ЛАБОРАТОРНО УПРАЖНЕНИЕ №2

Работа с комуникационни канали

Цел: Реализиране на Java конзолни приложения, които са свързани като тематика с практическото използване на възможностите на Java за създаване на входно-изходни комуникационни канали с цел тестване и комуникация с мрежови услуги.

Задача 1: Напишете Java конзолно приложение, което проверява дали на даден хост е инсталиран Web сървър и връща информация за него (име на сървъра, версия и др.).

При Web сървърите се използва протокол HTTP или протокол HTTPS (HTTP Security) за комуникация с клиентите. При протокол HTTP се подслушва порт 80, а при протокол HTTPS – порт 443. На транспортно ниво комуникацията се реализира чрез протокол TCP. При HTTPS се използва транспортен протокол TCP, но в комбинация с протокол SSL / TLS с цел шифриране на трафика по комуникационния канал. Възможни са два основни начина да решим задачата: 1) Да реализираме комуникация на ниско ниво чрез протокол TCP; и 2) Да използваме комуникация на високо ниво чрез протокол HTTP / HTTPS. За да намалим размера на програмния код ще използваме възможностите на Java за директна комуникация с Web сървъри. За целта се използва клас URL от библиотека java.net. Чрез методите от този клас клиентите изпращат заявки за съдържанието на желан ресурс, а сървърът връща неговото съдържание. *Как да получим информация за Web сървъра?* Всеки отговор съдържа заглавен блок (header) и данни. В заглавния блок има поле Server, което съдържа желаната информация за сървъра. От гледна точка на сигурност, администраторът на Web сървъра трябва да блокира показването на тази информация или да я минимизира.

На Фиг. 1 е показан програмния код чрез който се реализира Задача 1. Заявката към Web сървъра в стойността на константа REQUEST. По своята същност това е URL заявка към тествания сървър (име на протокол : // име на хост). В случая за име на протокол може да зададете само HTTP или HTTPS. Следва създаване на програмен обект server за комуникация с Web сървъра. За целта към конструктора на клас URL се подава низа-заявка. Създава се комуникационен канал за обмен със сървъра – conn. Това се реализира чрез метод openConnection(). Сървърът, ако има такъв и работи, веднага връща информация към клиента под формата на низ. Ако дължината на този низ е -1, то или има проблем с мрежовата връзка или указания протокол на се поддържа (например задали сте https, а сървъра поддържа само http). Чрез метод getHeaderField() е възможно прочитане на съдържанието на специфично поле от всички полета от заглавния блок на отговора. За нас представлява интерес само поле Server. Ако отговорът е null, то администратора е забранил това поле. В противен случай печатаме на конзолата стойността на полето. Изключенията, които са прехванати, са само две: 1) MalformedURLException – синтактична грешка в низа със заявката; и 2) IOException – неконкретизирана I/O грешка.

При тестване на Web сървъра на катедра КСТ се получава следната информация:

```
Apache/2.4.17 (Win32) OpenSSL/1.0.2d PHP/5.6.21
```

Вижда се подробна информация за Web сървъра: име (Apache); версия (2.4.17); операционна система (Microsoft Windows, 32-битова версия); инсталиран е протокол OpenSSL версия 1.0.2d (дали се използва с цел поддържане на SSL комуникации не е ясно) и PHP дистрибуция 5.6.21. Тази информация НЕ ТРЯБВА да се вижда толкова лесно, тъй като е отправна точка за избор на софтуер за атакуване на Web сървъра. Програмно може да се забрани показването на тази информация и да се вижда само името на сървъра. Ето какво връща приложението за Web сървъра на университета:

```
Apache/2.4.46 (FreeBSD)
```

Малко по-добре от гледна точка на сигурност, но се вижда версията на сървъра, както и инсталираната операционна система.

```
final static String REQUEST = "http://www.kst.tugab.bg";

public static void main(String[] args) {
    try {
        URL server = new URL(REQUEST);
        URLConnection conn = server.openConnection();
        String serverInfo = conn.getHeaderField("Server");
        if (serverInfo != null) {
            System.out.println(serverInfo);
        } else {
            if (conn.getContentLength() == -1) {
                System.out.println("Липсва мрежова свързаност, "
                    + "протоколът не се поддържа или няма такъв хост!");
            } else {
                System.out.println("Не е зададено име на Web сървъра!");
            }
        }
    } catch (MalformedURLException ex) {
        System.out.println("Невалиден синтаксис на заявката!");
    } catch (IOException ex) {
        System.out.println("Грешка при комуникация със сървъра!");
    }
}
```

Фиг. 1. Примерно решение на Задача 1

Домашна работа:

Като използвате приложението от Задача 1 попълнете следната таблица:

Име на хоста	Стойност на поле Server	Информация за сървъра	HTTPS?
www.tu-sofia.bg			
www.tu-varna.bg			
www.uni-ruse.bg			
www.yahoo.com			
www.twitter.com			
www.google.com			
www.nasa.gov			
www.intel.com			
www.fbi.gov			

Задача 2: Да се напише Java конзолно приложение, което проверява дали на даден хост има стартирана точно определена услуга, например Web сървър.

Една услуга е активна ако е възможно да създадем работен сокет с цел комуникация с нея. Единственото което трябва да знаем е кой порт „подслушва" услугата. При успешно създаване на сокета – услугата е активна. Ако се генерира изключение IOException при опит за създаване на сокета – услугата не е активна. Трябва да се предвиди възможност за програмно задаване на време за изчакване на отговор от страна на услугата (timeout). В противен случай времето за изчакване ще се задава от сървърната страна и ще бъде различно за различните услуги.

На Фиг. 2 е показан програмния код чрез който се реализира Задача 2. Заточаваме с получаване на IP адреса на тествания хост. За целта се използва метод getByName(). Ако не може да се получи IP адрес се генерира изключение UnknownHostException и програмата завършва. Следва създаване на обект socket – работен сокет за комуникация с хоста. За да можем да зададем timeout период ще използваме метод connect(). Първият аргумент към този метод е TCP работен сокет. Получава се чрез конструктора на клас InetSocketAddress. Времето за изчакване се задава в ms като костната TIMEOUT. Ако не се генерира изключение, портът (зададен чрез константа PORT) е „отворен". Това означава, че услугата, която подслушва този порт е активна. В случая се проверява има ли активен Web сървър, който използва протокол HTTP (порт 80). Ако се генерира изключение от тип SocketTimeoutException това означава, че сървъра не отговаря за времето, зададено като timeout (може да го увеличите). Ако се генерира изключение от тип IOException то портът е „затворен" – услугата не е активна в този момент.

```
static final int PORT = 80;
static final int TIMEOUT = 2500;
static final String HOST_NAME = "www.kst.tugab.bg";
// ----------------------------------------------------------------
public static void main(String[] args) {
    InetAddress ip;
    try {
        ip = InetAddress.getByName(HOST_NAME);
    } catch (UnknownHostException ex) {
        System.out.println("Неразпознато име на хост!");
        return;
    }
    Socket socket = new Socket();
    try {
        socket.connect(new InetSocketAddress(ip, PORT), TIMEOUT);
        System.out.println("Порт " + PORT + " е отворен.");
    } catch (SocketTimeoutException ste) {
        System.out.println("Сървърът не отговаря!");
    } catch (IOException ioe) {
        System.out.println("Порт " + PORT + " е затворен.");
    }
}
```

Фиг. 2. Примерно решение на Задача 2

Задача 3:

Променете програмния код на Задача 2 така, че да се проверяват множество портове, зададени чрез начален номер на порт и краен номер на порт. Смисълът на тази задача е да можем отдалечено да тестваме активните услуги на избран хост.

От гледна точка на сигурност, част от тези услуги може и да не са желани. Например, злонамерен софтуер от тип „задна врата" може да стартира сървър чрез който атакуващия да комуникира със софтуера, заразил хоста. Ако знаете кои портове използва този софтуер, лесно може да проверите дали хоста е заразен с конкретен тип злонамерен софтуер.

Анализирайте времето, необходимо за тестване на всички „добре познати портове" с номера от 1 до 1024.

Задача 4:

Реализирайте Задача 3 с използване на програмни нишки. Целта е да увеличим бързодействието на програмата, за да има тя практически смисъл.

Ще започнем с деклариране на необходимите константи и обекти:

```
static final int START_PORT = 1;
static final int END_PORT = 1024;
static final int TIMEOUT = 2500;
static final String HOST_NAME = "www.tugab.bg";

static Socket socket;
static InetAddress ip;
```

Ще създадем клас ScanThread (виж Фиг. 4) чрез който се стартира по една програмна нишка за проверка на един порт от зададения интервал. Клас ScanThread наследява клас Thread, за да може да напишем кода на нишката – метод run(). Метод run() трябва да „знае" IP адреса на хоста, номера на порта и timeout интервала. Тъй като IP адреса и timeout интервала са еднакви за всички нишки, те са декларирани в тялото на базовия клас и така се „виждат" от всяка нишка. Проблемен е номера на порта, тъй като е специфичен за всяка нишка. Следователно, трябва да напишем конструктор на ScanThread, който получава като аргумент номера на порта. Можем вместо да създаваме нов конструктор да използваме конструктора по подразбиране. Той очаква да му предадем името на нишката. Но ние вместо име ще подадем номера на порта, но не като число, а като низ. След това, когато на нишката и трябва номера на порта, тя ще прочете името си чрез метод getName() и ще го преобразува до цяло число от тип int. За целта ще използваме статичен метод parseInt(), който е разработен точно за това.

От тялото на нишките няма да обработваме изключение IOException, защото не се интересуваме от „затворените" портове, а само от „отворените". Ще използваме клауза finally чрез която ще унищожим създадения сокет. За целта задаваме стойност null за инстанцията му. Това ще накара Garbage Collector (GC) да изтрие от паметта този обект. Този начин на премахване на обекти от паметта е възможен, но не трябва да се използва, защото не е ясно кога точно GC ще унищожи обекта. Правилният подход е следния: Проверяваме дали е създаден обект socket (socket != null) и само тогава унищожаваме сокета чрез метод close().

```
static class ScanThread extends Thread {
    public ScanThread(String s) {
        super(s);
    }
    @Override
    public void run() {
        int port = Integer.parseInt(getName());
        try
        {
            Socket socket = new Socket();
            socket.connect(new InetSocketAddress(ip, port), TIMEOUT);
            System.out.println(port);
        } catch (IOException ex) {
        }
        finally
        {
            socket = null;
        }
    }
}
```

a) Клас ScanThread

```
public static void main(String ... pars) {
    try {
        ip = InetAddress.getByName(HOST_NAME);
    } catch (UnknownHostException uhs) {
        System.out.println("Невалидно има на хост!");
        return;
    }
    System.out.println("\"Отворени\" портове:");
    for (int i = START_PORT; i <= END_PORT; i++) {
        (new ScanThread("" + i)).start();
    }
}
```

b) Метод main()

Фиг. 3. Примерно решение на Задача 4

Всички нишки се стартират чрез for цикъл от тялото ма метод main(). Преди този цикъл се получава IP адреса на тествания хост по познатия ни вече начин чрез метод getByName(). Програмният код от тялото на цикъла стартира по една нишка за всеки номер на порт от указания интервал. Забележете как се преобразува номера на текущ порт i до низ – използва се слепване на празен низ и число: “” + i.

След стартиране на приложението се получава следната информация на конзолата:

```
"Отворени" портове:
80
22
21
443
```

Следователно на хоста www.tugab.bg има стартирани: Web сървър (HTTP); SSH сървър с цел отдалечен достъп до конзолата на хост (най-вероятно с цел администриране); FTP сървър и Web сървър (HTTPS).

Задача 5:

Реализирайте Задача 4 като използвате възможностите на Java 8+ за управление на множество нишки (thread pool).

Проблемът на решението с използване на нишки е, че една нишка се използва само за проверка на един порт. Това е разхищение на ресурси. По добре би било една нишка да се грижи за проверката на множество портове. *Как да разделим работата между нишките?* За целта може да използвате и собствен код, който решава проблема, но в Java има възможност това да стане много по-лесно. За целта ще използваме интерфейс Executors и клас ThreadPoolExecutor от библиотека java.util.concurrent. Интерфейс Executors предоставя методи чрез които може да се създаде thread pool по наколко различни начина, например:

- Задава се точно определен брой на нишките. За целта се използва метод newFixedThreadPool. Той изисква един аргумент – броя нишки, които ще решат задачата.
- Оставя се кода от библиотеката да избера броя на нишките с които ще работи. За целта се използва метод newCachedThreadPool(). Той не изисква задаване на аргументи.

И двата метода връщат обект от клас ThreadPoolExecutor, например executor. Чрез метод execute може да зададете обект-нишка, която да попадне в thread pool. След като стартирате чрез екзекутора всички нишки ви остава да чакате кога те ще завършат. За целта може да използвате следния код:

```
executor.shutdown();
executor.awaitTermination(TASK_TIMEOUT, TimeUnit.SECONDS);
```

Екзекутора ще изчака да завършат всички нишки в thread pool. Това се задава чрез метод shutdown(). Ако трябва да се зададе максимално време на изчакване нишките да завършат се използва метод awaitTermination(). В случая е указано да се чака интервал от време в секунди (TimeUnit.SECONDS). Времето на изчакване е стойността на константа TASK_TIMEOUT. Ако има нишки, които не са завършили за време TASK_TIMEOUT секунди, то екзекутора ще ги терминира автоматично.

Когато използвате метод newFixedThreadPool() трябва да зададете броя на нишките с които искате да работите. Ако зададете много на брой нишки няма да изпозлвате ефективно паметта. Ако използвате малко на брой нишки, ефекта от многонишковото програмиране няма да е голям. *Как се получава оптималния брой нишки?* Най-често броя нишки се получава след анализ на логиката на програмния код, на базата на опита на програмиста. Може да използвате и следната проста формула:

Number of threads = Number of CPU Cores * (1 + Wait time / Service time)

където:

- Number of CPU Cores е броят на ядрата на процесора.
- Wait time е времето за изчакване на отговор от услугата в ms.
- Service time е времето в ms за което нишката изпълнява слоя код.

Как програмно да получим броя на ядрата на процесора? Може да използвате метод availableProcessors():

```
int cores = Runtime.getRuntime().availableProcessors();
```

Домашна работа:

Реализирайте като Java конзолно приложение Задача 5.