COMP 3522 Lab #9: Abstract Factory

Christopher Thompson, Jeffrey Yim
jyim@bcit.ca

Due Friday 11:59pm

Introduction

Implement the abstract factory pattern in C++ using the following canonical example.

1 Set up your project

Start by creating a new project:

1. Clone your repo using github classroom: https://classroom.github.com/a/_RO7ETd5
2. Fill out your **name** and **student number** at the top of **main.cpp**
3. Ensure you commit and push your work frequently. You will not earn full marks if you don't

2 Requirements

1. Class maze_factory is abstract. It builds mazes, rooms, walls, and doors between rooms. A maze_factory can be instantiated and passed to some other module to be used to create the parts of a maze.
    - maze_factory contains a virtual member function called make_maze that returns a new maze.
    - maze_factory contains a virtual member function called make_wall that returns a new wall.
    - maze_factory contains a virtual member function called make_room that returns a new room.
    - maze_factory contains a virtual member function called make_door that accepts references/pointers to two rooms and returns a new door that 'connects' them.
2. Class maze_game is not abstract. It contains a method called create_maze that accepts a reference to a maze_factory and uses the maze_factory to build a maze. The create_maze method returns a pointer to the newly created maze.
3. Classes, maze, room, wall, and door should be abstract. They will be part of the factory. They may have one or more virtual functions related to printing out their contents and describing their surroundings. The following is a suggested solution, your exact solution may differ slightly
    - Maze contains containers of pointers to rooms in the maze, and functions to add rooms to the maze.
    - Rooms contain pointers to 4 walls and a door
    - Door contains pointers to the 2 rooms its connected
    - Walls will simply print out the wall description

4. Create concrete implementations of maze_factory, maze, room, wall, and door. Create two versions: an enchanted fairy land, and a futuristic dystopian maze.
5. Ensure your implementation includes overridden output methods that describe the rooms.
6. There should be no use of concrete types in any class “above" the boundary (refer to slides).
7. You will need a driver.cpp that contains a main method. Inside the main method, instantiate a maze_game and pass to its create_maze function a reference to an enchanted fairy factory. Print the description of the created maze. Do the same thing with the futuristic maze.
8. Remember that in order to earn full marks the low level implementation details must not appear outside of main or the implementing enchanted forest and futuristic classes.
9. Comment your code.
10. Things to note:
    - Don’t make new files for every new class or else we’ll have over 20+ files. Put related classes into one file. eg: Maze, Fairy_Maze, Dystopian_Maze classes can be in one set of maze.hpp/cpp files
    - Your output order doesn’t have to exactly match the sample output below
11. YOU’RE DONE!

Good luck, and have fun!

3 Sample output

```
A pretty, magical fairy maze
fairy Room 0: This room has fairies in it
fairy Room 1: This room has fairies in it
This door has a fairy wing for a handle.
This door connects fairy Room 0 and fairy Room 1
This wall has fairy wings on it
This wall has fairy wings on it
This wall has fairy wings on it
This wall has fairy wings on it
This wall has fairy wings on it
This wall has fairy wings on it
This wall has fairy wings on it
This wall has fairy wings on it

An Orwellian dystopian maze
Dystopian Room 0: This room is run down and cracked
Dystopian Room 1: This room is run down and cracked
This door has a pipe for a handle.
This door connects Dystopian Room 0 and Dystopian Room 1
This wall has surveillance cameras on it
This wall has surveillance cameras on it
This wall has surveillance cameras on it
This wall has surveillance cameras on it
This wall has surveillance cameras on it
This wall has surveillance cameras on it
This wall has surveillance cameras on it
This wall has surveillance cameras on it
```

4 Grading

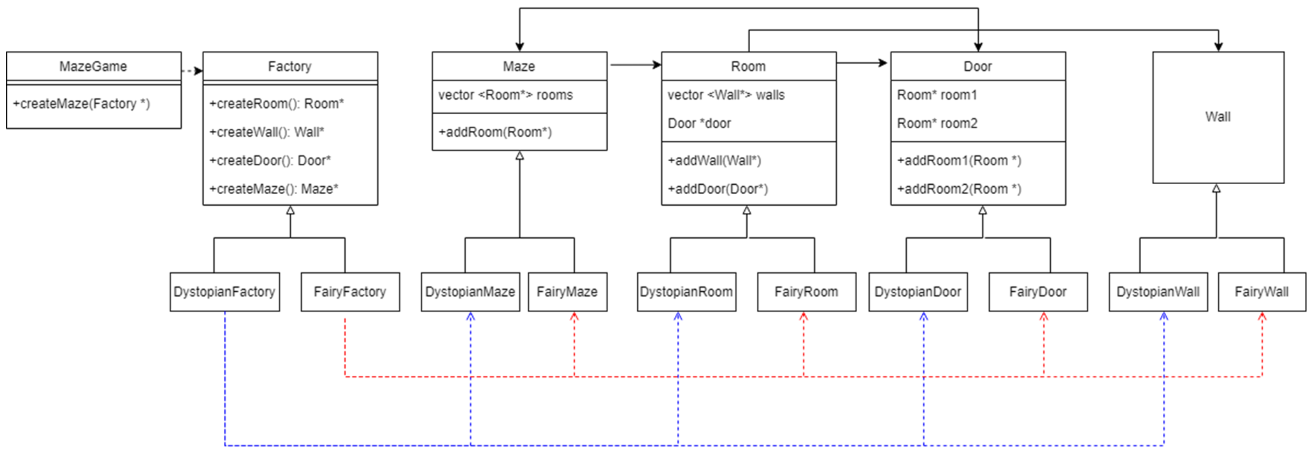
This activity will be marked out of 10. For full marks this week, you must:

1. (2 point) Commit and push to GitHub after each non-trivial change to your code
2. (6 points) Successfully implement the requirements as described in this document
3. (2 points) Write code that is consistently commented and formatted correctly using good variable names, efficient design choices, atomic functions, constants instead of magic numbers, etc.

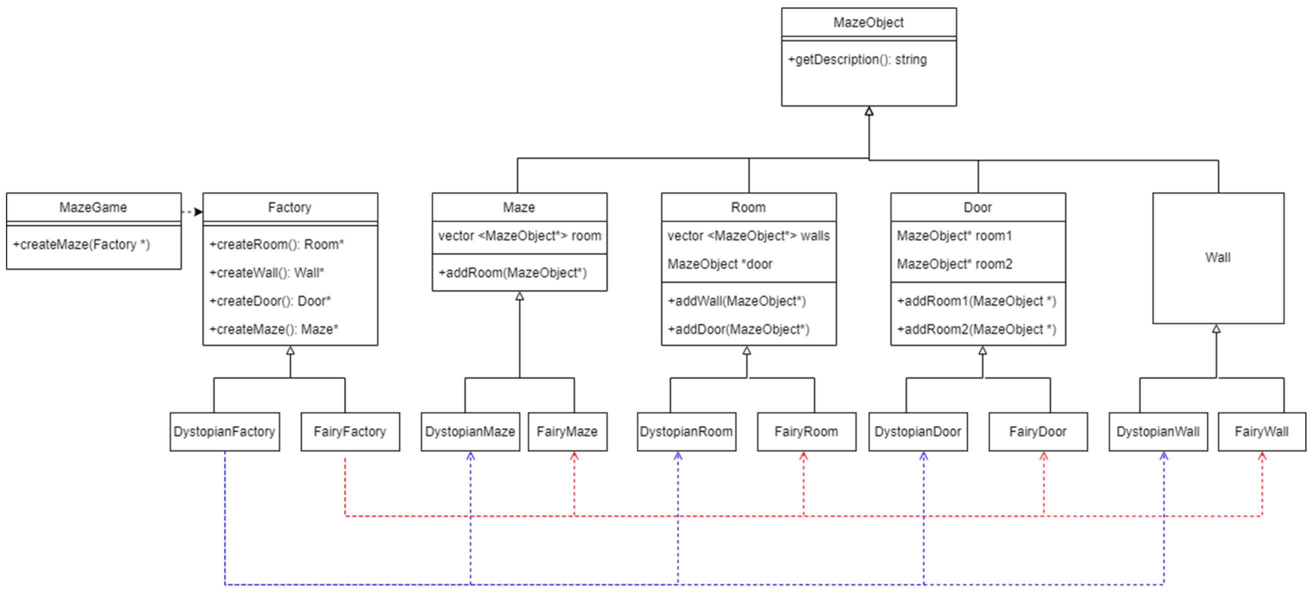
TIPS

- You will be using a lot of dynamic memory to instantiate the maze, room, wall, and doors. Remember to delete them all. Where would be appropriate places to delete them?
- Beware of forward declaration issues. room.hpp depends on door.hpp, and door.hpp depends on room.hpp. How do we properly solve this circular dependency with forward declaration?

Below is one possible solution class diagram for your solution:



Below is another possible solution that aggregates all the getDescription functions of the maze components into a generic MazeObject interface:


MazeObject
+getDescription(): string
MazeGame
+createMaze(Factory *)
Factory
+createRoom(): Room*
+createWall(): Wall*
+createDoor(): Door*
+createMaze(): Maze*
Maze
vector <MazeObject*> room:
+addRoom(MazeObject*)
Room
vector <MazeObject*> walls
MazeObject *door
+addWall(MazeObject*)
+addDoor(MazeObject*)
Door
MazeObject* room1
MazeObject* room2
+addRoom1(MazeObject *)
+addRoom2(MazeObject *)
Wall
DystopianFactory
FairyFactory
DystopianMaze
FairyMaze
DystopianRoom
FairyRoom
DystopianDoor
FairyDoor
DystopianWall
FairyWall