

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №1**

**з дисципліни «Компонентні технології програмного забезпечення»**

**на тему:**

**«**Розробка клієнт-серверної взаємодії віддалених об’єктів з використанням технології JAVA-RMI.**»**

Виконав:

Студент гр. ПЗ2011

Кулик С.В.

Прийняв:

Андрющенко В. О.

Дніпро, 2023

**Тема.** Розробка клієнт-серверної взаємодії віддалених об’єктів з використанням технології JAVA-RMI.

**Теоретичні відомості**

Java RMI (Remote Method Invocation) - це механізм, який дозволяє викликати методи віддалених об'єктів у розподілених системах. Використання RMI дозволяє програмістам створювати додатки, що працюють в розподілених середовищах, як один з найбільш простих і зручних способів.

Розподілена об'єктна модель, що описує, яким чином здійснюється виклик віддалених методів, що працюють на іншій віртуальній машині Java. 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, а віддалений об'єкт – сервернийм об’єктом. 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 у спеціальному об’єкті-заглушці (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. Процес кодування параметрів називається розгортанням параметрів (parameter marshaling). Основна мета розгортання параметрів - перетворення їх у формат, придатний для передачі параметрів від однієї віртуальної машини до іншої.

**Текст програми**

**Файл LeasingService.java**

import java.rmi.\*;  
public interface LeasingService extends Remote  
{  
 // Реєстрація машини  
 public void addNewCar(Car car) throws RemoteException;  
  
 // Оплата лізингу машини  
 public void leasePayment(Car car, double money) throws RemoteException;  
  
 // Перевірка періоду лізингу машини  
 public double getLeasingPeriod(Car car) throws RemoteException;  
}

**Файл LeasingServiceImpl.java**

import java.util.List;  
import java.util.ArrayList;  
import java.rmi.RemoteException;  
import java.rmi.registry.Registry;  
import java.rmi.registry.LocateRegistry;  
import java.rmi.server.UnicastRemoteObject;  
  
public class LeasingServiceImpl extends UnicastRemoteObject  
 implements LeasingService  
{  
 private static final long *serialVersionUID* = 1L;  
 private List<Car> cars;  
 // инициализация сервера  
 public LeasingServiceImpl() throws RemoteException  
 {  
 super();  
 cars = new ArrayList<Car>();  
 }  
 @Override  
 public void addNewCar(Car car) throws RemoteException  
 {  
 cars.add(car);  
 System.*out*.println("We get new car: " + car.getCar());  
 }  
 @Override  
 public void leasePayment(Car car, double money)  
 throws RemoteException {  
 for (Car c : cars) {  
 if (c.equals(car)) {  
 c.setMoney(c.getMoney() + money);  
 System.*out*.println("Payment: "  
 + "from " + car.getDriver()  
 + ", car - " + car.getCar()  
 + ", summa = " + money);  
 break;  
 }  
 }  
 }  
 @Override  
 public double getLeasingPeriod(Car car)  
 throws RemoteException {  
 double period = 0;  
 for (Car c : cars) {  
 if (c.equals(car)) {  
 period = c.getLeasing\_term();  
 System.*out*.println("Driver: " + car.getDriver()  
 + ", car - " + car.getCar()  
 + ", leasing term = " + period);  
 break;  
 }  
 }  
 return period;  
 }  
 */\*\*  
 \* Старт удаленного RMI объекта BillingService  
 \** ***@param*** *args аргументы  
 \** ***@throws*** *Exception  
 \*/* public static void main (String[] args) throws Exception  
 {  
 String localhost = "127.0.0.1";  
 String RMI\_HOSTNAME = "java.rmi.server.hostname";  
 try {  
 System.*setProperty*(RMI\_HOSTNAME, localhost);  
 // Создание удаленного RMI объекта  
 LeasingService service = new LeasingServiceImpl();  
 // Определение имени удаленного RMI объекта  
 String serviceName = "LeasingService";  
 System.*out*.println("Initializing " + serviceName);  
 Registry registry = LocateRegistry.*createRegistry*(1099);  
 registry.rebind(serviceName, service);  
 System.*out*.println("Start " + serviceName);  
 } catch (RemoteException e) {  
 System.*err*.println("RemoteException : "+e.getMessage());  
 System.*exit*(1);  
 } catch (Exception e) {  
 System.*err*.println("Exception : " + e.getMessage());  
 System.*exit*(2);  
 }  
 }  
}

**Файл Car.java**

import java.io.Serializable;  
public class Car implements Serializable  
{  
 private static final long *serialVersionUID* = 1L;  
 private String car;  
 private String driver;  
 private double money ;  
 private double leasing\_term ;  
 public Car(String car, String driver, double money, double leasing\_term)  
 {  
 super();  
 this.car = car;  
 this.driver = driver;  
 this.money = money;  
 this.leasing\_term = leasing\_term;  
 }  
 public String getDriver() { return driver; }  
 public String getCar() { return car; }  
 public double getMoney() { return money; }  
 public double getLeasing\_term() { return leasing\_term;}  
 public void setMoney(double money) { this.money = money; }  
   
 @Override  
 public boolean equals(Object car) {  
 Car c = (Car)car;  
 return this.getDriver().equals(c.getDriver());  
 }  
}

**Файл LeasingClient.java**

import java.net.MalformedURLException;

import java.rmi.\*;

public class LeasingClient

{

String localhost = "127.0.0.1";

String RMI\_HOSTNAME = "java.rmi.server.hostname";

String SERVICE\_PATH = "rmi://localhost/LeasingService";

String[][] CARS = {{"Toyota", "Dean Winchester"},

{"Mazda", "Robert Singer"}};

double[] MONEYS = {200, 1000};

public LeasingClient()

{

try {

System.*setProperty*(RMI\_HOSTNAME, localhost);

// URL удаленного объекта

String objectName = SERVICE\_PATH;

LeasingService ls;

ls = (LeasingService) Naming.*lookup*(objectName);

System.*out*.println("\nRegister cars ...");

registerCards(ls);

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

addMoney(ls);

System.*out*.println("Get leasing term ...\n");

getLeasingPeriod(ls);

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (RemoteException e) {

e.printStackTrace();

} catch (NotBoundException e) {

System.*err*.println("NotBoundException : " +

e.getMessage());

}

}

private Car createCar(final int idx)

{

return new Car(CARS[idx][0], CARS[idx][1], 100, 1);

}

private void registerCards(LeasingService ls)

{

for (int i = 0; i < CARS.length; i++) {

Car car = createCar(i);

try {

ls.addNewCar(car);

} catch (RemoteException e) {

System.*err*.println("RemoteException : " +

e.getMessage());

}

}

}

private void addMoney(LeasingService ls)

{

for (int i = 0; i < CARS.length; i++) {

Car car = createCar(i);

try {

ls.leasePayment(car, MONEYS[i]);

} catch (RemoteException e) {

System.*err*.println("RemoteException : " +

e.getMessage());

}

}

}

private void getLeasingPeriod(LeasingService ls)

{

for (int i = 0; i < CARS.length; i++) {

Car card = createCar(i);

try {

System.*out*.println("car : " +

card.getCar() +

", leasing term = " +

ls.getLeasingPeriod(card));

} catch (RemoteException e) {

System.*err*.println("RemoteException : " +

e.getMessage());

}

}

}

public static void main(String[] args)

{

new LeasingClient();

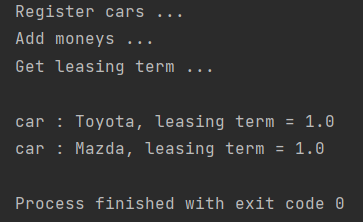
System.*exit*(0);

}

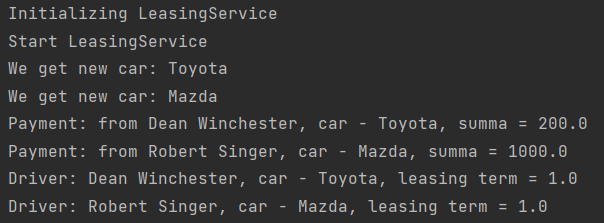
}

**Результати роботи програми**

Повідомлення клієнтського додатку.



Повідомлення серверного додатку.



Можна побачити що програма працює коректно, та виконує поставлені завдання.

**Висновок**

Під час виконання лабораторної роботи було набуто практичні навички роботи з RMI технологіями, та реалізовано клієнт-серверний додаток, що демонструє взаємодію віддалених об’єктів з використанням технології JAVA-RMI.