Middle East Technical University Department of Computer Engineering

CENG 443

Introduction to Object Oriented Programming Languages and Systems

Spring 2018-2019

Homework 3 - RMI Maze Hub

Due date: 26 05 2019, Sunday, 23:59

1 Objective

This assignment aims to familiarize you with the development of Remote Method Invocation (RMI) using Java. Your task is to implement a RMI client and server application that can control multiple grid based mazes. RMI Server hosts the hub that access mazes for various operations. RMI Clients receives input from standard input and sends necessary commands to the server with remote invocation of functions. Towards this end, you will be using rmi interfaces and classes that are present in java.

Keywords— Remote, UnicastRemoteObject, Serializable, RMI

2 Problem Definition

There are four main classes you need to implement in this homework. They are explained in the following subsections.

2.1 IMaze Remote Interface

This is the first remote interface you need to implement to be used in the client. You should define a class that implements this interface for maze operations. Client will call this interface functions and they are going to be executed on the RMI Server. Behaviour of the functions of this interface is given below:

- **create:** Define the maze boundaries and other initialization operations of the maze instance.
- **getObject:** Returns the MazeObject at a given location. Returns null if there is no object there.
- **createObject:** If the position is empty in the maze, creates the MazeObject using MazeObjectType given as an argument and puts in the maze. If the type is an `AGENT`, it should create an Agent instance with an ID. IDs of agents start from 0 and incremented for every new agent instance created for that maze. Returns true if the position is empty and object can be created. Returns false otherwise.

- **deleteObject:** Deletes object resides in the given position from the maze. Return true upon successful deletion. Returns false if the position is empty.
- **getAgents:** Returns every agent within the simulation as an array.
- **moveAgent:** Moves the agent with the given id to the given position if manhattan distance to the new position is exactly 1. The are four situations for move operation and they are given below:

  1. If the new position is empty, move the agent to the new position.
  2. If there is a hole in the new position, remove the agent from the maze. This simulates falling into a hole in the maze.
  3. If there is a wall or another agent in the new position, agents stays in the same position.
  4. If there is gold in the new position, move the agent to the new position and collect the gold. The Agent's gold count should increase by 1 after this.

  Return true if the operation is possible and false if it is not.
- **print:** Converts the maze into a human readable string representation. It is explained in the I/O section.

2.2 IMazeHub Remote Interface

This is the second remote interface you need to implement to be used in the client. You should define a class to implement this interface. The class you have implemented should store all the mazes that is available in the hub. Only this class should be bound to the RMI registry in the server. Clients should use this class to obtain IMaze interface objects and use the obtained object for maze operations. Functions of the Interface is given below:

- **createMaze:** Creates and stores a empty maze with given width and height. You can consider the storage as a simple dynamic array and newly created mazes are appended at the end.
- **getMaze:** Returns the maze from the given index. Indexes start from 0. Returns null if the index is too large or smaller than 0.
- **deleteMaze:** Removes the maze from the hub. Returns true if removal is successful and false otherwise.

2.3 RMI Server

This class should bind the implementation of the IMazeHub to the rmi register in a specific address. This address is up to you. You will use this address to execute the remote functions from the RMIClient. Should not output anything.

2.4 RMI Client

This class read user commands from the standard input and calls remote interfaces to perform these operations. IMazeHub should be retrieved from the RMI registry and IMaze should be retrieved from IMazeHub function **getMaze** for operations. The input parser and basic skeleton implementation is provided to you. Write outputs to the standard output. Input and output format is explained in the I/O Section.

3 Input&Output

RMIClient has nine inputs it can receive. Enumeration is given in the template files and they are explained using these enumeration values:

- **CREATE_MAZE**: Creates a maze with given width and height.
  Format:

  ```
  cm <width> <height>
  ```

- **SELECT_MAZE**: Selects and records the maze with given index for maze operations.
  Format:

  ```
  sm <index>
  ```

- **DELETE_MAZE**: Deletes the maze with given index from the hub. Deletes the record of maze retrieved from select maze operation for safety.
  Format:

  ```
  dm <index>
  ```

- **CREATE_OBJECT**: Creates an object at the given location and using the given type. It uses the selected maze for this operation. Types are case insensitive values of `MazeObjectType` enumeration.
  Format:

  ```
  co <position_x> <position_y> <object_type>
  ```

  Example:

  ```
  co 5 5 wall
  ```

- **DELETE_OBJECT**: Deletes the object at the given location if there is one at that location. It uses the selected maze for this operation. Format:

  ```
  do <position_x> <position_y>
  ```

  Example:

  ```
  do 5 5
  ```

- **LIST_AGENTS**: List the agents in the selected maze. Format:

  ```
  la
  ```

  Output for every agent:

  ```
  Agent<agent_id> at (<position_x>, <position_y>). Gold collected: <agent_collected_gold>.\n
  ```

  Example:

  ```
  Agent5 at (4, 22). Gold collected: 2.\n
  ```

- **MOVE_AGENT**: Moves the agent to the given location. It uses the selected maze for this operation.
  Format:

  ```
  ma <agent_id> <position_x> <position_y>
  ```

  Example:

  ```
  ma 1 2 3
  ```

- **PRINT_MAZE**: Prints the selected maze.

  ```
  p
  ```

  Output for a maze:

  ```
  +<width # of - characters>+
  |<width # of object characters for height=0>|
  |<width # of object characters for height=1>|
  ...
  ...
  ...
  |<width # of object characters for height=maze_height-1>|
  +<width # of - characters>+
  ```

  Objects are represented with the following characters:

  - **WALL**: `X`
  - **HOLE**: `O`
  - **AGENT**: `A`
  - **GOLD**: `G`
  - **EMPTY**: A single space

  Example for 4x4 maze:

  ```
  +----+
  |XA X|
  |OOG |
  |X O |
  | GG |
  +----+
  ```

- **QUIT**: Quits the client. Format:

  ```
  q
  ```

RMIClient outputs two possible values for fail-able operations. They are given below:

- `Operation Success.\n` if operation succeeds.
- `Operation Failed.` if operation fails.

4 Specifications

- Your code must be written in Java.
- You must RMI to complete this homework. You need to use RMI interfaces and classes to create your solutions.
- Submissions will be evaluated with black box technique with different inputs. Consequently, output order and format is important.
- Submissions will also be evaluated with white box to make sure you are usin RMI and your code adheres to Object Oriented Programming principles.
- There will be penalty for non-compliance with OOP principles. Solutions that do not employ RMI will get 0.
- Everything you submit should be your own work. Usage of binary source files and codes found on internet is strictly forbidden.
- Please follow the course page on ODTUClass for updates and clarifications.
- Please ask your questions related to homework through ODTUClass instead of emailing directly to teaching assistants.

5 Submission

Submission will be done via ODTUClass. Create a zip file named `hw3.zip` that contains all your java source code files without any directory. All your code should be under default package. Your code should be able to compile and run using this command sequence (Assuming linux command line)

```
> javac *.java
> rmiregistry &
> java RMIServer &
> java RMIClient
```