DEVMASTER.VN

Bài 12
Đa luồng

Mục tiêu

- Giới thiệu luồng – Thread
- Tạo và sử dụng luồng
- Quản lý luồng
- Luồng chính – deamon thread
- Đa luồng
- Khóa và bất đồng bộ
- Deadlock và k

Giới thiệu

- Một **process** là một **chương trình** được thực hiện.
- Mỗi **process** có nguồn tài nguyên run-time của riêng mình chẳng hạn như **dữ liệu**, **biến**, **không gian bộ nhớ**.
- **Thread** là đơn vị cơ bản mà hệ điều hành phân bổ thời gian xử lý.
- **Thread** là thực thể trong một **process** có thể được hoạch định thực hiện.
- Một **process** được bắt đầu với **thread** duy nhất gọi là **thread chính**.

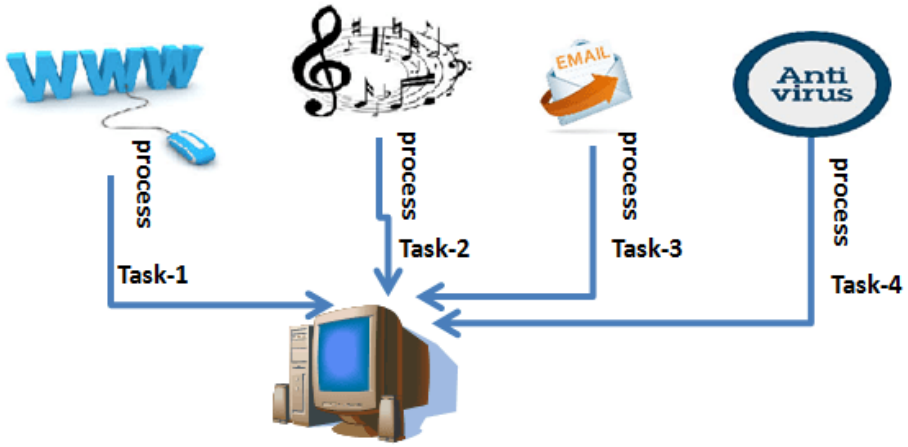
Giới thiệu

- **Thread** có thể chạy **độc lập** thời gian thực (**run-time**)
- **Thread** là đơn vị nhỏ nhất của mã thực thi trong ứng dụng mà thực hiện nhiệm vụ, công việc cụ thể.
- Nhiều **Thread** có thể thực hiện **đồng thời**, tạo điều kiện thực hiện một số **nhiệm vụ cùng lúc** trong ứng dụng.

Giới thiệu

Ứng dụng thread trong lập trình là:

- Phát nhạc và hiển thị hình ảnh cùng lúc.
- Hiển thị nhiều hình ảnh trên màn hình.
- Hiển thị di chuyển mẫu văn bản hoặc hình ảnh trên màn hình.



Multitasking On a Desktop System

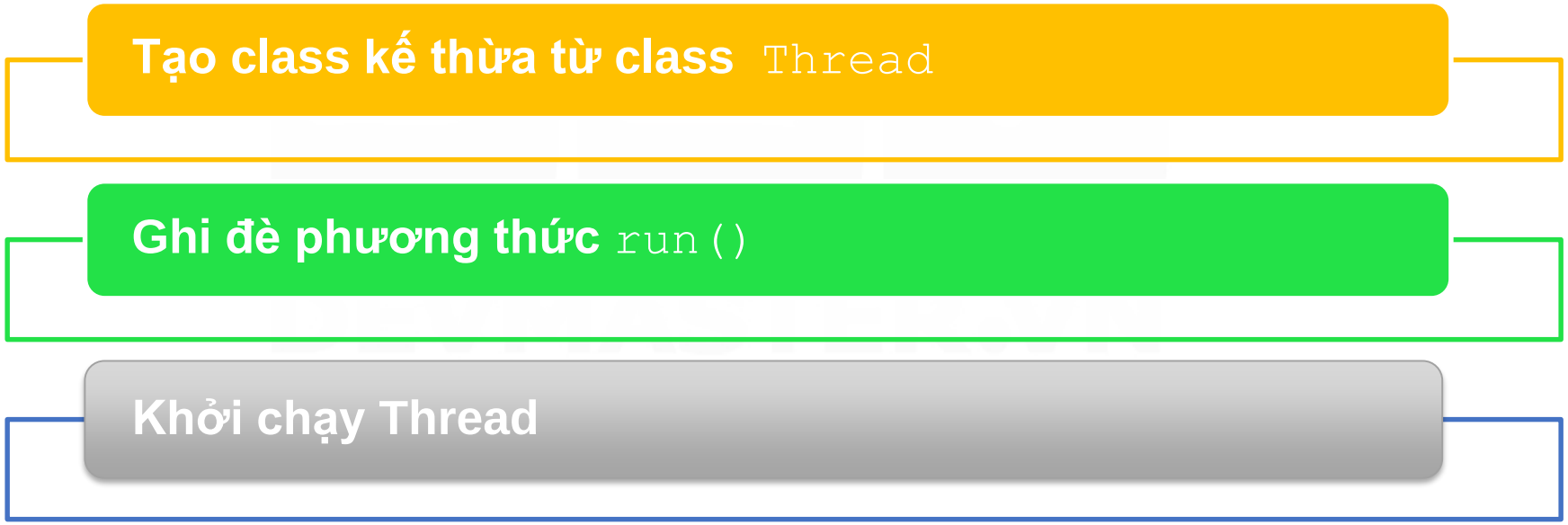
Tạo và sử dụng

Có 2 cách để tạo Thread.

- Tạo lớp kế thừa lớp `Thread`.
- Thực thị interface `Runnable`.

Kế thừa class Thead

Các bước tạo Thread từ class.



Kế thừa class Thead

Code demo: tạo class **MyThread**

Code Snippet

```
class MyThread extends Thread      //Extending Thread
    class
{
    //class definition
     . . .
}
```

Kế thừa class Thead

- Code demo: Override phương thức **run**()

Code Snippet

```
class MyThread extends Thread  //Extending Thread class
{
        // class definition
        public void run()
        //overriding the run() method
        {
        // implementation
        }
        . . .
 }
```

Kế thừa class Thead

DEVMASTER.VN

Code demo: Khởi chạy thread

Code Snippet

```
public class TestThread
{
. . .
    public static void main(String args[])
    {
         MyThread t=new MyThread(); //creating thread object
         t.start();      //Starting the thread
    }
}
```

Contructors của Thread

DEVMASTER.VN

Constructor	Description
`Thread()`	Default constructor
`Thread(Runnable objRun)`	Tạo thread với tham số là một Runnable
`Thread(Runnable objRun, String threadName)`	Tạo thread với tham số là một Runnable và tên của Thread (chuỗi)
`Thread(String threadName)`	Tạo thread với tham số là tên (chuỗi)
`Thread(ThreadGroup group, Runnable objRun)`	Tạo thread với tham số là một `ThreadGroup` và một Runnable

Phương thức của Thread

Method	Description
`static int activeCount()`	Trả về số luồng đang active trong luồng hiện tại của chương trình.
`static Thread currentThread()`	Trả về tham chiếu đến đối tượng thread đang thực hiện.
`ThreadGroup getThreadGroup()`	Trả về `ThreadGroup` mà thread hiện đang thuộc về
`static boolean interrupted()`	Kiểm tra thread hiện tại có bị gián đoạn không.
`boolean isAlive()`	Kiểm tra thread còn sống (hoạt động) không.
`boolean isInterrupted()`	Kiểm tra thread đã bị ngắt chưa.
`void join()`	Đợi thread hiện tại dừng
`void setName(String name)`	Đặt tên cho thread.

Demo

DEVMASTER.VN

Code demo:

```
/**
 * Creating threads using Thread class and using methods of the class
 */
package demo;
/**
 * NamedThread is created as a subclass of the class Thread
 */
 public class NamedThread extends Thread {
 /* This will store name of the thread */
       String name;
/**
  * This method of Thread class is overridden to specify the action
 * that will be done when the thread begins execution
  */
```

Demo

```
public void run() {
  // Will store the number of threads
     int count = 0;
    while(count<=3) {
    //Display the number of threads
         System.out.println(Thread.activeCount());
   //Display the name of the currently running thread
        name = Thread.currentThread().getName();
        count++;
        System.out.println(name);
  if (name.equals (“Thread1”))
      System.out.println(“Marimba”);
  else
      System.out.println(“Jini”);
 }
}
```

Demo

```
public static void main(String args[]) {
    NamedThread objNamedThread = new NamedThread();
    objNamedThread.setName("Thread1");
  //Display the status of the thread, whether alive or not
    System.out.println(Thread.currentThread().isAlive());
    System.out.println(objNamedThread.isAlive());
  /*invokes the start method which in turn will call
    run and begin thread execution
   */
    objNamedThread.start();
   System.out.println(Thread.currentThread().isAlive());
   System.out.println(objNamedThread.isAlive());
 }
}
```

Interface Runnable

Tạo **Thread** từ interface **Runnable**:

- **Interface Runnable** được thiết kế để cung cấp một tập các quy tắc cho các đối tượng có nhu cầu thực hiện mã trong một thread.
- Java không cho phép **đa kế thừa** nên cách tiếp cận này là một giải pháp khác.

Interface Runnable

Tạo **Thread** từ interface **Runnable**:

Interface Runnable

Code demo: tạo class thực thi interface

Code Snippet

```
// Declaring a class that implements Runnable interface
class MyRunnable implements Runnable
{
    . . .
}
```

Interface Runnable

Code demo: thực thi phương thức **run**()

Code Snippet

```
// Declaring a class that implements Runnable
interface class MyRunnable implements Runnable
{
    public void run()                    // Overriding the
    Run()
          {
        . . .
       // implementation
    }
}
```

Interface Runnable

DEVMASTER.VN

Code demo: khởi chạy thread

Code Snippet

```
class ThreadTest
{
    public static void main(String args[])
    {
      MyRunnable r=new MyRunnable();
      Thread thObj=new Thread(r);
      thObj.start(); //Starting a thread
    }
}
```

Demo code

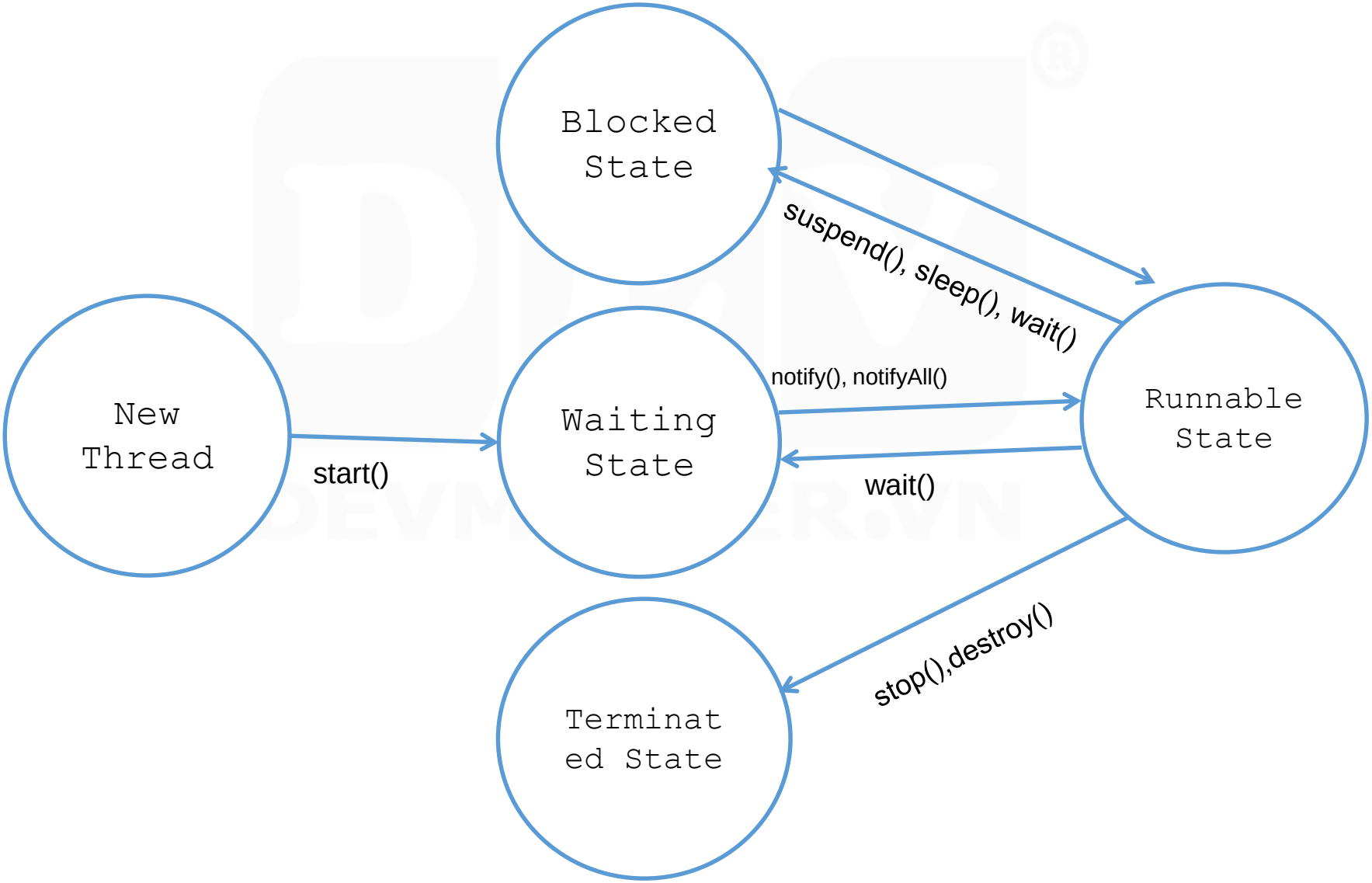
```
/*
 *Creating threads using Thread
 *class and using methods of the class
 */
package test;
/**
* NamedThread is created so as to implement the interface Runnable
*/
class NamedThread implements Runnable {
                /* this will store name of the thread */
                String name;
                 /**
                * This method of Runnable is implemented to specify the action          * that will be done
when the thread begins execution.
                 */
                 public void run()  {
                 int count = 0; //will store the number of threads
                 while(count < 3){
```

Demo code

```
                name = Thread.currentThread().getName();                System.out.println(name);
                count++;
                }
                }
 }
public class Main {
                public static void main(String args[])
        {

                NamedThread objNewThread= new NamedThread()
                Thread objThread = new Thread(objNewThread);            objThread.start();
                }
}
```

Vòng đời


Blocked State
suspend(), sleep(), wait()
New Thread
start()
Waiting State
notify(), notifyAll()
wait()
Runnable State
Terminated State
stop(),destroy()

Vòng đời

Giải thích:

- **New:** Một thread mới bắt đầu vòng đời của nó trong trạng thái new. Nó tồn tại trong trạng thái này tới khi chương trình bắt đầu thread này. Nó cũng được xem như là một thread mới sinh.
- **Runnable:** Sau khi một thread mới sinh ra được bắt đầu, thread trở thành runnable. Một thread trong trạng thái này được xem như đang thực hiện tác vụ của nó.
- **Waiting:** Đôi khi, một thread quá độ qua trạng thái waiting trong khi thread đợi cho thread khác thực hiện một tác vụ. Một thread chuyển về trạng thái runnable chỉ khi thread khác ra hiệu cho thread đang đợi để tiếp tục thực thi.

Vòng đời

- **Timed waiting:** Một thread trong trạng thái runnable có thể đi vào trạng thái timed waiting trong một khoảng thời gian nào đó. Một thread trong trạng thái này chuyển về trạng thái runnable khi khoảng thời gian đó kết thúc hoặc khi sự kiện nó đang đợi xuất hiện.
- **Terminated:** Một thread trong trạng thái runnable có thể đi vào trạng thái terminated khi nó hoàn thành tác vụ của nó hoặc nó chấm dứt.

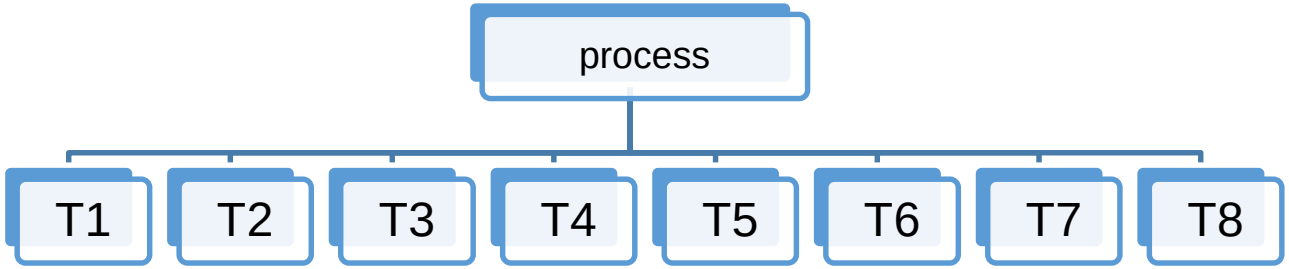
Phương thức class Thread

Một số phương thức quan trọng trong lớp Thread:

- getName()
- start()
- run()
- sleep()
- interrupt()

Quản lý luồng

- **Thread** là đơn vị có khả năng tự hoạt động trong chương trình.
- Trong một chương trình đơn lẻ, các thread có thể hoạt động độc lập.
- Đôi khi, các luồng lại dùng chung tài nguyên chia sẻ.
- Và, trong vài tình huống luồng đang chạy cần phải chặn để tài nguyên có thể sử dụng chỉ 1 luồng trong cùng một thời điểm.
- Do đó cần kiểm soát nội bộ, quản lý luồng thống nhất để chương trình thông suốt.



Độ ưu tiên

- Trong tình huống tài nguyên hoặc xử lý có sự tranh chấp giữa nhiều luồng thì cần xác định độ ưu tiên.
- Độ ưu tiên thể hiện tầm quan trọng của thread là khác nhau.

Độ ưu tiên

- Độ ưu tiên là số nguyên dao động giữa: **MIN_PRIORITY** và **MAX_PRIORITY**.
- Luồng có chỉ số ưu tiên cao được CPU ưu tiên xử lý hơn.
- Các mức ưu tiên:
    1. Thread.MAX_PRIORITY: Hằng số 10, cao nhất.
    2. Thread.NORM_PRIORITY: Hằng số 5, trung bình.
    3. Thread.MIN_PRIORITY: Hằng số 1. thấp nhất.

Deamon Thread

- Một deamon thread chạy liên tục để thực hiện một dịch vụ mà không cần bất kỳ kết nối với toàn bộ chương trình.
- Các đặc điểm của deamon thread:

1. Hoạt động nền, cung cấp dịch vụ cho các thread khác.
2. Phụ thuộc vào luồng người dùng.
3. Máy ảo JVM dừng một lần các luồng chết và chỉ deamon thread là sống.

Code demo: thực thi đa luồng

```
/**
* Creating multiple threads using a class derived from Thread
    class
*/
package test;
 /**
* MultipleThreads is created as a subclass of the class Thread
*/
public class MultipleThreads extends Thread {
    /* Variable to store the name of the thread */
    String name;
     /**
      * This method of Thread class is overridden to specify the
        action that will be done when the thread begins execution.
      */
```

Đa luồng

```
public void run() {
   while(true) {
          name = Thread.currentThread().getName();
          System.out.println(name);
           try
                  {
                  Thread.sleep(500);
                  }
           catch( InterruptedException e)
                  {
                   break;
                  }
                }
          }
    /**
    * This is the entry point for the MultipleThreads class.
     */
```

Đa luồng

```
     public static void main(String args[]) {
        MultipleThreads t1 = new MultipleThreads();
        MultipleThreads t2 = new MultipleThreads();
        t1.setName("Thread2");
        t2.setName("Thread3");
        t1.start();
        t2.start();
    System.out.println("Number of threads running: " + Thread.
    activeCount());
    }
}
```

Phương thức của Thread

isAlive(): kiểm tra luồng có đang hoạt động không

Code Snippet

```
. . .
 public static void main(String [] args)
{
ThreadDemo Obj = new ThreadDemo();
Thread t = new Thread(Obj);   System.out.println("The thread is
alive :" + t.isAlive());
}
 . . .
```

Phương thức của Thread

join(): đợi luồng hiện tại dừng

Code Snippet

```
try
{
System.out.println("I am in the main and waiting for the thread
to finish");
// objTh is a Thread object
objTh.join();
}
catch( InterruptedException e)
{
System.out.println("Main thread is interrupted");
}
. . .
```

Điều kiện tranh chấp

- Trong chương trình đa luồng, một số luồng có thể đồng thời cố gắng cập nhật nguồn tài nguyên nào đó ví dụ như tập tin.
- Điều này khiến tài nguyên xung đột.
- Do đó cần xác định điều kiện tranh chấp cho các luồng.

1. Hai hay nhiều luồng cùng chia sẻ tài nguyên, dữ liệu.
2. Hai hay nhiều luồng thực hiện đọc/ghi dữ liệu đồng thời trên cùng tài nguyên dữ liệu.

Điều kiện tranh chấp

DEVMASTER.VN

Khối Synchronized: áp dụng cho object

```
//synchronized block
synchronized(object)
{
        // statements to be synchronized
}
//synchronized method
synchronized method(...)
{
        // body of method
}
```

Điều kiện tranh chấp

Khối Synchronized: áp dụng cho phương thức

Code Snippet

```
. . .
class Account
{
    float balance = 0.0;
    public synchronized void deposit(float value)
    {
        balance = balance + value;
    }
}
. . .
```

Điều kiện tranh chấp

DEVMASTER.VN

Demo Synchronized phương thức

```
/**
* Demonstrating synchronized methods.
*/
package test;
class One {
// This method is synchronized to use the thread safely
synchronized void display(int num) {
 num++;
System.out.print(“” + num);
try {
 Thread.sleep(1000);
 } catch(InterruptedException e) {
  System.out.println(“Interrupted”);
 }
  System.out.println(“ done”);
 }
}
```

Điều kiện tranh chấp

```
class Two extends Thread {
int number;
One objOne;
public Two(One one_num, int num) {
 objOne = one_num;
 number = num;
}
public void run() {
// Invoke the synchronized method
objOne.display(number);
 }
}
class SynchMethod {
public static void main(String args[]) {
One objOne = new One();
int digit = 10;
```

Điều kiện tranh chấp

3 thread khởi chạy đồng thời, điểm khác biệt giữa việc yêu cầu syschronized là giữa các luồng sẽ chờ 1s.

```
// Create three thread objects
Two objSynch1 = new Two(objOne);
Two objSynch2 = new Two(objOne);
Two objSynch3 = new Two(objOne);
objSynch1.start();
objSynch2.start();
objSynch3.start();
}
}
```

Phương thức của Thread

wait(): luồng hiện tại rơi vào trạng thái tạm chờ

Code Snippet

```
. . .
public synchronized void takeup()
{
while (!available)  {
try      {
         System.out.println("Philosopher is waiting for the other
         chopstick");
         wait();
         }
         catch( InterruptedException e)
         {
         }
}
available = false;
}
. . .
```

Phương thức của Thread

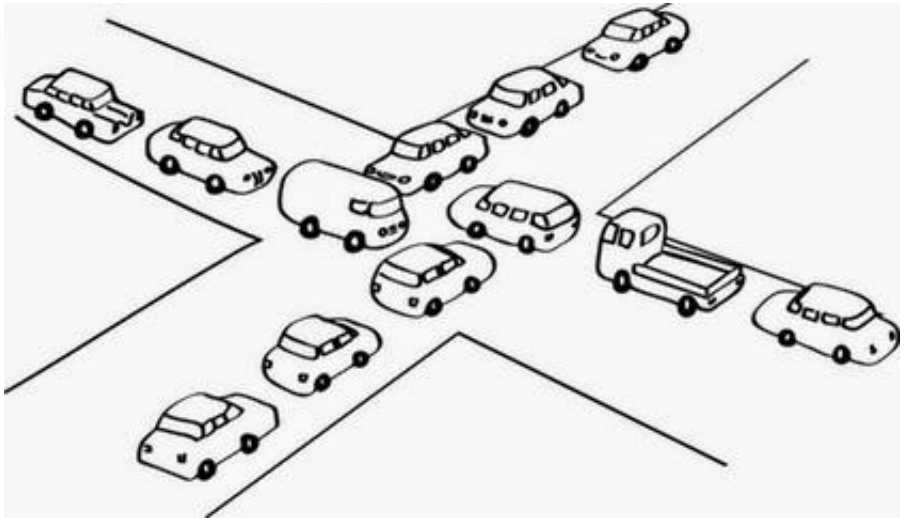
notify(): thông báo cho các luồng khác đang chờ

Code Snippet

```
. . .
public synchronized void putdown()
{
available = true;
notify();
}
. . .
```

Deadlock

- **Deadlock – bế tắc**: mô tả tình huống mà 2 hay nhiều luồng bị chặn mãi mãi do phải chờ đợi một tài nguyên được giải phóng.
- Đôi khi tình huống xảy ra khi 2 luồng bị khóa với tài nguyên của mình, chờ đợi ổ khóa tương ứng để trao đổi với nhau.



Deadlock

Demo:

```
/**
* Demonstrating Deadlock.
*/
package test;
/**
* DeadlockDemo implements the Runnable interface.
*/
public class DeadlockDemo implements Runnable
{
    public static void main(String args[])
    {
    DeadlockDemo objDead1 = new DeadlockDemo();
    DeadlockDemo objDead2 = new DeadlockDemo();
    Thread objTh1 = new Thread (objDead1);
    Thread objTh2 = new Thread (objDead2);
    objDead1.grabIt = objDead2;
    objDead2.grabIt = objDead1;
```

Deadlock

```
            objTh1.start();
            objTh2.start();
            System.out.println("Started");
            try {
                    objTh1.join();
                    objTh2.join();
                }
            catch( InterruptedException e) {
            System.out.println("error occurred");
            }
            System.exit(0);
    }
    DeadlockDemo grabIt;
    public synchronized void run() {
            try {
                    Thread.sleep(500);
```

Deadlock

```
                } catch( InterruptedException e) {
                        System.out.println("error occurred");
                 }
                grabIt.syncIt();
    }
    public synchronized void syncIt() {
        try {
                Thread.sleep(500);
                System.out.println("Sync");
        }
        catch(InterruptedException e) {
                System.out.println("error occurred");
         }
     System.out.println("In the syncIt() method");
    }
}// end class
```

Deadlock

Tránh deadlock:

1. • Tránh nhận nhiều khóa tại một thời điểm
2. • Đảm bảo trong chương trình có các ổ khóa theo thứ tự nhất quán được xác định

Tóm tắt bài học

- ✓ **Process** có thể bao gồm nhiều **thread**.
- ✓ **Thread** giúp chương trình có thể thực hiện nhiều tác vụ đồng thời.
- ✓ Có 2 cách tạo luồng là: kế thừa class **Thread** và thực thi **interface Runnable**.
- ✓ Độ ưu tiên xác định độ quan trọng giữa các thread.
- ✓ **Deamon thread** cung cấp dịch vụ cho các thread khác.
- ✓ Điều kiện tranh chấp giải quyết bởi khối **synchronized**.
- ✓ **Deadlock** là tình huống nhiều luồng cùng chờ đợi khóa chung được giải phóng.

Thank for watching!