Hackademics Hanoi
Hack your passion, Code your life!

CHƯƠNG 8

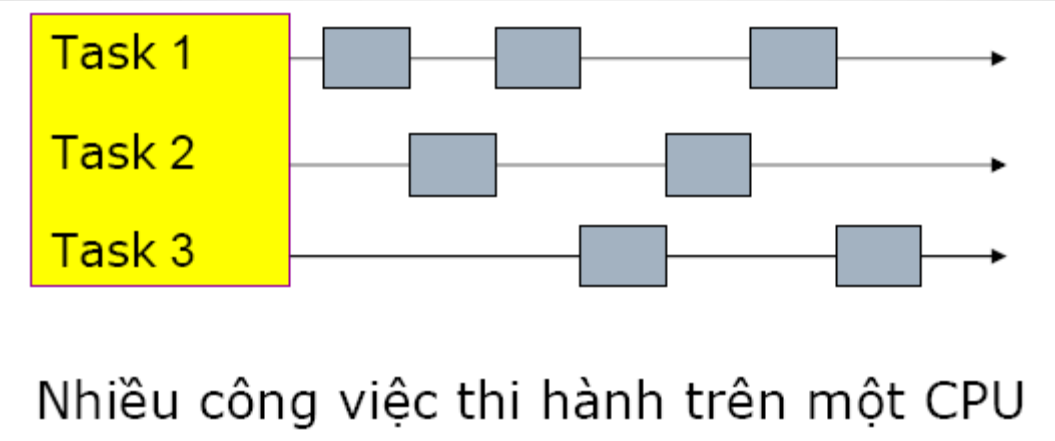
MULTI-THREADING

NỘI DUNG

- Đa nhiệm và đa tuyến
- Tạo lập và sử dụng tuyến
  - Lớp Thread
  - Giao tiếp Runnable
- Đồng bộ hoá các tuyến
- Tuyến ma
- Nhóm tuyến

ĐA NHIỆM (MULTITASKING)

- Đa nhiệm là kỹ thuật cho phép nhiều công việc được thực hiện cùng một lúc trên máy tính.
- Nếu có nhiều CPU, các công việc có thể được thực hiện song song trên từng CPU. Trong trường hợp nhiều công việc cùng chia sẻ một CPU, từng phần của mỗi công việc sẽ được CPU thực hiện xen kẽ.



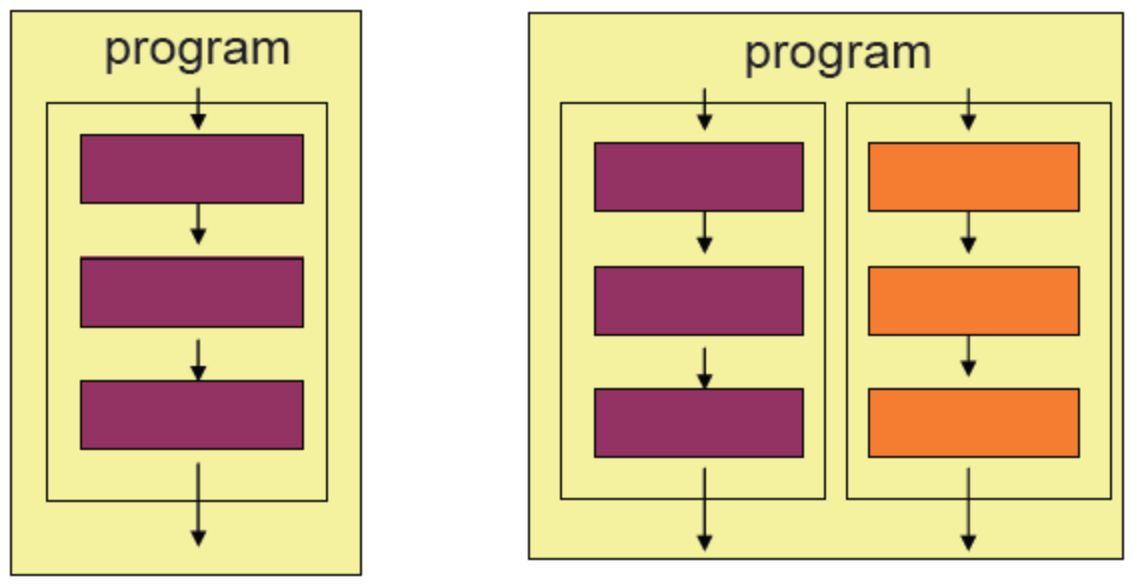
Nhiều công việc thi hành trên một CPU

ĐA NHIỆM (MULTITASKING)

- Hai kỹ thuật đa nhiệm cơ bản:
  - Đa tiến trình (Process-based multitasking): Nhiều chương trình chạy đồng thời. Mỗi chương trình có một vùng dữ liệu độc lập.
  - Đa tuyến (Thread-based multitasking): Một chương trình có nhiều tuyến cùng chạy đồng thời. Các tuyến dùng chung vùng dữ liệu của chương trình.

TUYẾN VÀ ĐA TUYẾN

- Tuyến là mạch thi hành độc lập của một tác vụ trong chương trình.
- Một chương trình có nhiều tuyến thực hiện cùng lúc gọi là đa tuyến.



TẠO TUYẾN

- Tuyến trong Java cũng là các đối tượng.
- Có hai cách để tạo tuyến
  - Thừa kế từ lớp java.lang.Thread
  - Cài đặt giao tiếp java.lang.Runnable

TẠO TUYẾN – CÁCH 1: KẾ THỪA TỪ THREAD

Tạo lớp MyThread kế thừa từ Thread và nạp chồng phương thức run() của lớp Thread.

```
class MyThread extends Thread {
    ....
    public void run() {
        ...
    }
}
```

Tạo và thực thi tuyến.

```
Thread th1 = new MyThread();
Thread th2 = new MyThread();
th1.start();
th2.start();
```

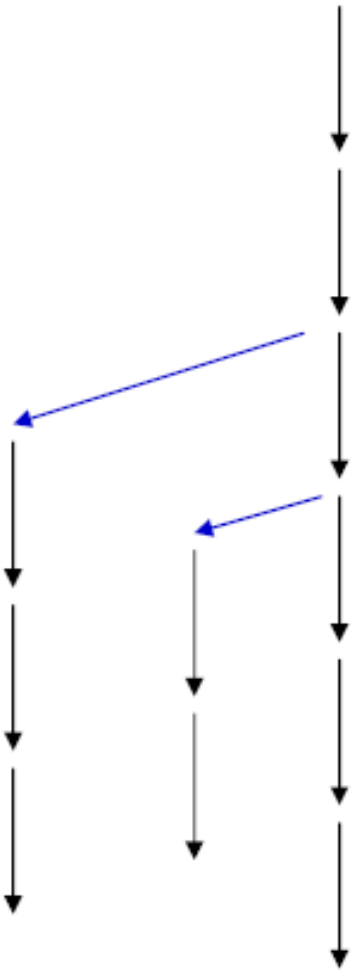
TẠO TUYẾN – CÁCH 1: KẾ THỪA TỪ THREAD

- Khi một tuyến được tạo ra, nó cần gọi start() để đặt tuyến ở trạng thái sẵn sàng. Tiếp theo hệ thống sẽ thực thi các câu lệnh trong run() của tuyến đó.
- Tuyến sẽ kết thúc khi làm hết lệnh trong run() hoặc khi stop() được gọi.

TẠO TUYẾN – CÁCH 1: KẾ THỪA TỪ THREAD

```
MyThread th1 = new MyThread();
MyThread th2 = new MyThread();
...
th1.start();
th2.start();
...
```

Tạo tuyến mới



Sẵn sàng bắt đầu thực thi tuyến

TẠO TUYẾN – CÁCH 2: CÀI ĐẶT RUNNABLE

Trong trường hợp lớp đã kế thừa từ một lớp khác, cần cài đặt giao tiếp Runnable để lớp có thể là một tuyến.

Runnable có duy nhất một phương thức run().

```
class MyClass extends SomeClass
    implements Runnable {
    ....
    public void run() {
        ...
    }
}
```

Tạo và thực thi tuyến.

```
Thread th1 = new Thread(new MyClass());
Thread th2 = new Thread(new MyClass());
th1.start();
th2.start();
```

ĐỘ ƯU TIÊN

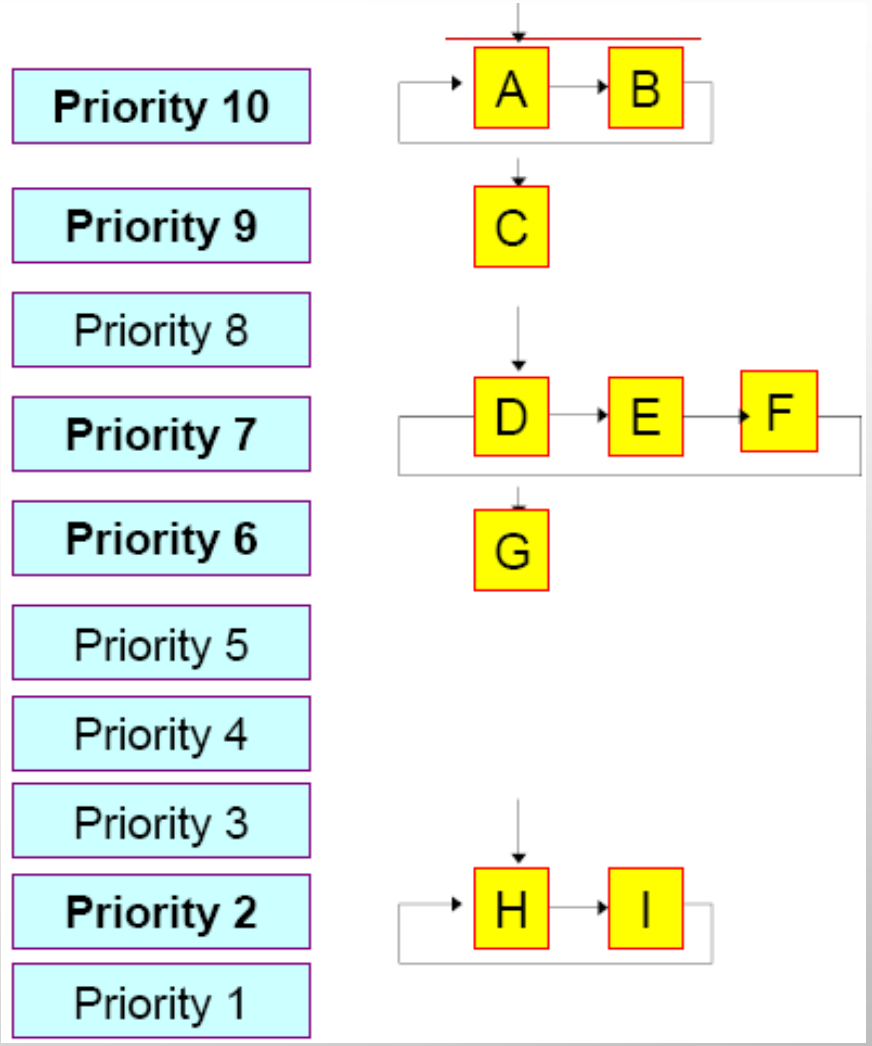
- Các tuyến trong Java có độ ưu tiên từ*Thread.MIN_PRIORITY* (giá trị 1) đến *Thread.MAX_PRIORITY* (giá trị 10)
- Tuyến có độ ưu tiên càng cao thì càng sớm được thực hiện và hoàn thành.
- Độ ưu tiên mặc định của các tuyến là *Thread.NORM_PRIORITY* (giá trị 5).
- Một tuyến mới sẽ thừa kế độ ưu tiên từ tuyến tạo ra nó.

BỘ LẬP LỊCH

- Bộ lập lịch (scheduler) của Java quản lý các tuyến theo cơ chế phân chia thời gian (timeslicing).
- Từng tuyến sẽ được cấp một khoảng thời gian ngắn (time quantum) để sử dụng CPU.
- Trong khi thực thi, nếu đã hết thời gian được cấp thì dù chưa kết thúc tuyến cũng phải tạm dừng để cho các tuyến khác cùng độ ưu tiên dùng CPU.
- Các tuyến cùng độ ưu tiên luân phiên sử dụng CPU theo kiểu xoay vòng (round-robin).

BỘ LẬP LỊCH

- Ví dụ: Tuyến A và B sẽ luân phiên nhau thực thi cho đến khi kết thúc. Tiếp theo tuyến C sẽ thực thi đến khi kết thúc.
- Tiếp theo tuyến D, E và F sẽ luân phiên thực thi đến khi kết thúc. Tiếp theo tuyến G thực thi đến khi kết thúc. Cuối cùng tuyến H và I luân phiên thực thi đến khi kết thúc.
- Nhận xét:Các tuyến có độ ưu tiên thấp sẽ có nguy cơ bị trì hoãn vô hạn định.



VÍ DỤ VỀ ĐA TUYẾN

- Tạo ra 3 tuyến với độ ưu tiên mặc định.
- Công việc của mỗi tuyến là ngủ trong một thời gian ngẫu nhiên từ 0 đến 5 giây. Sau khi ngủ xong, các tuyến sẽ thông báo ra màn hình.

VÍ DỤ VỀ ĐA TUYẾN (TT)

```
class PrintThread extends Thread {
   private int sleepTime;
   public PrintThread( String name ){
        super( name );
        sleepTime = (int)(Math.random()*5000);
        System.out.println( getName() +
              " have sleep time: " + sleepTime);
   }
```

VÍ DỤ VỀ ĐA TUYẾN (TT)

```
    // method run is the code to be executed by new thread
    public void run(){
        try{
            System.out.println(getName()+" starts to sleep");
            Thread.sleep( sleepTime );
        }
        //sleep() may throw an InterruptedException
        catch(InterruptedException e){
            e.printStackTrace();
        }
        System.out.println( getName() + " done sleeping" );
    }
}
```

VÍ DỤ VỀ ĐA TUYẾN (TT)

```
public class ThreadTest{
    public static void main( String [ ] args ){
        PrintThread thread1 = new PrintThread( "thread1" );
        PrintThread thread2 = new PrintThread( "thread2" );
        PrintThread thread3 = new PrintThread( "thread3" );
        System.out.println( "Starting threads" );
        thread1.start();  //start and ready to run
        thread2.start();  //start and ready to run
        thread3.start();  //start and ready to run
        System.out.println( "Threads started, main ends\n" );
    }
}
```

Ví dụ về đa tuyến (tt)

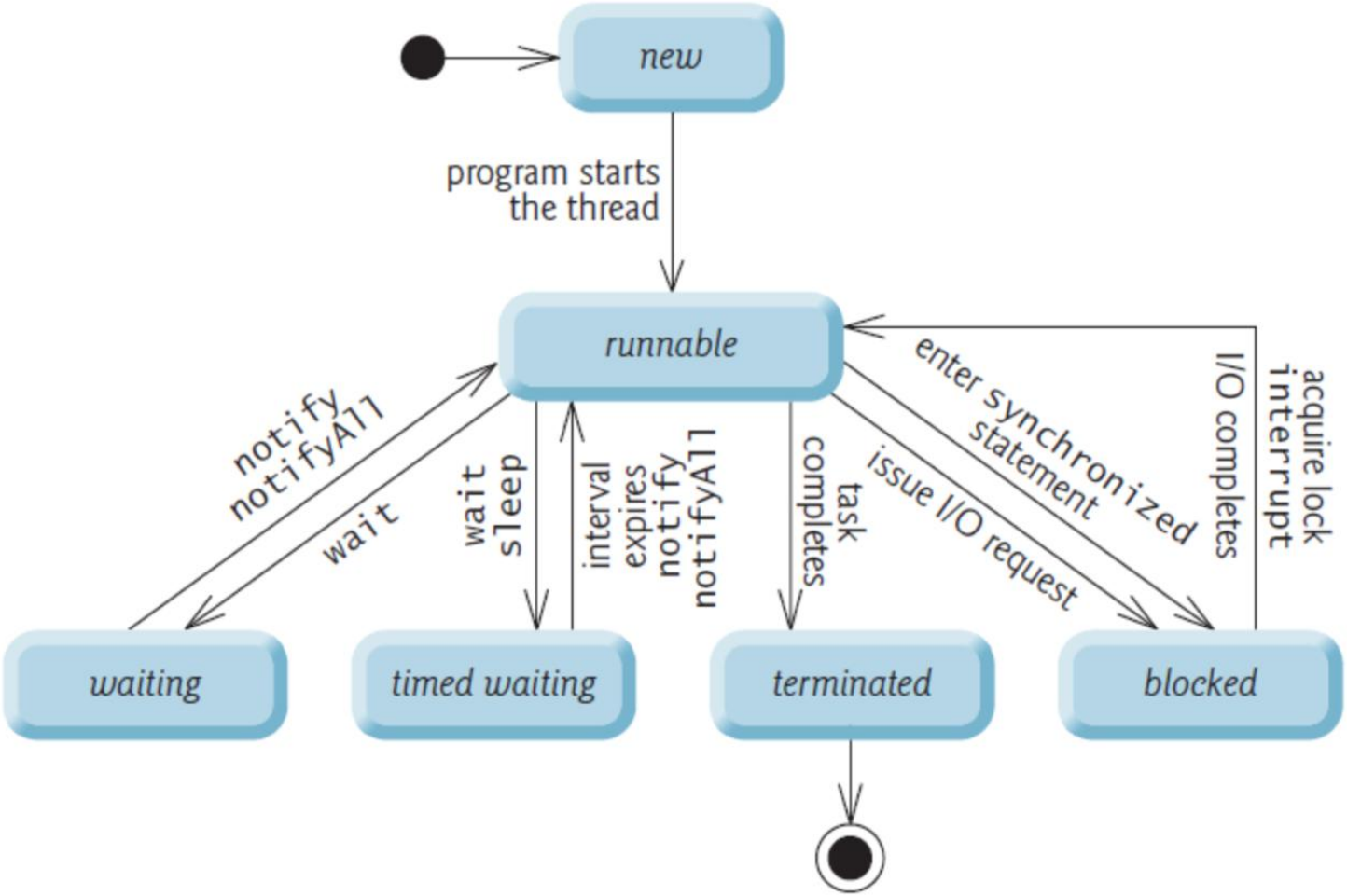
```
thread1 have sleep time: 622
thread2 have sleep time: 4543
thread3 have sleep time: 1622
Starting threads
Threads started, main ends

thread1 starts to sleep
thread2 starts to sleep
thread3 starts to sleep
thread1 done sleeping
thread3 done sleeping
thread2 done sleeping
```

MỘT SỐ PHƯƠNG THỨC CỦA THREAD

- void sleep(long millis); // ngủ
- void yield(); // nhường điều khiển
- void interrupt(); // ngắt tuyến
- void join(); // yêu cầu chờ kết thúc
- void suspend(); // deprecated
- void resume(); // deprecated
- void stop(); // deprecated

Vòng đời của tuyến



ĐỒNG BỘ HÓA TUYẾN

- Việc các tuyến trong chương trình cùng truy nhập vào một đối tượng có thể sẽ đem lại kết quả không như mong muốn. Ví dụ: Tuyến A cập nhật đối tượng X và tuyến B đọc dữ liệu từ X. Rất có thể xảy ra sự cố là tuyến B đọc dữ liệu chưa được cập nhật.
- Đồng bộ hoá tuyến (thread synchronization) giúp cho tại mỗi thời điểm chỉ có một tuyến có thể truy nhập vào đối tượng còn các tuyến khác phải đợi. Ví dụ: Trong khi tuyến A cập nhật X thì tuyến B chưa được đọc.

ĐỒNG BỘ HÓA TUYẾN

- Dùng từ khoá synchronized trên các phương thức để thực hiện đồng bộ hoá.
- Đối tượng khai báo phương thức synchronized sẽ có một bộ giám sát (monitor). Bộ giám sát đảm bảo tại mỗi thời điểm chỉ có một tuyến được gọi phương thức synchronized.
- Khi một tuyến gọi phương thức synchronized, đối tượng sẽ bị khoá. Khi tuyến đó thực hiện xong phương thức, đối tượng sẽ được mở khoá.

ĐỒNG BỘ HÓA THREAD

- Trong khi thực thi phương thức synchronized, một tuyến có thể gọi wait() để chuyển sang trạng thái chờ cho đến khi một điều kiện nào đó xảy ra. Khi tuyến đang chờ, đối tượng sẽ không bị khoá.
- Khi thực hiện xong công việc trên đối tượng, một tuyến cũng có thể thông báo (notify) cho các tuyến khác đang chờ để truy nhập đối tượng.
- Deadlock: Tuyến A chờ tuyến B và tuyến B cũng chờ tuyến A.

Quan hệ Producer - Consumer

- Giả sử có 2 tuyến: Producer ghi dữ liệu vào một buffer và Consumer đọc dữ liệu từ buffer

  => Cần có sự đồng bộ hoá nếu không dữ liệu có thể bị Producer ghi đè trước khi Consumer đọc được hoặc Consumer có thể đọc một dữ liệu nhiều lần khi Producer chưa sản xuất kịp.

Quan hệ Producer - Consumer

- Giải pháp đồng bộ hoá:
  - Trước khi tiếp tục sinh dữ liệu và đưa vào buffer, Producer phải chờ (wait) Consumer đọc xong dữ liệu từ buffer.
  - Khi Consumer đọc xong dữ liệu, nó sẽ thông báo (notify) cho Producer biết để tiếp tục sinh dữ liệu.
  - Nếu Consumer thấy trong buffer không có dữ liệu hoặc dữ liệu đó đã được đọc rồi, nó sẽ chờ (wait) cho tới khi nhận được thông báo có dữ liệu mới.
  - Khi Producer sản xuất xong dữ liệu, nó thông báo (notify) cho Consumer biết.

Ví dụ về P - C: Không đồng bộ

```
class Buffer{
    private int buffer = -1;
    public void set( int value ){
        buffer = value;
    }
    public int get(){
        return buffer;
    }
}
```

Ví dụ về P - C: Không đồng bộ

```
class Producer extends Thread
{
    private Buffer sharedBuffer;
    public Producer( Buffer shared )
    {
      super( "Producer" );
      sharedBuffer = shared;
    }
```

Ví dụ về P - C: Không đồng bộ

```
public void run(){
  for ( int count = 1; count <= 5; count++ ){
    try{
      Thread.sleep((int)(Math.random() * 3000));
      sharedBuffer.set( count );
      System.out.println( "Producer writes " + count);
    } catch (InterruptedException e){
      e.printStackTrace();
    }
  }
  System.out.println( getName() + " finished.");
}
}
```

Ví dụ về P - C: Không đồng bộ

```
class Consumer extends Thread
{
   private Buffer sharedBuffer;
   public Consumer( Buffer shared )
   {
        super( "Consumer" );
        sharedBuffer = shared;
   }
```

Ví dụ về P - C: Không đồng bộ

```
public void run(){
    for ( int count = 1; count <= 5; count++ ){
        try{
            Thread.sleep((int)(Math.random() *3000));
            System.out.println( "Consumer reads " +
                                sharedBuffer.get());
        }catch (InterruptedException e){
            e.printStackTrace();
        }
    }
    System.out.println( getName() + " finished.");
  }
}
```

Ví dụ về P - C: Không đồng bộ

```
public class SharedBufferTest1{
    public static void main( String [] args ){
        //create shared object used by threads
        Buffer sharedBuffer = new Buffer();
        //create producer and consumer objects
        Producer producer=new Producer(sharedBuffer);
        Consumer consumer=new Consumer(sharedBuffer);
        producer.start();    // start producer thread
        consumer.start();    // start consumer thread
    }
}
```

Kết quả khi không đồng bộ

```
----- Consumer đọc -1
Producer ghi 1
----- Consumer đọc 1
----- Consumer đọc 1
Producer ghi 2
Producer ghi 3
----- Consumer đọc 3
Producer ghi 4
----- Consumer đọc 4
Consumer finished.
Producer ghi 5
Producer kết thúc.
```

VÍ DỤ VỀ P – C: CÓ ĐỒNG BỘ

```
class Buffer{ // Thiết kế lại lớp Buffer
    private int buffer = -1;
    private boolean writable = true;
    public synchronized void set( int value ){
        while (!writable){
            try{
                wait();
            }catch (InterruptedException e){
                e.printStackTrace();
            }
        }
        buffer = value;
        writable = false;
        notify();
    }
```

VÍ DỤ VỀ P – C: CÓ ĐỒNG BỘ

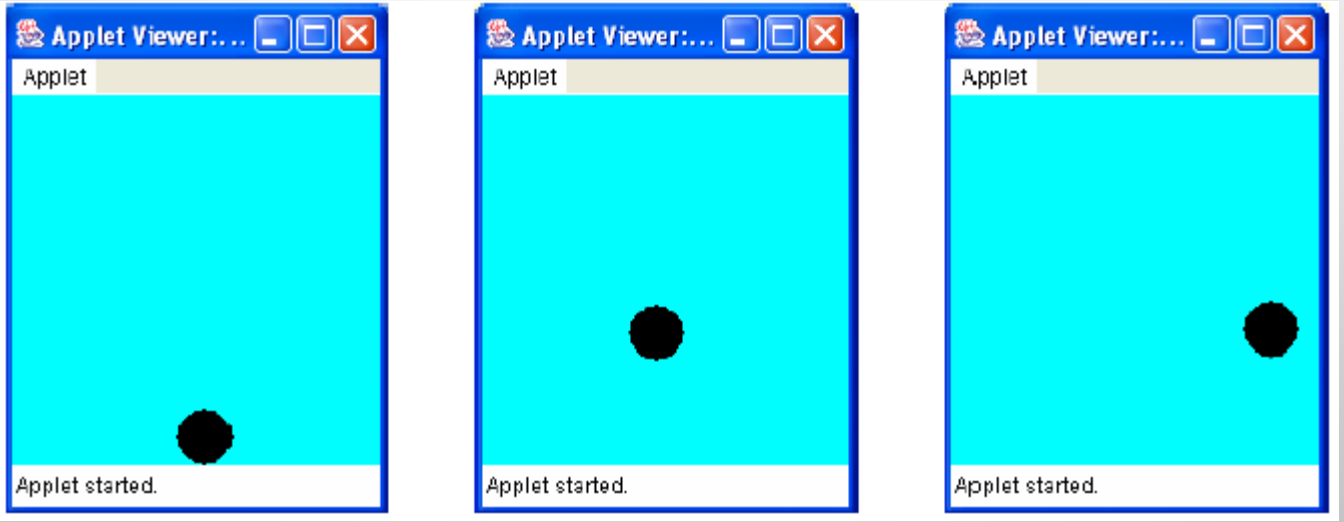
```
public synchronized int get(){
        while( writable ){
                try{
                        wait();
                } catch (InterruptedException e){
                        e.printStackTrace();
                }
        }
        writable = true;
        notify();
        return buffer;
    }
}
```

Kết quả khi có đồng bộ

```
Producer ghi 1
----- Consumer đọc 1
Producer ghi 2
----- Consumer đọc 2
Producer ghi 3
----- Consumer đọc 3
Producer ghi 4
----- Consumer đọc 4
Producer ghi 5
----- Consumer đọc 5
Producer kết thúc.
Consumer finished.
```

TẠO TUYẾN TỪ GIAO TIẾP RUNNABLE

- Một lớp có thể trở thành một tuyến khi cài đặt giao tiếp Runnable (giao tiếp này chỉ có một phương thức run() duy nhất).
- Ví dụ: Tạo applet có quả bóng chạy



TẠO TUYẾN TỪ GIAO TIẾP RUNNABLE

```
import java.awt.*;
import java.applet.*;
public class BallFlying extends Applet implements Runnable{
    Thread animThread = null;
    int ballX= 0, ballY=50;
    int dx=1, dy=2;
    boolean stopRun = false;
    public void start(){        //applet starts
        if (animThread == null){
                animThread= new Thread(this);
                animThread.start();
        }
    }
```

TẠO TUYẾN TỪ GIAO TIẾP RUNNABLE

```
public void stop(){ // applet stops
        stopRun = true;
}
publicvoid run(){
        this.setBackground(Color.CYAN);
        while (! stopRun){
                moveBall();
                delay(5);
        }
}
private void delay(intmiliSeconds){
        try{
                Thread.sleep(miliSeconds);
        }catch (Exception e){
                System.out.println("Sleeperror !");
        }
}
```

TẠO TUYẾN TỪ GIAO TIẾP RUNNABLE

```
    private void moveBall(){
        ballX += dx;
        ballY += dy;
        if (ballY> getSize().height -30)  dy =- dy;
        if (ballX> getSize().width -30)   dx =- dx;
        if (ballY< 0)                     dy =- dy;
        if (ballX< 0)                     dx =- dx;
        repaint();
    }
    publicvoid paint(Graphicsg){
        g.fillOval(ballX,ballY, 30, 30);
    }
}
```

TUYẾN MA (DAEMON THREAD)

- Tuyến ma thường là tuyến hỗ trợ môi trường thực thi của các tuyến khác. Ví dụ: garbage collector của Java là một tuyến ma.
- Chương trình kết thúc khi tất cả các tuyến không phải tuyến ma kết thúc.
- Các phương thức với tuyến ma:
  - void setDaemon(boolean isDaemon); // đặt tuyến trở thành tuyến ma
  - boolean isDaemon(); // kiểm tra tuyến có phải tuyến ma không

NHÓM TUYẾN (THREAD GROUP)

- Các tuyến có thể được đưa vào trong cùng một nhóm thông qua lớp ThreadGroup. Ví dụ: nhóm tuyến tìm kiếm dữ liệu trên các tập dữ liệu khác nhau.
- Một nhóm tuyến chỉ có thể xử lý trên các tuyến trong nhóm, ví dụ: ngắt tất cả các tuyến.
- Có thể tạo ra các nhóm tuyến là nhóm con của một nhóm tuyến khác.
- Nhóm tuyến đặc biệt: system, main

LỚP TIMER

- Hai lớp liên quan tới xử lý công việc theo thời gian:
  - javax.swing.Timer
  - java.util.Timer
- Lớp java.swing.Timer
  - Đơn giản, dễ dùng trên GUI
- Lớp java.util.Timer
  - Nhiều tính năng hơn java.swing.Timer

VÍ DỤ: ĐẾM NGƯỢC

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class CountDown extends Applet implements
    ActionListener{
    private TextField timeField;
    private Button startButton;
    private Button stopButton;
    private javax.swing.Timer timer;
    private int count;
    public void init(){
        timeField = new TextField(6);
```

VÍ DỤ: ĐẾM NGƯỢC

```
        timeField.setFont(new Font("sansserif", Font.PLAIN, 18));
        startButton = new Button("Start");
        stopButton = new Button("Stop");
        add(timeField);
        add(startButton);
        add(stopButton);
        startButton.addActionListener(this);
        stopButton.addActionListener(this);
        timer = new javax.swing.Timer(10, this);
        count = 0;
    } // end init()
```

VÍ DỤ: ĐẾM NGƯỢC

```
    public void actionPerformed(ActionEvent e){
        if (e.getSource() == startButton)
          timer.start();
        else if (e.getSource() == stopButton)
          timer.stop();
        else {
          count++;
          int hsecond = count%100;
          int totalSecond = (count/100);
          int h = totalSecond/3600;
          int secondLeft = totalSecond%3600;
          int m = secondLeft/60;
          int s = secondLeft%60;
          timeField.setText("" + h + ":" + m + ":" + s + ":" + hsecond);
        }
    }
}
```

HẾT

CHƯƠNG 8