MazeGame Exercise for Factory Pattern

This exercise contains two parts. Part 1 is the implementation of a simple MazeGame following the OO design. Part 2 is extending the simple MazeGame to a colorful MazeGame by implementing the factory patterns.

Part 1: Build a Simple Maze Game

Goal:

1. Check the familiarity with basic UML modeling
    a. Understand the given UML diagrams
    b. Understand Java code according to the UML diagram
2. Practice basic OO techniques using Java, including
    a. Using classes and methods
    b. Basic I/O
    c. Java compilation and configuration

Requirements:

1. Compile the given program to show an empty maze game
2. Modify methods in the given code to add rooms in the maze
3. Compile the modified program to show a simple maze with at least two rooms, and the rooms are connected by walls or doors,
4. Modify the program again to read from input files that specify maze structures.
5. Compile the modified program to show maze game as specified in the input files

Instructions

Step 1: Make the program running.

1. If you are using command line:

a) Select a working directory

b) Unzip the given *maze.zip* into the working directory, which will expand into a new *maze* directory.

c) Change the current directory to *maze*, and you will see two subdirectories, *src* and *bin*, as well as a *maze-ui.jar* file.

d) Read the .java files within the *src* directory according to the given UML diagrams and understand their relations

e) Type in the following command to compile the program:

i. javac -d bin -cp maze-ui.jar src\maze.java (Win) (1)*

ii. javac -d bin -cp maze-ui.jar src/maze/.java (Linux, Unix, Mac)*

f) Type in the following command to run the program:

i. java -cp bin;maze-ui.jar maze.SimpleMazeGame (Win) (2)

ii. java -cp bin:maze-ui.jar maze.SimpleMazeGame (Linux, Unix, Mac)

If successful, you will see an empty small window and a prompt:
"The maze does not have any rooms yet!"

2. **If you are using an IDE:**

a. Download and unzip the maze.zip to a folder

b. Import the project in your preferred IDE

c. Add the maze-ui.jar to your running environment as a UI library

d. Run the main program in SimpleMazeGame.java

Step 2: Build a maze with rooms

1. Fill in the *CreateMaze* function in the *SimpleMazeGame* class to create a maze with at least two rooms, connected with doors. Note:

a. please number your room starting from 0!

b. You need to call setCurrentRoom() to show the small ball moving around the maze.

2. Compile and run the new program using (1) and (2) to show a maze game with rooms

Step 3: Load a maze from a given file.

1. Two maze input files, *large.maze* and *small.maze* will be provided. The format of each line:

<Room/Door> <room no./door no.> < North> <South> <East> <West>
The order of direction is the same as the enumeration type: Direction.

2. Put these two files into a directory.

3. Fill in the *loadMaze* function in the *SimpleMazeGame* class to read a maze from a given file

4. Change the *main()* function so that

a. If the input path file is given as a parameter, then a corresponding maze should be produced;

b. If no parameter is given, then a maze should be created as you did in stage 2.

5. Compile the new program with command (1).

6. Run the new program. For example, if the input files are put into an *input* directory, then run the program as follows:

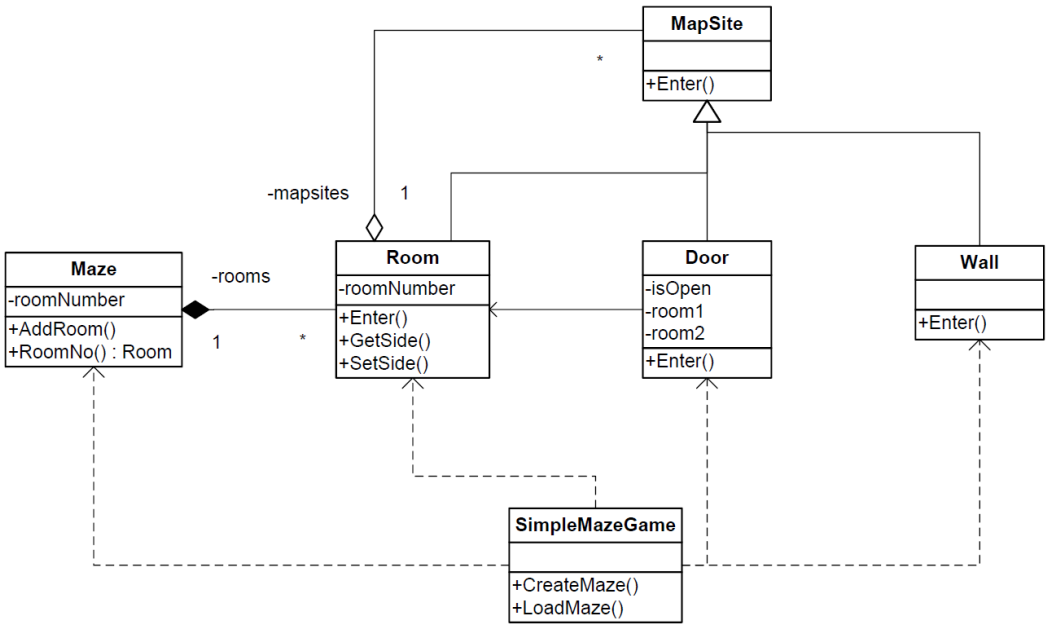
java -cp bin;maze-ui.jar maze.SimpleMazeGame input\small.maze (Win) (2)
java -cp bin;maze-ui.jar maze.SimpleMazeGame input/small.maze (Linux, Unix, Mac)
or you can run it on an IDE

After Step 3, pack the final program into a zip file and submit it through Canvas.

Grading Guidelines (50%)

(1) The modified program can show at least 2 rooms: 5%

(2) The modified program can show maze games according to the two configuration files: 45%



Part 2: Build a Colorful MazeGame

Goal:

Practice Factory Method Pattern and Abstract Factory Pattern

Requirements:

1. Adding the Red Maze Game and Blue Maze Game feature in addition to the basic function. The Red Maze Game will have pink rooms, red walls and green doors. The Blue Maze Game will have light blue rooms, brown door and dark blue Walls.
2. Modify the maze game you accomplished in hw7 to use Factory Method pattern.
3. Compile the new program to show basic, red or blue maze games.
4. Modify the maze game you accomplished in hw7 to use Abstract Factory pattern.
5. Compile the new program to show basic, red or blue maze games.

Instructions

This part has the following two stages:

Stage 1: Using Factory Method Pattern

1. Select a working directory
2. Copy and paste the basic maze game in hw7 to the working directory
3. Modify the basic maze game according to the UML Factory Method model:
    a. Modify the *SimpleMazeGame* class into a *MazeGameCreator* class, add factory methods and change the *main* method accordingly
    b. Run the new program

4. Add a Red Maze Game Feature
    a. Add necessary red maze game classes, such as the pink room class.
    b. Add a new *RedMazeGameCreator* class
    c. Change the *main* method to accept a “red” parameter
    d. Run the new program with “red” option

5. Add a Blue Maze Game Feature
    a. Add necessary blue maze game classes, such as a new room class.
    b. Add a new BlueMazeGameCreator class
    c. Change the *main* method to read both “red” and “blue” parameters

d. Run the new program with both “red” and “blue” options

Stage 2: Using Abstract Factory Pattern

1. Select a working directory
2. Copy and paste the basic maze game in hw7 to the working directory
3. Modify the basic maze game program according to the UML Abstract Factory diagram:

    a. Modify the *SimpleMazeGame* class into a *MazeGameAbstractFactory* class, and change the *main* method accordingly

    b. Create a *MazeFactory* interface

    c. Run the new program

4. Add a Red Maze Game Feature
    a. Add necessary red maze game classes.

    b. Add a new *RedMazeFactory* class

    c. Change the *main* method to read a “red” parameter

    d. Run the new program with “red” option

5. Add a Blue Maze Game Feature
    a. Add necessary blue maze game classes.

    b. Add a new BlueMazeFactory class

    c. Change the *main* method to read both “red” and “blue” parameters

Grading Guidelines

(1) Application of Factory Method pattern

a. The code conforms to the UML class diagram of FM pattern: 10%

b. Being able to create the maze game according to given parameters: 10%

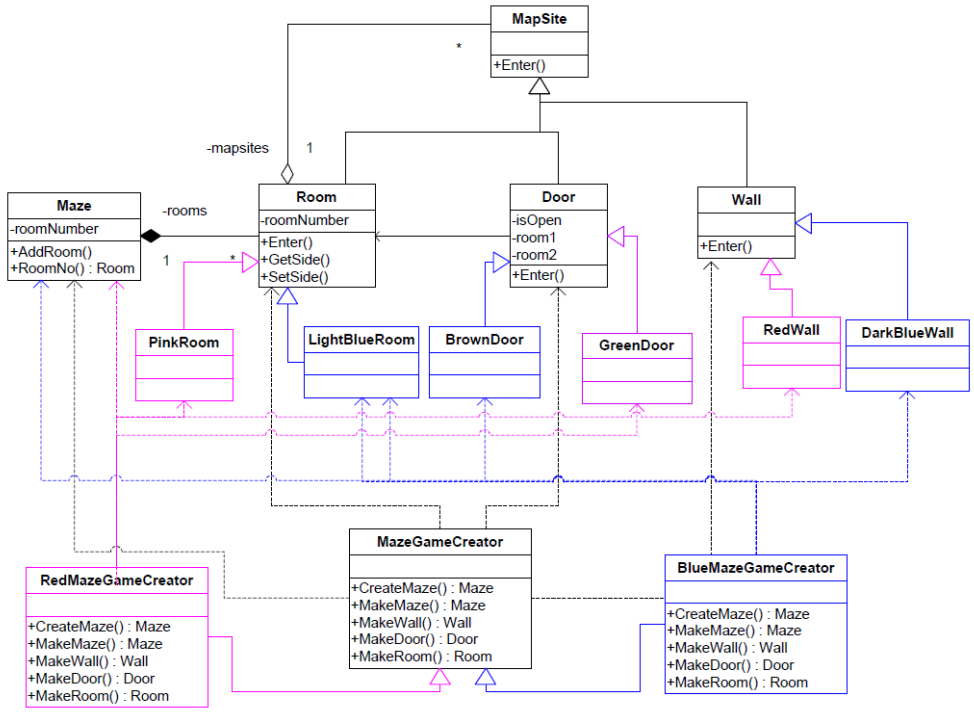
(2) Application of Abstract Factory pattern

a. The code conforms to the UML class diagram of AF pattern: 10%

b. Being able to create the maze game according to given parameters: 10%

(3) The quality of overall design (i.e. single responsibility, proper usage of data structure): 10%

Appendix A: Maze Game Factory Method Pattern



Appendix B: Maze Game Abstract Factory Pattern