Java Fundamental

Bài 10

Đa luồng – Multi Thread

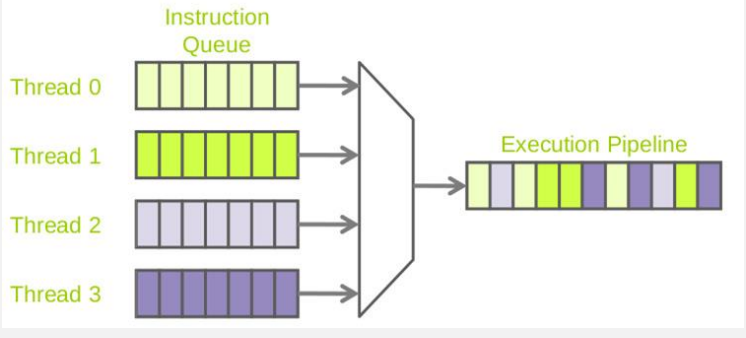
Company: DEVPRO VIỆT NAM

Website: devpro.edu.vn

Design by Minh An

Đa luồng - Multithreading

- Đa luồng là kỹ thuật giúp thực thi đồng thời hai hay nhiều phần của một chương trình để có thể tận dụng tối đa CPU.
- Mỗi phần như vậy được gọi là một luồng.
- Luồng có thể coi như là các tiến trình con trong một tiến trình xử lý của hệ thống (light-weight processes within a process).



Design by Minh An

Tạo luồng

- Có hai cách để tạo luồng
  - Tạo luồng bằng cách kế thừa từ lớp Thread (java.lang.Thread)
  - Tạo luồng bằng cách triển khai từ Interface Runnable

Kế thừa từ lớp Thread

- Tạo một lớp mới kế thừa từ lớp Thread
- Override phương thức run() ở lớp này
- Mọi công việc mà luồng thực hiện phải được viết trong phương thức run()
- Mọi công việc trong phương thức run() được thực thi khi luồng bắt đầu chạy, khi chạy xong các câu lệnh trong phương thức run() luồng sẽ tự hủy.

```
public class SampleThread extends Thread{
    @Override
    public void run() {
        System.out.println("Doing your works here...!");
    }
}
```

Kế thừa từ lớp Thread

- **Để thực thi luồng:**
  - **Tạo một đối tượng của lớp vừa khai báo**
  - **Gọi phương thức start() của đối tượng**

```
public class SampleThreadUtils {

    public static void main(String[] args) {
        SampleThread myThread = new SampleThread();
        myThread.start();
    }

}
```


Triển khai từ Interface Runnable

- **Tạo một lớp mới triển khai từ Interface Runnable**
- **Hiện thực phương thức run() của lớp này**

```
public class SampleRunable implements Runnable{
    @Override
    public void run() {
        System.out.println("Doing your works here...!");
    }
}
```

Triển khai từ Interface Runnable

- Tạo một thể hiện của lớp vừa tạo
- Tạo một thể hiện của lớp Thread
- Gọi phương thức start() của thể hiện lớp Thread.

```
public class SampleRunableUtils {

    public static void main(String[] args) {
        SampleRunable runable = new SampleRunable();
        Thread myThread = new Thread(runable);
        myThread.start();
    }
}
```

Design by Minh An

Sử dụng class Thread hay Interface Runnable?

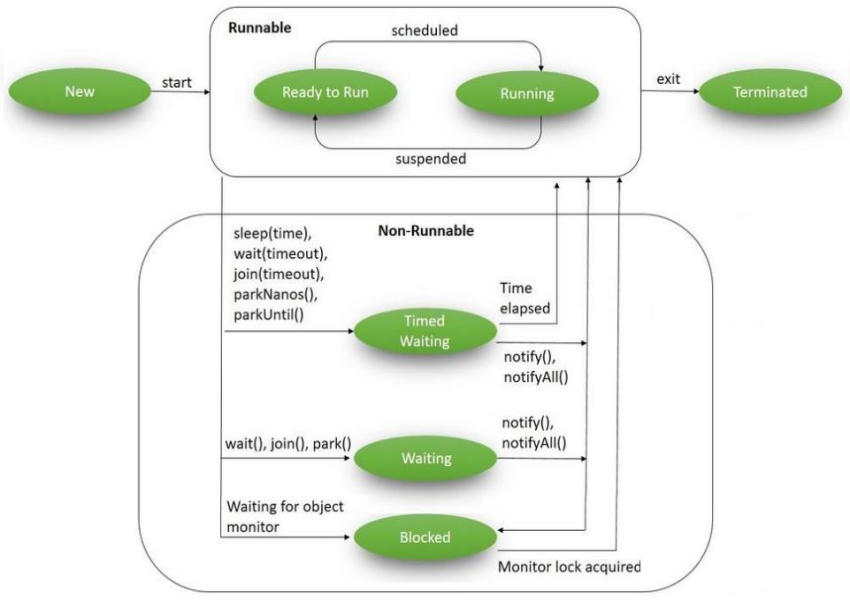
- Kế thừa class Thread, class tạo ra sẽ không thể kế thừa class nào khác nữa vì Java không hỗ trợ đa kế thừa.
- Triển khai interface Runnable, class tạo vẫn có thể kế thừa một class khác.
- Kế thừa class Thread, class tạo ra có các phương thức cơ bản của một luồng bởi vì trong class Thread đã có sẵn các method như yield(), interrupt(), ... Các method này không có sẵn trong interface Runnable

Design by Minh An

Vòng đời và các trạng thái của luồng

- Class Thread chứa một static enum là State, enum này định nghĩa các trạng thái có thể có của Thread.
- Trong vòng đời của một Thread ở bất cứ thời điểm nào thread cũng chỉ nhận một trong các trạng thái này.
- Mỗi Thread có 6 trạng thái:
  - NEW
  - RUNNABLE
  - BLOCKED
  - WAITING
  - TIMED_WAITING
  - TERMINATED

Vòng đời và các trạng thái của luồng



Vòng đời và các trạng thái của luồng

- NEW - thread mới được khởi tạo và chưa được chạy.
- RUNNABLE - đang chạy hoặc đã sẵn sàng chạy nhưng đang chờ được cấp phát tài nguyên
- BLOCKED - thread phải đợi tài nguyên hệ thống, hoặc đợi để giành được khóa (monitor lock) để có thể chạy một khối lệnh hoặc một method đồng bộ (synchronized method/block). Thông thường khi một thread truy cập vào tài nguyên hệ thống nó sẽ tự động bị block cho tới khi chiếm được quyền truy cập.

Vòng đời và các trạng thái của luồng

- WAITING - đợi một thread khác thực thi một công việc cụ thể nào đó trong một thời gian không giới hạn.
- TIMED_WAITING - đợi một trong một khoảng thời gian nhất định. Thread sẽ ở vào trạng thái TIMED_WAITING cho tới khi hết thời gian đợi hoặc nó nhận được thông báo chấm dứt đợi.
- TERMINATED - thread đã chạy xong các lệnh trong phương thức run() hoặc bị dừng bởi lỗi hoặc ngoại lệ không được xử lý. Một thread ở trạng thái TERMINATED sẽ không thể chạy tiếp.

Ví dụ: extends class Thread

```
public class GotoSchool extends Thread{
    private int amount;
    public GotoSchool(int amount) {
        this.amount = amount;
    }
    @Override
    public void run() {
        try {
            for (int i=1; i<=amount; i++) {
                System.out.println("HV thu " + i + " lop "
                                   + getName() + " den");
            }
            Thread.sleep(50);
        }
        catch(InterruptedException e) {
            System.out.println("Thread " + getName() +
                                        "is interrupted");
        }
    }
}
```


Ví dụ: extends class Thread

```
public class GotoSchoolUtils {

    public static void main(String[] args) {
        GotoSchool java = new GotoSchool(5);
        GotoSchool php = new GotoSchool(4);
        GotoSchool tes = new GotoSchool(7);
        java.setName("Java");
        php.setName("PHP");
        tes.setName("Tester");
        java.start();
        php.start();
        tes.start();
        //System.out.println("\nHV cac lop da den du");
    }
}
```

Ví dụ: implements interface Runnable

```
public class SendOrReceive implements Runnable{
    private String action;
    private int times;
    public SendOrReceive(String action, int times) {
        this.action = action;
        this.times = times;
    }
    @Override
    public void run() {
        for (int i=1; i<=times; i++)
            System.out.println("You are " + action +
                                "ing an email");
    }
}
```


Ví dụ: implements interface Runnable

```
public class SendOrReceiveUtils {

    public static void main(String[] args) {
        Thread send = new Thread(new SendOrReceive("send", 5));
        Thread receive = new Thread(new SendOrReceive("receiv", 6));
        send.start();
        receive.start();
    }

}
```

Một số phương thức của lớp Thread

- **suspend():** Tạm dừng hoạt động của 1 luồng nào đó bằng các ngưng cung cấp CPU cho luồng này.
- **resume():** cho luồng chạy lại khi luồng bị dừng do phương thức suspend().
- **isAlive() :** Phương thức này kiểm tra xem luồng còn active hay không.
- **yeild():** Ngừng cấp CPU cho luồng trong một lần nhận
- **sleep(long):** tạm dừng luồng trong một khoảng thời gian millisecond.
- **getName():** Trả về tên của thread.
- **setName(String name) :** Thay đổi tên của thread.
- **getId():** Trả về id của thread.
- **getState():** trả về trạng thái của thread.
- …

Bài tập

1. Cài đặt chương trình tạo một mảng sao cho các phần tử có chỉ số chẵn chứa các số chia hết 7, các phần tử có chỉ số lẻ chứa các số chia hết cho 4. Chương trình sử dụng hai luồng, luồng 1 gán giá trị cho các phần tử có chỉ số chẵn, luồng 2 gán giá trị cho các phần tử có chỉ số lẻ. Hiển thị mảng ra màn hình
2. Cài đặt chương trình với 2 luồng, luồng 1 tìm 50 số Fibonacci đầu tiên, luồng 2 tìm 50 số nguyên tố lớn hơn 30. Các số nguyên tố được lưu ở nửa đầu mảng a, các số Fibonacci được lưu ở nửa cuối mảng a. Hiển thị mảng ra màn hình.