



**Πανεπιστήμιο Δυτικής Αττικής
Σχολή Μηχανικών
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών**

**ΕΡΓΑΣΤΗΡΙΟ ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ
ΦΙΛΙΠΠΟΣ ΠΑΠΑΓΕΩΡΓΙΟΥ - 21390174
Άσκηση #2 - Ημ. Παράδοσης: 9/6/2024**

ΟΜΑΔΑ ΕΡΓΑΣΤΗΡΙΟΥ:

ΣΤ4-Α (Τετάρτη 4-6)

Υπευθύνος Ομάδας:

ΙΟΡΔΑΝΑΚΗΣ ΜΙΧΑΛΗΣ

Πίνακας Περιεχομένων

Εισαγωγή2

Υλοποίησης και Προβλήματα	4
Σχολιασμός κώδικας.....	7
Ενδεικτικά Παραδείγματα Εκτέλεσης	21

Εισαγωγή

Σύντομη Επεξήγηση της Εργασίας

Αυτή η εργασία επικεντρώνεται στην ανάπτυξη μιας εφαρμογής client-server με χρήση Java RMI, η οποία επιτρέπει την ταυτόχρονη πρόσβαση πολλών πελατών (clients) για την κράτηση δωματίων σε ένα ξενοδοχείο. Το σύστημα περιλαμβάνει έναν server (HRServer), έναν client (HRCClient), ένα remote interface (HRInterface), και την υλοποίηση του interface αυτού (HRImpl).

Στόχοι

1. Ανάπτυξη βασικής client-server εφαρμογής:

- Δυνατότητα πολλαπλών ταυτόχρονων πελατών.
- Κρατήσεις δωματίων συγκεκριμένου τύπου χωρίς καθορισμό ημερομηνιών.
- Υποστήριξη διαφορετικών τύπων δωματίων με αντίστοιχες τιμές.

2. Διαδικασίες που Υποστηρίζονται:

- Λίστα διαθέσιμων δωματίων.
- Κράτηση δωματίων.
- Λίστα πελατών και κρατήσεων.
- Ακύρωση κρατήσεων.

3. Επέκταση της εφαρμογής:

- Εγγραφή πελατών για ειδοποίηση όταν ακυρωθεί κράτηση δωματίου που επιθυμούσαν.
- Αποστολή ειδοποιήσεων από τον server στους εγγεγραμμένους πελάτες κατά την ακύρωση κρατήσεων.

Υλοποίησης και Προβλήματα

Υλοποίηση

Η εργασία συντάχθηκε και υλοποιήθηκε επιτυχώς σε ένα περιβάλλον εικονικής μηχανής με λειτουργικό σύστημα Ubuntu 16.04 **Λόγο αυτού δεν μπορούσα να κάνω σχολία στα ελληνικά και ο κωδικός θα σχολιάθει πλήρως στο Word.** Η ανάπτυξη περιλάμβανε τη δημιουργία των ακόλουθων βασικών συστατικών:

1. HRServer: Ο server που διαχειρίζεται τις κρατήσεις δωματίων και τις επικοινωνίες με τους clients.
2. HRClient: Ο client που επιτρέπει στους χρήστες να κάνουν κρατήσεις, να ακυρώνουν κρατήσεις, να βλέπουν διαθέσιμα δωμάτια και να καταχωρούνται για ειδοποιήσεις.
3. HRInterface: Το απομακρυσμένο interface που δηλώνει τις μεθόδους που μπορούν να καλέσουν οι clients.
4. HRImpl: Η υλοποίηση του HRInterface που περιλαμβάνει τη λογική των κρατήσεων και των ακυρώσεων.
5. HRListener: για την διαχείριση πολλαπλών clients με την υλοποίηση Runnable.

Κατά την εκτέλεση της εργασίας, τα εξής λειτουργικά μέρη ολοκληρώθηκαν:

Καταγραφή και Διαχείριση Διαθέσιμων Δωματίων Ο server διατηρεί μια λίστα με τα διαθέσιμα δωμάτια για κάθε τύπο.

Κρατήσεις Δωματίων: Οι clients μπορούν να κάνουν κρατήσεις δωματίων μέσω απομακρυσμένων κλήσεων.

Ακύρωση Κρατήσεων: Οι clients μπορούν να ακυρώνουν κρατήσεις και να βλέπουν το ενημερωμένο σύνολο κρατήσεων.

Εμφανίσει εγγεγραμμένων clients με κρατήσεις και λεπτομέρειες των κρατησεών.

Οι βασικές εντολές για την εκτέλεση των λειτουργιών της εφαρμογής ήταν:

- `java HRClient list <hostname>`
- `java HRClient book <hostname> <type> <number> <name>`
- `java HRClient guests <hostname>`
- `java HRClient cancel <hostname> <type> <number> <name>`

Προβλήματα

Παρά την επιτυχή υλοποίηση του πρώτου μέρους της εργασίας, η επέκταση στο δεύτερο Β μέρος δεν κατέστη δυνατή λόγω μη επιλύσιμων σφαλμάτων. Οι κύριες προκλήσεις που αντιμετωπίστηκαν περιλαμβάνουν:

1. Σφάλματα Σύνταξης και Εκτέλεσης

- Αν και ο κώδικας συντάχθηκε (compiled) χωρίς σφάλματα, κατά την εκτέλεση εμφανίστηκαν runtime errors τα οποία δεν μπόρεσαν να επιλυθούν πλήρως.
- Τα σφάλματα αφορούσαν κυρίως τη διαχείριση ταυτόχρονων προσβάσεων και την ενημέρωση των clients μέσω των λιστών ειδοποίησης.

2. Διαχείριση Ειδοποιήσεων:

- Η λογική για την εγγραφή των clients στις λίστες ειδοποίησης και η αποστολή ειδοποιήσεων κατά την ακύρωση κρατήσεων προκάλεσε προβλήματα συγχρονισμού και ασυνεπείς καταστάσεις δεδομένων.

Παρά τις προαναφερθείσες δυσκολίες, η εργασία παρέχει μια λειτουργική βάση για την υλοποίηση μιας εφαρμογής κρατήσεων δωματίων με χρήση Java RMI και επιτυγχάνει τους βασικούς στόχους του πρώτου μέρους. Για την πλήρη επίλυση των προβλημάτων και την ολοκλήρωση του δεύτερου μέρους, απαιτείται περαιτέρω ανάλυση και δοκιμή του κώδικα, καθώς και πιθανές τροποποιήσεις στις δομές δεδομένων και τις μεθόδους συγχρονισμού.

Σχολίασμος κώδικας

HRInterface:

Το HRInterface καθορίζει τις μεθόδους που μπορούν να κληθούν απομακρυσμένα από τους clients μέσω RMI. Οι κύριες μέθοδοι είναι:

- `checkAvailability()`: Επιστρέφει τη διαθεσιμότητα των δωματίων στο ξενοδοχείο.
- `addListener()` και `removeListener()`: Προσθέτουν ή αφαιρούν listeners για να προσθέτω νέους clients ώστε να διαχειρίζομαι με ειδοποιήσεις.
- `makeReservation()`: Κάνει κράτηση δωματίων για έναν πελάτη.
- `getGuests()`: Επιστρέφει τη λίστα των πελατών και τις κρατήσεις τους.
- `CanselRes()`: Ακυρώνει κρατήσεις δωματίων για έναν πελάτη.

Στους παράμετρους θα δούμε `HRLListener` που αυτό αφορά τον συγκεκριμένο client που κάλεσε την μέθοδο και τον χρησιμοποιώ για να γύρισω notifications σε αυτόν μόνο.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HRInterface extends Remote {
    String checkAvailability() throws RemoteException;
    void addListener(HRLListener listener) throws RemoteException;
    void removeListener(HRLListener listener) throws RemoteException;
    void makeReservation(String roomType, int numberOfRooms, String
clientName,HRLListener client) throws RemoteException;
    void getGuests(HRLListener client) throws RemoteException;
    boolean CanselRes(HRLListener client ,String roomType ,int
numberOfRooms , String clientName) throws RemoteException;
}
```

HRListener

Το HRListener είναι ένα απομακρυσμένο interface που επιτρέπει στον server να στέλνει ειδοποιήσεις στους clients. Οι μέθοδοι του είναι:

- `notifyUpdate(String message)`: Στέλνει ένα μήνυμα ενημέρωσης στον client. Αυτή η μέθοδος χρησιμοποιείται για να ενημερώνει τον client για γεγονότα όπως ακυρώσεις κρατήσεων που μπορεί να τον ενδιαφέρουν.
- `getClientName()`: Επιστρέφει το όνομα του πελάτη. Αυτή η μέθοδος επιτρέπει στον server να γνωρίζει το όνομα του πελάτη που συνδέεται με τον listener.

Αυτό το interface είναι σημαντικό για την υποστήριξη της λειτουργίας ειδοποίησης, όπου οι clients μπορούν να ενημερώνονται σε πραγματικό χρόνο για τις αλλαγές που τους αφορούν.

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface HRListener extends Remote {  
    void notifyUpdate(String message) throws RemoteException;  
    String getClientName() throws RemoteException;  
}
```

HRServer

Η κλάση HRServer είναι υπεύθυνη για την εκκίνηση του server που θα παρέχει τις υπηρεσίες κρατήσεων δωματίων μέσω Java RMI. Ο κώδικας περιλαμβάνει τα εξής βήματα:

. Δημιουργία Αντικειμένου HRImpl:

Δημιουργείται ένα αντικείμενο της κλάσης `HRImpl` που υλοποιεί τις μεθόδους του απομακρυσμένου interface `HRInterface`.

. Εκκίνηση του RMI Registry:

Το `LocateRegistry.createRegistry(1099)` δημιουργεί και εκκινεί ένα RMI registry στην προκαθορισμένη θύρα 1099.

Bind του Αντικειμένου στο Registry:

- Το `registry.rebind("HotelReservation", obj)` καταχωρεί το αντικείμενο `HRImpl` στο registry με το όνομα "HotelReservation". Αυτό επιτρέπει στους clients να αναζητήσουν και να καλέσουν τις απομακρυσμένες μεθόδους αυτού του αντικειμένου.

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class HRServer {
    public static void main(String[] args){
        try {
            HRImpl obj = new HRImpl();
            Registry registry =
LocateRegistry.createRegistry(1099);
            registry.rebind("HotelReservation", obj);
            System.out.println("HRServer is ready.");
        } catch (Exception e) {
            System.err.println("HRServer exception: " +
e.toString());
            e.printStackTrace();
        }
    }
}
```

HRImpl

κλάση **HRImpl** υλοποιεί το απομακρυσμένο interface **HRInterface** και περιλαμβάνει την λογική διαχείρισης των κρατήσεων δωματίων.

- Reservation κλάση υλοποιεί μια κράτηση δωματίων από έναν πελάτη
- Η κλάση HRImpl υλοποιεί το απομακρυσμένο interface HRInterface και διαχειρίζεται τις κρατήσεις δωματίων του ξενοδοχείου επίσης με την runnable διαχειρίζεται πολλαπλούς clients και την μέθοδο @Override run()
- Στην συνέχεια δήλωνουμε τις δομές δεδομενων

```
private Map<String, Integer> roomInventory; :
```

-Η οποία αποθηκευει τον τύπο δωματίου και την διαθεσιμότητατους.

Με ένα set κλειδιών δηλαδή μας <<ενωνεί Π.χ το “Α” με το 40>>

```
private Map<String, Integer> roomPrices;
```

-Η οποία αποθηκευει τον τύπο δωματίου και την τιμη του.
παρομοίως

```
private List<HRListener> listeners;
```

-ArrayList με τύπο HRListener για να διαχειριζομαστε να προσθετούμε και να αφαιρούμε clients

```
private Vector<Reservation> reservations;
```

-Vector με τύπο Reservation για να αποθέκουμε όλα τα bookings και να μπορούμε να τις εμφανίσουμε στην μέθοδο guests.

- **checkAvailability()**
Αυτή η μέθοδος επιστρέφει την τρέχουσα διαθεσιμότητα των δωματίων, συμπεριλαμβανομένου του τύπου δωματίου, του αριθμού διαθέσιμων δωματίων και της τιμής ανά βραδιά.
- **makeReservation()**
Αυτή η μέθοδος κάνει κράτηση δωματίων για έναν πελάτη, ενημερώνει τον πελάτη για το αποτέλεσμα και τον εγγράφει στη λίστα ειδοποιήσεων για ακυρώσεις.
Ελέγχει τη διαθεσιμότητα του ζητούμενου τύπου δωματίου, ενημερώνει την αποθήκη δωματίων και άνολγα αν έχει διαθέσιμα δώματα ή όχι ενημερώνει αναλογα τον client.

- `getGuests()`
Επιστρέφει μια λίστα με τους πελάτες και τις κρατήσεις τους.
Δημιουργεί μια συμβολοσειρά που περιέχει πληροφορίες για όλες τις κρατήσεις και την στέλνει στον πελάτη μέσω της μεθόδου `notifyUpdate` του `client`
- `CanselRes()`
Δημιουργώ μια για τις κρατήσεις που θα παραμείνουν μετά την ακύρωση μετά προσπερνάω όλες τις κρατησεις και ελεγχούμε ποια ταιργιαζει με τον πελάτη άμα ταιργιαξούν υπολογιζούμε τον αριθμό δωματιων που θα ακύρουν , ενημερώνουμε το `roomInventory`
Εάν η κράτηση έχει περισσότερα δωμάτια από αυτά που ακυρώθηκαν, προσθήκη της υπόλοιπης κράτησης στη λίστα Τέλος ελέγχω αν ακύρωθηκαν δωαμτία , ενημερωνω τις λίστα κρατήσεων και γύρνω τα ανάλογα μήνηματα.

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.util.HashMap;
import java.util.Map;
import java.util.List;
import java.util.ArrayList;
import java.util.Vector;

class Reservation
private String clientame;
    private String roomType;
    private int numberOfRooms;
    private int totalCost;

    public Reservation(String clientName, String roomType, int
numberOfRooms, int totalCost){
        this.clientName = clientName;
        this.roomType = roomType;
        this.numberOfRooms = numberOfRooms;
        this.totalCost = totalCost;
    }
}
```

```

    public String getClientName() {
        return clientName;
    }

    public String getRoomType() {
        return roomType;
    }

    public int getNumberOfRooms() {
        return numberOfRooms;
    }

    public int getTotalCost() {
        return totalCost;
    }

    @Override
    public String toString() {
        return "Client: " + clientName + ", Room Type: " + roomType +
            ", Number of Rooms: " + numberOfRooms + ", Total Cost: " + totalCost +
            " Euro\n";
    }
}

```

```

public class HRImp1 extends UnicastRemoteObject implements
    HRIInterface, Runnable{

```

```

    private Map<String, Integer> roomInventory;
    private Map<String, Integer> roomPrices;
    private List<HRIListener> listeners;
    private Vector<Reservation> reservations;

```

```

    protected HRImp1() throws RemoteException {
        super();
        roomInventory = new HashMap<>();
        roomInventory.put("A", 40);
        roomInventory.put("B", 35);
        roomInventory.put("C", 25);
        roomInventory.put("D", 30);
        roomInventory.put("E", 20);

        roomPrices = new HashMap<>();
        roomPrices.put("A", 75);
        roomPrices.put("B", 110);
    }
}

```

```

        roomPrices.put("C", 120);
        roomPrices.put("D", 150);
        roomPrices.put("E", 200);

        listeners = new ArrayList<>();
        reservations = new Vector<>();
    }

    @Override
    public synchronized String checkAvailability() throws
RemoteException{
        StringBuilder availability = new StringBuilder();
        for (Map.Entry<String, Integer> entry :
roomInventory.entrySet()) {
            String roomType = entry.getKey();
            int availableRooms = entry.getValue();
            int price = roomPrices.get(roomType);
            availability.append("Room type ").append(roomType).append(":
")
                .append(availableRooms).append(" rooms available -
price: ")
                .append(price).append(" Euro per night\n");
        }
        return availability.toString();
    }

    public synchronized void makeReservation(String roomType, int
numberOfRooms, String clientName, HRLListener client) throws
RemoteException{
        if (roomInventory.containsKey(roomType)){
            int availableRooms = roomInventory.get(roomType);
            int reservedRooms = Math.min(availableRooms,
numberOfRooms);
            roomInventory.put(roomType, availableRooms -
reservedRooms);
            int pricePerRoom = roomPrices.get(roomType);
            int totalCost = reservedRooms * pricePerRoom;

            String message;
            if (reservedRooms == numberOfRooms) {
                message = "Reservation successful for " +
reservedRooms + " rooms of type " + roomType + " in the name of " +
clientName + ". Total cost: " + totalCost + " Euro.";
            } else {

```

```

        message = "Only " + reservedRooms + " rooms of type "
+ roomType + " could be reserved in the name of " + clientName + ".
Total cost: " + totalCost + " Euro.";
    }
    reservations.add(new Reservation(clientName, roomType,
reservedRooms, totalCost));
    notificationLists.get(roomType).add(client);
    client.notifyUpdate(message);
    client.notifyUpdate("Subscribed to notifications for room
type " + roomType);
    }else{
        String message ="Room type " + roomType + " failed to Book
Reservation\n";
        client.notifyUpdate(message);
    }
}

```

```

@Override
public void getGuests(HRListener client) throws RemoteException{
    StringBuilder guestsList = new StringBuilder();
    if (reservations.isEmpty()) {
        String message = "No guests have made reservations yet.";
        client.notifyUpdate(message);
    } else {
        int count = 1;
        for (Reservation reservation : reservations) {
            guestsList.append(count).append(".
").append(reservation.toString()).append("\n");
            count++;
        }
        client.notifyUpdate(guestsList.toString());
    }
}

```

```

@Override
public boolean CanselRes(HRListener client ,String roomType ,int
numberOfRooms , String clientName) throws RemoteException
{
    List<Reservation> remainingReservations = new ArrayList<>();
    int roomsCancelled = 0;
    for (Reservation reservation : reservations){
        if (reservation.getClientName().equals(clientName) &&
reservation.getRoomType().equals(roomType) && roomsCancelled <
numberOfRooms)
        {

```

```

        int cancelCount = Math.min(numberOfRooms -
roomsCancelled, reservation.getNumberOfRooms());
        roomsCancelled += cancelCount;
        roomInventory.put(roomType,
roomInventory.get(roomType) + cancelCount);

        if (reservation.getNumberOfRooms() > cancelCount) {
            remainingReservations.add(new Reservation(clientName,
roomType, reservation.getNumberOfRooms() - cancelCount,
(reservation.getNumberOfRooms() - cancelCount) *
roomPrices.get(roomType)));
        }
    }else{
        remainingReservations.add(reservation);
    }
}

if(roomsCancelled > 0){

    reservations.clear();
    reservations.addAll(remainingReservations);
    StringBuilder result = new StringBuilder("Cancelled " +
roomsCancelled + " rooms of type " + roomType + " for " + clientName +
".\nRemaining reservations:\n");
    int count = 1;
    for (Reservation reservation : remainingReservations) {
        if (reservation.getClientName().equals(clientName)) {

result.append(reservation.toString()).append("\n");
        }
    }
    return true;
}else{
    client.notifyUpdate("No matching reservations found for
cancellation.");
}
return false;
}

@Override
public void addListener(HRListener listener) throws
RemoteException {
    if (listeners == null) {

```

```

        listeners = new ArrayList<>();
    }
    listeners.add(listener);
    System.out.println("Client Connected: " +
listener.getClientName());
    }

@Override
    public void removeListener(HRListener listener) throws
RemoteException {
        if (listeners != null) {
            listeners.remove(listener);
        }
        System.out.println("Client disconnected: " +
listener.getClientName());
    }

@Override
    public void run() {

    }

}

```

HRClient

Η κλάση HRClient υλοποιεί το interface HRListener και περιέχει τον κώδικα που επιτρέπει την επικοινωνία με τον server μέσω RMI. Περιλαμβάνει τις ακόλουθες βασικές μεθόδους και λειτουργίες:

1. notifyUpdate(String message)

Λαμβάνει ειδοποιήσεις από τον server και τις εκτυπώνει.

Εκτυπώνει το μήνυμα ειδοποίησης που στέλνει ο server.

2. getClientName()

Επιστρέφει το όνομα του πελάτη.

Επιστρέφει την τιμή της μεταβλητής `clientName`.

3.main(String[] args)

Διαχειρίζεται τις εισόδους από τη γραμμή εντολών και εκτελεί τις κατάλληλες λειτουργίες ανάλογα με τις παραμέτρους που δίνονται. Ελέγχει αν δόθηκαν παράμετροι. Αν όχι, εκτυπώνει τον τρόπο χρήσης και επιστρέφει.

- Διαχειρίζεται την εντολή `list` για να εμφανίσει τη διαθεσιμότητα δωματίων.
- Διαχειρίζεται την εντολή `book` για να κάνει κράτηση δωματίων. Διαχειρίζεται την εντολή `guests` για να εμφανίσει τη λίστα πελατών και των κρατήσεών τους.
- Διαχειρίζεται την εντολή `cancel` για να ακυρώσει κρατήσεις δωματίων.

Διαχείριση Εντολών στη main

- list:

- Συνδέεται με το RMI registry, δημιουργεί ένα αντικείμενο `HRClient`, προσθέτει τον client ως listener και καλεί τη μέθοδο `checkAvailability` του server για να λάβει και να εμφανίσει τη διαθεσιμότητα των δωματίων.

- book:

- Συνδέεται με το RMI registry, δημιουργεί ένα αντικείμενο `HRClient`, προσθέτει τον client ως listener και καλεί τη μέθοδο `makeReservation` του server για να κάνει κράτηση δωματίων. Εμφανίζει το αποτέλεσμα της κράτησης.

- guests:

- Συνδέεται με το RMI registry, δημιουργεί ένα αντικείμενο `HRClient`, προσθέτει τον client ως listener και καλεί τη μέθοδο `getGuests` του server για να λάβει και να εμφανίσει τη λίστα πελατών και των κρατήσεών τους.

- cancel:

Συνδέεται με το RMI registry, δημιουργεί ένα αντικείμενο `HRClient`, προσθέτει τον client ως listener και καλεί τη μέθοδο `CancelRes` του server για να ακυρώσει κρατήσεις δωματίων. Εάν η ακύρωση ήταν επιτυχής, καλεί τη μέθοδο `notifyCancelListeners` για να ειδοποιήσει τους εγγεγραμμένους πελάτες.

Τέλος έχω φτιαξεί σε περίπτωση ξεχωρίστα να κάνει καινούργιο client και όνομα και όταν χρησιμοποιήσει ο client την υπηρεσία περιμένει να πατήσει οτιδήποτε και μετά τον αποσυνδεεί από το server.

```
import java.rmi.registry.LocateRegistry;  
import java.rmi.registry.Registry;  
import java.rmi.RemoteException;  
import java.rmi.server.UnicastRemoteObject;  
import java.util.Scanner;
```

```

public class HRClient extends UnicastRemoteObject implements
HRListener {

    private String clientName;

    protected HRClient(String clientName) throws RemoteException {
        super();
        this.clientName = clientName;
    }

    @Override
    public void notifyUpdate(String message) throws RemoteException {
        System.out.println("Notification:\n" + message);
    }

    @Override
    public String getClientName() throws RemoteException {
        return clientName;
    }

    public static void main(String[] args){

        Scanner scanner = new Scanner(System.in);

        try{
            if (args.length == 0) {
                printUsage();
                return;
            }

            if (args.length == 2 && args[0].equals("list")) {
                Registry registry =
LocateRegistry.getRegistry("localhost");
                HRInterface stub = (HRInterface)
registry.lookup("HotelReservation");
                HRClient client = new HRClient(args[1]);
                stub.addListener(client);

                String availability = stub.checkAvailability();
                System.out.println(availability);
            }
        }
    }
}

```

```

        scanner.nextLine();
        stub.removeListener(client);
        return;
    }
    if (args.length == 5 && args[0].equals("book")) {
        Registry registry =
LocateRegistry.getRegistry("localhost");
        HRInterface stub = (HRInterface)
registry.lookup("HotelReservation");
        HRClient client = new HRClient(args[1]);
        stub.addListener(client);
        stub.makeReservation(args[2],
Integer.parseInt(args[3]),args[4],client);

        scanner.nextLine();
        stub.removeListener(client);
        return;
    }
    if (args.length == 2 && args[0].equals("guests")) {
        Registry registry =
LocateRegistry.getRegistry("localhost");
        HRInterface stub = (HRInterface)
registry.lookup("HotelReservation");
        HRClient client = new HRClient(args[1]);
        stub.addListener(client);
        stub.getGuests(client);

        scanner.nextLine();
        stub.removeListener(client);
        return;
    }
    if (args.length == 5 && args[0].equals("cancel")) {
        Registry registry =
LocateRegistry.getRegistry("localhost");
        HRInterface stub = (HRInterface)
registry.lookup("HotelReservation");
        HRClient client = new HRClient(args[1]);
        stub.addListener(client);
        boolean flag =
stub.CanselRes(client,args[2],Integer.parseInt(args[3]),args[4]);

        if(flag)

```

```

        {
            stub.notifyCancelListeneres(args[2]);
        }

        scanner.nextLine();
        stub.removeListener(client);
        return;
    }

    }catch (Exception e) {
        System.err.println("HRClient exception: " + e.toString());
        e.printStackTrace();
    }
}

private static void printUsage() {
    System.out.println("Usage:");
    System.out.println("java HRClient");
    System.out.println("java HRClient list <hostname>");
    System.out.println("java HRClient book <hostname> <type>
<number> <name>");
    System.out.println("java HRClient cancel <hostname>
<type> <number> <name>");
    System.out.println("java HRClient guests
<hostname>");
}
}
}

```

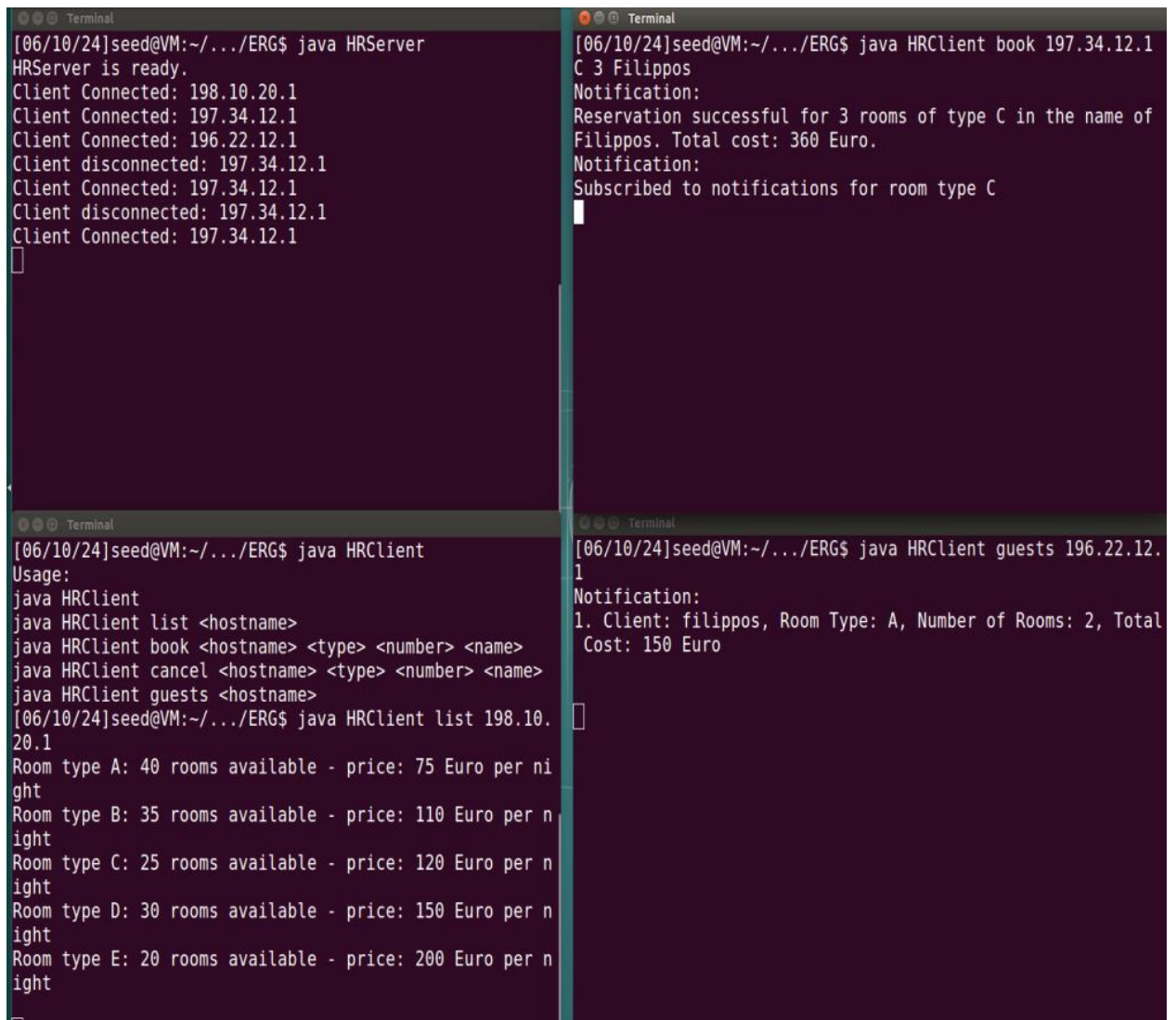
Ενδεικτικά Παραδείγματα Εκτέλεσης

Παρακάτω παρουσιάζονται κάποια παραδείγματα εκτέλεσης της εφαρμογής HRClient με τις διάφορες εντολές. Τα παραδείγματα περιλαμβάνουν τις εντολές `list`, `book`, `guests`, και `cancel`.

Για τα Ενδεικτικά παραδειγματα χρησιμοποιουνται ψευτικές IP και hostname στην ούσια όλοι είναι localhost

Παρατηρήσεις:

Στο αριστερό παράθυρο πάνω τρέχει ο server και εμφανίζει όλους τους Listeneres που προσθέτει και αφαιρεί καθώς στα άλλα , γίνονται οι διαφορές καλέσεις από τους clients προς τον server με τα διάφορα όρισματα



```
[06/10/24]seed@VM:~/.../ERG$ java HRServer
HRServer is ready.
Client Connected: 198.10.20.1
Client Connected: 197.34.12.1
Client Connected: 196.22.12.1
Client disconnected: 197.34.12.1
Client Connected: 197.34.12.1
Client disconnected: 197.34.12.1
Client Connected: 197.34.12.1

[06/10/24]seed@VM:~/.../ERG$ java HRClient book 197.34.12.1
C 3 Filippos
Notification:
Reservation successful for 3 rooms of type C in the name of
Filippos. Total cost: 360 Euro.
Notification:
Subscribed to notifications for room type C

[06/10/24]seed@VM:~/.../ERG$ java HRClient
Usage:
java HRClient
java HRClient list <hostname>
java HRClient book <hostname> <type> <number> <name>
java HRClient cancel <hostname> <type> <number> <name>
java HRClient guests <hostname>
[06/10/24]seed@VM:~/.../ERG$ java HRClient list 198.10.
20.1
Room type A: 40 rooms available - price: 75 Euro per ni
ght
Room type B: 35 rooms available - price: 110 Euro per n
ight
Room type C: 25 rooms available - price: 120 Euro per n
ight
Room type D: 30 rooms available - price: 150 Euro per n
ight
Room type E: 20 rooms available - price: 200 Euro per n
ight

[06/10/24]seed@VM:~/.../ERG$ java HRClient guests 196.22.12.
1
Notification:
1. Client: filippos, Room Type: A, Number of Rooms: 2, Total
Cost: 150 Euro
```

```
[06/10/24]seed@VM:~/.../ERG$ java HRServer
HRServer is ready.
Client Connected: 198.10.20.1
Client Connected: 197.34.12.1
Client Connected: 196.22.12.1
Client disconnected: 197.34.12.1
Client Connected: 197.34.12.1
Client disconnected: 197.34.12.1
Client Connected: 197.34.12.1
Client disconnected: 198.10.20.1
Client Connected: 193.23.11.3
Client disconnected: 193.23.11.3
Client Connected: 193.23.11.3
Client Connected: 196.22.12.1
Client disconnected: 193.23.11.3
Client Connected: 193.3.23.11.3
[]

[06/10/24]seed@VM:~/.../ERG$ java HRClient book 197.34.12.1
C 3 Filippos
Notification:
Reservation successful for 3 rooms of type C in the name of
Filippos. Total cost: 360 Euro.
Notification:
Subscribed to notifications for room type C
[]

[06/10/24]seed@VM:~/.../ERG$ java HRClient book 193.23.
11.3 A 10 Alex
Notification:
Reservation successful for 10 rooms of type A in the na
me of Alex. Total cost: 750 Euro.
Notification:
Subscribed to notifications for room type A

^C[06/10/24]seed@VM:~/.../ERG$ java HRClient book 193.2
11.3 C 8 SteliosPap
Notification:
Reservation successful for 8 rooms of type C in the nam
e of SteliosPap. Total cost: 960 Euro.
Notification:
Subscribed to notifications for room type C

^C[06/10/24]seed@VM:~/.../ERG$ java HRClient cancel 193
.23.11.3 A 5 Alex
Notification:
Cancelled 5 rooms of type A for Alex.
Remaining reservations:
Client: Alex, Room Type: A, Number of Rooms: 5, Total C
ost: 375 Euro

[06/10/24]seed@VM:~/.../ERG$ java HRClient book 197.34.12.1
C 3 Filippos
Notification:
Reservation successful for 3 rooms of type C in the name of
Filippos. Total cost: 360 Euro.
Notification:
Subscribed to notifications for room type C
[]

1
Notification:
1. Client: filippos, Room Type: A, Number of Rooms: 2, Total
Cost: 150 Euro

^C[06/10/24]seed@VM:~/.../ERG$
[06/10/24]seed@VM:~/.../ERG$ java HRClient guests 196.22.12.
1
Notification:
1. Client: filippos, Room Type: A, Number of Rooms: 1, Total
Cost: 75 Euro
2. Client: Filippos, Room Type: C, Number of Rooms: 3, Total
Cost: 360 Euro
3. Client: Alex, Room Type: A, Number of Rooms: 10, Total Co
st: 750 Euro
4. Client: SteliosPap, Room Type: C, Number of Rooms: 8, Tot
al Cost: 960 Euro
[]
```



```
[06/10/24]seed@VM:~/.../ERG$ java HRServer
```

```
HRServer is ready.
```

```
Client Connected: 198.10.20.1
```

```
Client Connected: 197.34.12.1
```

```
Client Connected: 196.22.12.1
```

```
Client disconnected: 197.34.12.1
```

```
Client Connected: 197.34.12.1
```

```
Client disconnected: 197.34.12.1
```

```
Client Connected: 197.34.12.1
```

```
Client disconnected: 198.10.20.1
```

```
Client Connected: 193.23.11.3
```

```
Client disconnected: 193.23.11.3
```

```
Client Connected: 193.23.11.3
```

```
Client Connected: 196.22.12.1
```

```
Client disconnected: 193.23.11.3
```

```
Client Connected: 193.3.23.11.3
```

```
Client Connected: 198.10.20.1
```

```
□
```

```
Terminal
```

```
[06/10/24]seed@VM:~/.../ERG$ java HRClient list 198.10.20.1
```

```
Room type A: 34 rooms available - price: 75 Euro per night
```

```
Room type B: 35 rooms available - price: 110 Euro per night
```

```
Room type C: 14 rooms available - price: 120 Euro per night
```

```
Room type D: 30 rooms available - price: 150 Euro per night
```

```
Room type E: 20 rooms available - price: 200 Euro per night
```