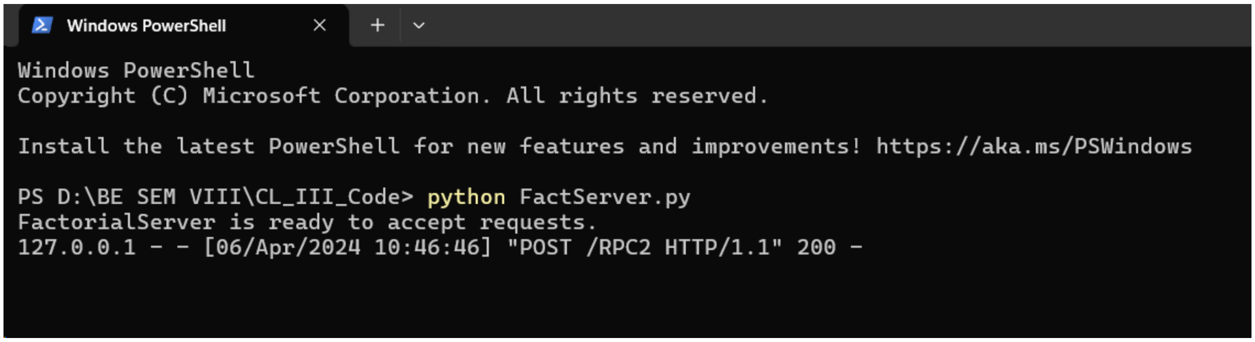
FactServer.py

```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler

class FactorialServer:
    def calculate_factorial(self, n):
        if n < 0:
            raise ValueError("Input must be a non-negative integer.")
        result = 1

        for i in range(1, n + 1):
            result *= i
        return result

# Restrict to a particular path.
class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

# Create server
with SimpleXMLRPCServer(('localhost', 8000),
requestHandler=RequestHandler) as server:
    server.register_introspection_functions()
    # Register the FactorialServer class
    server.register_instance(FactorialServer())
    print("FactorialServer is ready to accept requests.")
    # Run the server's main loop
    server.serve_forever()
```

FactClient.py

```
import xmlrpc.client

# Create an XML-RPC client
with xmlrpc.client.ServerProxy("http://localhost:8000/RPC2") as
proxy:
    try:
        # Replace 5 with the desired integer value
        input_value = 5
        result = proxy.calculate_factorial(input_value)
        print(f"Factorial of {input_value} is: {result}")
    except Exception as e:
        print(f"Error: {e}")
```

Output:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\BE SEM VIII\CL_III_Code> python FactServer.py
FactorialServer is ready to accept requests.
127.0.0.1 - - [06/Apr/2024 10:46:46] "POST /RPC2 HTTP/1.1" 200 -
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\BE SEM VIII\CL_III_Code> python FactClient.py
Factorial of 5 is: 120
PS D:\BE SEM VIII\CL_III_Code>
```

Server.py

```
import Pyro4

@Pyro4.expose
class StringConcatenationServer:
    def concatenate_strings(self, str1, str2):
        result = str1 + str2
        return result

def main():
    daemon = Pyro4.Daemon()  # Create a Pyro daemon
    ns = Pyro4.locateNS()  # Locate the Pyro nameserver

    # Create an instance of the server class
    server = StringConcatenationServer()

    # Register the server object with the Pyro nameserver
    uri = daemon.register(server)
    ns.register("string.concatenation", uri)

    print("Server URI:", uri)

    with open("server_uri.txt", "w") as f:
        f.write(str(uri))

    daemon.requestLoop()

if __name__ == "__main__":
    main()
```

Client.py

import Pyro4

```
def main():
    with open("server_uri.txt", "r") as f:
        uri = f.read()

    server = Pyro4.Proxy(uri)  # Connect to the remote server
    str1 = input("Enter the first string: ")
    str2 = input("Enter the second string: ")
    result = server.concatenate_strings(str1, str2)
    print("Concatenated Result:", result)
```

```
if __name__ == "__main__":
    main()
```

Output:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\BE SEM VIII\CL_III_Code> python Server.py
Server URI: PYRO:obj_f8b9737cdb2d45aa8e53a4664599663f@localhost:64475
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\BE SEM VIII\CL_III_Code> python Client.py
Enter the first string: dy
Enter the second string: patil
Concatenated Result: dypatil
PS D:\BE SEM VIII\CL_III_Code>
```

Code:

```
import numpy as np

# Function to perform Union operation on fuzzy sets
def fuzzy_union(A, B):
    return np.maximum(A, B)

# Function to perform Intersection operation on fuzzy sets
def fuzzy_intersection(A, B):
    return np.minimum(A, B)

# Function to perform Complement operation on a fuzzy set
def fuzzy_complement(A):
    return 1 - A

# Function to perform Difference operation on fuzzy sets
def fuzzy_difference(A, B):
    return np.maximum(A, 1 - B)

# Function to create fuzzy relation by Cartesian product of two fuzzy
sets
def cartesian_product(A, B):
    return np.outer(A, B)

# Function to perform Max-Min composition on two fuzzy relations
def max_min_composition(R, S):
    return np.max(np.minimum.outer(R, S), axis=1)

# Example usage
A = np.array([0.2, 0.4, 0.6, 0.8]) # Fuzzy set A
B = np.array([0.3, 0.5, 0.7, 0.9]) # Fuzzy set B

# Operations on fuzzy sets
union_result = fuzzy_union(A, B)
intersection_result = fuzzy_intersection(A, B)
complement_A = fuzzy_complement(A)
difference_result = fuzzy_difference(A, B)

print("Union:", union_result)
print("Intersection:", intersection_result)
print("Complement of A:", complement_A)
print("Difference:", difference_result)

# Fuzzy relations
```

```
R = np.array([0.2, 0.5, 0.4]) # Fuzzy relation R
S = np.array([0.6, 0.3, 0.7]) # Fuzzy relation S

# Cartesian product of fuzzy relations
cartesian_result = cartesian_product(R, S)

# Max-Min composition of fuzzy relations
composition_result = max_min_composition(R, S)

print("Cartesian product of R and S:")
print(cartesian_result)

print("Max-Min composition of R and S:")
print(composition_result)
```

Output:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\Fuzzy.py"
Union: [0.3 0.5 0.7 0.9]
Intersection: [0.2 0.4 0.6 0.8]
Complement of A: [0.8 0.6 0.4 0.2]
Difference: [0.7 0.5 0.6 0.8]
Cartesian product of R and S:
[[0.12 0.06 0.14]
 [0.3  0.15 0.35]
 [0.24 0.12 0.28]]
Max-Min composition of R and S:
[0.2 0.5 0.4]
PS D:\BE SEM VIII>
```

Code:

```
import random

class LoadBalancer:
    def __init__(self, servers):
        self.servers = servers
        self.server_index_rr = 0

    def round_robin(self):
        server = self.servers[self.server_index_rr]
        self.server_index_rr = (self.server_index_rr + 1) %
len(self.servers)
        return server

    def random_selection(self):
        return random.choice(self.servers)

def simulate_client_requests(load_balancer, num_requests):
    for i in range(num_requests):
        # Simulating client request
        print(f"Request {i+1}: ", end="")

        # Using Round Robin algorithm for load balancing
        server_rr = load_balancer.round_robin()
        print(f"Round Robin - Server {server_rr}")

        # Using Random algorithm for load balancing
        server_random = load_balancer.random_selection()
        print(f"Random - Server {server_random}")
        print()

if __name__ == "__main__":
    # List of servers
    servers = ["Server A", "Server B", "Server C"]

    # Create a LoadBalancer instance
    load_balancer = LoadBalancer(servers)

    # Simulate 10 client requests
    simulate_client_requests(load_balancer, 10)
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    PORTS    DEBUG CONSOLE

PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\Loadbalancer.py"
Request 1: Round Robin - Server Server A
Random - Server Server C

Request 2: Round Robin - Server Server B
Random - Server Server C

Request 3: Round Robin - Server Server C
Random - Server Server C

Request 4: Round Robin - Server Server A
Random - Server Server B

Request 5: Round Robin - Server Server B
Random - Server Server B

Request 6: Round Robin - Server Server C
Random - Server Server B

Request 7: Round Robin - Server Server A
Random - Server Server A

Request 8: Round Robin - Server Server B
Random - Server Server B

Request 9: Round Robin - Server Server C
Random - Server Server A

Request 10: Round Robin - Server Server A
Random - Server Server A

PS D:\BE SEM VIII>
```

Code:

```
import random
from deap import base, creator, tools, algorithms

# Define evaluation function (this is a mock function, replace this
with your actual evaluation function)
def evaluate(individual):
    # Here 'individual' represents the parameters for the neural
network
    # You'll need to replace this with your actual evaluation
function that trains the neural network
    # and evaluates its performance
    # Return a fitness value (here, a random number is used as an
example)
    return random.random(),

# Define genetic algorithm parameters
POPULATION_SIZE = 10
GENERATIONS = 5

# Create types for fitness and individuals in the genetic algorithm
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

# Initialize toolbox
toolbox = base.Toolbox()

# Define attributes and individuals
toolbox.register("attr_neurons", random.randint, 1, 100)  # Example:
number of neurons
toolbox.register("attr_layers", random.randint, 1, 5)    # Example:
number of layers
toolbox.register("individual", tools.initCycle, creator.Individual,
(toolbox.attr_neurons, toolbox.attr_layers), n=1)
toolbox.register("population", tools.initRepeat, list,
toolbox.individual)

# Genetic operators
toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutUniformInt, low=1, up=100,
indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
```

```
# Create initial population
population = toolbox.population(n=POPULATION_SIZE)

# Run the genetic algorithm
for gen in range(GENERATIONS):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5,
mutpb=0.1)

    fitnesses = toolbox.map(toolbox.evaluate, offspring)
    for ind, fit in zip(offspring, fitnesses):
        ind.fitness.values = fit

    population = toolbox.select(offspring, k=len(population))

# Get the best individual from the final population
best_individual = tools.selBest(population, k=1)[0]
best_params = best_individual

# Print the best parameters found
print("Best Parameters:", best_params)
```

Output:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\Genetic.py"
Best Parameters: [82, 3]
PS D:\BE SEM VIII>
```

Code:

```
import numpy as np

# Generate dummy data for demonstration purposes (replace this with
your actual data)
def generate_dummy_data(samples=100, features=10):
    data = np.random.rand(samples, features)
    labels = np.random.randint(0, 2, size=samples)
    return data, labels

# Define the AIRS algorithm
class AIRS:
    def __init__(self, num_detectors=10, hypermutation_rate=0.1):
        self.num_detectors = num_detectors
        self.hypermutation_rate = hypermutation_rate

    def train(self, X, y):
        self.detectors = X[np.random.choice(len(X),
self.num_detectors, replace=False)]

    def predict(self, X):
        predictions = []
        for sample in X:
            distances = np.linalg.norm(self.detectors - sample,
axis=1)
            prediction = int(np.argmin(distances))
            predictions.append(prediction)
        return predictions

# Generate dummy data
data, labels = generate_dummy_data()

# Split data into training and testing sets
split_ratio = 0.8
split_index = int(split_ratio * len(data))
train_data, test_data = data[:split_index], data[split_index:]
train_labels, test_labels = labels[:split_index],
labels[split_index:]

# Initialize and train AIRS
airs = AIRS(num_detectors=10, hypermutation_rate=0.1)
airs.train(train_data, train_labels)

# Test AIRS on the test set
predictions = airs.predict(test_data)
```

```
# Evaluate accuracy
accuracy = np.mean(predictions == test_labels)
print(f"Accuracy: {accuracy}")
```

Output:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\Clonal.py"
Accuracy: 0.05
PS D:\BE SEM VIII>
```

Code:

```
import random
from deap import base, creator, tools, algorithms

# Define the evaluation function (minimize a simple mathematical
function)
def eval_func(individual):
    # Example evaluation function (minimize a quadratic function)
    return sum(x ** 2 for x in individual),

# DEAP setup
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()

# Define attributes and individuals
toolbox.register("attr_float", random.uniform, -5.0, 5.0)  # Example:
Float values between -5 and 5
toolbox.register("individual", tools.initRepeat, creator.Individual,
toolbox.attr_float, n=3)  # Example: 3-dimensional individual
toolbox.register("population", tools.initRepeat, list,
toolbox.individual)

# Evaluation function and genetic operators
toolbox.register("evaluate", eval_func)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1,
indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)

# Create population
population = toolbox.population(n=50)

# Genetic Algorithm parameters
generations = 20

# Run the algorithm
for gen in range(generations):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5,
mutpb=0.1)

    fits = toolbox.map(toolbox.evaluate, offspring)
    for fit, ind in zip(fits, offspring):
```

```
        ind.fitness.values = fit

    population = toolbox.select(offspring, k=len(population))

# Get the best individual after generations
best_ind = tools.selBest(population, k=1)[0]
best_fitness = best_ind.fitness.values[0]

print("Best individual:", best_ind)
print("Best fitness:", best_fitness)
```

Output:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\DEAP.py"
Best individual: [-0.011174776506688588, -0.0063488374813361935, -0.033035424484573764]
Best fitness: 0.0012565226382148342
PS D:\BE SEM VIII>
```

Code:

HotelClient.java

```
import java.rmi.Naming;
import java.util.Scanner;
public class HotelClient {
 public static void main(String[] args) {
 try {
 // Look up the RMI server object from the registry
 HotelServiceInterface hotelService = (HotelServiceInterface)
Naming.lookup("rmi://localhost/HotelService");
 Scanner scanner = new Scanner(System.in);
 while (true) {
 System.out.println("1. Book a room");
 System.out.println("2. Cancel booking");
 System.out.println("3. Exit");
 System.out.print("Enter your choice: ");
 int choice = scanner.nextInt();
 scanner.nextLine(); // consume the newline character
 switch (choice) {
 case 1:
 System.out.print("Enter guest name: ");
 String guestName = scanner.nextLine();
 System.out.print("Enter room number: ");
 int roomNumber = scanner.nextInt();
 boolean booked = hotelService.bookRoom(guestName, roomNumber);
 if (booked) {
 System.out.println("Room booked successfully!");
 } else {
 System.out.println("Room booking failed.");
 }
 break;
 case 2:
 System.out.print("Enter guest name for cancellation: ");
 String cancelGuestName = scanner.nextLine();
 boolean canceled = hotelService.cancelBooking(cancelGuestName);
 if (canceled) {
 System.out.println("Booking canceled successfully!");
 } else {
 System.out.println("Booking cancellation failed.");
 }
 break;
 case 3:
 System.out.println("Exiting the client application.");
```

```
 System.exit(0);
 break;
 default:
 System.out.println("Invalid choice. Please enter a valid option.");
 }
 }
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}
```

HotelServer.java

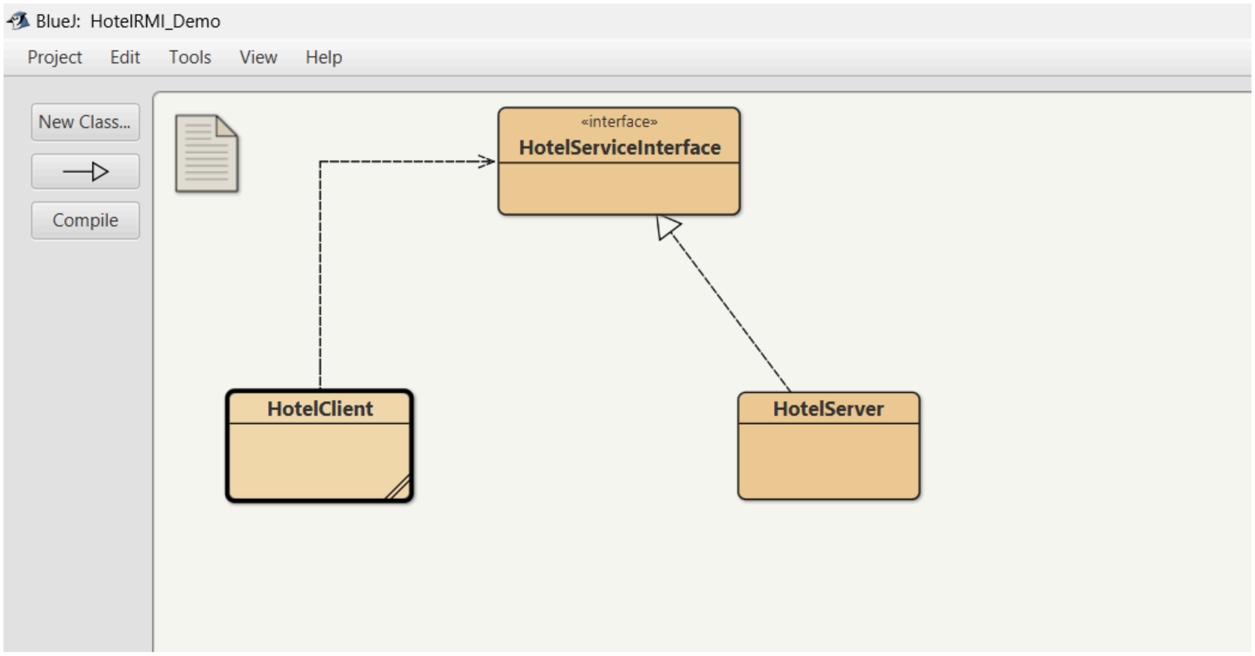
```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.HashMap;
import java.util.Map;
public class HotelServer extends UnicastRemoteObject implements
HotelServiceInterface {
 private Map<Integer, String> bookedRooms;
 public HotelServer() throws RemoteException {
 bookedRooms = new HashMap<>();
 }
 @Override
 public synchronized boolean bookRoom(String guestName, int
roomNumber)
throws RemoteException {
 if (!bookedRooms.containsKey(roomNumber)) {
 bookedRooms.put(roomNumber, guestName);
 System.out.println("Room " + roomNumber + " booked for guest: " +
guestName);
 return true;
 } else {
 System.out.println("Room " + roomNumber + " is already booked.");
 return false;
 }
 }
 @Override
 public synchronized boolean cancelBooking(String guestName) throws
RemoteException {
 for (Map.Entry<Integer, String> entry : bookedRooms.entrySet()) {
 if (entry.getValue().equals(guestName)) {
 bookedRooms.remove(entry.getKey());
```

```
 System.out.println("Booking for guest " + guestName + " canceled.");
 return true;
 }
 }
 System.out.println("No booking found for guest " + guestName);
 return false;
 }
 public static void main(String[] args) {
 try {
 HotelServer server = new HotelServer();
 // Create and export the RMI registry on port 1099
 java.rmi.registry.LocateRegistry.createRegistry(1099);
 // Bind the server object to the registry
 Naming.rebind("HotelService", server);
 System.out.println("Hotel Server is running...");
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}
```

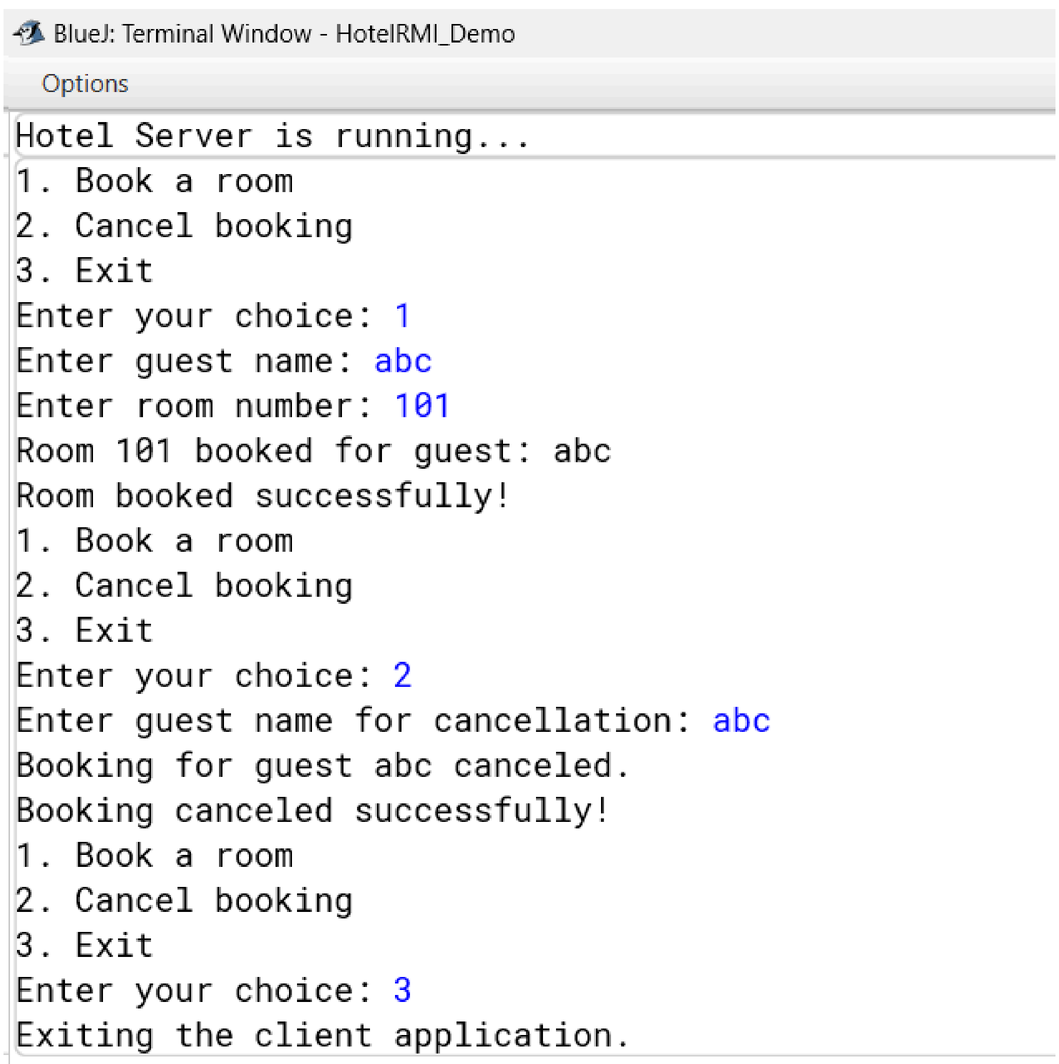
HotelServiceInterface.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface HotelServiceInterface extends Remote {
 boolean bookRoom(String guestName, int roomNumber) throws
RemoteException;
 boolean cancelBooking(String guestName) throws RemoteException;
}
```

Output:



BlueJ: HotelRMI_Demo
Project
Edit
Tools
View
Help
New Class...
Compile
«interface»
HotelServiceInterface
HotelClient
HotelServer



BlueJ: Terminal Window - HotelRMI_Demo
Options
Hotel Server is running...
1. Book a room
2. Cancel booking
3. Exit
Enter your choice: 1
Enter guest name: abc
Enter room number: 101
Room 101 booked for guest: abc
Room booked successfully!
1. Book a room
2. Cancel booking
3. Exit
Enter your choice: 2
Enter guest name for cancellation: abc
Booking for guest abc canceled.
Booking canceled successfully!
1. Book a room
2. Cancel booking
3. Exit
Enter your choice: 3
Exiting the client application.

Code:

```
import csv
from functools import reduce
from collections import defaultdict

# Define mapper function to emit (year, temperature) pairs
def mapper(row):
    year = row["Date/Time"].split("-")[0]  # Extract year from
"Date/Time" column
    temperature = float(row["Temp_C"])  # Convert temperature to
float
    return (year, temperature)

# Define reducer function to calculate sum and count of temperatures
for each year
def reducer(accumulated, current):
    accumulated[current[0]][0] += current[1]
    accumulated[current[0]][1] += 1
    return accumulated

# Read the weather dataset
weather_data = []
with open("weather_data.csv", "r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        weather_data.append(row)

# Map phase
mapped_data = map(mapper, weather_data)

# Reduce phase
reduced_data = reduce(reducer, mapped_data, defaultdict(lambda: [0,
0]))

# Calculate average temperature for each year
avg_temp_per_year = {year: total_temp / count for year, (total_temp,
count) in reduced_data.items()}

# Find coolest and hottest year
coolest_year = min(avg_temp_per_year.items(), key=lambda x: x[1])
hottest_year = max(avg_temp_per_year.items(), key=lambda x: x[1])

print("Coolest Year:", coolest_year[0], "Average Temperature:",
coolest_year[1])
```

```
print("Hottest Year:", hottest_year[0], "Average Temperature:",
hottest_year[1])
```

Output:

```
Coolest Year: 1/15/2012 8:00 Average Temperature: -23.3
Hottest Year: 6/21/2012 15:00 Average Temperature: 33.0
```

Code:

```
import numpy as np
import random

# Define the distance matrix (distances between cities)
# Replace this with your distance matrix or generate one based on
your problem
# Example distance matrix (replace this with your actual data)
distance_matrix = np.array([
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
])

# Parameters for Ant Colony Optimization
num_ants = 10
num_iterations = 50
evaporation_rate = 0.5
pheromone_constant = 1.0
heuristic_constant = 1.0

# Initialize pheromone matrix and visibility matrix
num_cities = len(distance_matrix)
pheromone = np.ones((num_cities, num_cities))  # Pheromone matrix
visibility = 1 / distance_matrix  # Visibility matrix (inverse of
distance)

# ACO algorithm
for iteration in range(num_iterations):
    ant_routes = []
    for ant in range(num_ants):
        current_city = random.randint(0, num_cities - 1)
        visited_cities = [current_city]
        route = [current_city]

        while len(visited_cities) < num_cities:
            probabilities = []
            for city in range(num_cities):
                if city not in visited_cities:
                    pheromone_value = pheromone[current_city][city]
                    visibility_value = visibility[current_city][city]
                    probability = (pheromone_value **
pheromone_constant) * (visibility_value ** heuristic_constant)
                    probabilities.append((city, probability))
```

```
                probabilities = sorted(probabilities, key=lambda x: x[1],
reverse=True)
                selected_city = probabilities[0][0]
                route.append(selected_city)
                visited_cities.append(selected_city)
                current_city = selected_city

            ant_routes.append(route)

        # Update pheromone levels
        delta_pheromone = np.zeros((num_cities, num_cities))

        for ant, route in enumerate(ant_routes):
            for i in range(len(route) - 1):
                city_a = route[i]
                city_b = route[i + 1]
                delta_pheromone[city_a][city_b] += 1 /
distance_matrix[city_a][city_b]
                delta_pheromone[city_b][city_a] += 1 /
distance_matrix[city_a][city_b]

        pheromone = (1 - evaporation_rate) * pheromone + delta_pheromone

# Find the best route
best_route_index =
np.argmax([sum(distance_matrix[cities[i]][cities[(i + 1) %
num_cities]] for i in range(num_cities)) for cities in ant_routes])
best_route = ant_routes[best_route_index]
shortest_distance = sum(distance_matrix[best_route[i]][best_route[(i
+ 1) % num_cities]] for i in range(num_cities))

print("Best route:", best_route)
print("Shortest distance:", shortest_distance)
```

Output:

```
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\TSP.py"
d:\BE SEM VIII\CL_III_Code\TSP.py:24: RuntimeWarning: divide by zero encountered in divide
  visibility = 1 / distance_matrix  # Visibility matrix (inverse of distance)
Best route: [0, 1, 3, 2]
Shortest distance: 80
PS D:\BE SEM VIII>
```