**Đồng bộ hóa đa luồng trong Java**

**1. Đồng bộ luồng trong Java**

Để các bạn hiểu về đồng bộ luồng trong Java, tôi đưa ra một ví dụ đơn giản cần sử dụng đến đồng bộ như sau: Giả sử một công ty cấp quyền cho 10 nhân viên có thể thực hiện giao dịch rút tiền và chuyển tiền từ tài khoản ngân hàng chung của công ty và số dư của tài khoản này hiện đang la 10.000$. Vào lúc 9h sáng, giám đốc của công ty này đi giao dịch với khách hàng và quyết định rút 2.000$ từ tài khoản. Trong khi hành động rút tiền của vị giám đốc này đang được thực hiện và vẫn chưa kết thúc thì vào lúc này, người kế toán trưởng của công ty bắt đầu thực hiện kiểm tra số dư tài khoản và tiến hành chuyển tiền cho đối tác của công ty với số tiền là 9.000$.

Lúc này, vì giao dịch rút tiền của người giám đốc vẫn chưa kết thúc nên khi người kế toán trưởng kiểm tra số dư thì máy ATM vẫn trả lại số dư là 10.000$ và giao dịch rút 9.000$ của người này vẫn được chấp nhận. Và nếu trong trường hợp này mà không có sự đồng bộ thì cả 2 hành động rút tiền và chuyển tiền đều được thực hiện thành công và vì vậy ngân hàng sẽ mất đi số tiền là 1.000$ → trong trường hợp này bắt buộc cần đến sự đồng bộ hóa giữa các luồng.

Tóm lại, đồng bộ luồng trong Java chính là việc sắp xếp thứ tự các luồng khi truy xuất vào cùng một đối tượng (*trong ví dụ trên là số tiền trong tài khoản*) sao cho không có sự xung đột dữ liệu. Cơ chế đồng bộ này sẽ khóa hay đồng bộ dữ liệu sử dụng chung để tại một thời điểm chỉ có một luồng được thực thi (*rút tiền hoặc chuyển tiền*). Chỉ khi nào việc thực thi luồng này kết thúc thì luồng khác mới được thực hiện. Đây chính là cơ chế đồng bộ luồng trong Java.

Để đồng bộ luồng trong Java, chúng ta sẽ sử dụng từ khóa synchronized. Phần dưới đây tôi sẽ trình bày chi tiết.

**2. Đồng bộ luồng sử dụng từ khóa synchronized**

Để đồng bộ luồng trong Java, chúng ta sẽ sử dụng từ khóa synchronized. Từ khóa này sẽ đứng trước kiểu trả về và đứng sau phạm vi truy cập của phương thức đó.

Để minh họa cách sử dụng từ khóa này, tôi có một ví dụ minh họa bài toán ngân hàng như sau:

**Customer.java**

package dongboluong;

public class Customer {

    private int taiKhoan = 10000;

    public Customer() {

        System.out.println("Tài khoản hiện có = " + taiKhoan);

    }

    private synchronized void rutTien (int soTienRut) {

        System.out.println("Giao dịch rút tiền đang được thực hiện với" +

                " số tiền = " + soTienRut + "...");

        if(taiKhoan < soTienRut) {

            System.out.println("Số tiền trong tài khoản không đủ!");

            try {

                wait(); // phương thức wail sẽ đưa Thread rơi vào trạng thái sleeping

            } catch (InterruptedException ie) {

                System.out.println(ie.toString());

            }

        }

        taiKhoan -= soTienRut;

        System.out.println("Rút tiền thành công. Số tiền hiện có trong tài khoản = " + taiKhoan);

    }

    private synchronized void nopTien(int soTienNop) {

        System.out.println("Giao dịch nộp tiền đang được thực hiện với" +

                " số tiền nộp = " + soTienNop + "...");

        taiKhoan += soTienNop;

        System.out.println("Nộp tiền thành công. Số tiền hiện có trong tài khoản = " + taiKhoan);

        notify();

    }

    public static void main(String[] args) {

        final Customer customer = new Customer();

        Thread t1 = new Thread(){

            public void run() {

                customer.rutTien(20000);

            }

        };

        t1.start();

        Thread t2 = new Thread(){

            public void run() {

                customer.nopTien(30000);

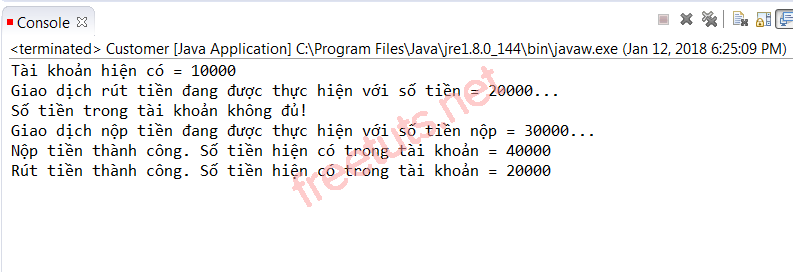
            }

        };

        t2.start();

    }

}

**Kết quả sau khi biên dịch chương trình:** 

**Giải thích hoạt động của chương trình trên:**

Trong ví dụ trên, tôi giả sử số tiền hiện có trong tài khoản ngân hàng là 10.000$. Trong hàm main(), chúng ta có 2 luồng: luồng t1 sẽ thực hiện việc rút tiền và luồng t2 sẽ thực hiện việc nộp tiền vào tài khoản. Trong Thread t1, khách hàng rút tiền với số tiền là 20.000$ thông qua phương thức rutTien(), lúc này số dư trong tài khoản không đủ nên hành động rút tiền này sẽ được đưa vào trạng thái sleeping thông qua dòng lệnh wait(). Sau đó Thread t2 sẽ được thực thi, lúc này khách hàng sẽ nộp vào tài khoản với số tiền là 40.000$ thông qua phương thức nopTien(). Trong phương thức nopTien() này, khi đã nộp tiền thành công thì dòng lệnh notify() sẽ đánh thức Thread đứng trước nó đang ở trạng thái sleeping vì phương thức wait() bị gọi (*ở đây là Thread t1*). Lúc này phương thức rutTien() sẽ kiểm tra số dư tài khoản là 40.000$ > số tiền cần rút là 20.000$ thì hành động rút tiền này sẽ thành công và số dư tài khoản còn lại là 20.000$.

**Lưu ý:** Trong khi sử dụng phương thức wait(), các bạn thấy có đoạn try...catch bao bọc bên ngoài, thì trong ví dụ này các bạn đừng để ý đến nó mà chỉ cần hiểu đây là điều bắt buộc khi cần sử dụng wait(). Chi tiết về try...catch tôi sẽ trình bày trong chương sau.