МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет “Львівська політехніка”

Кафедра телекомунікацій



курсова робота

З дисципліни:

«Розподілені сервісні системи та Cloud-технології»

на тему:

“***Створення розподіленого додатку для віддаленого використання клієнтами обчислювальних ресурсів високопродуктивних серверів***”

Виконав: ст. гр. ТРІМ-11

Ворко Р.В.

Перевірив: Асистент Лаврів О.А.

### Львів – 2019

ЗМІСТ

[ВСТУП……………………………………………………………………….…..…6](#_Toc9706690)

[1. ТЕХНОЛОГІЯ RMI………………………………………………………..……7](#_Toc9706691)

[2. СТВОРЕННЯ РОЗПОДІЛЕНОГО ДОДАТКУ ДЛЯ ВІДДАЛЕНОГО ВИКОРИСТАННЯ КЛІЄНТАМИ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ ВИСОКОПРОДУКТИВНИХ СЕРВЕРІВ……………………………………..…11](#_Toc9706692)

2.1 Створення інтерфейсів для взаємодії клієнта та сервера………...…....11

2.2 Серверна частина………………………………………………………....12

2.3Клієнтська частина ……………………………………………………....13

2.4 Створення основних полів та методів для реалізації розподіленої системи за допомогою технології RMI на стороні сервера………………..…...15

[ВИСНОВОК……………………………………………………………….…..…..1](#_Toc9706693)9

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…………………………………………](#_Toc9706694)20

**ВСТУП**

**RMI** (Remote Method Invocation) – віддалений виклик методів або механізм, який дозволяє об’єкту в одній Java-машині (в даному випадку клієнтська частина) викликати методи об’єкта в іншій (сервер).

Віддалений об'єкт є деякими даними, сукупність яких визначає його стан. Цей стан можна змінювати шляхом виклику його методів. Зазвичай можливий прямий доступ до даних віддаленого об'єкту, при цьому відбувається неявний віддалений виклик, необхідний для передачі значення поля даних об'єкту між процесами. Методи і поля об'єкту, які можуть використовуватися через віддалені виклики, доступні через деякий зовнішній інтерфейс класу об'єкту. Зовнішній інтерфейс компонента зазвичай збігається із зовнішнім інтерфейсом одного з класів компоненту.

На стороні клієнта створюється клієнтська заглушка, яка називається посередником (proxy). Посередник реалізує той самий інтерфейс, що і віддалений об'єкт. Викликаючий процес використовує методи посередника, який перетворює (маршалізує) їх параметри для передачі по мережі, і передає їх по мережі серверу. Проміжне середовище на стороні сервера десеріалізує параметри і передає їх заглушці на стороні сервера, яку називають каркасом (skeleton) або, як і у віддаленому виклику процедур, заглушкою. Каркас зв'язується з деяким екземпляром віддаленого об'єкту. Це може бути як новий, так і існуючий екземпляр об'єкту, залежно від моделі використання віддалених об'єктів.

Весь описаний процес називається маршалінгом віддаленого об'єкту за посиланням (marshal by reference).

***1. ТЕХНОЛОГІЯ RMI***

При доступі до об'єктів на іншому комп'ютері, можна викликати методи цього об'єкту. Необхідно тільки доставити параметри методу на інший комп'ютер, повідомити об'єкт про необхідність виконання методу, а потім передати назад значення (якщо метод не void). Механізм RMI дає можливість організувати виконання всіх цих операцій.

У термінах RMI об'єкт, який викликає віддалений метод, називається [клієнтським об'єктом](https://uk.wikipedia.org/w/index.php?title=%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82%D1%81%D1%8C%D0%BA%D0%B8%D0%B9_%D0%BE%D0%B1%27%D1%94%D0%BA%D1%82&action=edit&redlink=1), а віддалений об'єкт - [серверним об'єктом](https://uk.wikipedia.org/w/index.php?title=%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B8%D0%B9_%D0%BE%D0%B1%27%D1%94%D0%BA%D1%82&action=edit&redlink=1). Комп'ютери виступають у ролі клієнта і сервера тільки для конкретного виклику. Цілком можливо, що при виконанні наступної операції ці комп'ютери поміняються ролями, тобто сервер може сам стати клієнтом при зверненні до об'єкту на іншому комп'ютері.

При виклику методу віддаленого об'єкту, насправді викликається звичайний Java метод, [інкапсульований](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D1%96%D1%8F" \o "Інкапсуляція) у спеціальному [об'єкті-заглушці](https://uk.wikipedia.org/w/index.php?title=%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82-%D0%B7%D0%B0%D0%B3%D0%BB%D1%83%D1%88%D0%BA%D0%B0&action=edit&redlink=1) (stub), який є представником серверного об'єкту. Заглушка знаходиться на клієнтському комп'ютері, а не на сервері. Вона упаковує параметри віддаленого методу у блок байтів. Кожен параметр кодується за допомогою алгоритму, що забезпечує незалежність від апаратного забезпечення. Наприклад, числа завжди передаються в порядку, при якому спочатку передається старший байт (big-endian). При цьому об'єкти піддаються [серіалізації](https://uk.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D1%96%D0%B0%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F" \o "Серіалізація). Процес кодування параметрів називається [розгортанням параметрів](https://uk.wikipedia.org/w/index.php?title=%D0%A0%D0%BE%D0%B7%D0%B3%D0%BE%D1%80%D1%82%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D1%96%D0%B2&action=edit&redlink=1) (parameter marshaling). Основна мета розгортання параметрів - перетворення їх у формат, придатний для передачі параметрів від однієї віртуальної машини до іншої.

Метод, який належить заглушці, створює блок, до якого входять такі елементи:

* ідентифікатор віддаленого об'єкту;
* опис методу який викликається;
* розгорнуті параметри.

Потім метод заглушки посилає цю інформацію серверу. Далі об'єкт-одержувач виконує для кожного виклику віддаленого методу наступні дії:

* [згортання параметрів](https://uk.wikipedia.org/w/index.php?title=%D0%97%D0%B3%D0%BE%D1%80%D1%82%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D1%96%D0%B2&action=edit&redlink=1);
* пошук об'єкту, який викликається;
* виклик заданого методу;
* витягування та розгортання значення яке повертається або [виключення](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%B2%D0%B8%D0%BD%D1%8F%D1%82%D0%BA%D1%96%D0%B2), згенерованого цим методом;
* передача пакету, який складається із розгорнутих даних, які повертаються, об'єкту-заглушці на клієнтському комп'ютері.

Клієнтський об'єкт-заглушка згортає повернене значення або виключення, отримане із сервера. Результат згортання стає значенням методу заглушки. Якщо віддалений метод повертає виключення, то об'єкт-заглушка повторить його в середовищі об'єкту-клієнта.

Remote Method Invocation - механізм, який дозволяє викликати метод видалення об'єкта. Відповідно йому, всі операції по підготовці та передачі даних інкапсулюються в викликані методами клієнтського об'єкта-заглушки (заглушки). Сам же виклик метода нічим не відрізняється від вибору методів звичайного локального об'єкта, за невеликим виключенням:

* локальні об'єкти передаються за знанням (копії);
* при передачі віддаленого об'єкта, якщо він експортований, передається цей об'єкт;
* передавальні об'єкти повинні бути серіалізовані.

Крім всіх інших виняткових ситуацій, при викликанні віддаленого методу може створюватися виключення RemoteException.

Також потрібно відмітити, що при виході методу ми працюємо з віддаленим інтерфейсом, а не з віддаленим класом.

Задача RMI - організація клієнт-серверної взаємодії. Це означає, що не треба турбуватися про передачу та попередню обробку даних (а також протокол і т.д.). Якщо у клієнт-серверному середовищі є програми, написані не тільки на Java, то від RMI слід відмовитися.

RMI заснована на більш ранній технології віддаленого виклику процедури Remote Procedure Call (RPC), розробленої в 80-х роках і використовуваної для процедурного програмування.

RPC дозволяє встановлювати окремі додатки, що викликають функцію на іншому комп'ютері. Таким чином, RPC виконує всю роботу по організації мережевих взаємодій і маршалінгу даних Важливим недоліком RPC є необхідність використання спеціальної мови визначення інтерфейсу (IDL) для опису функцій. З метою подолання цих недоліків і була розроблена технологія RMI.

RMI містить набір об'єктів (класів) для організації віддаленої взаємодії з Java-додатком. Системи RMI часто включають два окремих додатки: сервер і клієнт. Серверний додаток, як правило, створює віддалені об'єкти , що роблять доступні посилання на ці об'єкти і перебуває в очікуванні вибору методів цих об'єктів. Клієнтський додаток отримує від сервера посилання на віддалені об'єкти, після чого він їх викликає. Технологія RMI забезпечує механізм взаємодії клієнта і сервера та передачі між ними відповідної інформації, реалізованої у вигляді java.rmi пакета, який містить цілий ряд вкладених підпакетів; java.rmi.server реалізує функції сервера RMI.

RMI забезпечує маршалінг даних по мережі і дозволяє додаткам передавати об'єкти з допомогою механізму серіалізації об'єктів та не вимагає від програми знання мови IDL. Крім того, RMI підтримує тільки Java.

***Опис віддаленого RMI об'єкта***

Серверний RMI об'єкт повинен успадковувати (extends) властивості класу java.rmi.server.UnicastRemoteObject, який представляє базові функціональні можливості, необхідні віддаленим об'єктам для обслуговування віддалених запитів. Конструктори і методи класу UnicastRemoteObject збуджують виняток RemoteException.

Конструктор класу UnicastRemoteObject забезпечує експорт об'єкта, щоб він був доступним для прийому віддалених викликів. Експорт дозволяє віддаленому RMI об'єкту очікувати з'єднань з клієнтами для здійснення взаємодії типу "точка-точка" з використанням стандартних з'єднань через сокети. Передбачається, що клієнти RMI повинні здійснювати з'єднання з використанням порту 1099 для пошуку віддаленого об'єкта в реєстрі RMI сервера. Перевантажений конструктор класу UnicastRemoteObject дозволяє визначити свій номер порту для експорту віддаленого об'єкта. Посилання встановлення зв'язку з віддаленим RMI об'єктом зазвичай має такий вигляд: rmi://host:port/object, де

* **Host** – являє собою ім'я комп'ютера, в якому визначено реєстр сервера rmiregistry для віддалених об'єктів із зареєстрованим віддаленим RMI об'єктом,
* **Port** – є номер порту, на якому працює серверний додаток. За замовчуванням для реєстру RMI об'єктів використовується порт 1099, який можна в цьому випадку в засланні не вказувати;
* **Оbject** – це ім'я віддаленого RMI об'єкта, зареєстрованого в реєстрі сервера.

Для зв'язування віддаленого RMI об'єкта з реєстром сервера використовується один з методів bind або rebind. Метод rebind реєструє об'єкт в реєстрі з попередньою перевіркою; якщо об'єкт був раніше зареєстрований під цим ім'ям, то метод замінить його новим об'єктом. Це може знадобитися при реєстрації нової версії існуючого віддаленого об'єкта.

**2. *СТВОРЕННЯ РОЗПОДІЛЕНОГО ДОДАТКУ ДЛЯ ВІДДАЛЕНОГО ВИКОРИСТАННЯ КЛІЄНТАМИ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ ВИСОКОПРОДУКТИВНИХ СЕРВЕРІВ***

Згідно до варіанту використовується алгоритм сортування з прямим включенням.

**Виклик віддалених методів**

Для цього застосовується:

**public static final** String **RMI\_SERVER\_NAME** = **"lpi.server.rmi"**-унікальне ім’я віддаленого об’єкту (сервера). Повинне бути однаковим на клієнтській частині і на сервері.

**private** Registry **rgs**-створення об’єкта “реєстр віддалених об’єктів”  
 **private** IServer **server**-створення об’єкту для віддаленого доступу.

**this**.**rgs**= LocateRegistry.getRegistry(**Host**, **Port**)-реєстрація об’єкта на конкретному імені та порті.

**this**.**server** = (IServer) **rgs**.lookup(**"lpi.server.rmi"**)-отримання проксі-об‘єкта у реєстра.

***2.1 Створення інтерфейсів для взаємодії клієнта та сервера***

Сервер повинен мати змогу виконувати будь-які алгоритми, які клієнт відправить для виконання. Для цього необхідно створити інтерфейс Task, який повинен реалізуватися класами, які містять сам алгоритм обробки вхідних даних та повернення результату.

**public interface** IServer **extends** Remote {  
 **public interface** Task<T> {  
 T execute();  
 }

Цей інтерфейс доступний як на стороні клієнта так і на стороні сервера. Клієнт створює об’єкт, який реалізує цей інтерфейс, та передає цей об’єкт на сервер.

Сервер викликає метод execute(), результат виконання якого потім повертається клієнтові.

Для доступу клієнта до сервера та запуску виконання клієнтського завдання необхідно створити інтерфейс серверного об’єкту:

<T> T executeTask(Task<T> t) **throws** RemoteException;

***2.2 Серверна частина***

Клас Rmi.java:

**private** IServer **proxy**;

**private int port** = 3000;

**private** Registry **registry**;

**public void** run() {  
 **try** {  
 **this**.**proxy** = (IServer) UnicastRemoteObject.exportObject(**this**, **this**.**port**);  
 **this**.**registry** = LocateRegistry.createRegistry(**this**.**port**);  
 **this**.**registry**.bind(**"lpi.server.rmi"**, **this**.**proxy**);  
 **this**.**sessionTimer**.schedule(**new** SessionCleanupTask(), **CLEANUP\_DELAY\_MS**, **CLEANUP\_DELAY\_MS**);  
  
 System.**out**.printf(**"The RMI server was started successfully on the port %s%n"**, **this**.**port**);  
  
 } **catch** (AlreadyBoundException | RemoteException e) {  
 **throw new** RuntimeException(**"Failed to start server"**, e);  
 }  
}

Клас Main.java:

**try** (RmiServer server = **new** RmiServer(args)) {  
 server.run();  
 System.**in**.read();  
}

В класі Клас Main.javа створюється екземпляр класу RmiServer.java та викликається його метод run(), в якому відбувається створення об’єкта “реєстр віддалених об’єктів”,створення об’єкту для віддаленого доступу, реєстрація об’єкта на конкретному імені та порті та отримання проксі-об‘єкта у реєстра.

***2.3 Клієнтська частина***

Створено клас, який буде містити алгоритм, що повинен виконатися на сервері. Конструктор класу приймає вхідні дані. Після цього створений об’єкт передається на сервер, де сервер запускає виконання алгоритму через метод execute() та повертає результат клієнтові.

**public static class** Algoritm **implements** Task <**byte**[]>, Serializable {  
 **private** Integer[] **file\_to\_sort**;  
 **int t** = 0;

**private byte**[] **b\_1**;   
 **private** Integer[] **Mass\_to\_sort**;

////////////конструктор класу  
 **public** Algoritm(File file\_sort) **throws** IOException {

//Створення масиву байтів з вхідного файлу   
 **b\_1** = Files.readAllBytes(file\_sort.toPath());  
 ///переведення в масиву байтів в тип String  
 String St = **new** String(**b\_1**);

//////Вирізання пробілів зі стрічки  
 String[] St\_1 = St.split(**" "**);

//Створення цілочисельного масиву та переписування в нього поперередній //масив, приведений до цілочисельного типу  
 **Mass\_to\_sort** = **new** Integer[St\_1.**length**];  
 **int** i=0;  
 **while**(i <=St\_1.**length**-1) {  
 **Mass\_to\_sort**[i] = Integer.parseInt(St\_1[i]);  
 i++;  
 }  
 System.**out**.println(**Mass\_to\_sort**.toString());  
 }  
 //// запуск виконання алгоритму через метод execute()  
 @Override  
 **public byte**[] execute() {  
 String buff1;  
 **byte**[] buff2;

///Виклик методу сортування  
 Sort(**Mass\_to\_sort**.**length**);  
 StringBuffer buff = **new** StringBuffer();

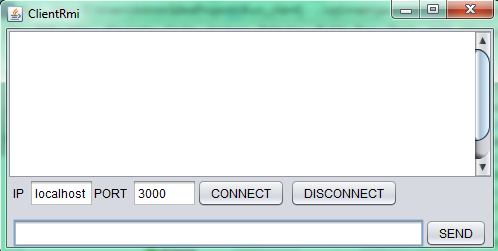
////Створення відсортованого масиву байтів   
 **int** i=0;  
 **while**(i <= **Mass\_to\_sort**.**length**-1) {  
 buff.append(**Mass\_to\_sort**[i]);  
 **if** (i!=(**Mass\_to\_sort**.**length**-1)) {  
 buff.append(**" "**);  
 }  
 i++;  
 }  
 buff1=buff.toString();  
 buff2=buff1.getBytes();

////////повернення відсортованого масиву байтів  
 **return** buff2;  
 }}}}

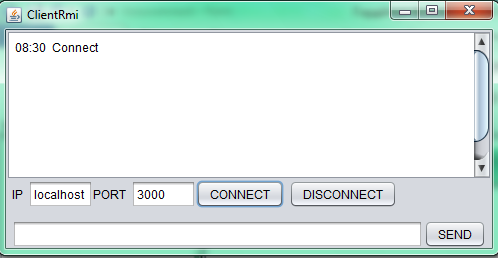
***2.4 Створення основних полів та методів для реалізації розподіленої системи за допомогою технології RMI на стороні сервера***

1. Вікно клієнтського додатку.

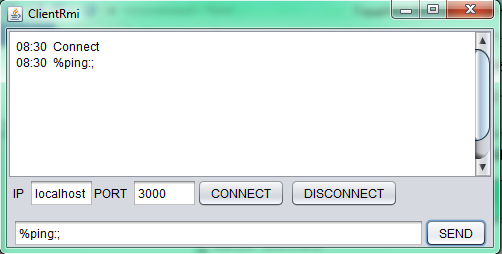
Він містить поле ІР (за замовчуванням localhost), PORT (3000), кнопки для встановлення і для розірвання з’єднання, та поле вводу команд.



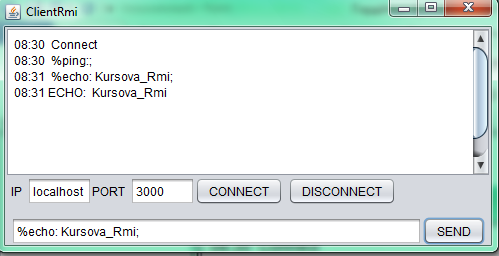
1. Підключення до сервера за вказаною ІР та портом.



1. Надсилання команди PING для перевірки зв’язку між клієнтською та серверною частинами. Для спрощення на серверній стороні відповіді не передбачено.



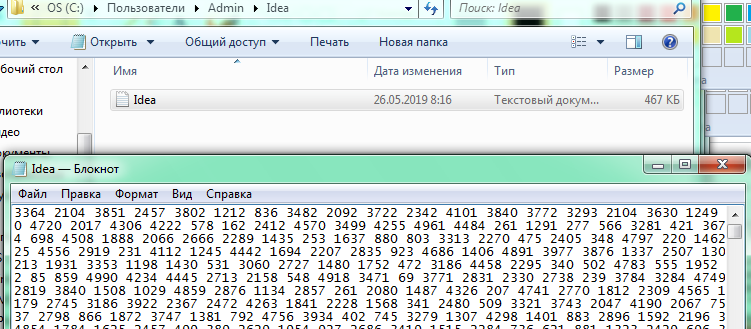
1. Команда Echo. Вона складається з двох частин: %echo-заголовок команди та певного повідомлення, яке сервер відсилає назад. Команда починається із символу %, який показує, що це початок команди. Метод (як і інші команди) є реалізований на серверній частині, а в клієнтській тільки викликається, що значно спрощує реалізацію клієнтської частини та необхідності формувати повністю повідомлення для правильного пересилання і обробки. На серверній частині є приймач віддалених об’єктів-“заглушка”, який відділяє заголовок і обробляє прийняте повідомлення.



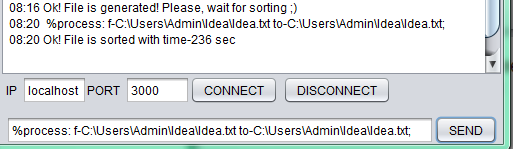
1. Команда process . Має вигляд %process: f-<Шлях до файлу з вхідними даними> to-<Шлях до файлу для збереження результатів обробки>;.

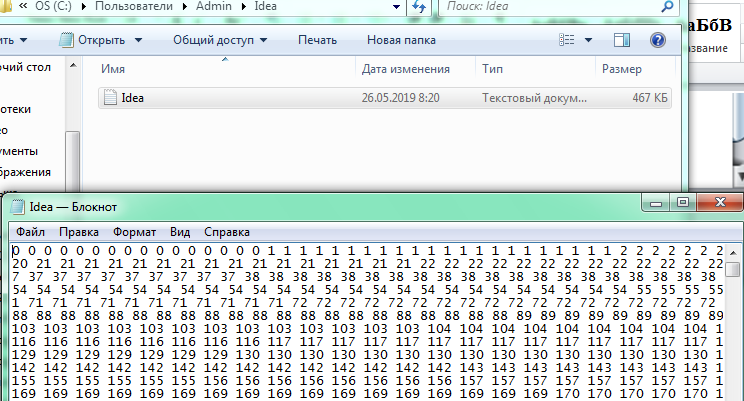
Внаслідок виконання команди вихідний файл рандомно заповнюється цілими числами. Потім файл надсилається на сервер для сортування і клієнт переходить в режим очікування, поки сервер поверне результат виконання алгоритму сортування- відсортований файл.

Згенерований файл:



Після сортування:





Увесь код знаходиться на GitHub:

Клієнтська частина: <https://github.com/Elen12344321/Kurs_client>

Сервер: <https://github.com/Elen12344321/server.rmi1>

**ВИСНОВОК**

В даній курсовій роботі було створено розподілений додаток для віддаленого використання клієнтами обчислювальних ресурсів високопродуктивних серверів. Для цього використалася технологія віддаленого виклику методів-Remote Method Invocation - механізм, який дозволяє викликати метод видалення об'єкта.

Відповідно йому, всі операції по підготовці та передачі даних інкапсулюються в викликані методами клієнтського об'єкта-заглушки (заглушки). Це механізм, який дозволяє об’єкту в одній Java-машині (в даному випадку клієнтська частина) викликати методи об’єкта в іншій (сервер). RMI містить набір об'єктів (класів) для організації віддаленої взаємодії з Java-додатком. Серверний додаток, як правило, створює віддалені об'єкти , що роблять доступні посилання на ці об'єкти і перебуває в очікуванні вибору методів цих об'єктів.

Було створено клієнтський додаток, який може взаємодіяти із сервером за допомогою технології RMI: підключатися до сервера, комунікувати, застосовуючи команди ping та echo. Також реалізована можливість передати файл на сервер, викликаючи метод сортування; потім відсортований файл сервер повертає назад. Для сортування використовується алгоритм прямого включення, його недоліком є досить повільне сортування великих файлів, що і зумовлює нечасте застосування даного алгоритму в сучасних обчислювальних системах.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Методичні вказівки до виконання курсової роботи з дисципліни «Розподілені сервісні системи та Cloud-технології (Distributed systems and Cloud technologies)».

2 <https://docs.oracle.com/javase/tutorial/rmi/overview.html>

3 <https://habr.com/ru/post/415935/>

4 <http://java-online.ru/java-rmi.xhtml>

5 <https://www.cs.usfca.edu/~galles/visualization/>

6 <https://prog-cpp.ru/algorithm-sort/>

7 <http://www.ejbtutorial.com/java-rmi/a-step-by-step-implementation-tutorial-for-java-rmi>