

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни

«Проектування комп'ютерних мереж»

для магістрів спеціальності 123 "Комп'ютерна інженерія", освітні
програми "Спеціалізовані комп'ютерні системи" та "Комп'ютерні
системи та мережі", усіх форм навчання

Розробка додатків у середовищі програмування Java.

Частина 1

Методичні вказівки до лабораторних робіт з дисципліни “Проектування комп’ютерних мереж” для магістрів спеціальності 123 “Комп’ютерна інженерія”, освітні програми “Спеціалізовані комп’ютерні системи” та “Комп’ютерні системи та мережі”, усіх форм навчання. Розробка додатків у середовищі програмування Java. Частина 1 / Укл. Г.Г.Киричек, С.Ю.Скрупський. – Запоріжжя: ЗНТУ, 2018. – 38 с.

Укладачі:

Г.Г. Киричек, доцент, к.т.н.

С.Ю. Скрупський, доцент, к.т.н.

Рецензент:

М.Ю. Тягунова доцент, к.т.н.

Відповідальний за випуск:

Г.Г. Киричек, доцент, к.т.н.

Затверджено
на засіданні кафедри КСМ
Протокол №3 від 24.09.2018

Затверджено
на засіданні НМК КНТ
Протокол №2 від 28.09.2018

ЗМІСТ

1 Лабораторна робота. Програмування TCP-сокетів.....	4
1.1 Загальні відомості	4
1.2 Перевірка налаштувань мережного інтерфейсу комп'ютерів	8
1.3 Установка середовища програмування Java	9
1.4 Створення клієнтської програми	10
1.5 Створення серверної програми	17
1.6 Підготовка до запуску на іншому ПК.....	19
1.7 Запуск програм	19
1.8 Завдання для самостійного програмування.....	20
1.9 Створення багатопоточного серверу	20
1.10 Модифікація багатопоточного мережного додатку.....	22
1.11 Відновлення комп'ютерів у початковий стан	23
1.12 Зміст письмового звіту.....	23
1.13 Контрольні питання	23
2 Лабораторна робота. Програмування UDP-сокетів	24
2.1 Загальні відомості	24
2.2 Перевірка налаштувань мережного інтерфейсу комп'ютерів	26
2.3 Установка середовища програмування Java	28
2.4 Створення клієнтської програми	28
2.5 Створення серверної програми	33
2.6 Підготовка до запуску на іншому ПК.....	35
2.7 Запуск програм	36
2.8 Завдання для самостійного програмування.....	36
2.9 Відновлення комп'ютерів у початковий стан	36
2.10 Зміст письмового звіту.....	37
2.11 Контрольні питання	37
Рекомендована література	38

1 ЛАБОРАТОРНА РОБОТА. Програмування TCP-сокетів

Мета роботи – ознайомлення з основами розробки мережних додатків.

1.1 Загальні відомості

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосувань і передбачає взаємодію та обмін даними між ними. Мережний додаток складається з двох програм — клієнта і сервера. Коли ці програми запускаються, створюються клієнтський і серверний процеси, що взаємодіють один з одним, обмінюючись повідомленнями через сокети. При створенні мережного додатку головним завданням розробника є написання програмного коду для клієнтської і серверної частин додатку.

Існують два види додатків з архітектурою клієнт/сервер. До першого виду відносяться додатки, які підтримують стандартний протокол, описаний в документах RFC. У цьому випадку як клієнтська, так і серверна частини додатку повинні відповідати всім описам, приведеним в RFC. Наприклад, якщо додаток використовує протокол FTP, то клієнтська і серверна програми додатку повинні бути побудовані відповідно до вимог документа RFC 959, що описує FTP-додатки. Дотримання стандартів дозволяє незалежним розробникам підключатися до створення різних частин одного і того ж додатку. Для додатків, розроблених за стандартом RFC, слід застосовувати порт з номером, відповідним протоколу, який використовується.

Другий вид додатків з архітектурою клієнт/сервер — нестандартні додатки. В таких додатках підтримка клієнтом і сервером яких-небудь угод, прийнятих в RFC, не обов'язкова. Розробник в особі одного або декількох чоловік розробляє клієнтську і серверну частини додатку, повністю контролюючи процес написання коду. При цьому інші розробники не зможуть створювати програми, які будуть здатні взаємодіяти з нестандартним додатком. Для

нестандартних додатків необхідно використовувати порти з номерами, відмінними від загальновідомих, вказаних в RFC.

В роботі будуть розглянуті ключові принципи розробки нестандартних додатків з архітектурою клієнт/сервер. Одне з перших рішень, яке вимушений приймати розробник, торкається вибору протоколу транспортного рівня (TCP або UDP) для свого додатку. Протокол TCP передбачає встановлення логічного з'єднання між хостами і забезпечує надійну передачу даних. Протокол UDP, навпаки, не встановлює логічного з'єднання, і передача пакетів здійснюється без гарантії їх доставки адресату.

В цій роботі ви створите простий мережний додаток, який підтримує протокол TCP. Програми написані на мові Java.

Процеси, що виконуються на різних комп'ютерних машинах, взаємодіють один з одним за допомогою повідомлень, що посилаються через сокети. Як показано на рисунку 1.1, сокет знаходиться між процесом і протоколом TCP. Розробник додатку повністю контролює ту частину сокета, яка відноситься до прикладного рівня, майже не маючи нагоди впливати на «транспортну частину» (за винятком завдання декількох параметрів протоколу TCP, таких як максимальний розмір буфера і сегменту).

Тепер розглянемо процес взаємодії клієнтської і серверної програм більш детально. У функції клієнта входить ініціювання з'єднання з сервером, а сервер повинен бути готовим до встановлення з'єднання. Це означає, що, по-перше, програма-сервер повинна бути запущена раніше, ніж клієнт зробить спробу встановити з'єднання, і, по-друге, що сервер повинен мати в своєму розпорядженні сокет, за допомогою якого встановлюється з'єднання.

Коли серверний процес запущений, клієнт може ініціювати установку TCP-з'єднання з сервером. Першою дією клієнтської програми є створення сокета, при цьому програма указує адресу серверного процесу, що складається з IP-адреси і номера порту процесу.

Після створення сокета клієнтська сторона протоколу TCP здійснює процедуру потрійного рукоштовування з сервером, що закінчується встановленням з'єднання. Відзначимо, що процедура рукоштовування ніяк не позначається на роботі додатку.

Після прийому запиту на серверному сокеті на встановлення з'єднання сервер створює новий сокет для взаємодії з клієнтом.



Рисунок 1.1 - Взаємодія процесів за допомогою TCP-сокетів

В прикладі нижче серверним сокетом є об'єкт **ServerSocket** з ім'ям **welcomeSocket**. Коли клієнт відсилає запит до цього сокету, викликається метод **accept()** об'єкту **welcomeSocket**, що створює новий сокет для клієнта.

Після закінчення процедури рукоштовування встановлюється TCP-з'єднання між сокетом клієнта і новим сокетом серверу, який називають *сокетом з'єднання* (рис. 1.2).

З погляду додатку TCP-з'єднання є прямим віртуальним каналом між сокетами з'єднання клієнта і серверу. Клієнт може здійснювати передачу будь-яких байтів через свій сокет, при цьому протокол TCP гарантує, що сервер отримає ці байти через свій сокет без спотворень і в тому ж порядку, в якому вони були передані.

Оскільки сокети грають центральну роль в роботі додатків клієнт/сервер, розробку таких додатків часто називають програмуванням сокетів.

Перш ніж навести приклад додатку клієнт/сервер, розглянемо поняття *потoku даних*. Під потоком даних розуміється послідовність символів, що передається між двома процесами. По відношенню до процесу виділяють *вхідний* і *вихідний* потоки. Вхідний потік процесу

зв'язується з деяким вхідним пристроєм (клавіатурою, сокетом і т.п.), а вихідний потік — з деяким вихідним пристроєм (монітором, сокетом і т.п.).

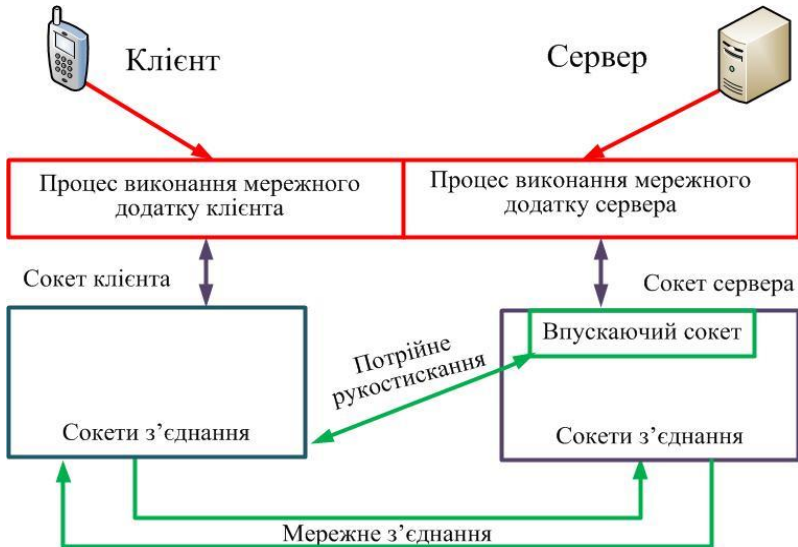


Рисунок 1.2 - Клієнтський сокет, серверний сокет і сокет з'єднання

Даний додаток функціонує таким чином:

- клієнт читає з стандартного пристрою вводу (клавіатури) рядок символів і посилає цей рядок серверу через свій сокет;
- сервер приймає рядок через свій сокет;
- сервер переводить всі символи рядка у верхній регістр;
- сервер відсилає модифікований рядок клієнту;
- клієнт одержує рядок і друкує його за допомогою стандартного пристрою виводу (монітора).

Примітка. Розробка вихідних кодів сервера та клієнта здійснюється на одному ПК, а перевірка та демонстрація викладачу – на двох ПК.

Методичні вказівки для цієї роботи складені з орієнтацією на виконання роботи на комп'ютерах з фізичними іменами **RMF01** і **RMF02**.

Для цих комп'ютерів, нагадаємо, при завантаженні на них клієнтської ОС **Microsoft Windows 7** маємо:

IP-адреса комп'ютера: **10.0.9.51** і **10.0.9.52** відповідно,

Мережне ім'я комп'ютера: **rmf51** і **rmf52** відповідно.

А оскільки більшість з вас працює за комп'ютерами з іншими фізичними іменами, то при виконанні роботи вам доведеться використовувати відповідно інші ("свої") IP-адреси, короткі і повні Мережні імена.

1.2 Перевірка налаштувань мережного інтерфейсу комп'ютерів

На комп'ютері з фізичним ім'ям **RMF01**:

- завантажить **Microsoft Windows 7** (ім'я користувача – **admin**, пароль – **1234567**);

- перевірте настройки мережного інтерфейсу:

- задайте **Пуск => Панель управления => Сеть и Интернет => Центр управления сетями и общим доступом => Изменение параметров адаптера**;

- у вікні **Сетевые подключения** запустить - **Подключение по локальной сети**;

- у вікні **Состояние Подключение по локальной сети** клацніть **Свойства**;

- установить курсор на **Протокол Интернета (TCP/IPv4)** і клацніть **Свойства**;

- виберіть опцію **Использовать следующий IP-адрес** і введіть:

IP-адреса: **10.0.9.51** (якщо фізичне ім'я вашого комп'ютера **RMF01+50**), (аналогічно, якщо фізичне ім'я вашого комп'ютера **RMF25**, то – **10.0.9.75**)

Маска подсети: **255.255.255.0**

Основной шлюз: **10.0.9.100**

Предпочитаемый DNS-сервер: **10.0.2.1**

і клацніть **ОК**;

- з поверненням у вікно **Подключение по локальной сети - свойства** клацніть **ОК**;

- закрийте вікно **Состояние–Подключение по локальной сети**;

- закрийте вікно **Просмотр основных сведений о сети и настройка подключений**.

На комп'ютері з фізичним ім'ям **RMF02**:

- завантажте **Microsoft Windows 7** (ім'я користувача – **admin**, пароль – **1234567**);

- перевірте настройки мережного інтерфейсу:

- задайте **Пуск => Панель управления => Сеть и Интернет => Центр управления сетями и общим доступом => Изменение параметров адаптера**;

- у вікні **Сетевые подключения** запустіть - **Подключение по локальной сети**;

- у вікні **Состояние Подключение по локальной сети** клацніть **Свойства**;

- установіть курсор на **Протокол Интернета (TCP/IPv4)** і клацніть **Свойства**;

- виберіть опцію **Использовать следующий IP-адрес** і введіть:

IP-адреса: **10.0.9.52** (якщо фізичне ім'я вашого комп'ютера **RMF02+50**)

Маска подсети: **255.255.255.0**

Основной шлюз: **10.0.9.100**

Предпочитаемый DNS-сервер: **10.0.2.1**

і клацніть **ОК**,

- з поверненням у вікно **Подключение по локальной сети - свойства** клацніть **ОК**;

- закрийте вікно **Состояние–Подключение по локальной сети**;

- закрийте вікно **Просмотр основных сведений о сети и настройка подключений**.

1.3 Установка среды программирования Java

Якщо на комп'ютерах з фізичними іменами **RMF01** і **RMF02** вже встановлено середовище програмування Java (про це свідчить наявність каталогу **G:\OpenJDK_7_u60**).

Якщо ні, то на обох комп'ютерах виконайте наступні операції.

Установіть середовище програмування Java, для чого:

- запустіть:

J:\Install\Java_install\install.exe

- з відкриттям вікна встановлення клацніть **Next**;

– у вікні **Licensing Agreements** оберіть **I accept the terms of this license agreement** та клацніть **Next**;

- у вікні **Target Path** вкажіть папку:

G:\OpenJDK_7_u60;

- клацніть **Next**, погодьтеся зі створенням папки;

- у вікні **Select Installation Packages** клацніть **Next**;

- дочекайтеся встановлення компонентів, та клацніть **Done**.

Створіть на диску **G** папку **Projects**.

1.4 Створення клієнтської програми

На комп'ютері з фізичним ім'ям **RMF02** (з мережним ім'ям **rmf52**; у вашого комп'ютера відповідне “своє мережне ім'я”):

- запустіть середовище розробки Eclipse (рис.1.3)

J:\Install\Eclipse\eclipse.exe;

– при першому запуску середи розробки можливо буде потрібно вказати шлях до каталогу робочого простору (**G:\Projects**);

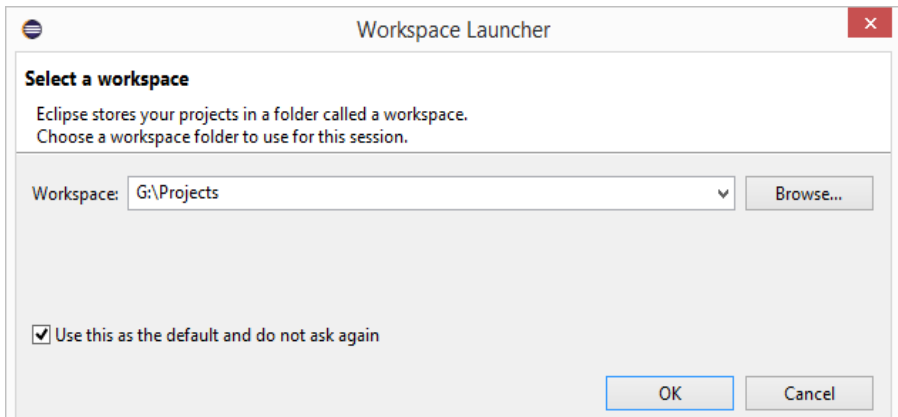


Рисунок 1.3 – Каталог робочого простору

– створюємо новий проект. Для цього обираємо пункт меню **File > New > Java Project**, та вказуємо в полі **Project Name** назву проекту, наприклад **Lab1**. В інших полях залишаємо значення за замовчанням (рис. 1.4);

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location: [Browse...](#)

JRE

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE (currently 'OpenJDK_7u60') [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets

Working sets: [Select...](#)

[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

Рисунок 1.4 – Створення нового проекту

– створюємо новий клас у створеному проєкті. Для цього обираємо пункт меню **File > New > Class**, та вказуємо в полях такі значення:

- 1) пакет (Package): **task1**;
- 2) назва (Name): **TCPCClient**;
- 3) помічаємо пункт **"public static void main(String[] args)"**.

Інші поля залишаємо із значеннями за замовчанням (рис. 1.5).

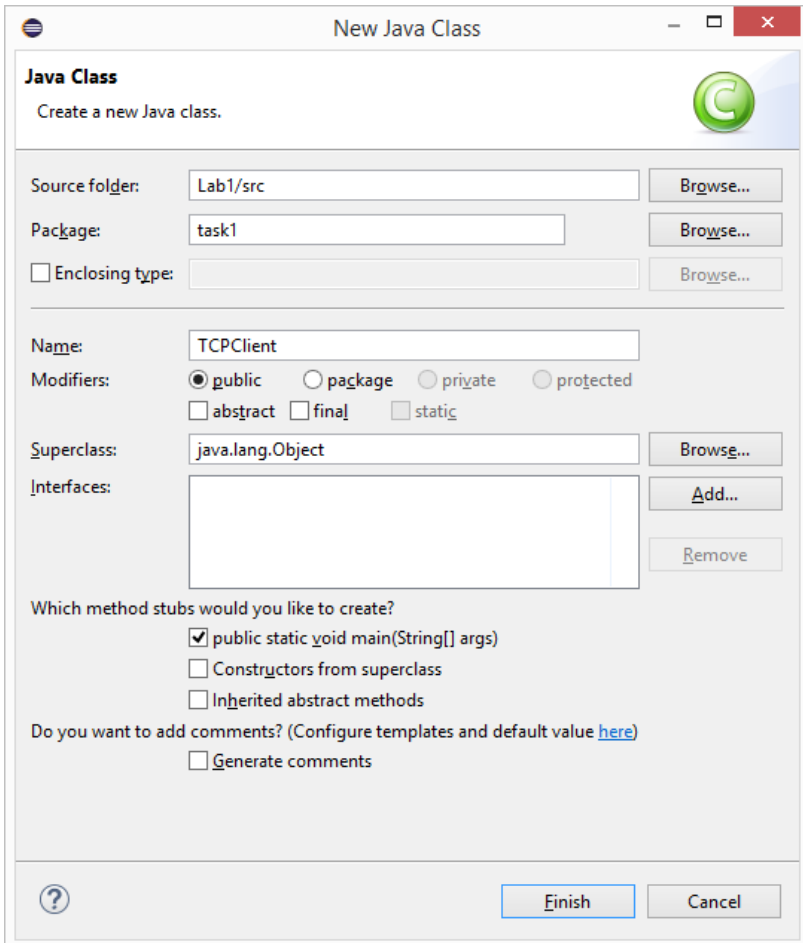


Рисунок 1.5 – Створення нового класу

Наберіть наступний текст програми.

```
package task1;

import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;

public class TCPClient {
    public static void main(String[] args) throws
    IOException
    {
        Socket clientSocket = new Socket("localhost",
6789);
        Scanner inFromUser = new Scanner(System.in);
        Scanner inFromServer = new Scanner(

clientSocket.getInputStream());
        DataOutputStream outToServer = new
DataOutputStream(

clientSocket.getOutputStream());
        System.out.print("Input sentence: ");
        String sentence = inFromUser.nextLine();
        outToServer.writeBytes(sentence + "\n");
        String modifiedSentence =
inFromServer.nextLine();
        System.out.println("Server's answer: " +
modifiedSentence);
        clientSocket.close();
    }
}
```

Далі даються пояснення для кожного рядка коду. Як показано на рисунку 1.6, програма **TCPClient** створює три потоки даних і один сокет. Сокет має назву **clientSocket**.

Потік **inFromUser** є вхідним потоком даних програми і пов'язаний із стандартним пристроєм вводу (клавіатурою). Всі

символи, що вводяться з клавіатури користувачем, потрапляють в потік **inFromUser**.

Інший вхідний потік даних програми, **inFromServer**, пов'язаний з сокетом і складається з символів, що передаються серверною стороною додатку.

Вихідний потік даних програми **TCPClient** має назву **outToServer** і також пов'язаний з сокетом. Цей потік містить символи, що передаються серверу для обробки.

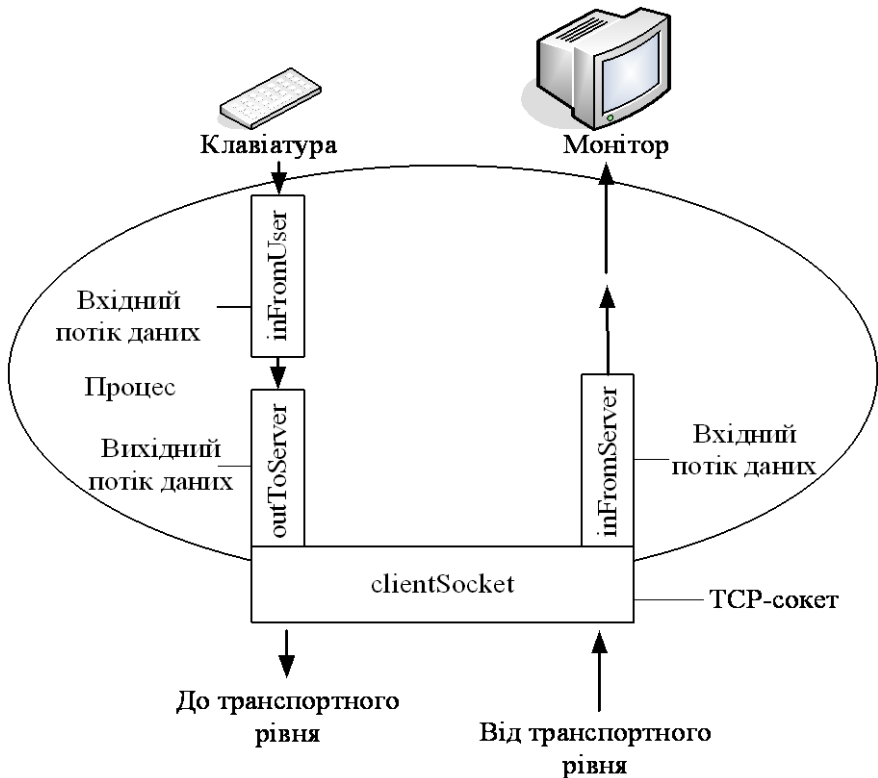


Рисунок 1.6 - Програма **TCPClient** організовує три потоки даних для символів

Перший рядок програми вказує на те, що клас, який розроблюється, буде знаходитися в пакеті **task1**.

Наступні чотири рядки програми містять імена Java-пакетів та класів, які імпортуються:

```
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;
```

Далі іде оголошення класу **TCPClient** та реалізація його методів:

```
class TCPClient {
    public static void main(String args[])
    throws IOException
    {.....}
}
```

Конструкції, аналогічні приведеній вище, є стандартними в практиці програмування на Java. Перший рядок є початком блоку визначення класу.

В ньому присутнє ключове слово **class** і ім'я визначуваного класу **TCPClient**. Клас може містити змінні і методи; в оголошенні класу вони укладені у фігурні дужки і фактично складають блок визначення класу. В класі **TCPClient** немає змінних, він містить єдиний метод з ім'ям **main()**. Методи схожі на процедури і функції мов, подібних C; метод **main()** також грає роль, схожу з функцією **main()** в C або C++.

Коли інтерпретатор Java виконує додаток (будучи викликаним керівним класом додатку), він звертається до методу **main()** цього класу.

Метод **main()** викликає всю решту методів, що використовуються в додатку. Якщо в даний момент ви вперше стикаєтеся з Java-додатком, можете не звертати уваги на ключові слова **public, static, void, main** і **throws Exception**.

```
Socket clientSocket = new Socket("localhost", 6789);
```

В приведеному рядку відбувається створення об'єкту **clientSocket** типу **Socket**. Крім того, при цьому ініціюється TCP-

з'єднання між клієнтом і сервером. Слово **localhost** необхідне замінити ім'ям хоста, зберігши лапки (наприклад, "**www.zntu.edu.ua**"). Перед початком установки TCP-з'єднання клієнт проводить DNS-запит IP-адреси хоста. Значення **6789** — це номер порту; ви можете вибрати інше число, проте необхідно пам'ятати, що клієнтська і серверна сторони повинні використовувати один і той же номер порту. Як згадувалося раніше, IP-адреса хоста в сукупності з номером порту додатку ідентифікує серверний процес.

```
Scanner inFromUser = new Scanner(System.in);
```

Тут ми створюємо потоковий об'єкт для читання даних типу **Scanner**. Ініціалізація потоку даних проводиться об'єктом **System.in**, що пов'язує потік із стандартним пристроєм вводу. Після виконання цієї команди клієнту починають передаватися символи, що вводяться користувачем з клавіатури.

```
Scanner inFromServer = new Scanner (  
clientSocket.getInputStream()); ;  
DataOutputStream outToServer = new  
DataOutputStream(clientSocket.getOutputStream()); ;
```

Тут відбувається створення поточкових об'єктів, пов'язаних з сокетом. Потік **outToServer** забезпечує вивід програми через сокет, а потік **inFromServer** призначений для прийому даних від серверної сторони (рис. 1.3).

```
System.out.print("Input sentence: ");
```

Цей метод виводить в командне вікно фразу запрошення ввести речення з клавіатури.

```
String sentence = inFromUser.nextLine();
```

Приведений рядок оголошує об'єкт **sentence** типу **String** та ініціалізує його послідовністю символів, що вводяться користувачем. Введення закінчується натисненням користувачем клавіші **Enter**. Як бачимо, для поміщення символів в змінну використовується потоковий об'єкт **inFromUser**.

```
outToServer.writeBytes(sentence + "\n");
```

Тут рядок **sentence** разом з символом повернення каретки поміщається у вихідний потік **outToServer**. Після цього він буде переданий через сокет в канал, що з'єднує клієнта з сервером.

```
String modifiedSentence = inFromServer.nextLine();
```

Відповідь сервера приймається у вхідний потік **inFromServer**, звідки прийнятий рядок копіюється в змінну **modifiedSentence** типу **String**. Поміщення символів в рядок **modifiedSentence** продовжується до тих пір, поки не буде отриманий символ повернення каретки.

```
System.out.println("Server's answer: " + modifiedSentence);
```

Тут відбувається вивід прийнятого від сервера рядка на монітор.

```
clientSocket.close();
```

Цей рядок закриває сокет, а отже, і TCP-з'єднання між клієнтом і сервером. Виконання цієї команди веде до відправки TCP-клієнтом повідомлення TCP-серверу.

1.5 Створення серверної програми

На першому кроці створіть клас **TCPServer** у пакеті **task1**.

Наберіть наступний текст програми:

```
package task1;

import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class TCPServer {
    public static void main(String[] args) throws
    IOException
    {
        ServerSocket welcomeSocket = new
        ServerSocket(6789);

        while (true) {
            Socket connectionSocket =
            welcomeSocket.accept();
            Scanner inFromClient = new Scanner(
            connectionSocket.getInputStream());
```

```

        DataOutputStream outToClient = new
DataOutputStream(

connectionSocket.getOutputStream());
        String clientSentence =
inFromClient.nextLine();
        String capitalizedSentence =
            clientSentence.toUpperCase() + "\n";
        outToClient.writeBytes(capitalizedSentence);
    }
}
}

```

Програма **TCPServer** має багато спільного з програмою **TCPClient**; розглянемо її докладніше, опустивши рядки, спільні з програмою **TCPClient** і описані вище.

ServerSocket welcomeSocket = new ServerSocket(6789);

Тут відбувається створення об'єкту **welcomeSocket** типу **ServerSocket**. Об'єкт **welcomeSocket** є серверним сокетом, за допомогою якого клієнт встановлює первинний контакт з сервером. Для цього сокета використовується порт з номером **6789**, що співпадає з номером порту сокета клієнта.

Протокол TCP встановлює прямий віртуальний канал між сокетом **clientSocket** на клієнтській стороні і **connectionSocket** на серверній стороні, після чого клієнт і сервер можуть вільно здійснювати обмін інформацією. Після створення сокета **connectionSocket** сервер може знов використовувати сокет **welcomeSocket** для ініціювання нових з'єднань з клієнтами (наведена програма не обробляє нових з'єднань, однак її можна модифікувати відповідним чином).

Далі програма створює декілька потокових об'єктів, аналогічних таким об'єктам у клієнтській програмі.

```

String clientSentence = inFromClient.nextLine();
String capitalizedSentence =
    clientSentence.toUpperCase() + "\n";

```

Ці команди основні в додатку. Перша виконує отримання рядка, переданого клієнтом, а друга – переведення рядка у верхній регістр за

допомогою методу **toUpperCase()** і додавання в кінець символу повернення каретки.

Всі інші команди програми є периферійними і не стосуються взаємодії серверу з клієнтом.

1.6 Підготовка до запуску на іншому ПК

Виконайте наступні дії:

- скопіюйте каталог G:\Projects із його вмістом на інший комп'ютер;

- запустіть середовище розробки Eclipse:

J:\Install\Eclipse\eclipse.exe;

- імпортуйте створений проект. Для цього треба обрати пункт **File > Import**, у вікні, що з'явилось, обрати **General > Existing projects into workspace**, натиснути **Next**, вказати шлях **G:\Projects** у полі **Select root directory**;

- оберіть у полі **Projects** проект **Lab1** для імпорту та натисніть кнопку **Finish**.

1.7 Запуск програм

Запустіть сервер на комп'ютері з фізичним ім'ям **RMF01**. Для цього треба відкрити файл **TCPServer.java**, та обрати пункт меню **Run > Run**, або значок запуску з панелі, як зазначено на рисунку 1.7. Після цього сервер перейде в режим очікування.

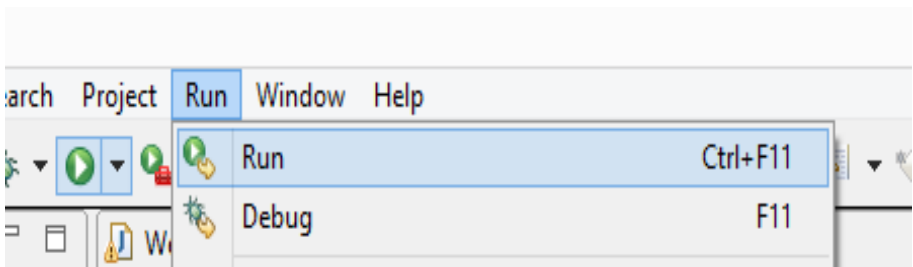


Рисунок 1.7 – Запуск додатку

На комп'ютері з фізичним ім'ям **RMF02** запустіть клієнт. Для цього треба відкрити файл **TCPClient.java**, та обрати пункт меню **Run > Run**, або значок запуску з панелі. В консолі головного вікна **Eclipse** з'явиться вікно терміналу для взаємодії з клієнтом:

- наберіть рядок, що містить латинські символи у нижньому та верхньому регістрі (наприклад, **aaaAAAbbbbBBB**), і натисніть **Enter**;
- запишіть результат роботи програм і проаналізуйте його;
- продемонструйте роботу програм викладачу і поясніть їх роботу.

1.8 Завдання для самостійного програмування

Змініть додаток так, щоб поведінка додатку була наступною:

- клієнт відсилає серверу рядок, що містить прізвище студента;
- сервер при отриманні виводить його на екран і відсилає клієнту рядок, що містить ім'я студента;
- клієнт при отриманні виводить його на екран.

1.9 Створення багатопоточного серверу

Зазвичай сервери постійно оброблюють багато підключень від багатьох клієнтів. Але створений сервер із пункту 1.5 може обробляти дані лише від одного клієнта у кожен момент часу. Його можна покращити, якщо відокремити код обробки клієнтів в окремий потік:

- створіть клас `TCPServerThread` у пакеті `task2`;
- наберіть наступний текст програми:

```
package task2;
```

```
import java.io.DataOutputStream;
import java.net.Socket;
import java.util.Scanner;
```

```
public class TCPServerThread extends Thread {
    protected Socket socket;
```

```
    TCPServerThread(Socket s) {
        socket = s;
```

```

    }

    @Override
    public void run() {
        try {
            Scanner inFromClient = new Scanner(
                socket.getInputStream());
            DataOutputStream outToClient = new
DataOutputStream(
                socket.getOutputStream());
            String clientSentence =
inFromClient.nextLine();
            String capitalizedSentence =
                clientSentence.toUpperCase() +
"\n";
            outToClient.writeBytes(capitalizedSentence);
        } catch (Exception e) {
            System.err.printf("Exception in thread:
%s\n",
e.getLocalizedMessage());
        }
    }
}

```

Цей файл містить реалізацію класу потоку, що буде оброблювати дані від клієнту. Саме він обробляє речення, яке передає йому клієнт.

Метод **run()**, описаний в цьому класі, виконується після запуску потоку. Саме в ньому має бути написаний код.

Варто зауважити, що цей клас має конструктор, ціль якого – отримати сокет підключеного клієнта від головного потоку.

Створіть клас **TCPThreadedServer** у пакеті **task2**.

Наберіть наступний текст програми:

```

package task2;

import java.io.IOException;
import java.net.ServerSocket;

```

```
import java.net.Socket;

public class TCPThreadedServer {
    public static void main(String[] args) throws
    IOException {

        ServerSocket welcomeSocket = new
        ServerSocket(6789);

        while (true) {
            Socket connectionSocket =
            welcomeSocket.accept();
            Thread serverThread = new

TCPServerThread(connectionSocket);
            serverThread.start();
        }
    }
}
```

Цей клас потрібний лише для створення серверного сокету та очікування клієнтів. Для кожного клієнту, що підключається до серверного сокету, створюється, і за допомогою метода **serverThread.start()** запускається новий потік **TCPServerThread**.

Запуск та перевірка багатопоточного серверу здійснюється за пунктом 1.7 за винятком того, що на ПК **RMF01** запускається **TCPThreadedServer** замість TCPServer.

1.10 Модифікація багатопоточного мережного додатку

Змініть додаток так, щоб поведінка додатку була наступною:

- одна серверна програма чекає з'єднання по двом портам;
- на клієнтській машині запустіть відразу дві клієнтських програми, що приєднуються до різних портів серверу. Вони зчитують по одному рядку з клавіатури і відправляють ці рядки серверу;
- сервер відправляє кожному з клієнтів рядок, рівний конкатенації отриманих від клієнтів рядків;
- клієнти виводять цей результат на екран.

1.11 Відновлення комп'ютерів у початковий стан

На обох комп'ютерах видаліть папку **G:\Projects**

1.12 Зміст письмового звіту

- хід роботи;
- основні функції;
- відповіді на контрольні питання.

1.13 Контрольні питання

1. Що таке сокет?
2. Укажіть розташування сокета у моделі взаємодії відкритих систем OSI.
3. Назвіть етапи встановлення TCP та з'єднання з сервером.
4. Розкрийте поняття потоку даних.
5. Вкажіть відмінності в реалізації однопоточних додатків від багатопоточних.

2 ЛАБОРАТОРНА РОБОТА.

Програмування UDP-сокетів

Мета роботи – ознайомлення з розробкою додатків для Інтернету.

2.1 Загальні відомості

З погляду процесів, взаємодія за протоколом TCP виконується через канал, який існує до тих пір, поки один з процесів не закрий його. Передача процесом інформації зводиться до того, що процес «скидає» байти безпосередньо в канал, при цьому немає необхідності забезпечувати байти адресою призначення, оскільки канал логічно пов'язаний з адресатом.

Крім того, передача по каналу являється надійною, тобто послідовність байтів, що приймається, точно відповідає послідовності, що передається.

Протокол UDP, як і TCP, реалізує взаємодію між двома процесами, які виконуються на різних хостах, проте UDP-взаємодія принципово відрізняється від TCP-взаємодії:

- по-перше, логічне з'єднання між хостами відсутнє, оскільки не існує процедури рукошлякування, в ході якої створювався б канал;
- по-друге, будь-яка послідовність байтів, що передається, повинна забезпечуватися адресою призначення, що є сукупністю IP-адреси хоста і номера порту процесу.

IP-адреса, номер порту і послідовність інформаційних байтів ми будемо називати пакетом.

Після того, як пакет створено, процес відправляє його адресату через свій сокет. Транспортний рівень (протокол UDP) здійснює дії для доставки пакету за адресою призначення, проте, не дає щодо доставки ніяких гарантій. Іншими словами, протокол UDP забезпечує ненадійну передачу даних між процесами.

В роботі, у вигляді прикладу, ми модифікуємо для протоколу UDP мережний додаток з попередньої лабораторної роботи. Як ви побачите, коди програм для протоколів TCP і UDP значно різняться:

- по-перше, через відсутність процедури рукошлякування немає

необхідності у серверному сокеті;

- по-друге, вхідні і вихідні потоки даних не пов'язані з сокетами;
- по-третє, при пересиланні кожної групи байтів вказується IP-адреса хоста-одержувача і номер порту сервера;
- по-четверте, приймаючій стороні доводиться виділяти інформаційні байти з прийнятих пакетів.

Нагадаємо порядок дій, що виконуються нашим додатком:

- клієнт зчитує з стандартного пристрою вводу (клавіатури) рядок символів і посилає цей рядок серверу через свій сокет;
- сервер приймає рядок через свій сокет;
- сервер переводить всі символи рядка у верхній регістр;
- сервер відсилає модифікований рядок клієнту;
- клієнт одержує рядок і друкує його за допомогою стандартного пристрою виводу (монітора).

На рисунку 2.1 приведена схема взаємодії клієнта і сервера при використанні протоколу UDP.

Примітка. Розробка вихідних кодів сервера та клієнта здійснюється на одному ПК, а перевірка та демонстрація викладачу – на двох ПК.

Методичні вказівки для цієї роботи складені з орієнтацією на виконання роботи на комп'ютерах з фізичними іменами **RMF01** і **RMF02**.

Нагадаємо, для цих комп'ютерів при завантаженні на них клієнтської ОС **Microsoft Windows 7** має бути:

IP-адреса комп'ютера: **10.0.9.51** і **10.0.9.52** відповідно,
мережне ім'я комп'ютера: **rmf51** і **rmf52** відповідно.

Оскільки більшість з вас працює за комп'ютерами з іншими фізичними іменами, то при виконанні роботи вам доведеться використовувати відповідно інші (“свої”) IP-адреси, короткі і повні мережні імена.

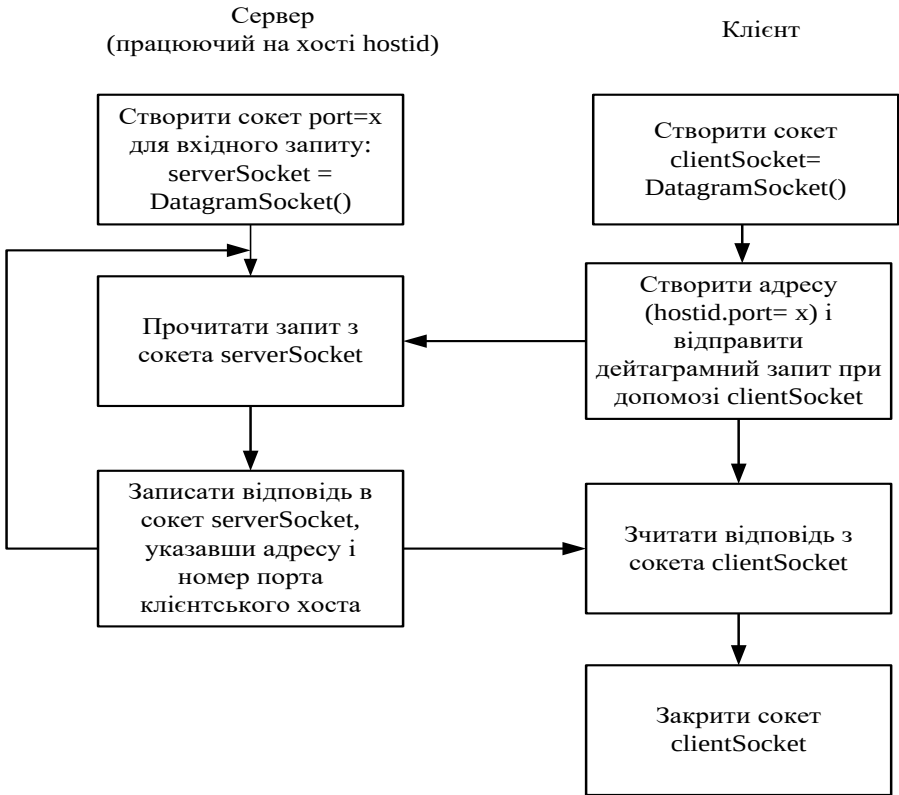


Рисунок 2.1 - Додаток клієнт/сервер, який використовує транспортні служби без установлення логічного з'єднання

2.2 Перевірка налаштувань мережного інтерфейсу комп'ютерів

На комп'ютері з фізичним ім'ям **RMF01**:

- завантажити **Microsoft Windows 7** (ім'я користувача – **admin**, пароль – **1234567**);
- перевірити налаштування мережного інтерфейсу;

– задайте **Пуск => Панель управления => Сеть и Интернет => Центр управления сетями и общим доступом => Изменение параметров адаптера**;

– у вікні **Сетевые подключения** запустіть - **Подключение по локальной сети**;

– у вікні **Состояние Подключение по локальной сети** клацніть **Свойства**;

– установіть курсор на **Протокол Интернета (TCP/IPv4)** і клацніть **Свойства**;

– виберіть опцію **Использовать следующий IP-адрес** і введіть:

IP-адреса: **10.0.9.51** (якщо фізичне ім'я вашого комп'ютера **RMF01+50**), (аналогічно, якщо фізичне ім'я вашого комп'ютера **RMF25**, то – **10.0.9.75**)

Маска подсети: **255.255.255.0**

Основной шлюз: **10.0.9.100**

Предпочитаемый DNS-сервер: **10.0.2.1**

і клацніть **ОК**;

– з поверненням у вікно **Подключение по локальной сети - свойства** клацніть **ОК**;

– закрийте вікно **Состояние–Подключение по локальной сети**;

– закрийте вікно **Просмотр основных сведений о сети и настройка подключений**.

На комп'ютері з фізичним ім'ям **RMF02**:

– завантажте **Microsoft Windows 7** (ім'я користувача – **admin**, пароль – **1234567**);

– перевірте настройки мережного інтерфейсу:

– задайте **Пуск => Панель управления => Сеть и Интернет => Центр управления сетями и общим доступом => Изменение параметров адаптера**;

– у вікні **Сетевые подключения** запустіть - **Подключение по локальной сети**;

– у вікні **Состояние Подключение по локальной сети** клацніть **Свойства**;

– установіть курсор на **Протокол Интернета (TCP/IPv4)** і клацніть **Свойства**;

– виберіть опцію **Использовать следующий IP-адрес** і введіть:

IP-адреса: 10.0.9.52 (якщо фізичне ім'я вашого комп'ютера **RMF02+50**)

Маска подсети: 255.255.255.0

Основной шлюз: 10.0.9.100

Предпочитаемый DNS-сервер: 10.0.2.1

і клацніть **ОК**;

– з поверненням у вікно **Подключение по локальной сети - свойства** клацніть **ОК**;

– закрийте вікно **Состояние–Подключение по локальной сети**;

– закрийте вікно **Просмотр основных сведений о сети и настройка подключений**.

2.3 Установка середовища програмування Java

Якщо на комп'ютерах з фізичними іменами **RMF01** і **RMF02** вже встановлено середовище програмування Java (про це свідчить наявність каталогу **G:\OpenJDK_7_u60**) — пропустіть цей крок. Якщо ні, то на обох комп'ютерах встановіть середовище програмування Java, як зазначено в першій лабораторній роботі в розділі 1.3. Після цього створіть на диску **G** папку **Projects**.

2.4 Створення клієнтської програми

На комп'ютері з фізичним ім'ям **RMF02** (мережним ім'ям **rmf52**; у вашого комп'ютера відповідне “своє” мережне ім'я):

– запустіть середовище розробки Eclipse;

– створіть проект з назвою "Lab2";

– створіть клас "UDPCClient" в пакеті "task1" (послідовність дій створення проекту та нового класу вказана в першій лабораторній роботі);

– наберіть наступний текст програми:

```
package task1;
```

```

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;
public class UDPCClient {
    public static void main(String[] args) throws
IOException
    {
        DatagramSocket clientSocket = new
DatagramSocket();
        InetAddress IPAddress =

InetAddress.getByName("localhost");

        Scanner inFromUser = new Scanner(System.in);
        System.out.print("Input sentence: ");
        String sentence = inFromUser.nextLine();

        byte[] sendData = sentence.getBytes();
        byte[] receiveData = new byte[1024];

        DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length,
IPAddress, 9876);
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket = new
DatagramPacket(
            receiveData, receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence = new String(
            receivePacket.getData(), 0,
receivePacket.getLength());
        System.out.println("Server's answer: " +
            modifiedSentence);
        clientSocket.close();
    }
}

```

Як показано на рисунку 2.2, програма **UDPCliant** створює один потік даних і один сокет.

Сокет має тип **DatagramSocket** і ім'я **clientSocket**.

Зверніть увагу, що при використанні протоколів UDP і TCP типи сокетів відрізняються: у випадку TCP звичайно застосовують тип **Socket**. Потік **inFromUser** є вхідним для програми і пов'язаний з стандартним пристроєм вводу (клавіатурою). Коли користувач вводить символи, вони потрапляють в програму через вхідний потік даних. Аналогічним чином вхідний потік був визначений в програмі **TCPCliant**.

Слід зазначити одну важливу деталь: при використанні протоколу UDP жоден з потоків (ні вхідний, ні вихідний) не пов'язаний з сокетом. Відправка байтів здійснюється в пакетах за допомогою об'єкту типу **DatagramSocket**.

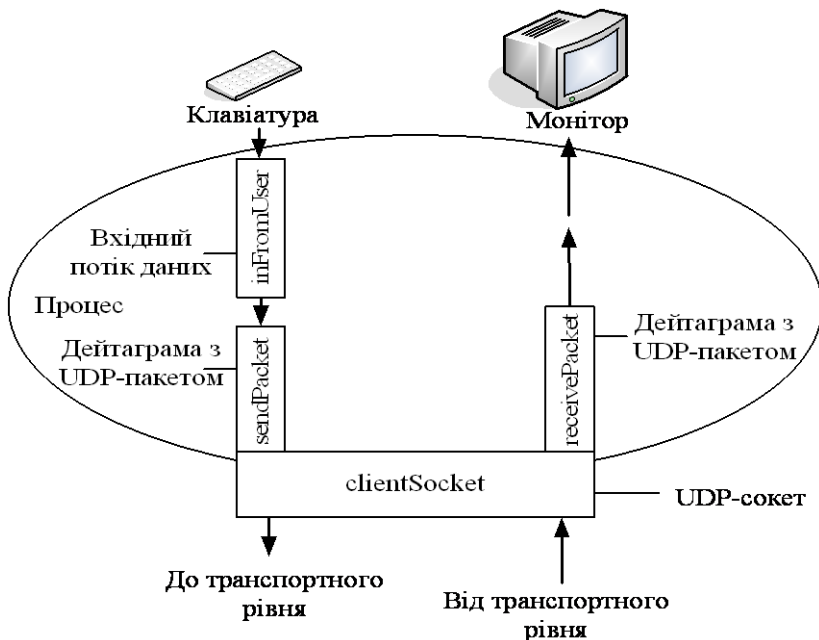


Рисунок 2.2 - В програмі **UDPCliant** є єдиний потік даних; сокет приймає і передає пакети процесу

Тепер більш детально розглянемо рядки коду програми **UDPCClient**.

```
DatagramSocket clientSocket = new DatagramSocket();
```

Ця команда створює об'єкт **clientSocket** типу **DatagramSocket**, однак на відміну від програми **TCPClient** не ініціює установлення TCP-з'єднання. З цієї причини конструктор **DatagramSocket()** не приймає як аргументи ім'я хоста сервера і номер порту. Таким чином, дія приведеного рядка зводиться лише до створення сокета клієнта.

```
InetAddress IPAddress = InetAddress.getByName("localhost");
```

Для того, щоб мати можливість передавати віддаленому процесу інформацію, необхідно мати в своєму розпорядженні його адресу. Як ми знаємо, частиною адреси процесу є IP-адреса хоста, на якому він виконується. Даний рядок створює DNS-запит IP-адреси хоста з ім'ям **localhost** (як і в програмі **TCPClient**, хост **localhost** необхідно замінити реальним ім'ям хоста). Фактично робота з DNS-запитом здійснюється методом **getByName()**, який приймає як аргумент ім'я хоста і повертає отриману IP-адресу. IP-адреса поміщається в об'єкт **IPAddress** типу **InetAddress**.

```
Scanner inFromUser = new Scanner(System.in);
```

Тут ми створюємо потоковий об'єкт для читання даних типу **Scanner**. Ініціалізація потоку даних проводиться об'єктом **System.in**, що пов'язує потік із стандартним пристроєм вводу. Після виконання цієї команди клієнту починають передаватися символи, що вводяться користувачем з клавіатури.

```
System.out.print("Input sentence: ");
```

Цей метод виводить в командне вікно фразу запрошення ввести речення з клавіатури.

```
String sentence = inFromUser.nextLine();
```

Приведений рядок оголошує об'єкт **sentence** типу **String** та ініціалізує його послідовністю символів, що вводяться користувачем. Введення закінчується натисненням користувачем клавіші **Enter**. Як бачимо, для запису символів в змінну використовується потоковий об'єкт **inFromUser**.

```
byte[] sendData = sentence.getBytes();
```

Рядок **sentence** за допомогою методу **getBytes()** перетворюється в масив байтів, який зберігається в змінну **sendData**.

```
byte[] receiveData = new byte[1024];
```

Байтовий масив **receiveData** призначений для зберігання інформації, що приймається.

```
DatagramPacket sendPacket = new DatagramPacket(  
    sendData, sendData.length, IPAddress, 9876);
```

Тут відбувається створення пакету **sendPacket**, призначеного для відправки серверу через сокет. В пакет входять дані, що містяться в змінній **sendData**, розмір цих даних, IP-адреса хоста призначення і номер порту додатку (в даному випадку ми вибрали номер **9876**). Зверніть увагу на те, що пакет має тип **DatagramPacket**.

```
clientSocket.send(sendPacket);
```

Метод **send()** об'єкту **clientSocket** відправляє пакет **sendPacket** (як згадувалося раніше, **clientSocket** є клієнтським сокетом додатку). Звернемося до механізмів передачі даних, що здійснюються протоколами TCP і UDP. У випадку TCP рядок символів включається у вихідний потік, що має логічний зв'язок з сервером. У випадку UDP нам необхідно створити пакет, що містить адресу сервера. Після передачі пакета клієнт чекає прийому пакета від сервера.

```
DatagramPacket receivePacket = new DatagramPacket(  
    receiveData, receiveData.length);
```

Тут клієнт створює змінну для зберігання пакету **receivePacket** типу **DatagramPacket**.

```
clientSocket.receive(receivePacket);
```

Клієнт знаходиться в стані очікування до тих пір, поки не буде початий прийом пакету від сервера; символи, що приймаються, поміщаються в змінну **receivePacket**.

```
String modifiedSentence = new String(  
    receivePacket.getData(), 0, receivePacket.getLength());
```

В цьому рядку відбувається витягання даних з об'єкту **receivePacket** і перетворення типу з масиву байтів в рядок з подальшим присвоюванням змінній **modifiedSentence**. Другий параметр конструктора вказує, що рядок починається з першого елементу масива, а останній задає довжину рядку.

```
System.out.println("Server's answer: " + modifiedSentence);
```

Цей рядок, присутній раніше в програмі **TCPClient**, друкує на екрані клієнта рядок **modifiedSentence**.

```
clientSocket.close();
```

Виклик методу **close()** об'єкту **clientSocket** приводить до закриття сокета. Оскільки протокол UDP не використовує логічне

з'єднання, при закритті сокета клієнт не посилає серверу повідомлень транспортного рівня (як це було у випадку TCP).

2.5 Створення серверної програми

Створіть клас **"UDPServer"** в пакеті **"task1"**;

Наберіть наступний текст програми:

```
package task1;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class UDPServer {
    public static void main(String[] args) throws
    IOException
    {
        DatagramSocket serverSocket = new
        DatagramSocket(9876);

        byte[] receiveData = new byte[1024];
        byte[] sendData;

        while (true)
        {
            DatagramPacket packet = new DatagramPacket(
                receiveData,
                receiveData.length);
            serverSocket.receive(packet);

            String sentence = new
            String(packet.getData(),
                                0,
            packet.getLength());
            String capitalizedSentence =
            sentence.toUpperCase();
```

```

        sendData = capitalizedSentence.getBytes();
        packet.setData(sendData);
        serverSocket.send(packet);
    }
}

```

Як показано на рисунку 2.3, програма **UDPServer** створює єдиний сокет з ім'ям **serverSocket** типу **DatagramSocket**; цей же тип використовувався нами в програмі-клієнті додатку. Аналогічно, з сокетом не пов'язаний жоден з потоків даних.

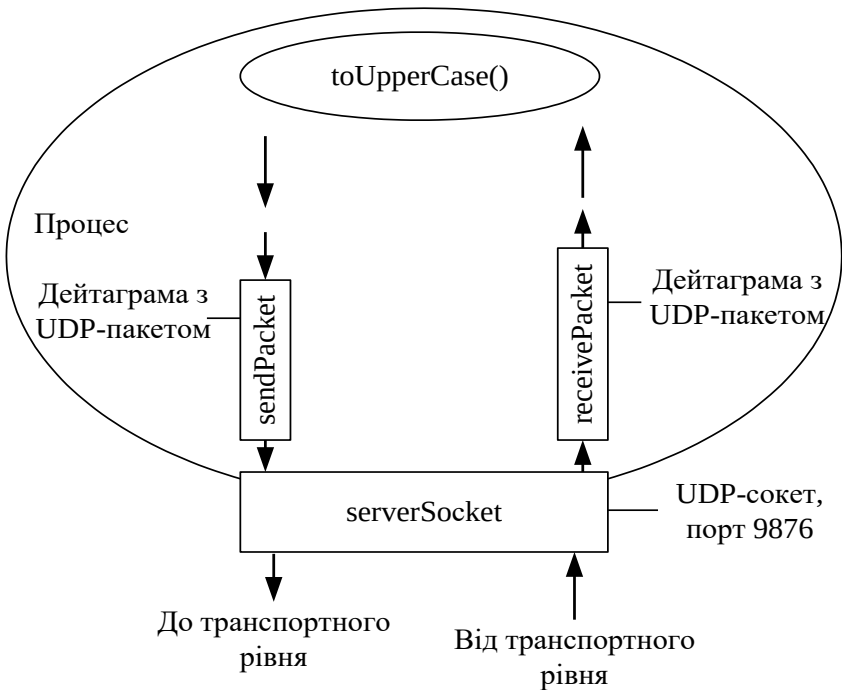


Рисунок 2.3 - Програма **UDPServer** не має потоків;
сокет приймає і передає пакети процесу

```
DatagramSocket serverSocket = new DatagramSocket(9876);
```

Цей рядок створює об'єкт **serverSocket** типу **DatagramSocket** з номером порту **9876**. Передача і прийом всіх даних сервером здійснюється за допомогою цього об'єкту.

Оскільки протокол UDP не передбачає встановлення логічного з'єднання, у нас немає необхідності підтримувати окремий сокет для кожного клієнта і спеціальний серверний сокет для рукописних з новими клієнтами, як для протоколу TCP. У випадку декількох клієнтів всі вони здійснюватимуть обмін даними з сервером через єдиний сокет **serverSocket**.

```
String sentence = new String(packet.getData(),  
                                0, packet.getLength());  
String capitalizedSentence = sentence.toUpperCase();
```

Вищенаведені рядки здійснюють розпакування пакету, переданого клієнтом.

Перша команда витягає з пакету дані і поміщає їх в рядковий об'єкт **sentence** (аналогічна команда є в програмі **UDPClient**). Наступна команда здійснює перетворення рядку **sentence** у верхній регістр.

```
sendData = capitalizedSentence.getBytes();  
packet.setData(sendData);
```

За допомогою цих рядків змінене речення готується до відправки клієнту. Перша команда повертає масив байт речення, а наступна встановлює цей масив як дані пакету, що будуть відправлені. При цьому використовується об'єкт пакету, що був прийнятий раніше, щоб не вказувати для вихідного пакету адресу та порт приймача (клієнта).

2.6 Підготовка до запуску на іншому ПК

Виконайте наступні дії:

- скопіюйте каталог G:\Projects із його вмістом на другий комп'ютер;

- запустіть середовище розробки Eclipse:

```
J:\Install\Eclipse\eclipse.exe
```

- імпортуйте створений проект. Для цього треба обрати пункт **File > Import**, у вікні, що з'явилося, обрати **General > Existing projects**

into workspace, натиснути **Next**, вказати шлях **G:\Projects** у полі **Select root directory**;

– оберіть у полі **Projects** проект **Lab2** для імпорту та натисніть кнопку **Finish**.

2.7 Запуск програм

Запустіть сервер на комп'ютері з фізичним ім'ям **RMF01**. Для цього треба відкрити файл **UDPServer.java**, та обрати пункт меню **Run > Run**, або значок запуску з панелі. Після цього сервер перейде в режим очікування.

На комп'ютері з фізичним ім'ям **RMF02** запустіть клієнт. Для цього треба відкрити файл **UDPClient.java**, та обрати пункт меню **Run > Run**, або значок запуску з панелі. В консолі головного вікна **Eclipse** з'явиться вікно терміналу для взаємодії з клієнтом.

– наберіть рядок, що містить латинські символи в нижньому і верхньому регістрі (наприклад, **aaaAAAbbbbBBB**) і натисніть **Enter**;

– запишіть результат роботи програм і проаналізуйте його;

– продемонструйте роботу програм викладачу і поясніть їх роботу.

2.8 Завдання для самостійного програмування

Змініть додаток так, щоб його поведінка була наступною:

– клієнт генерує ряд випадкових чисел, довжина якого задається з клавіатури та відправляє його серверу;

– сервер при отриманні обчислює суму елементів ряду і відсилає її у відповідь клієнту;

– клієнт отримує результат з сервера та також обчислює суму елементів ряду, після цього виводить обидва результати на екран;

2.9 Відновлення комп'ютерів у початковий стан

На обох комп'ютерах видаліть папку **G:\Projects**

2.10 Зміст письмового звіту

- хід роботи;
- основні функції;
- відповіді на контрольні питання.

2.11 Контрольні питання

1. Укажіть принципові відзнаки протоколів UDP та TCP.
2. Розкрийте поняття UDP-дейтаграми.
3. У яких випадках доцільно здійснювати передачу даних повідомлення за протоколом UDP?
4. Вкажіть методики передавання ряду чисел через мережу мовою Java (або будь-якою іншою).

РЕКОМЕНДОВАНА ЛИТЕРАТУРА

1. Хорстманн К. Java. Библиотека профессионала. Том 1. Основы / К. Хорстманн. – 10-е изд.: Пер. с англ. – М.: ООО "И.Д. Вильямс", 2016. – 864 с.
2. Олифер В. Г. Компьютерные сети. Принципы, технологии, протоколы / В.Г. Олифер, Н.А.Олифер. // Учебник для вузов. – 5-е изд. – СПб.: Питер, 2016. – 992 с.
3. Таненбаум Э. Компьютерные сети / Э. Таненбаум, Д.Уэзеролл. – 5-е изд. – СПб.: Питер, 2012. – 960 с.
4. Одом, Уэнделл. Официальное руководство Cisco по подготовке к сертификационным экзаменам CCENT/CCNA ICND1 100-101 / У. Одом. – акад. изд.: Пер. с англ. – М.; ООО "И. Д. Вильямс", 2015. – 912 с. – ISBN 978-5-8459-1906-9.
5. Одом, Уэнделл. Официальное руководство Cisco по подготовке к сертификационным экзаменам CCNA ICND2 200-101: маршрутизация и коммутация / У. Одом. – акад. изд.: Пер. с англ. – М.; ООО "И. Д. Вильямс", 2015. – 736 с. – ISBN 978-5-8459-1907-6.