data;

    public MyThreadRbl(String data) {
        this.data = data;
    }

    public void run() {
        System.out.println("I am a thread");
        System.out.println("The data is : " + data);
    }
}
```

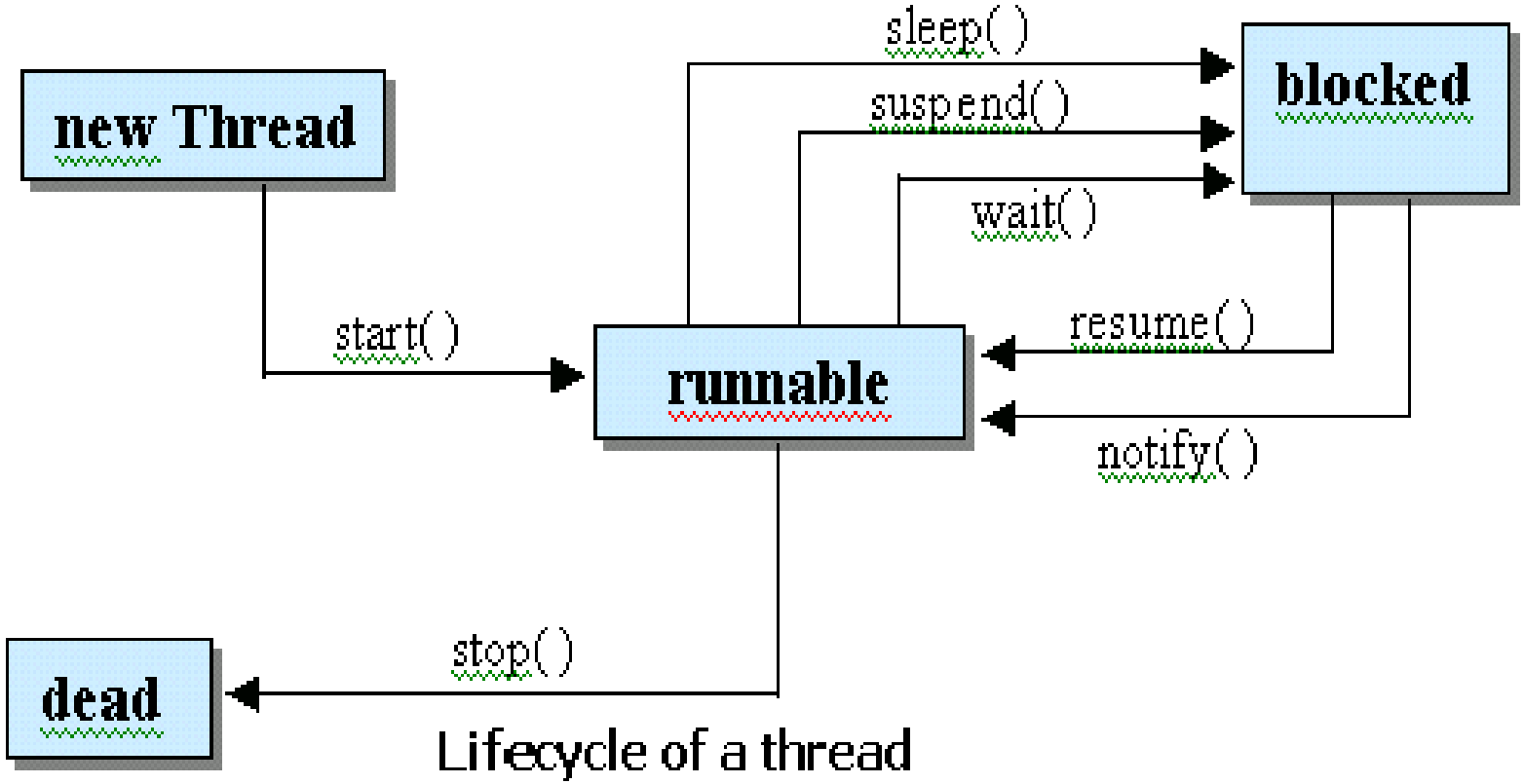
Giao tiếp Runnable

- Để tạo ra một luồng mới từ một đối tượng hiện thực giao tiếp Runnable, bạn phải khởi tạo một đối tượng Thread mới với đối tượng Runnable như đích của nó

```
public class MyThreadStart
{
    public static void main(String[] args) {
        MyThreadRbl thrbl = new MyThreadRbl("my data");
        Thread myThread = new Thread(thrbl);
        myThread.start();
        System.out.println("I am the main thread!");
    }
}
```

- Khi gọi start() trên đối tượng luồng sẽ tạo ra một luồng mới và phương thức run() của đối tượng Runnable sẽ được thực hiện.

Vòng đời của một luồng



Lifecycle of a thread

Điều phối luồng

- JVM chọn luồng để chạy theo “giải thuật quyền ưu tiên cố định”
- Mọi luồng có một quyền ưu tiên trong khoảng phạm vi Thread.MIN_PRIORITY và Thread.MAX_PRIORITY.
- Theo mặc định một luồng được khởi tạo với cùng quyền ưu tiên với luồng tạo ra nó.
- Bạn có thể thay đổi quyền ưu tiên sử dụng phương thức setPriority() của lớp Thread.

Điều phối luồng

- Các luồng với quyền ưu tiên cao có một cơ hội nhận thời gian sử dụng CPU để hoàn thành trước các luồng với quyền ưu tiên thấp hơn.
- JVM sử dụng giải thuật không độc quyền. Vì thế, nếu một luồng quyền ưu tiên thấp đang được chạy, luồng quyền có quyền ưu tiên cao hơn có thể giành quyền sử dụng CPU của nó.
- Nếu các luồng có cùng quyền ưu tiên đang chờ đợi để thực hiện, một luồng tùy ý sẽ được lựa chọn.

Điều phối luồng

- Khi một luồng giành quyền sử dụng CPU, nó sẽ thực hiện cho đến khi một sự kiện sau xuất hiện:
  - Phương thức run() kết thúc
  - Một luồng quyền ưu tiên cao hơn
  - Nó gọi phương thức sleep() hay yield() – nhượng bộ
- Khi gọi yield(), luồng đưa cho các luồng khác với cùng quyền ưu tiên cơ hội sử dụng CPU. Nếu không có luồng nào khác cùng quyền ưu tiên tồn tại, luồng tiếp tục thực hiện
- Khi gọi sleep(), luồng ngủ trong một số mili-giây xác định, trong thời gian đó bất kỳ luồng nào khác có thể sử dụng CPU.

Một số phương thức khác

- Phương thức join()
  - Khi một luồng (A) gọi phương thức join() của một luồng nào đó (B), luồng hiện hành (A) sẽ bị khóa chờ (blocked) cho đến khi luồng đó kết thúc (B).
- Ví dụ:

```
public class MyThreadStart
{
    public static void main(String[] args) {
        Thread thrd = new MyThread("my data");
        thrd.start();
        System.out.println("I am the main thread!");
        thrd.join();
        System.out.println("The 'thrd' finished!");
    }
}
```

Một số phương thức khác

- Phương thức interrupt()
  - Đặt trạng thái luồng ngắt (không ngừng hẳn luồng).
- Phương thức interrupted()
  - Phương thức này trả lại một giá trị boolean cho biết trạng thái ngắt quãng của luồng hiện thời.
  - Phương thức này cũng đặt lại trạng thái của luồng hiện thời thành không ngắt.
- Kết hợp sử dụng hai phương thức này có thể được dùng làm phương pháp yêu cầu một luồng nhượng bộ, ngủ hoặc kết thúc chính nó.

Sự đồng bộ hóa

- Trường hợp nhiều luồng cùng truy cập trên các tài nguyên đồng thời.
  - Đọc/ghi trên cùng một file
  - Sửa đổi cùng một đối tượng/biến
  - …
- Trong những trường hợp này, bạn phải cẩn thận phối hợp các thao tác này như thế nào để các tài nguyên kết thúc trong một trạng thái an toàn.
- Java có sẵn cơ chế cho sự phối hợp này → đồng bộ hóa luồng.

Bài toán Producer/Consumer

- Có hai luồng, một sản xuất và một tiêu thụ cả hai truy cập cùng môt đối tượng CubbyHole (chổ ấm áp).
- CubbyHole là một đối tượng đơn giản lưu giữ một giá trị đơn như nội dung của nó.
- Luồng sản xuất phát sinh ngẫu nhiên các giá trị và cất giữ chúng trong đối tượng CubbyHole
- Luồng tiêu thụ lấy các giá trị này khi chúng được sinh ra bởi luồng sản xuất.

Lớp CubbyHole

```
public class CubbyHole
{
    private int contents = 0;

    public int get() {
        return contents;
    }
    public void put(int value) {
        contents = value;
    }
}
```

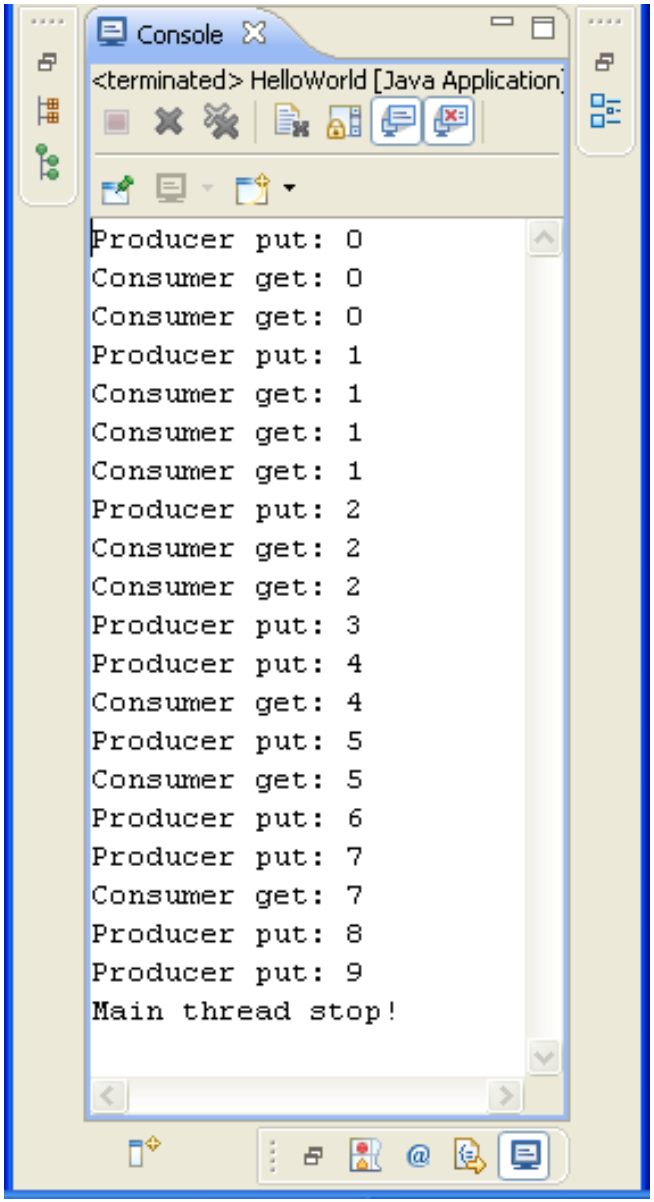
Luồng sản xuất (Producer)

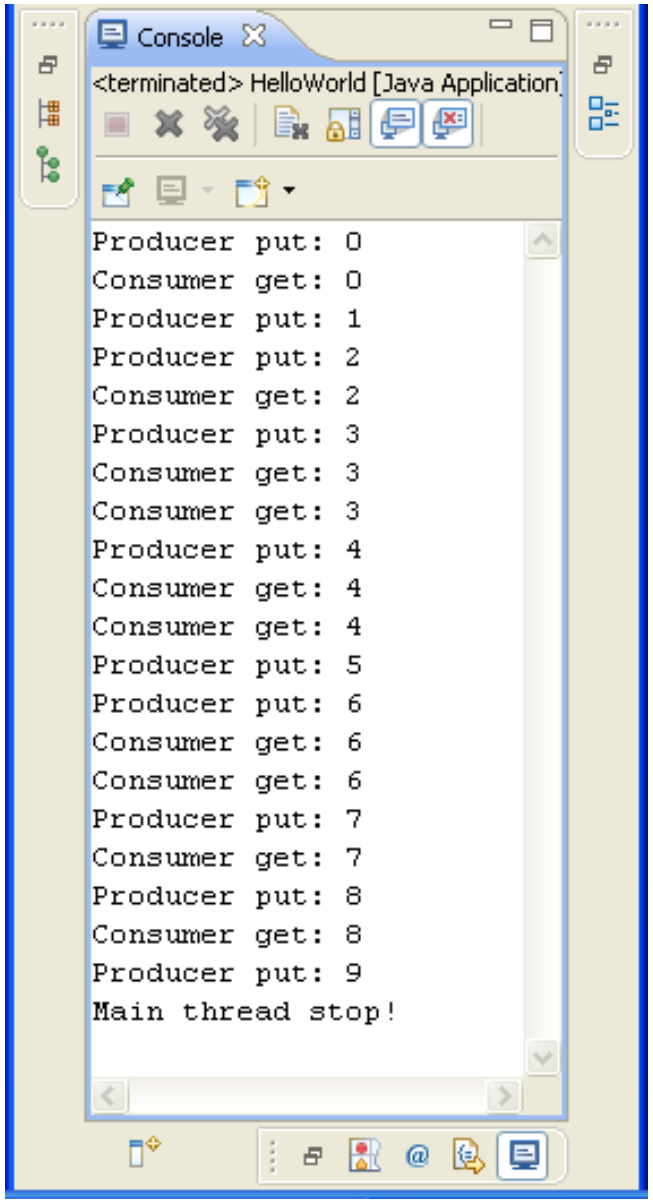
```
public class Producer extends Thread
{
    private CubbyHole cubbyhole;
    private int who;
    public Producer(CubbyHole c, int who) {
        cubbyhole = c;
        this.who = who;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            cubbyhole.put( i);
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) { }
        }
    }
}
```

Luồng tiêu thụ (Customer)

```
public class Consumer extends Thread
{
    private CubbyHole cubbyhole;
    private int who;
    public Consumer(CubbyHole c, int who) {
        cubbyhole = c;
        this.who = who;
    }
    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = cubbyhole.get();
        }
    }
}
```

Bài toán Producer/Customer





Các vấn đề Producer/Customer

- Khi luồng sản xuất sinh ra một giá trị, nó cất giữ nó vào CubbyHole và sau đó luồng tiêu thụ chỉ phải lấy nó một và chỉ một lần.
- Phụ thuộc vào các luồng được điều phối như thế nào
  - Chẳng hạn luồng sản xuất có thể sinh ra hai giá trị trước khi tiêu thụ có thể lấy một.
  - Luồng tiêu thụ có thể lấy cùng giá trị hai lần trước đây sản xuất có được sinh ra giá trị tiếp theo.
- Nếu luồng sản xuất và tiêu thụ truy cập CubbyHole cùng lúc, chúng đã có thể sinh ra một trạng thái mâu thuẫn hay thiếu một giá trị được sản xuất.

Giải pháp đồng bộ hóa

- Xây dựng đối tượng với các phương thức đồng bộ hóa với từ khóa synchronized
- Ví dụ:

```
public class CubbyHole
{
    private int contents;
    private boolean available = false;
    public synchronized int get(int who) {
        ...
    }
    public synchronized void put(int who, int value) {
        ...
    }
}
```

Giải pháp đồng bộ hóa

- Khi một luồng gọi thực hiện một phương thức đồng bộ hóa của một đối tượng, nó sẽ khóa đối tượng đó.
- Khi đó, các phương thức đồng bộ hóa được gọi bởi luồng khác trên đối tượng đó sẽ không được thực hiện cho đến khi đối tượng được mở khóa.

Giải pháp đồng bộ hóa

- Các phương thức đồng bộ hóa ngăn chặn luồng sản xuất và luồng tiêu thụ sửa đổi CubbyHole cùng lúc.
- Tuy nhiên, chúng ta vẫn còn cần phối hợp các luồng sản xuất và tiêu thụ sao cho chúng không sinh ra hay tiêu thụ không đúng thứ tự.
- Các phương thức wait() và notifyAll() được sử dụng để thóa khóa trên một đối tượng và thông báo các luồng đang đợi các chúng có thể có lại điều khiển.

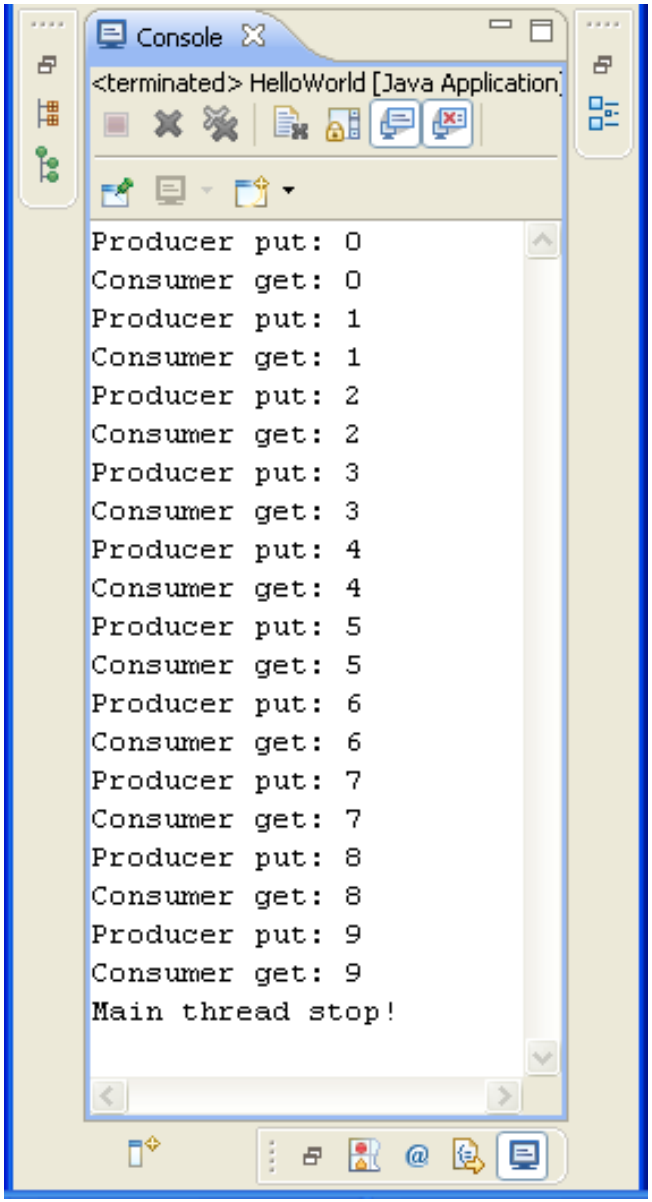
Giải pháp đồng bộ hóa

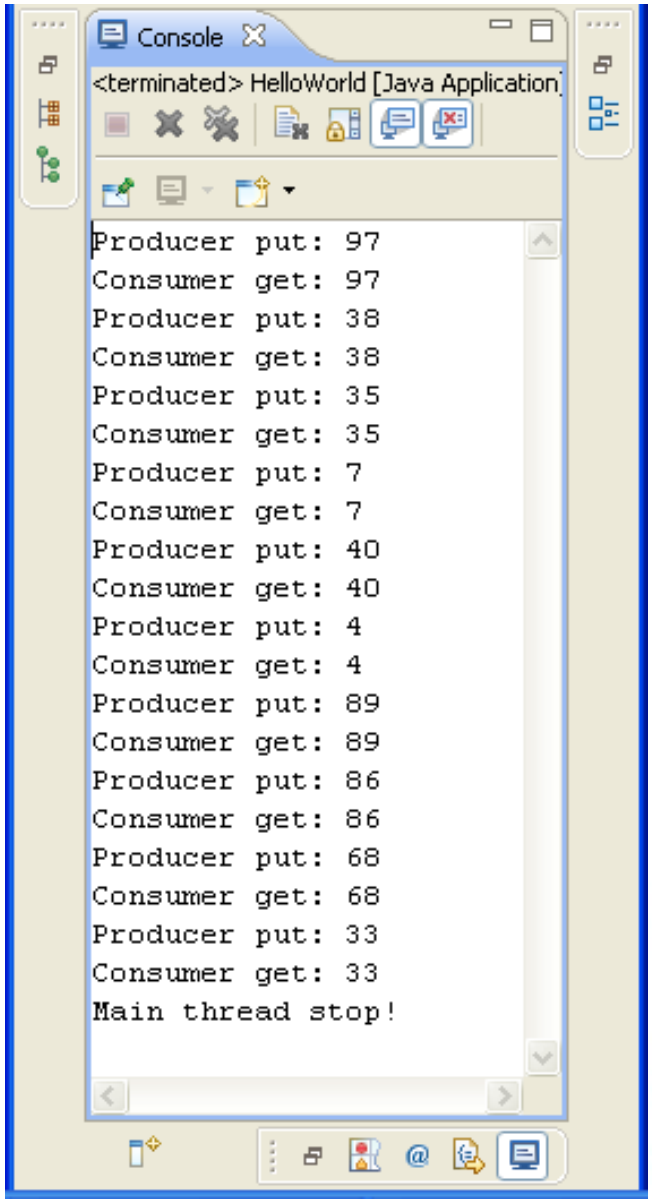
```
public synchronized int get() {
    while (available == false) {
        try {
            //wait for Producer to put value
            wait();
        } catch (InterruptedException e) { }
    }
    available = false;
    //notify Producer that value has been retrieved
    notifyAll();
    return contents;
}
```

Giải pháp đồng bộ hóa

```
public synchronized void put(int value) {
    while (available == true) {
        try {
            //wait for Consumer to get value
            wait();
        } catch (InterruptedException e) { }
    }
    contents = value;
    available = true;
    //notify Consumer that value has been set
    notifyAll();
}
```

Giải pháp đồng bộ hóa





Thanks for listenning!!!