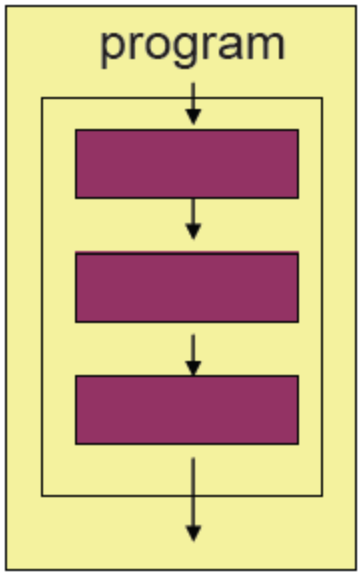
LẬP TRÌNH ĐA TUYẾN

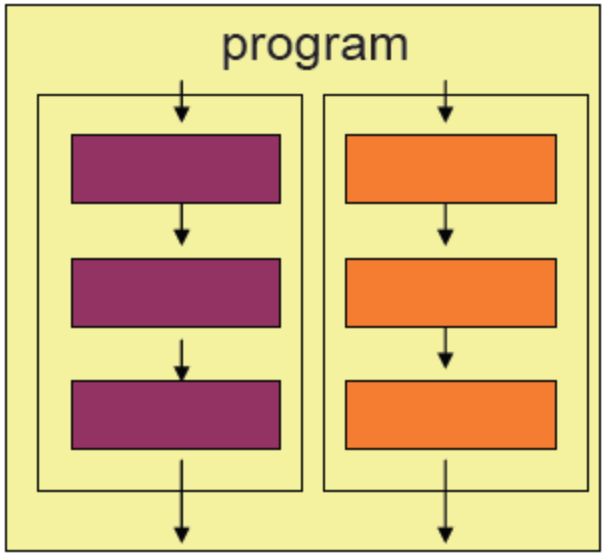
Giới Thiệu

- **Hệ điều hành đa nhiệm cổ điển:**
  - Đơn vị cơ bản sử dụng CPU là quá trình (process).
  - Quá trình là đoạn chương trình độc lập đã được nạp vào bộ nhớ.
  - Mỗi quá trình thi hành một ứng dụng riêng.
  - Mỗi quá trình có một không gian địa chỉ và một không gian trạng thái riêng.
  - Các quá trình liên lạc với nhau thông qua HĐH, tập tin, mạng.

Giới Thiệu

- 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.





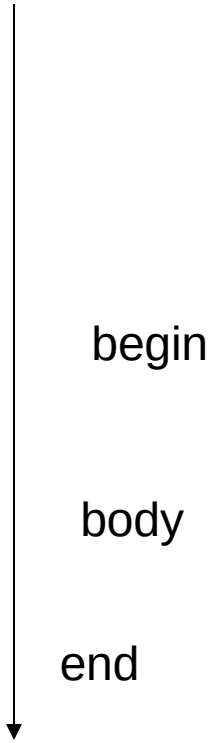
Giới Thiệu

- **Hệ điều hành đa nhiệm hiện đại, hỗ trợ luồng:**
  - Đơn vị cơ bản sử dụng CPU là luồng (thread).
  - Luồng một đoạn các câu lệnh được thi hành.
  - Mỗi quá trình có một không gian địa chỉ và nhiều luồng điều khiển.
  - Mỗi luồng có bộ đếm chương trình, trạng thái các thanh ghi và ngăn xếp riêng.
  - Luồng của một quá trình có thể chia sẻ nhau không gian địa chỉ : Biến toàn cục, tập tin, chương trình con, hiệu báo, . . .
  - Luồng chia sẻ thời gian sử dụng CPU => Luồng cũng có các trạng thái:

    Sẵn sàng (ready), Đang chạy (running), Nghẽn(Block) như quá trình.
  - Luồng cung cấp cơ chế tính toán song song trong các ứng dụng.

Chương trình đơn tuyến

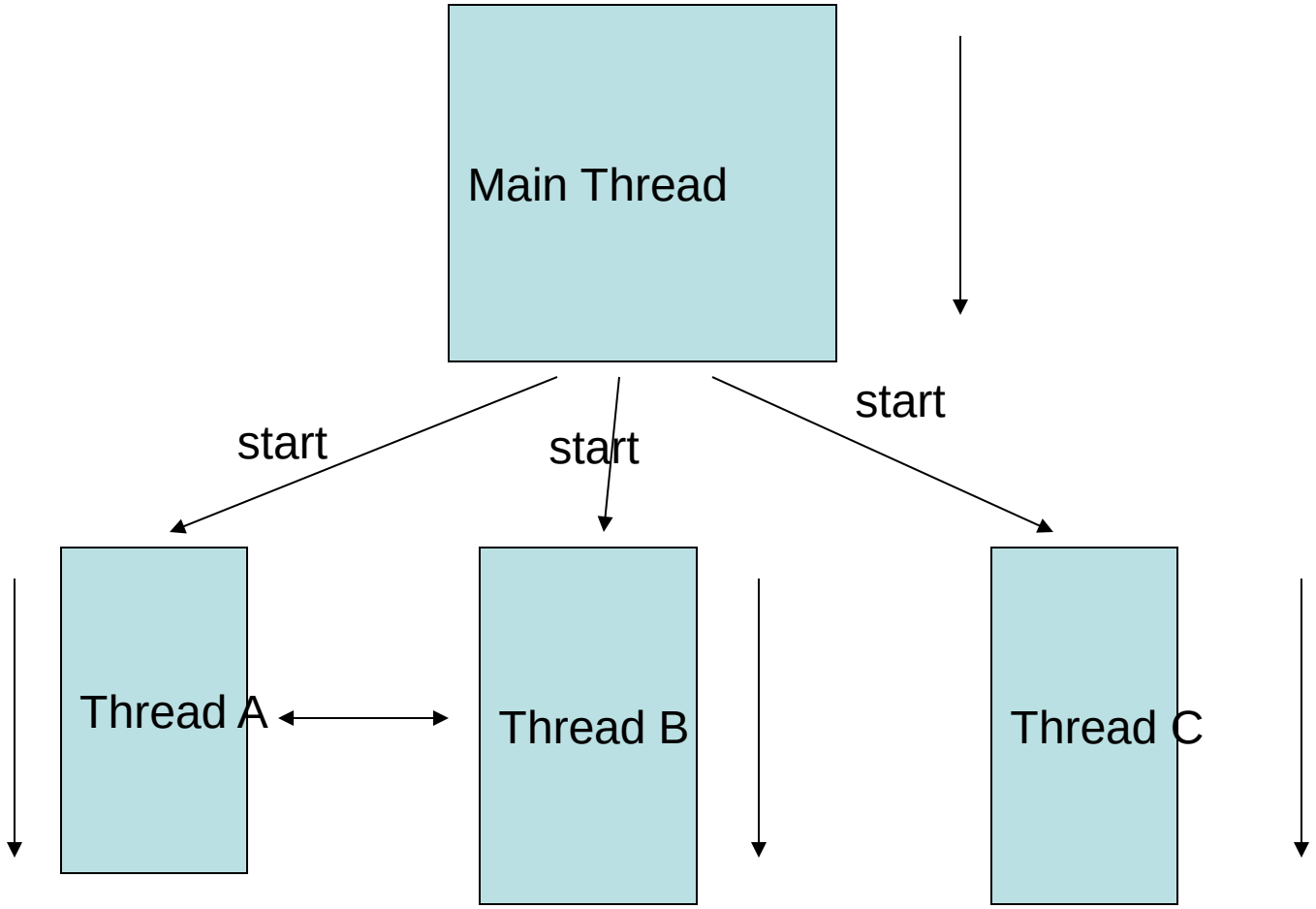
```
class ABC
{
….
    public void main(..)
    {
    …
    ..
    }
}
```



Đa tuyến

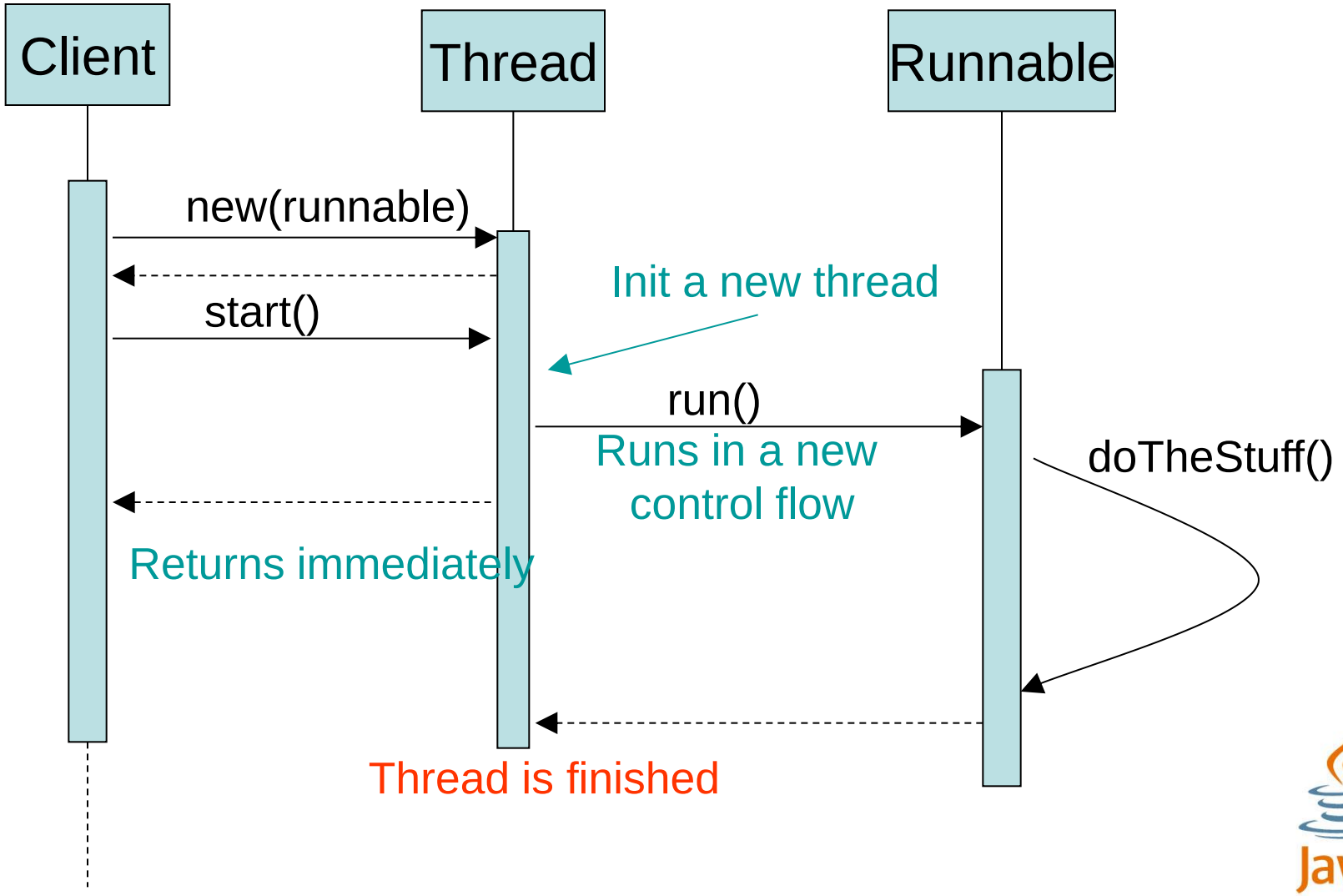
- Là khả năng làm việc với nhiều tuyến
- Đa tuyến chuyên sử dụng cho việc thực thi nhiều công việc đồng thời
- Đa tuyến giảm thời gian rỗi của hệ thống đến mức thấp nhất.

A Multithreaded Program

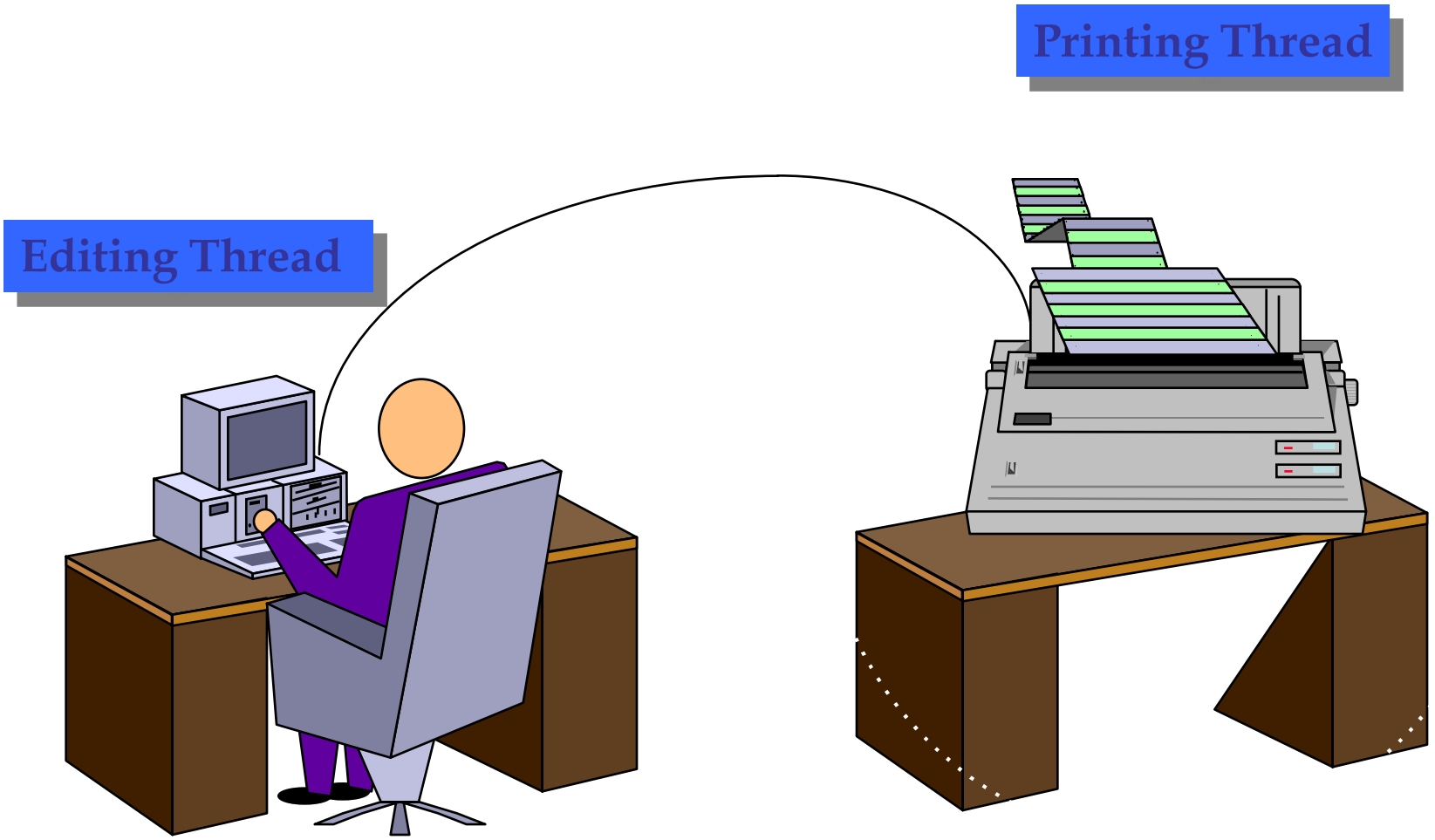


Các thread có thể chuyển đổi dữ liệu với nhau

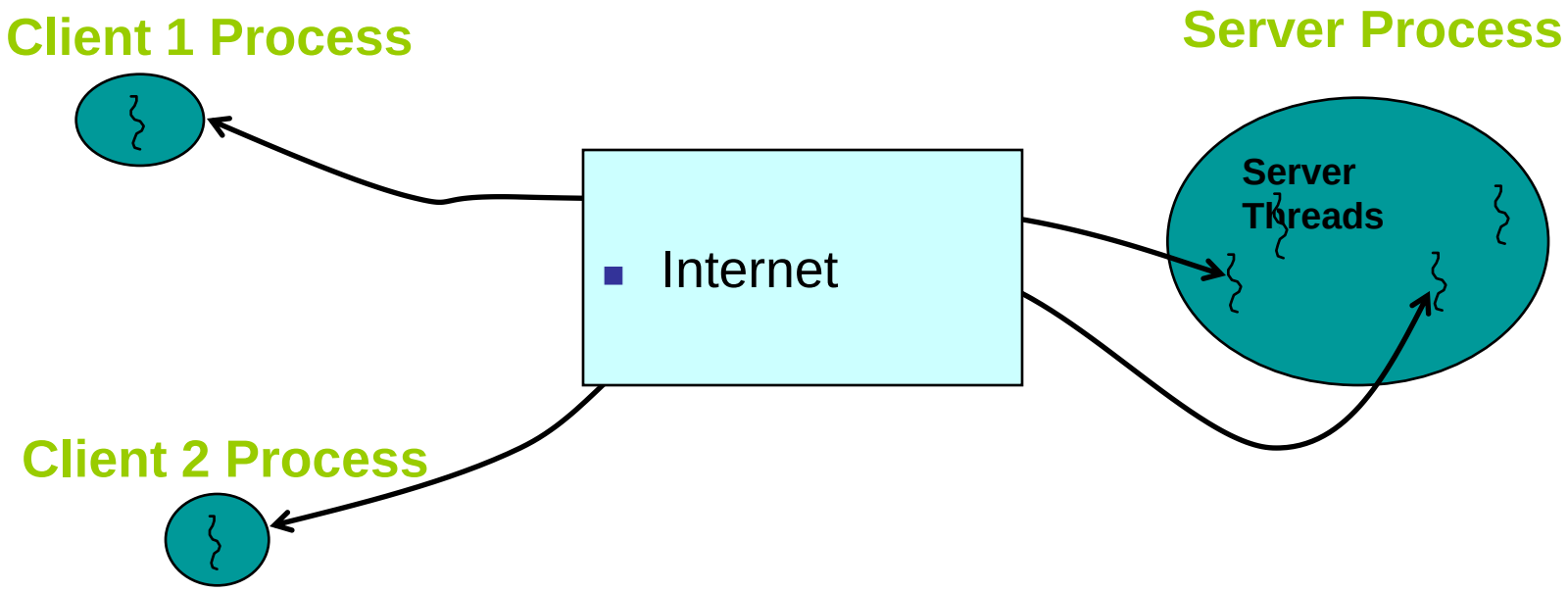
Cơ Chế Thi Hành



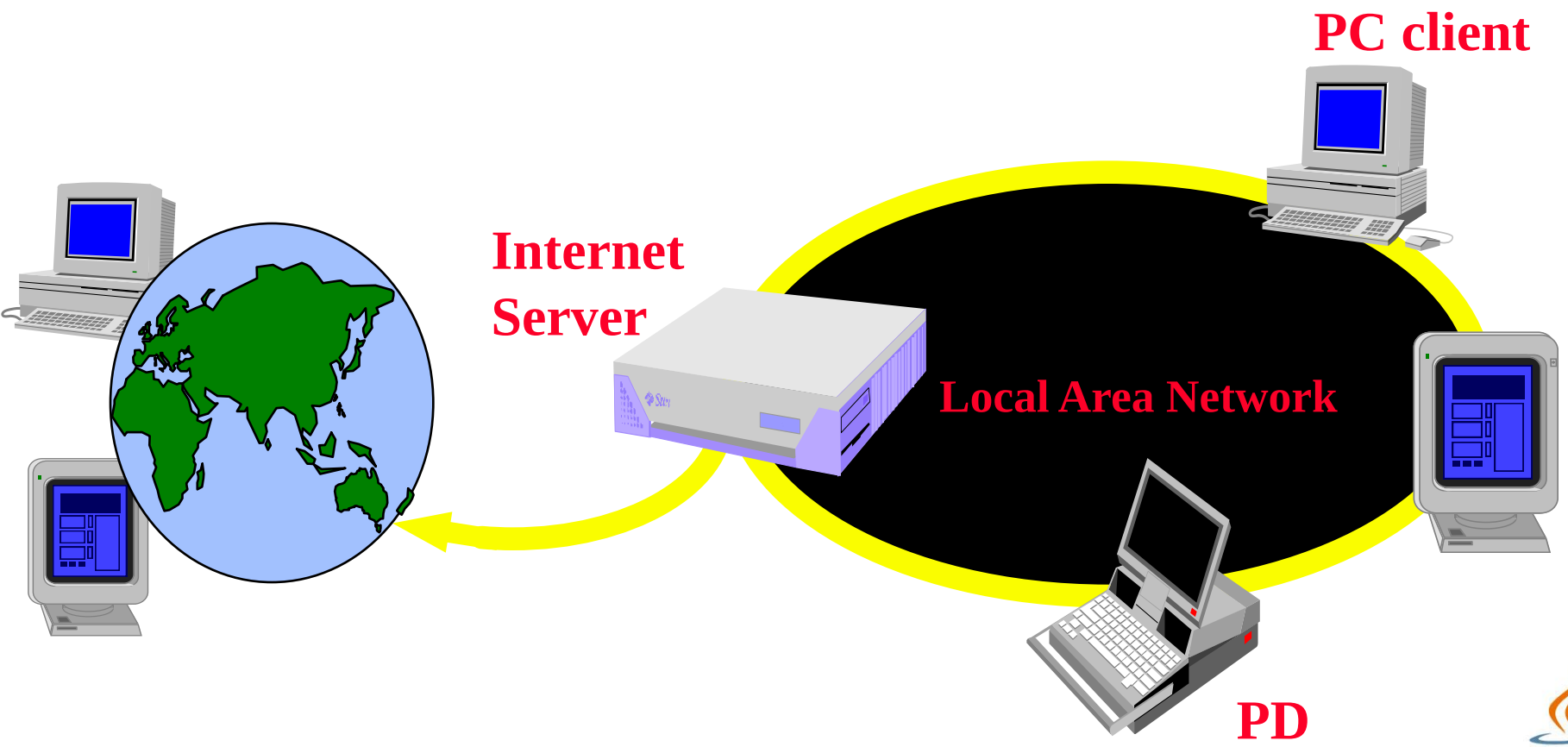
Ứng Dụng Multithreading



Multithreaded Server



Web/Internet Applications



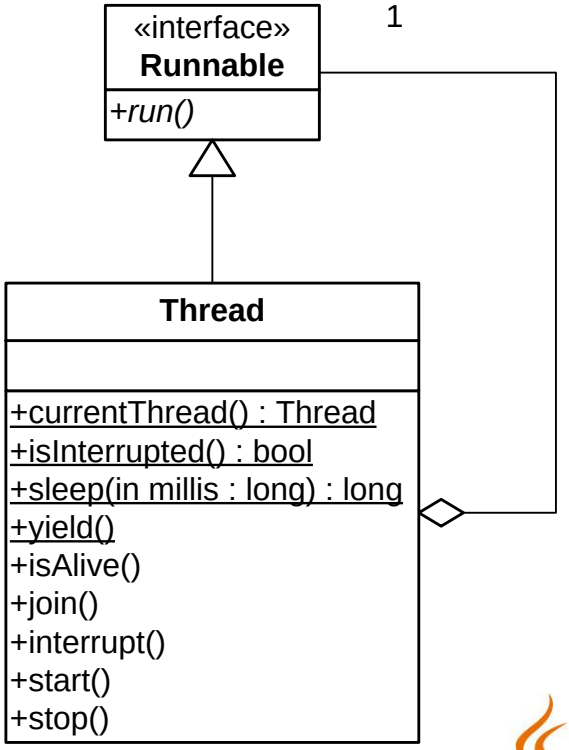
Lập trình đa tuyến với Java

- Cách thực hiện
  - Sử dụng lớp java.lang.Thread

```
public class Thread extends
    Object { … }
```

  - Sử dụng giao diện java.lang.Runnable

```
public interface Runnable {
    public void run();   // work ⇒ thread
}
```



Lớp java.lang.Thread

- Luồng trong java là một đối tượng của lớp **java.lang.Thread**
- Một chương trình cài đặt luồng bằng cách tạo ra các lớp con của lớp Thread.
- Lớp **Thread có 3 phương thức cơ bản:**
  - *public static synchronized void start() :*
    - Chuẩn bị mọi thứ cần thiết để thực hiện luồng.
  - *public void run():*
    - Chứa mã lệnh thực hiện công việc thực sự của luồng.
    - run() được gọi một cách tự động bởi start().
  - *public void stop() : kết thúc một luồng.*
  - Luồng kết thúc khi:
  - Hoặc tất cả các lệnh trong run() đã được thực thi.
  - Hoặc phương thức stop() của luồng được gọi.

Tạo và quản lý tuyến

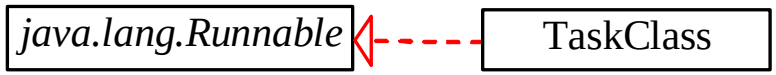
- Khi chương trình Java thực thi hàm main() tức là tuyến main được thực thi. Tuyến này được tạo ra một cách tự động. tại đây :

  - Các tuyến con sẽ được tạo ra từ đó

  - Nó là tuyến cuối cùng kết thúc việc thực hiện. Trong chốc lát tuyến chính ngừng thực thi, chương trình bị chấm dứt

- Tuyến có thể được tạo ra bằng 2 cách:
  - Dẫn xuất từ lớp Thread
  - Dẫn xuất từ Runnable.

Tạo Luông Công Việc




```
// Custom task class
public class TaskClass implements Runnable {
  ...
  public TaskClass(...) {
    ...
  }

  // Implement the run method in Runnable
  public void run() {
    // Tell system how to run custom thread
    ...
  }
  ...
}
```

```
// Client class
public class Client {
  ...
  public void someMethod() {
    ...
    // Create an instance of TaskClass
    TaskClass task = new TaskClass(...);

    // Create a thread
    Thread thread = new Thread(task);

    // Start a thread
    thread.start();
    ...
  }
  ...
}
```

Tạo thread sử dụng lớp Thread

- Cài đặt lớp kế thừa từ lớp Thread và override phương thức run()

```
class MyThread extends Thread
{
    public void run()
   {
      // thread body of execution
   }
}
```

- Tạo thread:

```
MyThread thr1 = new MyThread();
```

- Thi hành thread:

```
thr1.start();
```

- Tạo và thi hành thread:

```
new MyThread().start();
```

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

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)

```
thread1will sleep: 1438
thread2will sleep: 3221
thread3will sleep: 1813
thread1starts to sleep
Theads started. Thread main finised
thread3starts to sleep
thread2starts to sleep
thread1done sleeping
thread3done sleeping
thread2done sleeping
BUILD SUCCESSFUL (total time: 4 seconds)
```

```
thread1will sleep: 970
thread2will sleep: 950
thread3will sleep: 2564
thread1starts to sleep
Theads started. Thread main
finised
thread2starts to sleep
thread3starts to sleep
thread2done sleeping
thread1done sleeping
thread3done sleeping
```

Tạo thread sử dụng Runnable

```
class MyThread implements Runnable
{
   .....
   public void run()
   {
      // thread body of execution
   }
}
```

- **Tạo đối tượng:**

```
MyThread myObject = new MyThread();
```

- **Tạo thread từ đối tượng:**

```
Thread thr1 = new Thread( myObject );
```

- **Thi hành thread:**

```
thr1.start();
```

Ví Dụ

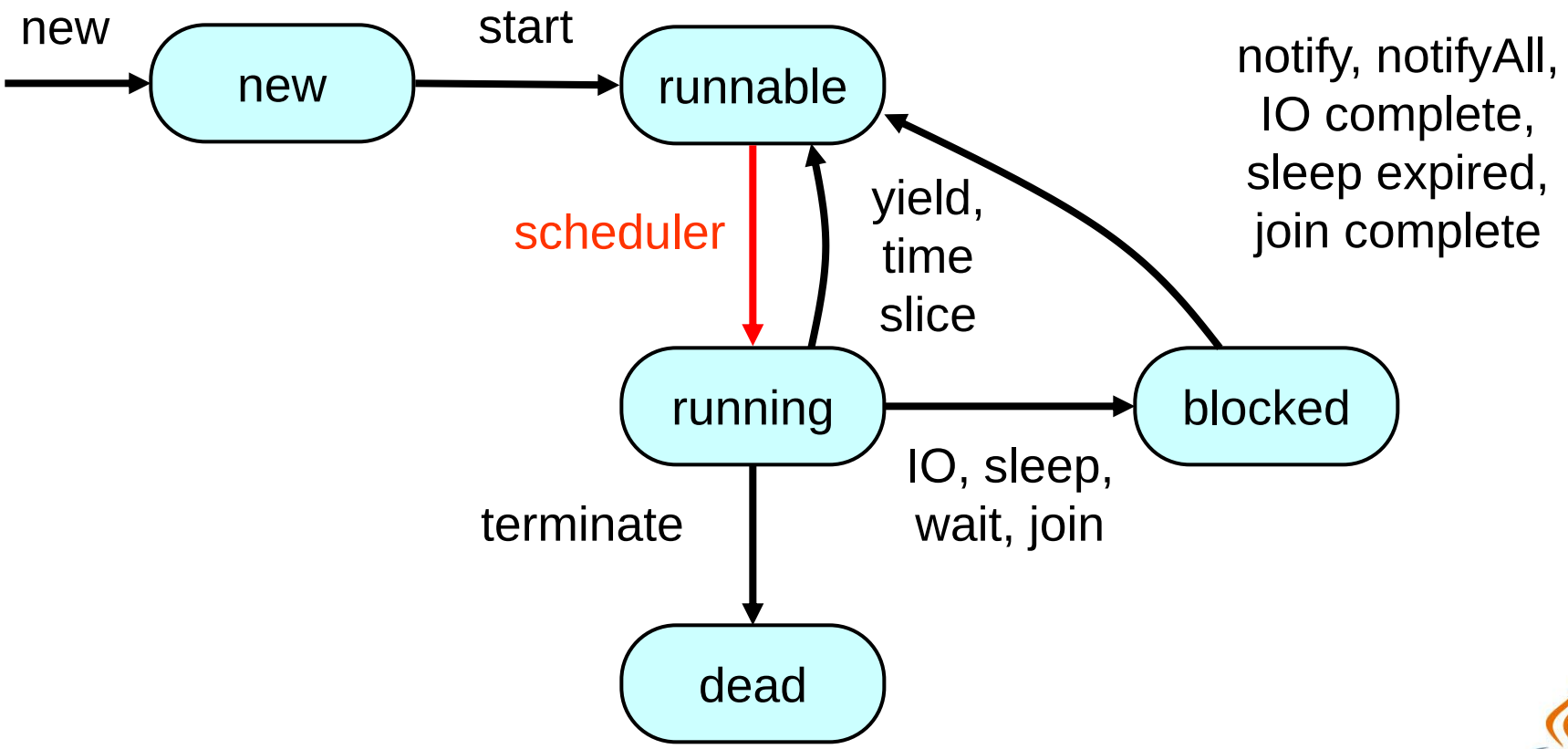
```
class MyThread implements Runnable  {
        public void run() {
                System.out.println(" this thread is running ... ");
        }
} // end class MyThread

class ThreadEx2 {
        public static void main(String [] args  ) {
                Thread t = new Thread(new MyThread());
                        // due to implementing the Runnable interface
                        // I can call start(), and this will call run().
                t.start();
        } // end main()
}       // end class ThreadEx2
```

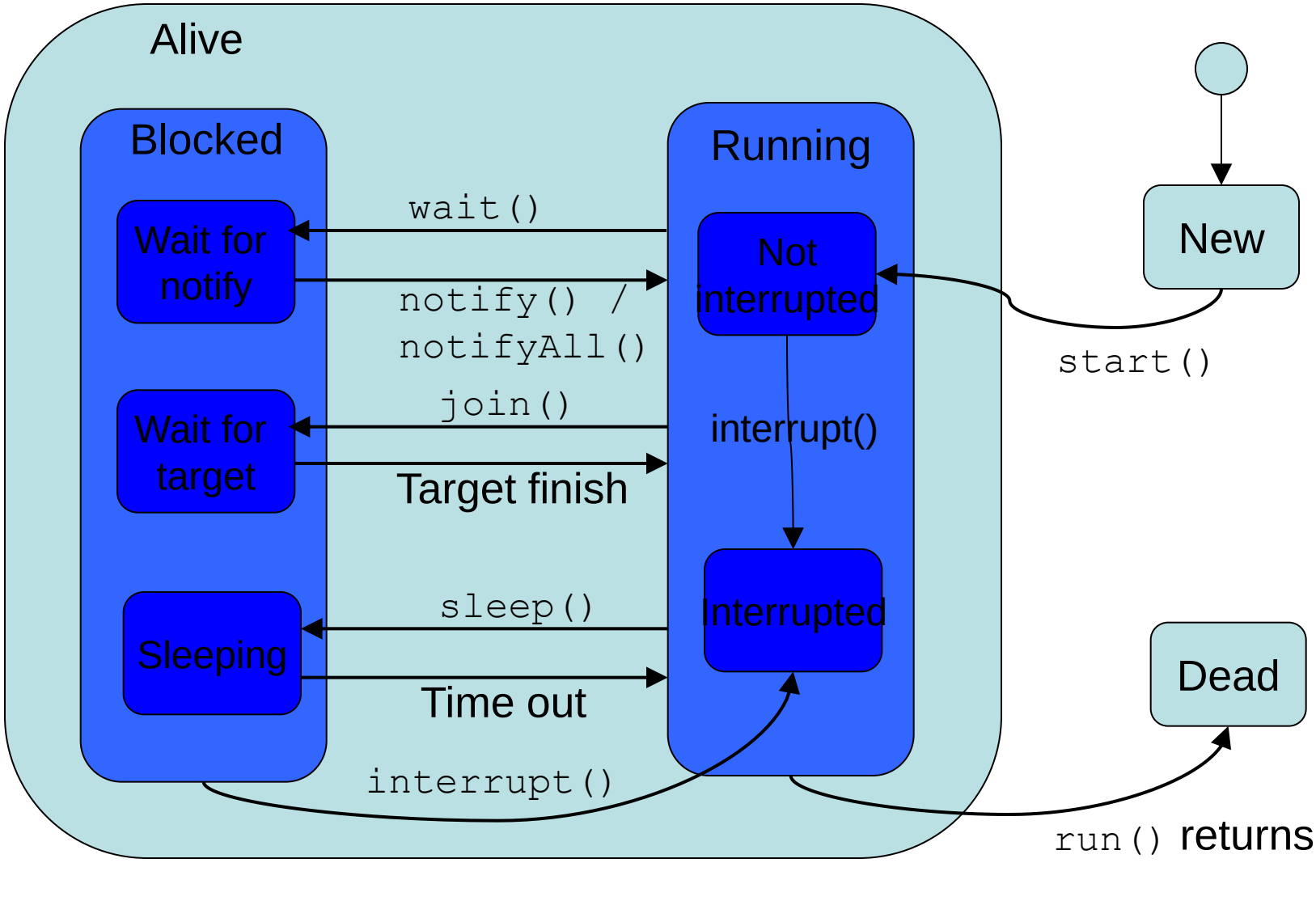
Threads – Thread States

- Các trạng thái của thread:
  - New – thread được tạo ra trong bộ nhớ
  - Runnable – thread có thể được thi hành
  - Running – thread đang thi hành
  - Blocked – thread đang bị treo (I/O, etc.)
  - Dead – thread kết thúc
- Việc chuyển đổi trạng thái thread thực hiện bởi:
  - Thi hành các phương thức trong lớp Thread
    - new(), start(), yield(), sleep(), wait(), notify()…
  - Các sự kiện bên ngoài
    - Scheduler, I/O, returning from run()…

Vòng Đời Của Thread

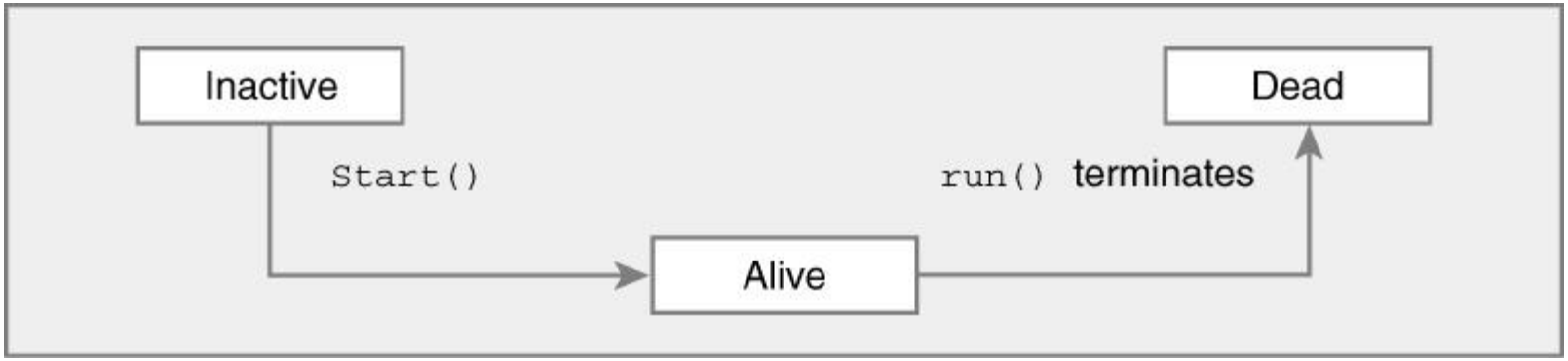


A thread's life cycle



Lưu Ý

– Thread chỉ được thi hành sau khi gọi phương thức start()



– Runnable là giao tiếp
  - Có thể hỗ trợ đa kế thừa
  - Thường dùng khi cài đặt giao diện GUI

Ví Dụ

- Viết chương trình thi hành song song 3 thread

Ví Dụ

```
class A extends Thread
{
      public void run()
       {
           for(int i=1;i<=5;i++)
            {
                 System.out.println("\t From ThreadA: i= "+i);
            }
             System.out.println("Exit from A");
       }
}

class B extends Thread
{
      public void run()
       {
           for(int j=1;j<=5;j++)
            {
                 System.out.println("\t From ThreadB: j= "+j);
            }
             System.out.println("Exit from B");
       }
}
```

```
class C extends Thread
{
      public void run()
       {
          for(int k=1;k<=5;k++)
           {
                System.out.println("\t From ThreadC: k= "+k);
           }

            System.out.println("Exit from C");
       }
}

class ThreadTest
{
        public static void main(String args[])
        {
                new A().start();
                new B().start();
                new C().start();
        }
}
```

Run 1

```
java ThreadTest
        From ThreadA: i= 1
        From ThreadA: i= 2
        From ThreadA: i= 3
        From ThreadA: i= 4
        From ThreadA: i= 5
Exit from A
        From ThreadC: k= 1
        From ThreadC: k= 2
        From ThreadC: k= 3
        From ThreadC: k= 4
        From ThreadC: k= 5
Exit from C
        From ThreadB: j= 1
        From ThreadB: j= 2
        From ThreadB: j= 3
        From ThreadB: j= 4
        From ThreadB: j= 5
Exit from B
```

Run2

```
java ThreadTest
        From ThreadA: i= 1
        From ThreadA: i= 2
        From ThreadA: i= 3
        From ThreadA: i= 4
        From ThreadA: i= 5
        From ThreadC: k= 1
        From ThreadC: k= 2
        From ThreadC: k= 3
        From ThreadC: k= 4
        From ThreadC: k= 5
Exit from C
        From ThreadB: j= 1
        From ThreadB: j= 2
        From ThreadB: j= 3
        From ThreadB: j= 4
        From ThreadB: j= 5
Exit from B
Exit from A
```

Daemon Threads

- Các lọai thread trong Java
  - User
  - Daemon
    - Cài đặt các dịch vụ
    - Chạy ngầm bên dưới hệ thống
    - Thi hành phương thức setDaemon() trước khi thi hành start()
- Chương trình kết thúc khi:
  1. Tất cả thread hoàn tất.
  2. Daemon threads bị kết thúc bởi JVM
  3. Chương trình chính kết thúc

Độ Ưu Tiên

- Trong Java, mỗi thread được gán 1 giá trị để chỉ mức độ ưu tiên của thread. Khi thread được tạo ra có độ ưu tiên mặc định (NORM_PRIORITY) sẽ được thi hành theo quy tắc FCFS.
  - Sử dụng phương thức setPriority() để thay đổi độ ưu tiên của thread:
    - ThreadName.setPriority(intNumber)
      - MIN_PRIORITY = 1
      - NORM_PRIORITY=5
      - MAX_PRIORITY=10

Ví Dụ Thread Priority

```
class A extends Thread
{
      public void run()
       {
            System.out.println("Thread A started");
            for(int i=1;i<=4;i++)
             {
                  System.out.println("\t From ThreadA: i= "+i);
             }
              System.out.println("Exit from A");
       }
}
class B extends Thread
{
      public void run()
       {
            System.out.println("Thread B started");
            for(int j=1;j<=4;j++)
             {
                  System.out.println("\t From ThreadB: j= "+j);
             }
              System.out.println("Exit from B");
       }
}
```

Ví Dụ Thread Priority

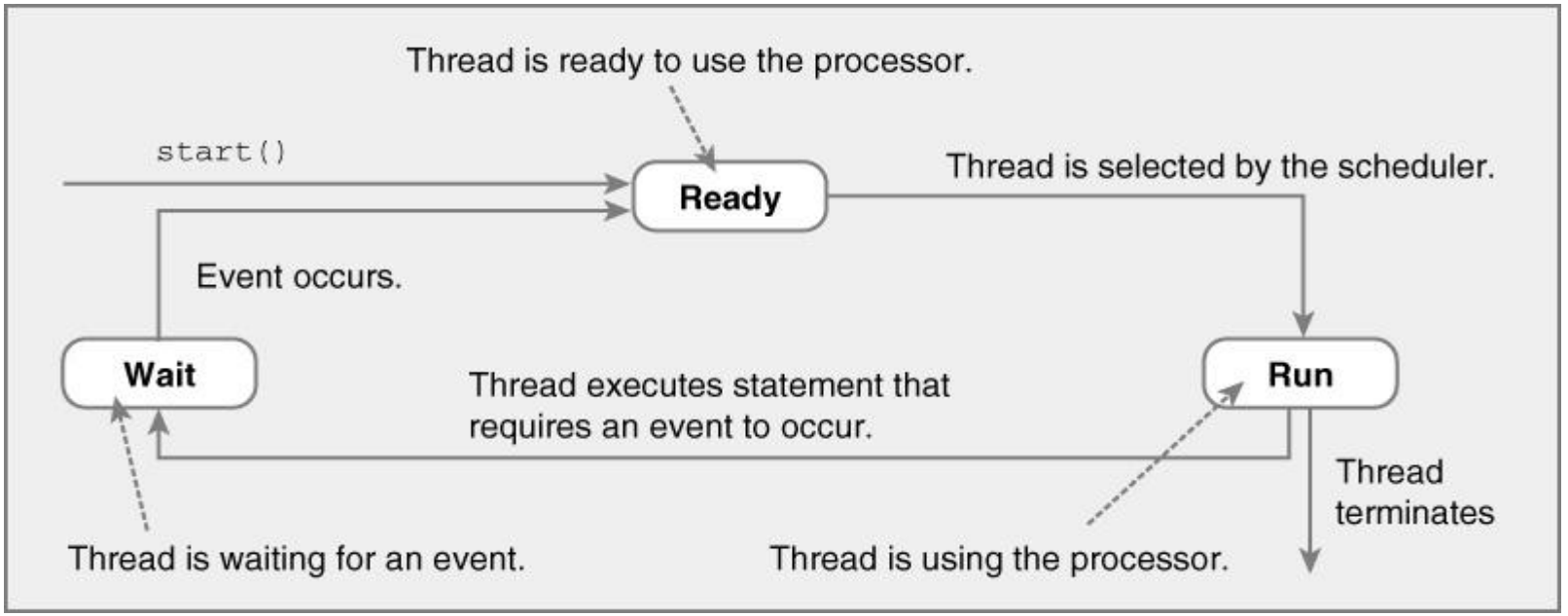
```
class C extends Thread
{
      public void run()
       {
           System.out.println("Thread C started");
           for(int k=1;k<=4;k++)
            {
                 System.out.println("\t From ThreadC: k= "+k);
            }
             System.out.println("Exit from C");
       }
}
class ThreadPriority
{
        public static void main(String args[])
         {
                  A threadA=new A();
                  B threadB=new B();
                  C threadC=new C();
                threadC.setPriority(Thread.MAX_PRIORITY);
                threadB.setPriority(threadA.getPriority()+1);
                threadA.setPriority(Thread.MIN_PRIORITY);
                System.out.println("Started Thread A");
                 threadA.start();
                System.out.println("Started Thread B");
                 threadB.start();
                System.out.println("Started Thread C");
                 threadC.start();
                 System.out.println("End of main thread");
         }
}
```

Threads – Scheduling

- Bộ lập lịch
  - Xác định thread nào sẽ thi hành
  - Có thể thực hiện dựa trên độ ưu tiên
  - Là một phần của HĐH hoặc Java Virtual Machine (JVM)
- Kiểu lập lịch
  - Nonpreemptive (cooperative) scheduling
  - Preemptive scheduling

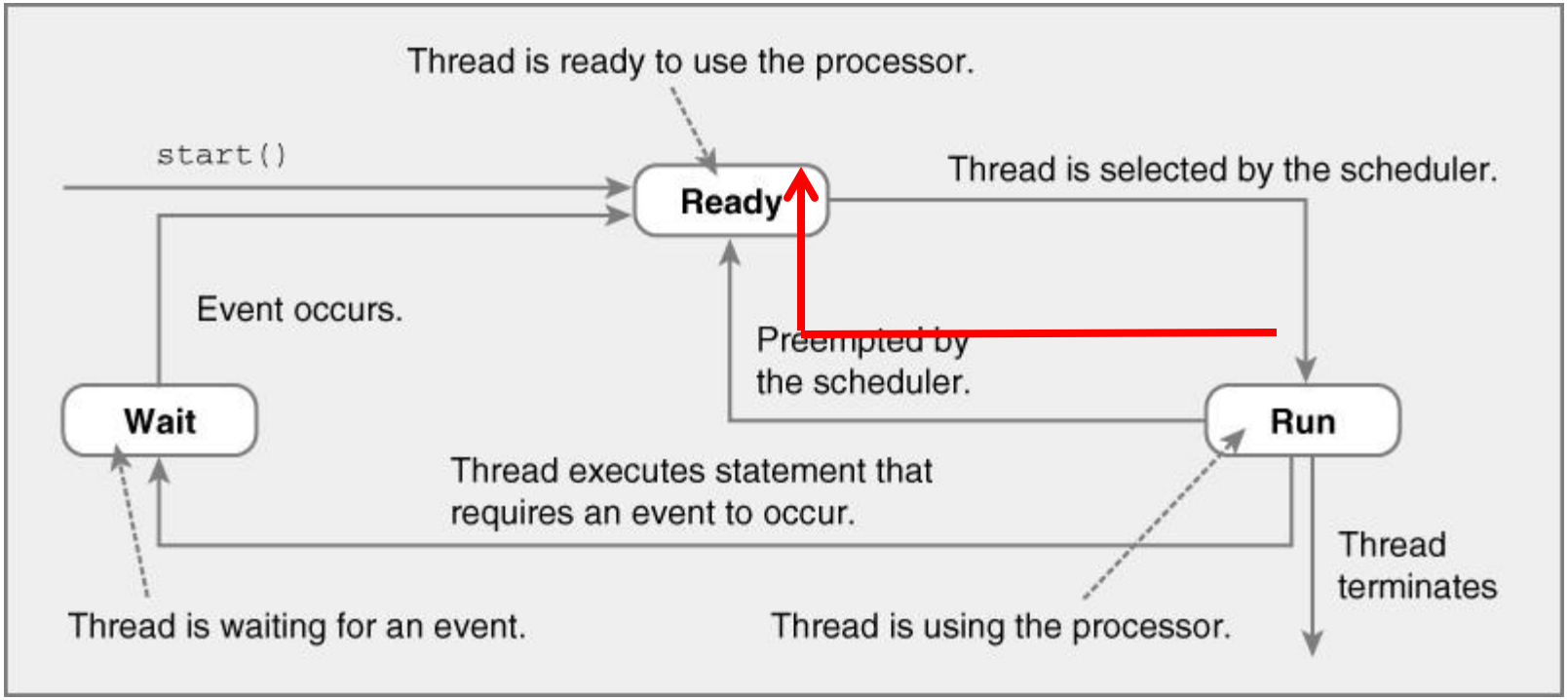
Non-preemptive Scheduling

- Thread thi hành cho đến khi
  - Hoàn tất công việc
  - Phải chờ sự kiện bên ngoài (IO,…)
  - Thread chủ động kết thúc thi hành (gọi phương thức yield hoặc sleep)



Preemptive Scheduling

- Threads thi hành cho đến khi
  - Tương tự non-preemptive scheduling
  - Preempted bởi bộ lập lịch



Thread Scheduling

- Ví dụ
  - Tạo lớp kế thừa Thread
  - Sử dụng phương thức sleep()
  - Công việc
    - Tạo 4 thread chạy song sọng, mỗi thread sẽ tạm ngưng thi hành một khoảng thời gian ngẫu nhiên
    - Sau khi kết thúc sleeping sẽ in ra tên thread.

```
1  // ThreadTester.java
2  // Show multiple threads printing at different intervals.
3
4  public class ThreadTester {
5     public static void main( String args[] )
6     {
7        PrintThread thread1, thread2, thread3, thread4;
8
9        thread1 = new PrintThread( "thread1" );
10       thread2 = new PrintThread( "thread2" );
11       thread3 = new PrintThread( "thread3" );
12       thread4 = new PrintThread( "thread4" );
13
14       System.err.println( "\nStarting threads" );
15
16       thread1.start();
17       thread2.start();
18       thread3.start();
19       thread4.start();
20
21       System.err.println( "Threads started\n" );
22    }
23 }
24
25 class PrintThread extends Thread {
26    private int sleepTime;
27
28    // PrintThread constructor assigns name to thread
29    // by calling Thread constructor
```

main kết thúc khi thread cuối cùng kết thúc.

```
30    public PrintThread( String name )
31    {
32       super( name );
33
34       // sleep between 0 and 5 seconds
35       sleepTime = (int) ( Math.random() * 5000 );
36
37       System.                                  e() +
38                                                eepTime );
39    }
40
41    // execute the thread
42    public void run()
43    {
44       // put thread to sleep for a random interval
45       try {
46          System.err.println( getName() + " going to sleep" );
47          Thread.sleep( sleepTime );
48       }
49       catch ( InterruptedException exception ) {
50          System.err.println( exception.toString() );
51       }
52
53       // print thread name
54       System.err.println( getName() + " done sleeping" );
55    }
56 }
```

Gọi constructor lớp cha

Công việc của thread

Sleep có thể ném ra biệt lệ

Truy Cập Tài Nguyên Dùng Chung

- Các ứng dụng truy cập vào tài nguyên dùng chung cần có cơ chế phối hợp để tránh đụng độ.
  - Máy in (2 công việc in không thể thực hiện cùng lúc)
  - Không thể thực hiện 2 thao tác đồng thời trên một tài khỏan
  - Việc gì sẽ xảy ra nếu vừa thực hiện đồng thời
    - Deposit()
    - Withdraw()

Đồng Bộ Hóa

- Lớp monitor
  - Là đối tượng có các phương thức **`synchronized`**
    - Bất kỳ đối tượng nào cũng có thể là monitor
  - Khai báo phương thức **`synchronized`**
    - **`public synchronized int myMethod( int x )`**
    - Chỉ duy nhất 1 thread được thực hiện phương thức **`synchronized tại 1 thời điểm`**

Đồng Bộ Hóa

- Đăng ký truy cập tài nguyên dùng chung
  - Gọi phương thức wait trong khi thi hành phương thức **`synchronized`**
    - Thread chuyển sang trạng thái chờ
  - Các thread khác cố gắng truy cập đối tượng monitor
    - Gọi **`notify`** để thông báo chuyển từ trạng thái wait sang ready
    - **`notifyAll`** – thông báo tất cả thread ở trạng thái wait chuyển sang ready

```
1  // HoldIntegerSynchronized.java
2  // Definition of class HoldIntegerSynchronized that
3  // uses thread synchronization to ensure that both
4  // threads access sharedInt at the proper times.
5  public class HoldIntegerSynchronized {
6     private static int sharedInt = -1;
7     private static
8
9     public synchron
10    {
11       while ( !writeable ) {  // not the producer's turn
12          try {
13             wait();
14          }
15          catch ( InterruptedException e ) {
16             e.printStackTrace();
17          }
18       }
19
20       System.err.println( Thread.currentThread().getName() +
21          " setting sharedInt to " + val );
22       sharedInt = val;
23
24       writeable = false;
25       notify();  // tell a waiting thread to
26    }
27
```

Test the condition variable. If it is not the producer's turn, then **`wait`**.

If **`writeable`** is **`true`**, write to the buffer, toggle **`writeable`**, and **`notify`** any waiting threads (so they may read from the buffer).

```
28    public synchronized int getSharedInt()
29    {
30       while ( writeable ) {   // not the consumer's turn
31          try {
32             wait();
33          }
34          catch ( InterruptedException e ) {
35             e.printStackTrace();
36          }
37       }
38
39       writeable = true;
40       notify();  // tell a waiting thread to become ready
41
42       System.err.println( Thread.currentThread().getName() +
43          " retrieving sharedInt value " + sharedInt );
44       return sharedInt;
45    }
46 }
```

As with **`setSharedInt`**, test the condition variable. If not the consumer's turn, then **`wait`**.

If it is ok to read (**`writeable`** is **`false`**), set **`writeable`** to **`true`**, **`notify`**, and **`return sharedInt`**.

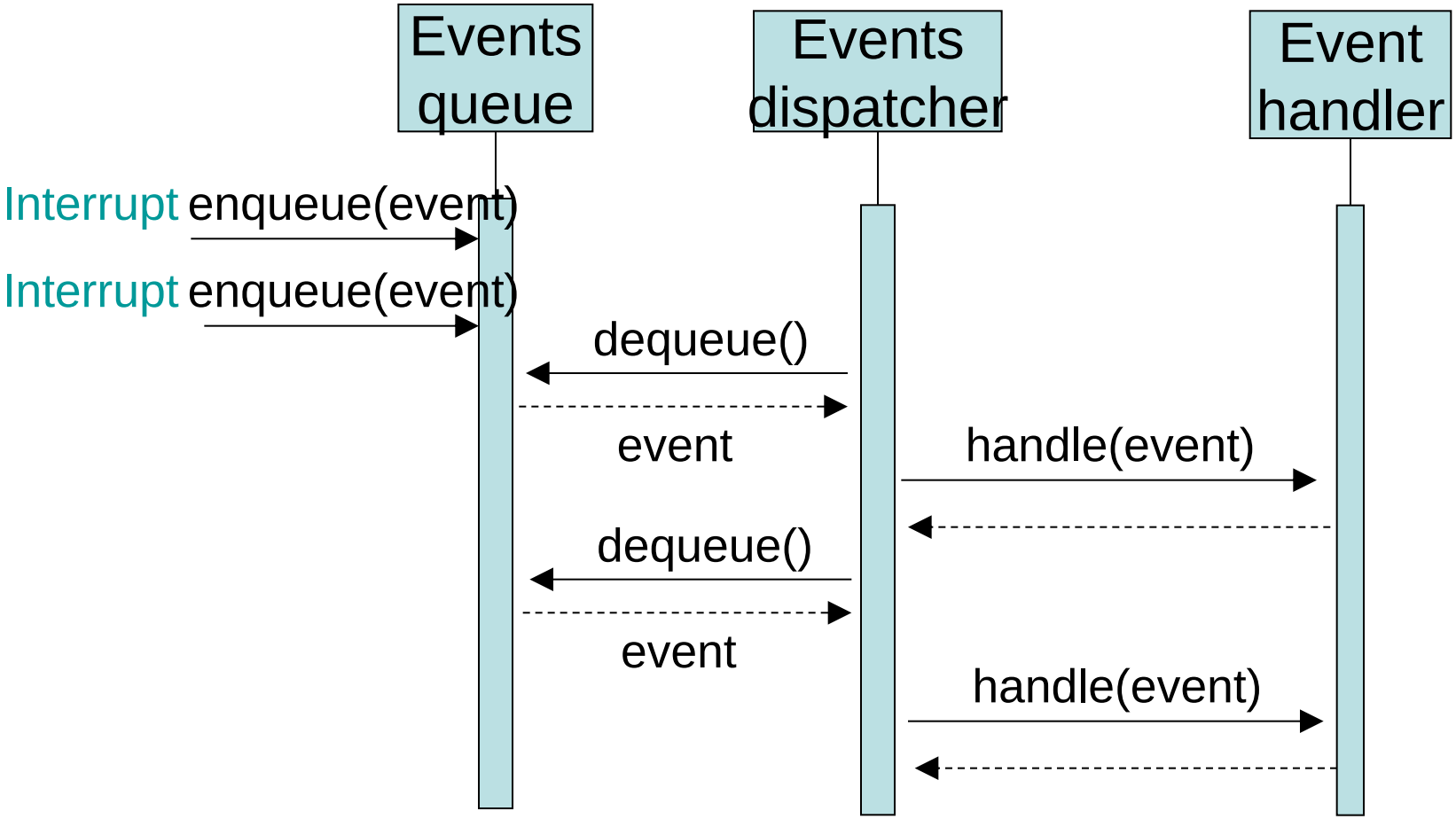
Swing's threads

- Công việc trong ứng dụng Swing không thi hành trong thread main
  - main thread chỉ khởi tạo GUI
- Mọi xử lý được thực hiện bởi thread đặc biệt là event-dispatching thread
- event dispatcher đảm nhận:
  - Thi hành các sự kiện trong hàng đợi
  - Vẽ và cập nhật "damaged" widgets
- Swing's components không thread-safe!
  - **Chỉ** event dispatcher được phép thực hiện công việc cập nhật GUI

Thi Hành Sự Kiện

- Tạo events:
  - JVM xử lý OS interrupts
  - interrupts được chuyển thành events
  - events được đặt vào hàng đợi
- event-dispatcher's loop:
  - Lấy event tiếp theo từ hàng đợi
  - Thi hành event-handler (nếu có)
  - Repaint widgets (nếu cần thiết)

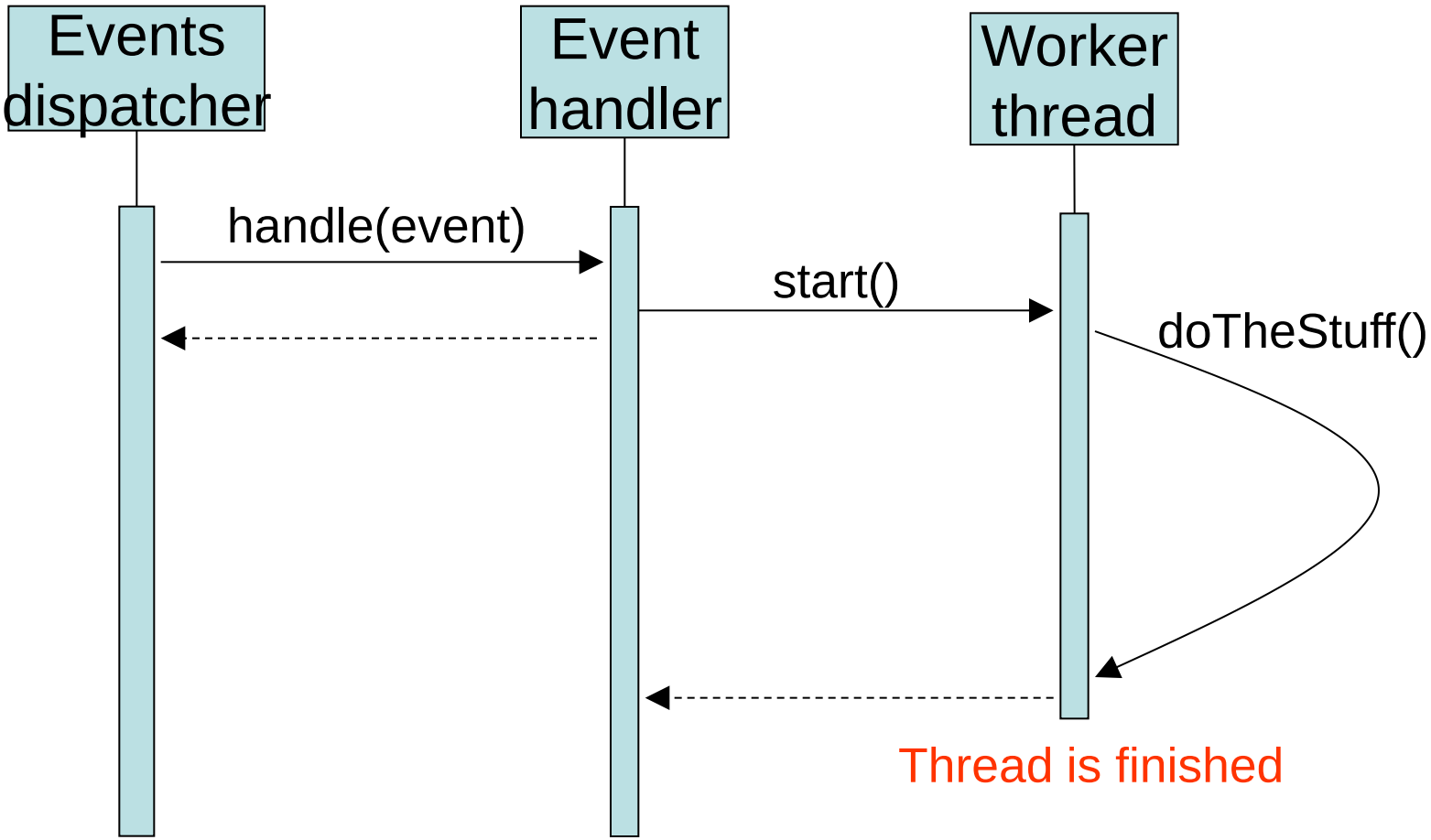
Thi Hành Sự Kiện (2)



Events handlers và threads

- Event handlers được thi hành tuần tự bởi event dispatcher thread.
- Trong khi thi hành event handlers sẽ không thực hiện xử lý trên GUI
  - Không có event khác được thi hành
  - Không repainting or updating
- Do đó, event handlers nên:
  - Ngắn gọn
  - Sử dụng 1 thread khác để giải phóng dispatcher

Sử Dụng thread



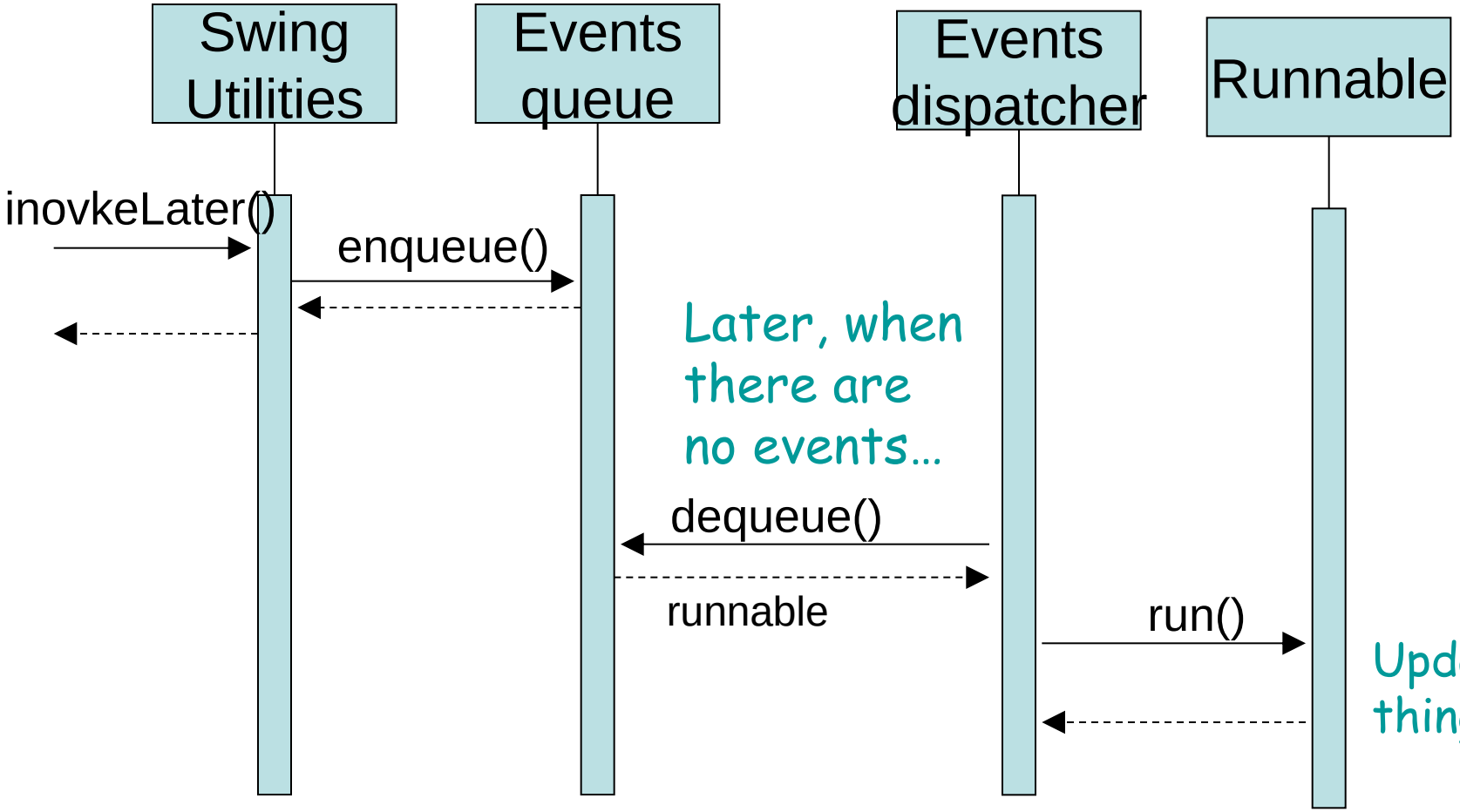
Vấn Đề Với multithreaded GUI

- Swing components là không thread-safe!
- Chỉ được phép cập nhật bởi 1 thread
  - Thông thường là event-dispatching thread
  - event handlers có thể cập nhật GUI nhưng các thread khác thì không
  - Ngoại lệ: `repaint()` và `revalidate()` thì thread-safe.

Thêm Xử Lý Vào event dispatcher

- `SwingUtilities` có phương thức để xin phép được đặt 1 công việc vào hàng đợi của event dispatcher
  - `invokeLater()`
  - `invokeAndWait()`
- Cả 2 có tham số kiểu `Runnable`
  - `run()` được event dispatcher tự động gọi

Thi Hành Công Việc Trong Hàng Đợi



Bắt Đầu ƯD Swing

```
public static void main(String[] args) {
SwingUtilities.invokeLater(new Runnable()
   {
    public void run()
          {
          createAndShowGUI(); // << method to start it
           }
       });
   }
```

createAndShowGUI

```
private static void createAndShowGUI()
{
  //Create and set up the window.
  JFrame frame = new JFrame("Hi..");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  //Add a label.
  JLabel label = new JLabel("Hello World");
  frame.getContentPane().add(label);
  //Display the window.
  frame.pack();
  frame.setVisible(true);
  }
```

Multithreading with GUI

- Các xử lý cần thời gian tính toán lâu nên được thực hiện bởi 1 thread riêng.
- Lớp `SwingWorker` (in package `javax.swing`) implements interface `Runnable`
  - Dùng để cài đặt thread thực hiện các tính toán đòi hỏi nhiều thời gian xử lý
  - Cập nhật Swing components từ hàng đợi event dispatch thread dựa trên kết quả thực hiện của worker thread

Method	Description
`doInBackground`	Defines a long computation and is called in a worker thread.
`done`	Executes on the event dispatch thread when `doInBackground` returns.
`execute`	Schedules the `SwingWorker` object to be executed in a worker thread.
`get`	Waits for the computation to complete, then returns the result of the computation (i.e., the return value of `doInBackground`).
`publish`	Sends intermediate results from the `doInBackground` method to the process method for processing on the event dispatch thread.
`process`	Receives intermediate results from the `publish` method and processes these results on the event dispatch thread.
`setProgress`	Sets the progress property to notify any property change listeners on the event dispatch thread of progress bar updates.

Fig. 23.24 | Commonly used `SwingWorker` methods.

Performing Computations in a Worker Thread

- Sử dụng `SwingWorker`
  - Thừa kế lớp `SwingWorker`
  - Overrides phương thức `doInBackground` và done
  - `doInBackground` thực hiện các xử lý và trả về kết quả
  - done hiển thị các kết quả trả về từ doInBackground trên GUI
- `SwingWorker` là 1 generic class
  - Tham số đầu tiên là kiểu trả về của `doInBackground`
  - Tham số thứ 2 là kiểu dữ liệu trao đổi giữa các phương thức dùng để lưu kết quả trung gian.
- `ExecutionException` được ném ra nếu quá trình tính toán có vấn đề.

```
1  // Fig. 26.25: BackgroundCalculator.java
2  // SwingWorker subclass for calculating Fibonacci numbers
3  // in a background thread.
4  import javax.swing.SwingWorker;
5  import javax.swing.JLabel;
6  import java.util.concurrent.ExecutionException;
7
8  public class BackgroundCalculator extends SwingWorker< Long, Object >
9  {
10    private final int n; // Fibonacci number to calculate
11    private final JLabel resultJLabel; // JLabel to display the result
12
13    // constructor
14    public BackgroundCalculator( int number, JLabel label )
15    {
16       n = number;
17       resultJLabel = label;
18    } // end BackgroundCalculator constructor
19
20    // long-running code to be run in a worker thread
21    public Long doInBackground()
22    {
```

Fig. 26.25 | `SwingWorker` subclass for calculating Fibonacci numbers in a background thread. (Part 1 of 3.)

```
23          long nthFib = fibonacci( n );
24          return String.valueOf( nthFib );
25       } // end method doInBackground
26
27       // code to run on the event dispatch thread when doInBackground returns
28       protected void done()
29       {
30          try
31          {
32             // get the result of doInBackground and display it
33             resultJLabel.setText( get().toString() );
34          } // end try
35          catch ( InterruptedException ex )
36          {
37             resultJLabel.setText( "Interrupted while waiting for results." );
38          } // end catch
39          catch ( ExecutionException ex )
40          {
41             resultJLabel.setText(
42                "Error encountered while performing calculation." );
43          } // end catch
44       } // end method done
```

Fig. 26.25 | `SwingWorker` subclass for calculating Fibonacci numbers in a background thread. (Part 2 of 3.)

```
45
46      // recursive method fibonacci; calculates nth Fibonacci number
47      public long fibonacci( long number )
48      {
49         if ( number == 0 || number == 1 )
50            return number;
51         else
52            return fibonacci( number - 1 ) + fibonacci( number - 2 );
53      } // end method fibonacci
54   } // end class BackgroundCalculator
```

Fig. 26.25 | `SwingWorker` subclass for calculating Fibonacci numbers in a background thread. (Part 3 of 3.)

```
1  // Fig. 26.26: FibonacciNumbers.java
2  // Using SwingWorker to perform a long calculation with
3  // results displayed in a GUI.
4  import java.awt.GridLayout;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JFrame;
9  import javax.swing.JPanel;
10 import javax.swing.JLabel;
11 import javax.swing.JTextField;
12 import javax.swing.border.TitledBorder;
13 import javax.swing.border.LineBorder;
14 import java.awt.Color;
15 import java.util.concurrent.ExecutionException;
16
17 public class FibonacciNumbers extends JFrame
18 {
19    // components for calculating the Fibonacci of a user-entered number
20    private final JPanel workerJPanel =
21       new JPanel( new GridLayout( 2, 2, 5, 5 ) );
22    private final JTextField numberJTextField = new JTextField();
```

Fig. 26.26 | Using `SwingWorker` to perform a long calculation with results displayed in a GUI. (Part 1 of 7.)

```
23      private final JButton goJButton = new JButton( "Go" );
24      private final JLabel fibonacciJLabel = new JLabel();
25
26      // components and variables for getting the next Fibonacci number
27      private final JPanel eventThreadJPanel =
28         new JPanel( new GridLayout( 2, 2, 5, 5 ) );
29      private long n1 = 0; // initialize with first Fibonacci number
30      private long n2 = 1; // initialize with second Fibonacci number
31      private int count = 1; // current Fibonacci number to display
32      private final JLabel nJLabel = new JLabel( "Fibonacci of 1: " );
33      private final JLabel nFibonacciJLabel =
34         new JLabel( String.valueOf( n2 ) );
35      private final JButton nextNumberJButton = new JButton( "Next Number" );
36
37      // constructor
38      public FibonacciNumbers()
39      {
40         super( "Fibonacci Numbers" );
41         setLayout( new GridLayout( 2, 1, 10, 10 ) );
42
43         // add GUI components to the SwingWorker panel
44         workerJPanel.setBorder( new TitledBorder(
45            new LineBorder( Color.BLACK ), "With SwingWorker" ) );
```

Fig. 26.26 | Using `SwingWorker` to perform a long calculation with results displayed in a GUI. (Part 2 of 7.)

```
46          workerJPanel.add( new JLabel( "Get Fibonacci of:" ) );
47          workerJPanel.add( numberJTextField );
48          goJButton.addActionListener(
49             new ActionListener()
50             {
51                public void actionPerformed( ActionEvent event )
52                {
53                   int n;
54
55                   try
56                   {
57                      // retrieve user's input as an integer
58                      n = Integer.parseInt( numberJTextField.getText() );
59                   } // end try
60                   catch( NumberFormatException ex )
61                   {
62                      // display an error message if the user did not
63                      // enter an integer
64                      fibonacciJLabel.setText( "Enter an integer." );
65                      return;
66                   } // end catch
67
```

Fig. 26.26 | Using `SwingWorker` to perform a long calculation with results displayed in a GUI. (Part 3 of 7.)

```
68                // indicate that the calculation has begun
69                fibonacciJLabel.setText( "Calculating..." );
70
71                // create a task to perform calculation in background
72                BackgroundCalculator task =
73                   new BackgroundCalculator( n, fibonacciJLabel );
74                task.execute(); // execute the task
75             } // end method actionPerformed
76          } // end anonymous inner class
77       ); // end call to addActionListener
78       workerJPanel.add( goJButton );
79       workerJPanel.add( fibonacciJLabel );
80
81       // add GUI components to the event-dispatching thread panel
82       eventThreadJPanel.setBorder( new TitledBorder(
83          new LineBorder( Color.BLACK ), "Without SwingWorker" ) );
84       eventThreadJPanel.add( nJLabel );
85       eventThreadJPanel.add( nFibonacciJLabel );
86       nextNumberJButton.addActionListener(
87          new ActionListener()
88          {
89             public void actionPerformed( ActionEvent event )
90             {
```

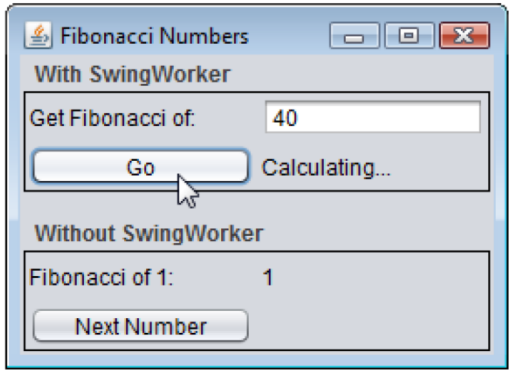
Fig. 26.26 | Using `SwingWorker` to perform a long calculation with results displayed in a GUI. (Part 4 of 7.)

```
91                  // calculate the Fibonacci number after n2
92                  long temp = n1 + n2;
93                  n1 = n2;
94                  n2 = temp;
95                  ++count;
96
97                  // display the next Fibonacci number
98                  nJLabel.setText( "Fibonacci of " + count + ": " );
99                  nFibonacciJLabel.setText( String.valueOf( n2 ) );
100              } // end method actionPerformed
101           } // end anonymous inner class
102        ); // end call to addActionListener
103        eventThreadJPanel.add( nextNumberJButton );
104
105        add( workerJPanel );
106        add( eventThreadJPanel );
107        setSize( 275, 200 );
108        setVisible( true );
109     } // end constructor
110
```

Fig. 26.26 | Using `SwingWorker` to perform a long calculation with results displayed in a GUI. (Part 5 of 7.)

```
111     // main method begins program execution
112     public static void main( String[] args )
113     {
114        FibonacciNumbers application = new FibonacciNumbers();
115        application.setDefaultCloseOperation( EXIT_ON_CLOSE );
116     } // end main
117  } // end class FibonacciNumbers
```

a) Begin calculating Fibonacci of 40 in the background.



b) Calculating other Fibonacci values while Fibonacci of 40 continues calculating.

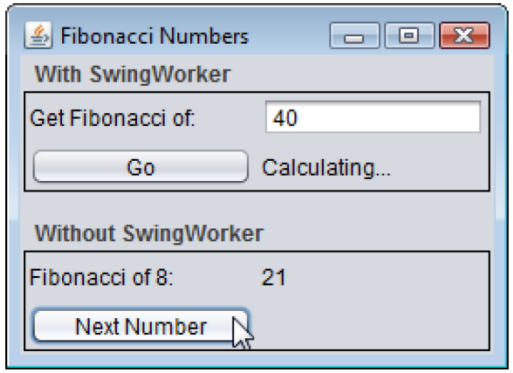


Fig. 26.26 | Using `SwingWorker` to perform a long calculation with results displayed in a GUI. (Part 6 of 7.)

c) Fibonacci of 40 calculation finishes.

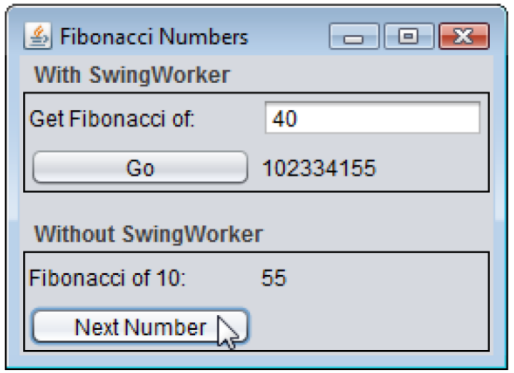


Fig. 26.26 | Using `SwingWorker` to perform a long calculation with results displayed in a GUI. (Part 7 of 7.)