TÀI LIỆU THỰC TẬP LẬP TRÌNH MẠNG: LAB 05

DANH MỤC THUẬT NGỮ TIẾNG ANH

Từ	Nghĩa của từ
abstract	Trừu tượng
break	Dừng vòng lặp
catch	Từ khóa đầu của một khối bắt ngoại lệ
continue	Bỏ qua phần cuối vòng lặp, tiếp tục sang bước tiếp theo
default	Giá trị mặc định của phương thức switch()
extends	Kế thừa
final	Một hằng số, phương thức hay một lớp không được ghi đè
finally	Một phần của khối xử lý ngoại lệ try luôn được thực hiện
implements	Thực hiện giao diện
import	Khai báo một gói thư viện
instanceof	Kiểm tra một đối tượng là một thể hiện của lớp
interface	Giao diện
new	Tạo một đối tượng mới của lớp
null	Tham chiếu rỗng
package	Gói
private	Tiền tố chỉ được truy cập bởi phương thức của lớp
protected	Tiền tố được truy cập bởi phương thức của lớp, lớp con của và các lớp khác trong cùng một gói
public	Tiền tố có thể được truy cập bởi phương thức của tất cả các lớp
return	Trả về của một phương thức
super	Gọi phương thức của lớp cha
synchronized	Đồng bộ
this	Tham chiếu đến đối tượng hiện tại

DANH MỤC CHỮ VIẾT TẮT

Chữ viết tắt	Ý nghĩa
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
URL	*Uniform Resource Locator*
CSDL	Cơ Sở Dữ Liệu
JDBC	Java Database Connectivity
CNTT	Công Nghệ Thông Tin
HĐH	Hệ Điều Hành
MVC	Model-View-Control
DNS	Domain Name System
API	Application Programming Interface
FTP	File Transfer Protocol
JDK	Java Development Kit
GB	GigaByte
UCLN	Ước Chung Lớn Nhất
BCNN	Bội Chung Nhỏ Nhất
RAM	Random Access Memory
RMI	Remote Method Invocation
JVM	Java Virtual Machine
NIC	Network Interface Card
ĐH KTKT CN	Đại học Kinh tế Kỹ thuật Công nghiệp

LỜI NÓI ĐẦU

Ngày nay do nhu cầu thực tế và do sự phát triển mạnh mẽ của nhiều công nghệ tích hợp, dẫn đến các chương trình ứng dụng hầu hết đều có khả năng thực hiện trên môi trường mạng. Ngôn ngữ JAVA là ngôn ngữ phù hợp để viết các ứng dụng mạng. So với lập trình thông thường, lập trình mạng đòi hỏi người lập trình hiểu biết và có kỹ năng tốt để viết các chương trình giao tiếp và trao đổi dữ liệu giữa các máy tính với nhau.
Để hỗ trợ sinh viên chuyên ngành CNTT trong nhà trường tiếp cận với kỹ thuật lập trình mới này, tiếp theo cuốn tài liệu học tập lý thuyết "**Công nghệ JAVA**", chúng tôi xây dựng cuốn "**Bài tập lập trình mạng**", nhằm cung cấp cho sinh viên những kiến thức và kỹ thuật cơ bản nhất để phát triển các chương trình ứng dụng mạng, thông qua các dạng bài tập từ cơ bản đến nâng cao qua các chủ đề: lập trình cơ bản, lập trình hướng đối tượng, lập trình CSDL JDBC, lập trình mạng dùng socket, lập trình phân tán với RMI. Sinh viên sẽ thực hiện các bài thực hành này trên phòng máy nhà trường.

Nội dung cuốn tài liệu bao gồm 12 bài lab chia thành các chủ đề khác nhau. Trong mỗi chủ đề chúng tôi đưa ra tóm tắt lý thuyết, bài tập mẫu, sau đó là bài tập tương tự, và bài tập tổng hợp. Kết quả qua những bài lab, sinh viên được rèn và thành thạo các kỹ năng lập trình hướng đối tượng, lập trình CSDL, lập trình với giao thức truyền thông có sẵn và khả năng tích hợp trong các ứng dụng khác nhau, nhất là các giao thức truyền thông thời gian thực, từ đó sinh viên có thể viết được các phần mềm quản lý theo mô hình MVC, xây dựng được các ứng dụng mạng, các ứng dụng tích hợp và triệu gọi lẫn nhau trên mạng Intranet (mạng cục bộ), mạng Internet (mạng toàn cầu), các hệ thống xử lý truy xuất dữ liệu phân tán hoàn chỉnh. Nội dung biên soạn phù hợp với chuẩn đầu ra của ngành CNTT và ngành mạng máy tính và truyền thông dữ liệu về kỹ năng và kiến thức. Sau khi học xong học phần này sinh viên có thể viết phần mềm quản lý, truyền thông.

Chúng tôi xin chân thành cảm ơn Thầy Nguyễn Hoàng Chiến, phó chủ nhiệm khoa, phụ trách khoa CNTT trường ĐH KTKT CN cùng với các đồng nghiệp đã đóng góp ý kiến cho cuốn tài liệu này. Vì tài liệu được biên soạn lần đầu, chúng tôi đã cố gắng hoàn chỉnh, song không tránh khỏi thiếu sót. Rất mong nhận được sự góp ý của bạn đọc để tài liệu học tập được hoàn thiện hơn.

Xin trân trọng cảm ơn!

Nhóm tác giả

LAB 5. QUẢN LÝ THREAD [11, 12]

A. MỤC TIÊU

- Hướng dẫn sinh viên cách tạo luồng.
- Khai thác giao tiếp giữa các luồng trong java.
- Khai thác sự phối hợp giữa các luồng: sử dụng từ khóa **synchronized, wait, notify**.

B. NỘI DUNG

- Tạo luồng.
- Quản lý luồng: Đồng bộ hóa luồng, mối quan hệ giữa các luồng.

C. KẾT QUẢ SAU KHI HOÀN THÀNH

Viết chương trình mô phỏng đồng bộ luồng, áp dụng vào các bài toán cụ thể: Xử lý đồng bộ các giao dịch trong ngân hàng.

D. YÊU CẦU PHẦN CỨNG, PHẦN MỀM

- Máy tính cài HĐH windows, RAM tối thiểu 1GB.
- Phần mềm NETBEAN 8.0, JDK 1.8.

E. HƯỚNG DẪN

1. Tạo Thread

a)Tạo một lớp con kế thừa class Thread

```
public class A extends Thread
{
    public void run()
    {
        //code for the new thread to execute
    }
 }
A a = new A();      //create the thread object
a.start();          // start the new thread executing
```

b) Thực hiện interface Runnable

```
class B implements Runnable
  {
    public void run()
    {
        //code for the new thread to execute
    }
  }
  B b = new B();               // Tạo đối tượng Runnable
  Thread t = new Thread(b);    // Tạo đối tượng luồng
```

```
t.start();                    // Khởi động luồng
```

Một số phương của Thread

- **public static void** sleep(**long** millis) **throws** InterruptedException

  Thread hiện tại ngừng hoạt động trong một thời gian millis.

- **public void** start()

  - Tạo một đối tượng thread. JVM sẽ tự động gọi phương thức run của đối tượng
  - Thread được bắt đầu hoạt động sau khi phương thức này được gọi thành công.

- **public final boolean** isAlive()

  - Kiểm tra thread còn hiệu lực hay không (còn sống).

- **public final void** join(**long** millis) **throws** InterruptedException

  - Đây là phương thức được gọi bởi một thread khác. Phương thức này làm cho thread gọi phải ngưng hoạt động và chờ trong một khoảng thời gian millis hoặc chờ cho đến khi thread gọi kết thúc thì mới tiếp tục hoạt động.

Đồng bộ luồng-Synchronization

Khi hai hay nhiều thread cùng sử dụng chung một biến hay một phương thức thì xảy ra vấn đề : Race condition.

- Race condition (điều kiện tranh chấp) : xảy ra khi có hai hay nhiều thread cùng chia sẻ dữ liệu, khi chúng cùng đọc, cùng ghi dữ liệu đó đồng thời.
- Sử dụng từ khoá synchronized cho phương thức hay đoạn mã cần bảo vệ.

public synchronized void protectedMethod(object obj)

Bài 1. Tạo một Thread kế thừa từ Thread **class**.

Hướng dẫn:

```
public class MyThread extends Thread
{
    public static void main(String[] args)
    {
        MyThread th = new MyThread();
        th.start();
        System.out.println("This is the main thread");
    }

    public void run()
    {
        while (true)
        {
            try
            {
                System.out.println("This is the child Thread");
```

```
                sleep(1000);
            } catch (InterruptedException ex)
            {
               ex.printStackTrace();
            }
        }
    }
}
```

Bài 2. Tạo một Thread **implements** từ Runnable **interface**.

Hướng dẫn:

```
public class MyThread2 implements Runnable
{
    public static void main(String[] args)
    {
        MyThread2 myRunnable = new MyThread2();
        Thread thread = new Thread(myRunnable);
        thread.start();
        System.out.println("This is the main thread");
    }
        public void run()
         {
           while (true)
             {
            try
               {
                System.out.println("This is the child Thread");
                sleep(1000);
               } catch (InterruptedException ex)
            {
               ex.printStackTrace();
            }
        }
    }
}
```

Bài 3. Tạo 2 Thread: Thread 1 – Cứ mỗi 2 giây sinh ra một số ngẫu nhiên từ 1 đến 20. Thread 2: Lấy số ngẫu nhiên do Thread 1 sinh ra tính bình phương và hiển thị.

Hướng dẫn:

Bước 1: Tạo **int**erface Listener

```
public interface Listener {
public void addNumber(int number);
}
```

Bước 2: Tạo 1 **class** Generator tạo các một số ngẫu nhiên 1->20, 1 **class** square để bình phương số mà generator vừa tạo ra. **Class** generator sẽ thông báo cho square khi nó tạo được 1 số ngẫu nhiên bằng callback.

Class Generator

```
public class Generator implements Runnable{
    Thread t;
    Listener listener;
    public Generator(Listener listener){
     this.listener = listener;
    }
@Override
  public void run(){
      try {
         while(true) {
           int number = new Random().nextInt(20) + 1;
                System.out.println("Generator: " + number);
                 // thông báo số vừa tạo ra cho listener
                listener.addNumber(number);
                Thread.sleep(2000);
             }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    public void start(){
           if(t == null) {
            t = new Thread(this, "Generator");
            t.start();
        }
    }
}
```

Class Square

```
public class Square implements Runnable, Listener{
      Thread t;
```

```
	int number;
	boolean flag = false;
// nhận số được tạo ra từ Generator
	@Override
	public voi daddNumber(int number){
		this.number = number;
// flag = true khi nhận được 1 số mới tạo ra
		this.flag = true;
	}
@Override
	public void run(){
		try {
			while(true) {
				if(this.flag) {
		System.out.println("Square: " + number * number);
					this.flag = false;
				}
				Thread.sleep(1000);
			}
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
	public void start(){
		if(t == null) {
			t = new Thread(this, "Square");
			t.start();
		}
	}
}
```

Phương thức main

```
public class Demo {
	public static void main(String[] args) {
		Square square = new Square();
```

```
            square.start();
            Generator generator = new Generator(square);
            generator.start();
      }
}
```

Bài 4.

- Tạo lớp ThreadUtils cài đặt giao diện Runnable, thực hiện chức năng tính tổng các số nguyên từ fromIndex đến toIndex, lưu trữ tổng kết quả tính được.

- Tạo lớp ThreadTest, khai báo một thể hiện của mảng các phần từ số nguyên, kích thước mảng là 10,000.

- Viết phương thức để gán các giá trị trị cho mảng số nguyên vừa khai báo ở trên, giá trị là các số nguyên sinh ngẫu nhiên từ 0 →? 10,000.

- Tạo hai tiến trình thực hiện chức năng tính tổng các giá trị của mảng vừa tạo. Tiến trình 1 thực hiện chức năng tính tổng các số nguyên từ index 0 →? 5000. Tiến trình hai thực hiện chức năng tính tổng các số nguyên từ 5001 →? 9999. Cộng tổng kết quả từ hai tiến trình rồi hiển thị ra màn hinh.

Hướng dẫn:

Bước 1: Tạo TheadUtils

```
public class ThreadUtils implements Runnable {
    static int arr[] = new int[1000];
    private int fromIndex;
    private int toIndex;
    private long total;
    public ThreadUtils(int fromIndex,int toIndex, int[] aaa)
    {
        this.fromIndex=fromIndex;
        this.toIndex=toIndex;
        this.arr = aaa;
        this.total = 0L;
    }
    public void run()
    {
        for(int i = fromIndex; i<=toIndex;i++)
             total += arr[i];
```

```
    }
    public long getSum() {
        return total;
    }
}
```

Bước 2: Tạo class ThreadTest

```
import java.util.Random;
public class ThreadTest {
    public static int[] mang = new int[1000];
    public static void gan()
    {
        for(int i =0; i<mang.length;i++)
        {
                int Random= new Random().nextInt(1000);
                mang[i]=Random;
        }
    }
    public static void main(String[] args)
    {
        gan();
        ThreadUtils tu1 = new ThreadUtils(0,500,mang);
        Thread t1 = new Thread(tu1);
        t1.start();
        try {
            t1.join();
        } catch(InterruptedException ex)
        {
        }
        long tong1 = tu1.getSum();
        ThreadUtils tu2 = new ThreadUtils(501, 999,mang);
        Thread t2 = new Thread(tu2);
        t2.start();
        try {
            t2.join();
```

```
        } catch(InterruptedException ex)
        {
        }
        long tong2 = tu2.getSum();
       System.out.println("Tong mang 1: " + tong1);
       System.out.println("Tong mang 2: " + tong2);
       System.out.println("Tong hai gia tri: " + tong1 + tong2);
    }
}
```

Bài 5. Sử dụng MultiThread và synchronized.Tạo hai luồng:

Luồng 1: Hiển thị các số nhỏ hơn 1000

Luồng 2: Dựa trên số hiển thị trong luồng 1, hiển thị thông tin tương ứng là "nguyên tố" hay "không là số nguyên tố". Viết Phương thức main ( ) minh họa hai luồng này.

Hướng dẫn:

Bước 1: Tạo class Thread1

```
public class Thread1 extends Thread{
    Number number;
    public Thread1(Number number) {
        this.number = number;
    }
    @Override
    public void run() {
        for (int i= 0; i < 1000; i++) {
            number.inSo();
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
            }
        }
    }
}
```

Bước 2: Tạo class Thread2

```
public class Thread2 extends Thread{
```

```
    Number number;
    public Thread2(Number number) {
        this.number = number;
    }
  @Override
    public void run() {
        for (int i= 0; i < 1000; i++) {
            number.kiemtra();
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
            }
        }
    }
}
```

Bước 3: Tạo class Number

```
public class Number {
    private int num = 1;
    private boolean flag = false;
    public synchronized void inSo(){
        while (flag) {
            try {
                wait();
            } catch (Exception e) {
            }
        }
        System.out.println("So: " + num);
        flag = true;
        notifyAll();
    }
    public synchronized void kiemtra(){
        while (!flag) {
            try {
                wait();
```

```
            } catch (Exception e) {
            }
        }
            if (laSoNgTo())
            System.out.println(num+ " la so nguyen to.");
        else
        System.out.println(num+ " khong phai la so nguyen to.");
        num++;
        flag = false;
        notifyAll();
    }
    private boolean laSoNgTo() {
        if (num == 1) return false;
        for (int i= 2; i < num; i++) {
            if (num % i == 0)
                return false;
        }
        return true;
    }
}
```

Bước 4: Tạo class Test

```
public class Test{
    public static void main(String[] args) {
        Number num = new Number();
        Thread1 t1 = new Thread1(num);
        Thread2 t2 = new Thread2(num);
        t1.start();
        t2.start();
    }
}
```

Bài 6.

Tạo ứng dụng java sử dụng MultilThread và đồng bộ luồng sử dụng (synchronized)

Luồng 1: Hiển thị ngày, giờ hệ thống.

Luồng 2: Dựa trên số giây hiển thị trong luồng 1, hiển thị thông tin tương ứng là “giây là chẵn” hay “giây là lẻ”.

Viết phương thức main ( ) minh họa hai luồng này. Luồng 1: now 12:20:22

Luồng 2: Giây là lẻ

Hướng dẫn:

Bước 1: Tạo class myObject

```
public class myObject {
    boolean flag = false;
    DateFormat df;
    int second;
    public synchronized void thoiGian() {
        while (flag) {
            try {
                wait();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        df = new SimpleDateFormat("HH:mm:ss");
        String format = df.format(new Date());
        System.out.println(format);
        second = Calendar.getInstance().get(Calendar.SECOND);
        flag = true;
        notifyAll();
    }
    public synchronized void kiemtra() {
        while (!flag) {
            try {
                wait();
            } catch (Exception e) {
                e.printStackTrace();
            }
   }
        if(second % 2 ==0 )
```

```
            System.out.println("The "+ second +" is Even");
        else
            System.out.println("The "+ second +" is Odd");
        flag = false;
        notifyAll();
    }
```

Bước 2: Tạo Thread 1

```
public class Thread1 extends Thread{
    myObject obj;
    public Thread1(myObject obj) {
        this.obj = obj;
    }
    @Override
    public void run() {
    //myObject obj = new myObject();
        try {
            while (true) {
                obj.thoiGian();
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Bước 3: Tạo Thread2

```
public class Thread2 extends Thread{
    myObject obj;
    public Thread2(myObject obj) {
        this.obj = obj;
    }
    @Override
```

```
    public void run() {
        try {
            while (true) {
                obj.kiemtra();
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Bước 4: Tạo class Main

```
public class Main {
    public static void main(String[] args) {
        myObject obj = new myObject();
        Thread1 t1 = new Thread1(obj);
        Thread2 t2 = new Thread2(obj);
        t1.start();
        t2.start();
    }
}
```

Bài 7.

- Tạo **class Bank** gồm các trường và phương thức sau:

- Tạo 1 mảng: **double**[] accounts
- Tạo 1 constructors với 2 tham số: **int** n, và double initBalance, n là số phần tử mảng accounts, initBalance là giá trị ban đầu gán cho từng phần tử trong mảng accounts.
- Tạo phương thức tên **size**(), trả về kích thước của mảng
- Tạo phương thức tên **getTotalBalance**(), trả về tổng giá trị của mảng. Phương thức này được đồng bộ hoá.
- Tạo phương thức **transfer (int from, int to, double amount)** bắt lỗi InterruptedException thực hiện công việc sau:
- Khi giá trị của phần tử **from** trong mảng nhỏ hơn **amount**, thread rơi vào trạng thái **wait**
- In ra tên của thread hiện thời

- Chuyển tiền với giá trị là **amount** từ **from** sang **to**. Mỗi lần chuyển tiền đều phải đưa ra thông báo. Ví dụ: Chuyển 200 từ account 12 sang account 24.
- In ra tổng giá trị của tất cả các account, sử dụng **getTotalBalance**().
- Sau khi thực hiện xong phải đánh thức các thread khác.
- Phương thức phải được đồng bộ hoá

- Tạo **class TransferMoney**, thực thi giao diện **Runnable**, gồm các trường và phương thức sau:

- Tạo 4 trường như sau: **Bank bank, int fromAcc, double maxAmount, int delay**.
- Tạo phương thức khởi tạo cho **bank, fromAcc** và **maxAmount**. Trường **delay** luôn có giá trị là 1000.
- Override phương thức **run**() để thực hiện công việc sau:
- Chuyển tiền từ account **fromAcc** sang account **toAcc** của **bank**, biết rằng **toAcc** là 1 account lấy ngẫu nhiên trong số các account còn lại của **bank**. Lượng tiền chuyển **amount** cũng là ngẫu nhiên, trong khoảng từ 0 tới **maxAmount**.
- Cho thread nghỉ 1 khoảng thời gian từ 0 đến 1 giây.

- Tạo **class SynchBank** để thực hiện 2 **class** trên như sau:

- Khởi tạo 1 **bank** có 100 account với giá trị ban đầu là 1000 usd.
- Chạy các thread **TransferMoney** đối với 100 account của **bank**

Hướng dẫn:

Bước 1: Tạo class Bank

```
public class Bank
{
    double[] accounts;
        public Bank(int n, double initBalance)
    {
        accounts = new double[n];
        for (int i = 0; i < accounts.length; i++)
        {
            accounts[i] = initBalance;
        }
    }
 public int size()
    {
        return accounts.length;
    }
  public synchronized double getTotalBalance()
       {
        double total = 0;
        for (int i = 0; i < accounts.length; i++)
        {
```

```
            total+=accounts[i];
        }
        return total;
       }
  public synchronized void transfer(int from, int to, double amount)
         {
        try
           {
            while(accounts[from] < amount)
            {
      System.out.println(Thread.currentThread().getName()+"đợi đủ tiền");
                wait();
System.out.println(Thread.currentThread().getName()+"tiếp tục giao
dịch");
            }
            accounts[from] -= amount;
            accounts[to] += amount;
 System.out.println("Chuyển " + amount + " từ account " + from + " sang
account " + to);
            System.out.println("Tổng tiền của các account: " +
getTotalBalance());
            notifyAll();
        }
        catch (InterruptedException ex)
        {
            InterruptedException("Giao dịch bị gián đoạn");
        }
    }
}
```

Bước 2: Tạo class TransferMoney

```
public class TransferMoney implements Runnable
{
    Bank bank;
    int fromAcc;
    double maxAmount;
    int delay = 1000;
    public TransferMoney(Bank bank, int fromAcc, double maxAmount) {
        this.bank = bank;
        this.fromAcc = fromAcc;
        this.maxAmount = maxAmount;
```

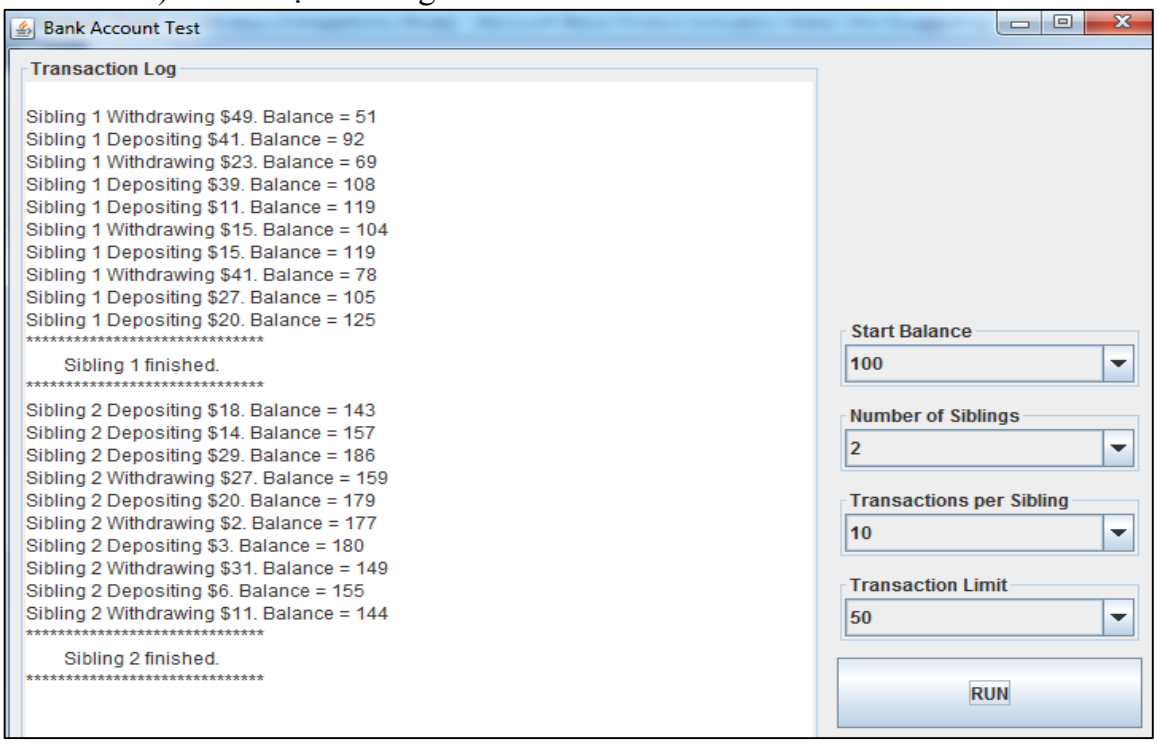
```
    }
        @Override
    public void run()
    {
        Random rd = new Random();
        int toAcc = 0;
        double amount = 0;
        try
        {
            while (true)
            {
                do
                {
                 toAcc = rd.nextInt(bank.size());
                }while (toAcc==fromAcc);
                amount = rd.nextInt((int)maxAmount);
                bank.transfer(fromAcc, toAcc, amount);
                Thread.sleep(rd.nextInt(delay));
            }
        }
        catch(InterruptedException ex)
        {
            InterruptedException("Giao dịch chuyển tiền từ account " +
fromAcc + " sang account " + toAcc + " bị gián đoạn");
        }
      }
    }
```

Bước 3. Tạo class SynchBank

```
public class SynchBank
{
    public static void main(String[] args) throws InterruptedException {
        Bank bank = new Bank(100, 1000);
        int size = bank.size();
        for (int i = 0; i < size; i++)
        {
          TransferMoney transferMoney = new TransferMoney(bank, i, 1000);
            Thread thread = new Thread(transferMoney);
            thread.start();
        }
```

```
    }
}
```

Bài 9. Tổng hợp Viết chương trình minh họa việc gửi và rút tiền tại ngân hàng (sử dụng mô hình MVC). Giao diện chương trình như sau:



Hình 3. Màn hình giao dịch ngân hàng

Yêu cầu: Một số anh chị em được cha mẹ cấp quyền truy cập vào tài khoản ngân hàng. Đối với mỗi lần chạy mô phỏng gồm các thông số:

- Số dư của tài khoản ngân hàng- số tiền ban đầu (start balance)
- Số lượng người dùng có quyền truy cập tài - số anh chị em (Number of Siblings)
- Đối với mỗi anh chị em, có số lần giao dịch giới hạn. (Transactions per Sibling).
- Giới hạn giao dịch: Transaction Limit.
- Mỗi lần giao dịch bao gồm hành động gửi và rút với số tiền xác định. Tất cả đều được tạo ngẫu nhiên.
- Mỗi anh chị em thực hiện danh sách các giao dịch của mình, với ràng buộc là số dư không thể nhỏ hơn 0

Kết quả của mỗi giao dịch được hiển thị trong nhật ký giao dịch.

Hướng dẫn:

Thiết kế kiến trúc phần mềm Model-View-Controller (MVC) trong đó:

- Model: Tài khoản ngân hàng
- View: Nhật ký giao dịch
- Control: code cho các chức năng trong combobox và button run

Bước 1: Tạo class BankAccount, lớp này kế thừa từ class Observable.

Lớp Observable được sử dụng để tạo các lớp con mà các phần khác của chương trình có thể quan sát. Khi một đối tượng của lớp con đó xảy ra một sự thay đổi, các lớp quan sát sẽ nhận được thông báo.

```
import java.util.Observable;
public class BankAccount extends Observable {
private int balance; //số dư ban đầu
//Phương thức thiết lập số dư
  public void setBalance(int balance) {
        this.balance = balance;
        setChanged();
        notifyObServers(null);
    }
 //Phương thức nạp tiền vào tài khoản
public void deposit(int amount, BankAccountUser user) {
        log("\n" +user.getName()+ " Depositing $" +amount);
        int newBalance = balance + amount;
        balance = balance + amount;
        log(". Balance = " + balance);
        checkFinished(user);
        assert(balance == newBalance);
}
//Phương thức rút tiền
public void withdraw(int amount, BankAccountUser user) {
        log("\n" +user.getName() + " Withdrawing $" + amount);
        if (amount > balance) {
        throw new RuntimeException("Amount (" +amount+ ") must not be
greater than " +balance+ ".");
        }
        int newBalance = balance - amount;
        balance = balance - amount;
        log(". Balance = " + balance);
        checkFinished(user);
        assert(balance == newBalance);
    }
```

```
//Phương thức trả về số dư
    public int getBalance() {
        return balance;
    }
}
 //Phương thức kiểm tra kết thúc
private void checkFinished(BankAccountUser user) {
        if ( user.isOneMore() ) {
            log("\n****\n " + user.getName() +  " finished.\*******");
            user.setFinished(true);
        }
    }
//Phương thức theo dõi tín hiệu từ tài khoản bằng cách gửi đến tất cả các đối tượng đang  quan sát.
private void log(String message) {
        setChanged();
        notifyObServers(message);
}
```

Bước 2: Tạo class BankAccountUser gồm các thuộc tính: name, account, list giao dịch

```
public class BankAccountUser {
  private String name;
    private BankAccount account;
    private List<Integer> transactions;
    private boolean oneMore;
    private boolean finished;
    public BankAccountUser(String name, BankAccount account,
List<Integer> transactions) {
        this.name = name;
        this.account = account;
        this.transactions = transactions;
        oneMore = false;
        finished = false;
    }
```

```
    public String getName() {
        return name;
    }
    public BankAccount getAccount() {
        return account;
    }
    public boolean isFinished() {
        return finished;
    }
    public void setFinished(boolean finished) {
        this.finished = finished;
    }
    public boolean isOneMore() {
        return oneMore;
    }
    public void setOneMore(boolean oneMore) {
        this.oneMore = oneMore;
    }
    public void run() {
        Iterator<Integer> iter = transactions.iterator();
            while (iter.hasNext()) {
                int amount = iter.next();
                if ( !iter.hasNext() ) {
                    oneMore = true;
                }
                if (amount > 0) {
                    account.deposit(amount, this);
                }
                else if (amount < 0) {
                    account.withdraw(Math.abs(amount), this);
                } // amount == 0 ignored
          }
    }
}
```

Bước 3: Tạo class BankAccountControl

```
public class BankAccountControl extends JComponent {
    private static Random generator = new Random();
    private BankAccount account;
    private BankAccountUser[] users;
    private JComboBox balanceChoices;
    private JComboBox usersChoices;
    private JComboBox limitChoices;
    private JComboBox transactionsChoices;
    private JButton runButton;
    private int startBalance;
    private int numUsers;
    private int amountLimit;
    private int numTransactions;
    public BankAccountControl(BankAccount account) {
    this.account = account;
 balanceChoices = makeComboBox("Start Balance", new int[] {100, 500,
1000,   10000});
      usersChoices = makeComboBox("Number of Siblings", new int[] {1, 2,
3, 4, 5});
      limitChoices = makeComboBox("Transaction Limit", new int[] {50,
100, 200, 500});
  transactionsChoices = makeComboBox("Transactions per Sibling", new
int[] {10, 25, 50, 100});
         runButton = new JButton("RUN");
         runButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                setup();
                run();
            }
        });
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(5, 1, 0, 10));
```

```
        panel.add(balanceChoices);
        panel.add(usersChoices);
        panel.add(transactionsChoices);
        panel.add(limitChoices);
        panel.add(runButton);
        setLayout(new FlowLayout());
        add(panel);
    }
  private JComboBox makeComboBox(String title, int[] options) {
        JComboBox comboBox = new JComboBox();
        comboBox.setPreferredSize(new Dimension(title.length()*8, 50));
        comboBox.setBorder(new TitledBorder(title));
        for (int i = 0; i < options.length; i++) {
            comboBox.addItem(options[i]);
        }
        return comboBox;
    }
  private void setup() {
        startBalance = (Integer) balanceChoices.getSelectedItem();
        numUsers = (Integer) usersChoices.getSelectedItem();
        amountLimit = (Integer) limitChoices.getSelectedItem();
       numTransactions = (Integer) transactionsChoices.getSelectedItem();
                account.setBalance(startBalance);
        users = new BankAccountUser[numUsers];
                for (int i = 0; i < users.length; i++) {
          users[i] = new BankAccountUser("Sibling " + (i+1), account,
makeTransactions());
        }
    }
        private void run() {
        try {
            for (int i = 0; i < users.length; i++) {
                users[i].run();
            }
```

```
        }
        catch(RuntimeException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
        }
    }
    private List<Integer> makeTransactions() {
        List<Integer> transactions = new ArrayList<Integer>();
        for (int i = 0; i < numTransactions; i++) {
            int amount = generator.nextInt(amountLimit) + 1;
transactions.add(generator.nextBoolean() ? amount : -amount);
        }
        return transactions;
    }
}
```

Bước 4: Tạo lớp BankAccountComponent

```
public class BankAccountComponent extends JComponent {
        public BankAccountComponent() {
        BankAccount account = new BankAccount();
        LogView view = new LogView("Transaction Log");
        account.addobServer(view);
        BankAccountControl control = new BankAccountControl(account);
        setLayout(new FlowLayout());
        add(view);
        add(control);
    }
}
```

Bước 5: Tạo lớp BankAccountFrame

```
public class BankAccountFrame extends JFrame {
       public BankAccountFrame() {
        super("Bank Account Test");
        add(new BankAccountComponent());
        pack();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        setVisible(true);
    }
public static void main(String[] args) {
        new BankAccountFrame();
    }
}
```

Bước 6: Tạo class LogView

```
public class LogView extends JScrollPane implements ObServer {
        public LogView(String title) {
        super(transactionArea);
        setBorder(new TitledBorder(title));
    }
public void update(Observable account, Object message) {
        if ( message == null ) {
            transactionArea.setText("");
        }
        else {
                transactionArea.append((String)message);
             }
    }
 private static JTextArea transactionArea = new JTextArea(40,45);
  }
```

TÀI LIỆU THAM KHẢO

[1]. Cay S. Horstmann, *Core Java Volum I - Fundamentals, Tenth Edition*, NewYork : Prentice Hall, 2016.

[2]. Cay S. Horstmann. *Core Java Volum II - Advanced Features, Tenth Edition*, New York : Prentice Hall, 2017.

[3].Eng.haneen Ei-masry, *Java database connection*, Islamic University of Gaza Faculty of Engineering Department of Computer Engineering ECOM 4113: DataBase Lab, 2014.

[4]. Angelos Stavrou, *Advanced Network Programming Lab using Java*, Network Security, ISA 656, Angelos Stavrou.

[5]. Marenglen Biba, Ph.D, *Manual for Lab practices, Remote Method Invocation Three Tier Application with a Database Server*, Department of Comsputer Science, University of New York.

[6].Elliotte Rusty Harold, *Java Network Programming, Fourth Edition*, O'Reilly Media, 2013.

[7]. Đoàn Văn Ban, *Lập trình hướng đối tượng với JAVA*, Nhà xuất bản Khoa học và Kỹ thuật, 2005.

[8]. ThS. Dương Thành Phết, *Bài tập thực hành Chuyên đề 1 CNPM- Java*, Khoa CNTT- Trường ĐH Công nghệ TP.HCM.

[9]. https://www.oracle.com/technetwork/java/socket-140484.html#

[10]. https://personales.unican.es/corcuerp/java/Labs/LAB_22.htm

[11]. http://www.nrcmec.org/pdf/Manuals/CSE/student/2-2%20java16-17.pdf

[12]. http://cse.mait.ac.in/pdf/LAB%20MANUAL/JAVA.pdf

[13]. https://www.academia.edu/35283541/Bài tập môn lập trình hướng đối tượng