**LẬP TRÌNH MẠNG**

Nhóm OTTI:

1. Phan Văn Thông
2. Zơ Râm Guân
3. Nguyễn Hoàng Lân
4. Thượng Thị Thu Hiệp.

**BÁO CÁO - CHƯƠNG 2**

1. Viết chương trình tạo ra 4 tuyến thực hiện các công việc:
   1. In các số số lẻ nhỏ hơn số nguyên n
   2. In các số số chẵn nhỏ hơn số nguyên n
   3. In các số từ 1 🡪 n
   4. In các kí tự hoa trong bảng mã ASCII

Bài làm

**package** MultiThreading;

**public** **class** C21 {

**public** **static** **void** main(String argv[])

{

**try**

{

InChan ic = **new** InChan("So Chan", 250);

InLe il = **new** InLe("So le",255);

InNhoHonN nh= **new** InNhoHonN("Nho hon N", 200);

InHoa ih = **new** InHoa("In hoa");

}

**catch**(Exception ie)

{

System.***out***.println("Pipe Echo Error: "+ie);

}

}

}

**class** InLe **extends** Thread{

String name;

**private** **final** **int** n;

InLe(String name, **int** n)

{

**this**.name= name;

**this**.n=n;

System.***out***.println("Tao luong "+name);

start();

}

**public** **void** run()

{

**for**(**int** j=0;j<n;j++)

{

**if**(j%2!=0)

System.***out***.println("So le: "+j);

}

}

}

**class** InChan **extends** Thread{

String name;

**private** **final** **int** n;

InChan(String name, **int** n)

{

**this**.name= name;

**this**.n=n;

System.***out***.println("Tao luong "+name);

start();

}

**public** **void** run()

{

**for**(**int** j=0;j<n;j++)

{

**if**(j%2==0)

System.***out***.println("So chan: "+j);

}

}

}

**class** InNhoHonN **extends** Thread{

String name;

**private** **final** **int** n;

InNhoHonN(String name, **int** n)

{

**this**.name= name;

**this**.n=n;

System.***out***.println("Tao luong "+name);

start();

}

**public** **void** run()

{

**for**(**int** j=0;j<n;j++)

{

System.***out***.println("So nho hon: "+n+": "+j);

}

}

}

**class** InHoa **extends** Thread{

String name;

InHoa(String name)

{

**this**.name= name;

System.***out***.println("Tao luong "+name);

start();

}

**public** **void** run()

{

**for**(**int** j=65;j<91;j++)

{

System.***out***.println("Ky tu: "+(**char**)j);

}

}

}

1. Viết chương trình tạo ra 3 tuyến:
   1. Nhập vào 2 số thực là 2 cạnh của hình chữ nhật
   2. Tính S hình chữ nhật
   3. In ra kết quả diện tích và chu vi hình chữ nhật

Bài làm

package com.company;  
  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 HinhChuNhat hinhChuNhat = new HinhChuNhat();  
  
 // nhập chiều dài và chiều rộng hình chữ nhật  
 hinhChuNhat.nhap();  
  
 // hiển thị chiều dài và chiều rộng  
 hinhChuNhat.hienThi(hinhChuNhat.chieuDai, hinhChuNhat.chieuRong);  
  
 // tính chu vi và diện tích  
 double chuVi = hinhChuNhat.tinhChuVi(hinhChuNhat.chieuDai, hinhChuNhat.chieuRong);  
 double dienTich = hinhChuNhat.tinhDienTich(hinhChuNhat.chieuDai, hinhChuNhat.chieuRong);  
  
 // hiển thị chu vi và diện tích  
 hinhChuNhat.hienThiChuViVaDienTich(chuVi, dienTich);  
 }  
 }

package com.company;  
  
import java.util.Scanner;  
  
public class HinhChuNhat {  
 protected double chieuDai, chieuRong, chuVi, dienTich;  
 Scanner scanner = new Scanner(System.*in*);  
  
 public void nhap() {  
 do {  
 System.*out*.println("Nhập chiều dài hình chữ nhật: ");  
 chieuDai = scanner.nextDouble();  
 System.*out*.println("Nhập chiều rộng hình chữ nhật: ");  
 chieuRong = scanner.nextDouble();  
 } while (chieuDai < chieuRong);  
 }  
  
 public void hienThi(double dai, double rong) {  
 System.*out*.println("Chiều dài và chiều rộng của hình chữ nhật lần lượt là "  
 + chieuDai + " và " + chieuRong);  
 }  
  
 public double tinhChuVi(double dai, double rong) {  
 chuVi = (dai + rong) \* 2;  
 return chuVi;  
 }  
  
 public double tinhDienTich(double dai, double rong) {  
 dienTich = dai \* rong;  
 return dienTich;  
 }  
  
 public void hienThiChuViVaDienTich(double cv, double dt) {  
 System.*out*.println("Chu vi hình chữ nhật = " + cv);  
 System.*out*.println("Diện tích hình chữ nhật = " + dt);  
 }  
  
}

1. Viết chương trình tạo ra 3 tuyến:
   1. Cứ mỗi 2000 miligiay sẽ tạo ra 1 số nguyên ngẫu nhiên từ 1🡪 20
   2. Cứ mỗi 5000 miligiay sẽ tạo ra 1 số nguyên ngẫu nhiên mà luồng tạo ra tính bình phương của nó và hiện thị ra màn hình
   3. Tạo và khởi động 2 luồng này

Bài làm

**package** MultiThreading;

**import** java.util.Random;

**public** **class** C23 {

**public** **static** **int** *so*=0;

**public** **static** **void** main(String argv[])

{

**try**

{

SoNgauNhien snn= **new** SoNgauNhien();

TinhBP tbp= **new** TinhBP();

System.***out***.println("Nhan enter de dung");

System.***in***.read();

snn.~~stop~~();

tbp.~~stop~~();

}

**catch**(Exception e)

{

System.***out***.println("Pipe Echo Error: "+e);

}

}

**class** SoNgauNhien **extends** Thread

{

Random taoSo=**new** Random();

SoNgauNhien()

{

System.***out***.println("Tao luong so ngau nhien");

start();

}

**public** **void** run()

{

**while**(1>0)

{

**try**

{

C23.*so* = taoSo.nextInt(20)+1;

System.***out***.println("So ngau nhien cua tao = : "+ C23.*so*);

Thread.*sleep*(2000);

}

**catch**(Exception e)

{

System.***out***.println("Error: "+e);

}

}

}

}

**class** TinhBP **extends** Thread

{

TinhBP()

{

System.***out***.println("Tao luong tinh binh phuong");

start();

}

**public** **void** run()

{

**while**(1>0)

**try**

{

System.***out***.println();

System.***out***.println("Binh phuong cua:" +C23.*so*+" = " +C23.*so*\*C23.*so*);

Thread.*sleep*(5000);

}**catch**(Exception e)

{

System.***out***.println("Error "+e);

}

}

}

}

1. Cho người dùng nhập 2 số nguyên a và b, sau đó dùng n thread để tìm số nguyên tố nằm trong khoảng 2 số đã nhập, theo trật tự từ nhỏ đến lớn.

Bài làm

**class** Main {

**public** Main()

{ array = **new** **int**[10];

**for** (**int** i = 0; i < array.length; i++)

{ array[i] = ran.nextInt(10);

} }

**int**[] array;

Random ran = **new** Random();

**public** **void** sapXep(**int** startNum, **int** endNum) {

**for** (**int** i = startNum; i < endNum - 1; i++)

{ **for** (**int** j = i + 1; j < endNum; j++) {

**if** (array[i] > array[j]) {

**int** temp = array[i]; array[i] = array[j]; array[j] = temp; } } } }

**public** **void** in(**int** startNum, **int** endNum) {

**for** (**int** i = startNum; i < endNum; i++)

{ System.***out***.println(array[i] + " ");

} } }

// <<<< extend the class Main >>>>>>> class TaoLuong extends Main implements Runnable { public TaoLuong(int startNum, int endNum) { this.startNum = startNum; this.endNum = endNum; }

**int** startNum, endNum; @Override **public** **void** run() { //synchronized (main) { sapXep(startNum, endNum); in(startNum, endNum); //} } }

**public** **class** SapXep { **public** **static** **void** main(String[] args) {

Main main = **new** Main();

TaoLuong taoLuong1 = **new** TaoLuong(0, main.array.length / 2);

Thread t1 = **new** Thread(taoLuong1); t1.start();

TaoLuong taoLuong2 = **new** TaoLuong(main.array.length / 2, main.array.length);

Thread t2 = **new** Thread(taoLuong2); t2.start(); } }}

1. Viết chương trình tạo ra 2 tuyến, tuyến 1 tuyến tìm số nguyên tố từ 1-1000 và tuyến 2 tính tổng giá trị của các số nguyên tố mà tuyến 1 tìm được

Bài làm

**package** MultiThreading;

**public** **interface** Buffer {

**public** **void** set(**int** value) **throws** InterruptedException;

**public** **int** get() **throws** InterruptedException;

**public** **void** setTerminate(**boolean** terminate);

**public** **boolean** isTerminate();

}

package MultiThreading;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class SharedBufferTest {

public static void main(String[] args) {

ExecutorService app = Executors.newCachedThreadPool();

Buffer sharedLocation = new SynchronizedBuffer();

app.execute(new FindPrime(sharedLocation));

app.execute(new SumPrime(sharedLocation));

app.shutdown();

}

}

**package** MultiThreading;

**public** **class** FindPrime **implements** Runnable {

**private** **final** Buffer sharedLocation;

**private** **int** i = 0;

**public** FindPrime(Buffer shared) {

sharedLocation = shared;

}

@Override

**public** **void** run() {

**try** {

**for** (**int** n = 1; n < 1000; n++) {

**if** (*isPrime*(n)) {

sharedLocation.set(n);

System.***out***.printf("%5d ", n);

**if** ((++i) % 10 == 0) {

System.***out***.println("");

}

}

}

sharedLocation.setTerminate(**true**);

sharedLocation.set(0);

} **catch** (InterruptedException exception) {

exception.printStackTrace();

}

}

**public** **static** **boolean** isPrime(**int** i) {

**int** c;

**for** (c = 2; c <=Math.*sqrt*(i); c++) {

**if** (i % c == 0) {

**return** **false**;

}

}

**return** **true**;

}

}

package MultiThreading;

import java.util.logging.Level;

import java.util.logging.Logger;

@SuppressWarnings("unused")

public class SumPrime implements Runnable {

private final Buffer sharedLocation;

private int sum = 0;

private boolean FindPrimeTask = false;

public SumPrime(Buffer shared) {

sharedLocation = shared;

}

@Override

public void run() {

try {

while (true) {

if (sharedLocation.isTerminate()) {

break;

}

System.out.print("."); // wait

sum += sharedLocation.get();

}

} catch (InterruptedException ex) {

ex.printStackTrace();

}

System.out.println("\n Sum Prime: " + sum);

}

}

**package** MultiThreading;

**public** **class** SynchronizedBuffer **implements** Buffer {

**private** **int** buffer = -1;

**private** **boolean** finded = **false**;

**private** **boolean** terminate = **false**;

**public** **synchronized** **void** set(**int** value) **throws** InterruptedException {

**while** (finded) {

wait();

}

buffer = value;

finded = **true**;

notifyAll();

}

**public** **synchronized** **int** get() **throws** InterruptedException {

**if**(isTerminate())

Thread.*interrupted*();

**while** (!finded) {

wait();

}

finded = **false**;

notifyAll();

**return** buffer;

}

**public** **boolean** isTerminate() {

**return** terminate;

}

**public** **void** setTerminate(**boolean** terminate) {

**this**.terminate = terminate;

}

}

1. Viết lại ví dụ trong chương 2

Ví dụ: Viết chương trình để kiểm tra trên 1024 cổng đầu tiên những cổng nào đang có server hoạt động

import java.net.\*;

import java.io.\*;

class PortScanner

{

public static void main(String[] args)

{

String host=”localhost”;

if(args.length>0){

host=args[0];

}

for(int i=0;i<1024;i++){

try{

Socket s=new Socket(host,i);

System.out.println(“Co mot server dang hoat dong tren cong:”+i);

}

catch(UnknownHostException e){

System.err.println(e);

}

catch(IOException e){

System.err.println(e);

}

}

}

}

public Socket(InetAddress host, int port)throws IOException

Tương tự như constructor trước, constructor này tạo một socket TCP với thông tin là địa chỉ của một host được xác định bởi một đối tượng InetAddres và số hiệu cổng port, sau đó nó thực hiện kết nối tới host. Nó đưa ra ngoại lệ IOException nhưng không đưa ra ngoại lệ UnknownHostException. Constructor đưa ra ngoại lệ trong trường hợp không kết nối được tới host.

public Socket (String host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException

Constructor này tạo ra một socket với thông tin là địa chỉ IP được biểu diễn bởi một đối tượng String và một số hiệu cổng và thực hiện kết nối tới host đó. Socket kết nối tới host ở xa thông qua một giao tiếp mạng và số hiệu cổng cục bộ được xác định bởi hai tham số sau. Nếu localPort bằng 0 thì Java sẽ lựa chọn một cổng ngẫu nhiên có sẵn nằm trong khoảng từ 1024 đến 65535.

public Socket (InetAddress host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException

Constructor chỉ khác constructor trên ở chỗ địa chỉ của host lúc này được biểu diễn bởi một đối tượng InetAddress.

4.2. Nhận các thông tin về Socket

Đối tượng Socket có một số trường thông tin riêng mà ta có thể truy nhập tới chúng thông qua các phương thức trả về các thông tin này.

public InetAddress getInetAddress()

Cho trước một đối tượng Socket, phương thức getInetAddress() cho ta biết host ở xa mà Socket kết nối tới, hoặc liên kết đã bị ngắt thì nó cho biết host ở xa mà Socket đã kết nối tới

public int getPort()

Phương thức này cho biết số hiệu cổng mà Socket kết nối tới trên host ở xa.

public int getLocalPort()

Thông thường một liên kết thường có hai đầu: host ở xa và host cục bộ. Để tìm ra số hiệu cổng ở phía host cục bộ ta gọi phương thức getLocalPort().

public InetAddress getLocalAddress()

Phương thức này cho ta biết giao tiếp mạng nào mà một socket gắn kết với nó.

public InputStream getInputStream() throws IOException

Phương thức geInputStream() trả về một luồng nhập để đọc dữ liệu từ một socket vào chương trình. Thông thường ta có thể gắn kết luồng nhập thô InputStream tới một luồng lọc hoặc một luồng ký tự nhằm đưa các chức năng tiện ích (chẳng hạn như các luồng InputStream, hoặc InputStreamReader). Để tâng cao hiệu năng, ta có thể đệm dữ liệu bằng cách gắn kết nó với luồng lọc BufferedInputStream hoặc BufferedReader.

public OutputStream getOutputStream() throws IOException

Phương thức getOutputStream() trả về một luồng xuất thô để ghi dữ liệu từ ứng dụng ra đầu cuối của một socket. Thông thường, ta sẽ gắn kết luồng này với một luồng tiện lợi hơn như lớp DataOuputStream hoặc OutputStreamWriter trước khi sử dụng nó. Để tăng hiệu quả ghi.

Hai phương thức getInputStream() và getOutputStream() là các phương thức cho phép ta lấy về các luồng dữ liệu nhập và xuất. Như đã đề cập ở chương 3 vào ra trong Java được tiến hành thông qua các luồng, việc làm việc với các socket cũng không phải là một ngoại lệ. Để nhận dữ liệu từ một máy ở xa ta nhận về một luồng nhập từ socket và đọc dữ liệu từ luồng đó. Để ghi dữ liệu lên một máy ở xa ta nhận về một luồng xuất từ socket và ghi dữ liệu lên luồng. Dưới đây là hình vẽ để ta hình dung trực quan hơn.

InputStream

OutputStream

Socket

Chương trình

Hình 4.5

4.3. Đóng Socket

Đến thời điểm ta đã có đầy đủ các thông tin cần thiết để triển khai một ứng dụng phía client. Khi viết một chương trình ứng dụng phía client tất cả mọi công việc đều chuyển về việc quản lý luồng và chuyển đổi dữ liệu từ luồng thành dạng thức mà người sử dụng có thể hiểu được. Bản thân các socket rất đơn giản bởi vì các phần việc phức tạp đã được che dấu đi. Đây chính là lý do để socket trở thành một lựa chọn có tính chiến lược cho lập trình mạng.

public void close() throws IOException

Các socket được đóng một cách tự động khi một trong hai luồng đóng lại, hoặc khi chương trình kết thúc, hoặc khi socket được thu hồi bởi gabbage collector. Tuy nhiên, thực tế cho thấy việc cho rằng hệ thống sẽ tự đóng socket là không tốt, đặc biệt là khi các chương trình chạy trong khoảng thời gian vô hạn. Để đóng một socket ta có thể dùng phương thức close().

Mỗi khi một Socket đã bị đóng lại, ta vẫn có thể truy xuất tới các trường thông tin InetAddress, địa chỉ cục bộ, và số hiệu cổng cục bộ thông qua các phưong thức getInetAddress(), getPort(), getLocalHost(), và getLocalPort(). Tuy nhiên khi ta gọi các phương thức getInputStream() hoặc getOutputStream() để đọc dữ liệu từ luồng đọc InputStream hoặc ghi dữ liệu OuputStream thì ngoại lệ IOException được đưa ra.

Các socket đóng một nửa (Half-closed socket)

Phương thức close() đóng cả các luồng nhập và luồng xuất từ socket. Trong một số trường hợp ta chỉ muốn đóng một nửa kết nối, hoặc là luồng nhập hoặc là luồng xuất. Bắt đầu từ Java 1.3, các phương thưc shutdownInput() và shutdownOutput() cho phép ta thực hiện điều này.

public void shutdownInput() throws IOException

public void shutdownOutput() throws IOException

Các phương thức này không thực sự ngắt liên kết. Tuy nhiên, nó chỉ điều chỉnh luồng kết nối tới nó sao cho.

Trong Java 1.4 đưa thêm vào hai phương thức các luồng nhập và luồng xuất mở hay đóng

public boolean isInputShutdown()

public boolean isOutputShutdown()

4.4. Thiết lập các tùy chọn cho Socket

4.4.1. TCP\_NODELAY

public void setTcpNoDelay(boolean on) throws SocketException

public boolean getTcpNoDelay() throws SocketException

Thiết lập giá trị TCP\_NODELAY là true để đảm bảo rằng các gói tin được gửi đi nhanh nhất có thể mà không quan tâm đến kích thước của chúng. Thông thường, các gói tin nhỏ được kết hợp lại thành các gói tin lớn hơn trước khi được gửi đi. Trước khi gửi đi một gói tin khác, host cục bộ đợi để nhận các xác thực của gói tin trước đó từ hệ thống ở xa.

4.4.2. SO\_LINGER

public void setSoLinger(boolean on, int seconds) throws SocketException

public int getSoLinger() throws SocketException

Tùy chọn SO\_LINGER xác định phải thực hiện công việc gì với datagram vẫn chưa được gửi đi khi một socket đã bị đóng lại. Ở chế độ mặc định, phương thức close() sẽ có hiệu lực ngay lập tức; nhưng hệ thống vẫn cố gắng để gửi phần dữ liệu còn lại. Nếu SO\_LINGER được thiết lập bằng 0, các gói tin chưa được gửi đi bị phá hủy khi socket bị đóng lại. Nếu SO\_LINGER lớn hơn 0, thì phương thức close() phong tỏa để chờ cho dữ liệu được gửi đi và nhận được xác thực từ phía nhận. Khi hết thời gian qui định, socket sẽ bị đóng lại và bất kỳ phần dữ liệu còn lại sẽ không được gửi đi.

4.4.3. SO\_TIMEOUT

public void setSoTimeout(int milliseconds) throws SocketException

public int getSoTimeout() throws SocketException

Thông thường khi ta đọc dữ liệu từ mộ socket, lời gọi phương thức phong tỏa cho tới khi nhận đủ số byte. Bằng cách thiết lập phương thức SO\_TIMEOUT, ta sẽ đảm bảo rằng lời gọi phương thức sẽ không phong tỏa trong khoảng thời gian quá số giây quy định.

4.5. Các phương thức của lớp Object

Lớp Socket nạp chồng phương thức chuẩn của lớp java.lang.Object, toString(). Vì các socket là các đối tượng tạm thời và thường chỉ tồn tại khi liên kết tồn tại.

public String toString()

Phương thức toString() tạo ra một xâu ký tự như sau:

Socket[addr=www.oreilly.com/198.122.208.11,port=80,localport=50055]

Phương thức này thường hữu ích cho việc gỡ rối.

4.6. Các ngoại lệ Socket

Hầu hết các phương thức của lớp Socket được khai báo đưa ra ngoại lệ IOException, hoặc lớp con của lớp IOExcepton là lớp SocketException.

4.7. Các lớp SocketAddress

Lớp SocketAddress bắt đầu có từ phiên bản Java 1.4, biểu diễn một đầu cuối của liên kết. Lớp SocketAddress là một lớp trừu tượng mà không có phương thức nào ngoài construtor mặc định. Lớp này có thể được sử dụng cho cả các socket TCP và socket không phải là TCP. Các lớp con của lớp SocketAddress cung cấp thông tin chi tiết hơn thích hợp cho kiểu socket. Trong thực tế, chỉ hỗ trợ TCP/IP.

Mục đích chính của lớp SocketAddress là cung cấp một nơi lưu trữ các thông tin liên kết socket tạm thời (như địa chỉ IP và số hiệu cổng) có thể được sử dụng lại để tạo ra socket mới.

public SocketAddress getRemoteSocketAddress()

public SocketAddress getLocalSocketAddress()

Cả hai phương thức này trả về giá trị null nếu socket vẫn chưa kết nối tới.

5. Lớp ServerSocket

Lớp ServerSocket có đủ mọi thứ ta cần để viết các server bằng Java. Nó có các constructor để tạo các đối tượng ServerSocket mới, các phương thức để lắng nghe các liên kết trên một cổng xác định, và các phương thức trả về một Socket khi liên kết được thiết lập, vì vậy ta có thể gửi và nhận dữ liệu.

Vòng đời của một server

Một ServerSocket mới được tạo ra trên một cổng xác định bằng cách sử dụng một constructor ServerSocket.

ServerSocket lắng nghe liên kết đến trên cổng đó bằng cách sử dụng phương thức accept(). Phương thức accept() phong tỏa cho tới khi một client thực hiện một liên kết, phương thức accept() trả về một đối tượng Socket mà liên kết giữa client và server.

Tùy thuộc vào kiểu server, hoặc phương thức getInputStream(), getOutputStream() hoặc cả hai được gọi để nhận các luồng vào ra để truyền tin với client.

server và client tương tác theo một giao thức thỏa thuận sẵn cho tới khi ngắt liên kết.

Server, client hoặc cả hai ngắt liên kết

Server trở về bước hai và đợi liên kết tiếp theo.

5.1. Các constructor

public ServerSocket(int port) throws IOException, BindException

Constructor này tạo một socket cho server trên cổng xác định. Nếu port bằng 0, hệ thống chọn một cổng ngẫu nhiên cho ta. Cổng do hệ thống chọn đôi khi được gọi là cổng vô danh vì ta không biết số hiệu cổng. Với các server, các cổng vô danh không hữu ích lắm vì các client cần phải biết trước cổng nào mà nó nối tới (giống như người gọi điện thoại ngoài việc xác định cần gọi cho ai cần phải biết số điện thoại để liên lạc với người đó).

Ví dụ: Để tạo một server socket cho cổng 80

try{

ServerSocket httpd = new ServerSocket(80);

}

catch(IOException e)

{

System. err.println(e);

}

Constructor đưa ra ngoại lệ IOException nếu ta không thể tạo và gán Socket cho cổng được yêu cầu. Ngoại lệ IOException phát sinh khi:

Cổng đã được sử dụng

Không có quyền hoặc cố liên kết với một cổng nằm giữa 0 và 1023.

Ví dụ;

import java.net.\*;

import java.io.\*;

public class congLocalHost

{

public static void main(String[] args)

{

ServerSocket ss;

for(int i=0;i<=1024;i++)

{

try{

ss= new ServerSocket(i);

ss.close();

}

catch(IOException e)

{

System.out.println(“Co mot server tren cong “+i);

}

}

}

}

public ServerSocket(int port, int queuelength, InetAddress bindAddress)throws IOException

Constructor này tạo một đối tượng ServerSocket trên cổng xác định với chiều dài hàng đợi xác định. ServerSocket chỉ gán cho địa chỉ IP cục bộ xác định. Constructor này hữu ích cho các server chạy trên các hệ thống có nhiều địa chỉ IP.

5.2. Chấp nhận và ngắt liên kết

Một đối tượng ServerSocket hoạt động trong một vòng lặp chấp nhận các liên kết. Mỗi lần lặp nó gọi phương thức accept(). Phương thức này trả về một đối tượng Socket biểu diễn liên kết giữa client và server. Tương tác giữ client và server được tiến hành thông qua socket này. Khi giao tác hoàn thành, server gọi phương thức close() của đối tượng socket. Nếu client ngắt liên kết trong khi server vẫn đang hoạt động, các luồng vào ra kết nối server với client sẽ đưa ra ngoại lệ InterruptedException trong lần lặp tiếp theo

public Socket accept() throws IOException

Khi bước thiết lập liên kết hoàn thành, và ta sẵn sàng để chấp nhận liên kết, cần gọi phương thức accept() của lớp ServerSocket. Phương thức này phong tỏa; nó dừng quá trình xử lý và đợi cho tới khi client được kết nối. Khi client thực sự kết nối, phương thức accept() trả về đối tượng Socket. Ta sử dụng các phương thức getInputStream() và getOutputStream() để truyền tin với client.

Ví dụ:

try{

ServerSocket theServer = new ServerSocket(5776);

while(true)

{

Socket con = theServer.accept();

PrintStream p = new PrintStream(con.getOutputStream());

p.println(“Ban da ket noi toi server nay. Bye-bye now.”);

con.close();

}

}

catch(IOException e)

{

System.err.println(e);

}

public void close() throws IOException

Nếu ta đã kết thúc làm việc với một đối tượng server socket thì cần phải đóng lại đối tượng này.

Ví dụ: Cài đặt một server daytime

import java.net.\*;

import java.io.\*;

import java.util.Date;

public class daytimeServer{

public final static int daytimePort =13;

public static void main(String[]args)

{

ServerSocket theServer;

Socket con;

PrintStream p;

try{

theServer = new ServerSocket(daytimePort);

try{

p= new PrintStream(con.getOutputStream());

p.println(new Date());

con.close();

}

catch(IOException e)

{

theServer.close();

System. err.println(e);

}

}

catch(IOException e)

{

System. err.println(e);

}

}

}

public void close() throws IOException

Nếu đã hoàn thành công việc với một ServerSocket, ta cần phải đóng nó lại, đặc biệt nếu chương trình của ta tiếp tục chạy. Điều này nhằm tạo điều kiện cho các chương trình khác muốn sử dụng nó. Đóng một ServerSocket không đồng nhất với việc đóng một Socket.

Lớp ServerSocket cung cấp một số phương thức cho ta biết địa chỉ cục bộ và cổng mà trên đó đối tượng server đang hoạt động. Các phương thức này hữu ích khi ta đã mở một đối tượng server socket trên một cổng vô danh và trên một giao tiếp mạng không

public InetAddress getInetAddress()

Phương thức này trả về địa chỉ được sử dụng bởi server (localhost). Nếu localhost có địa chỉ IP, địa chỉ này được trả về bởi phương thức InetAddress.getLocalHost()

Ví dụ:

try{

ServerSocket httpd = new ServerSocket(80);

InetAddress ia = httpd.getInetAddress();

}

catch(IOException e)

{

}

public int getLocalHost()

Các contructor ServerSocket cho phép ta nghe dữ liệu trên cổng không định trước bằng cách gán số 0 cho cổng. Phương thức này cho phép ta tìm ra cổng mà server đang nghe.