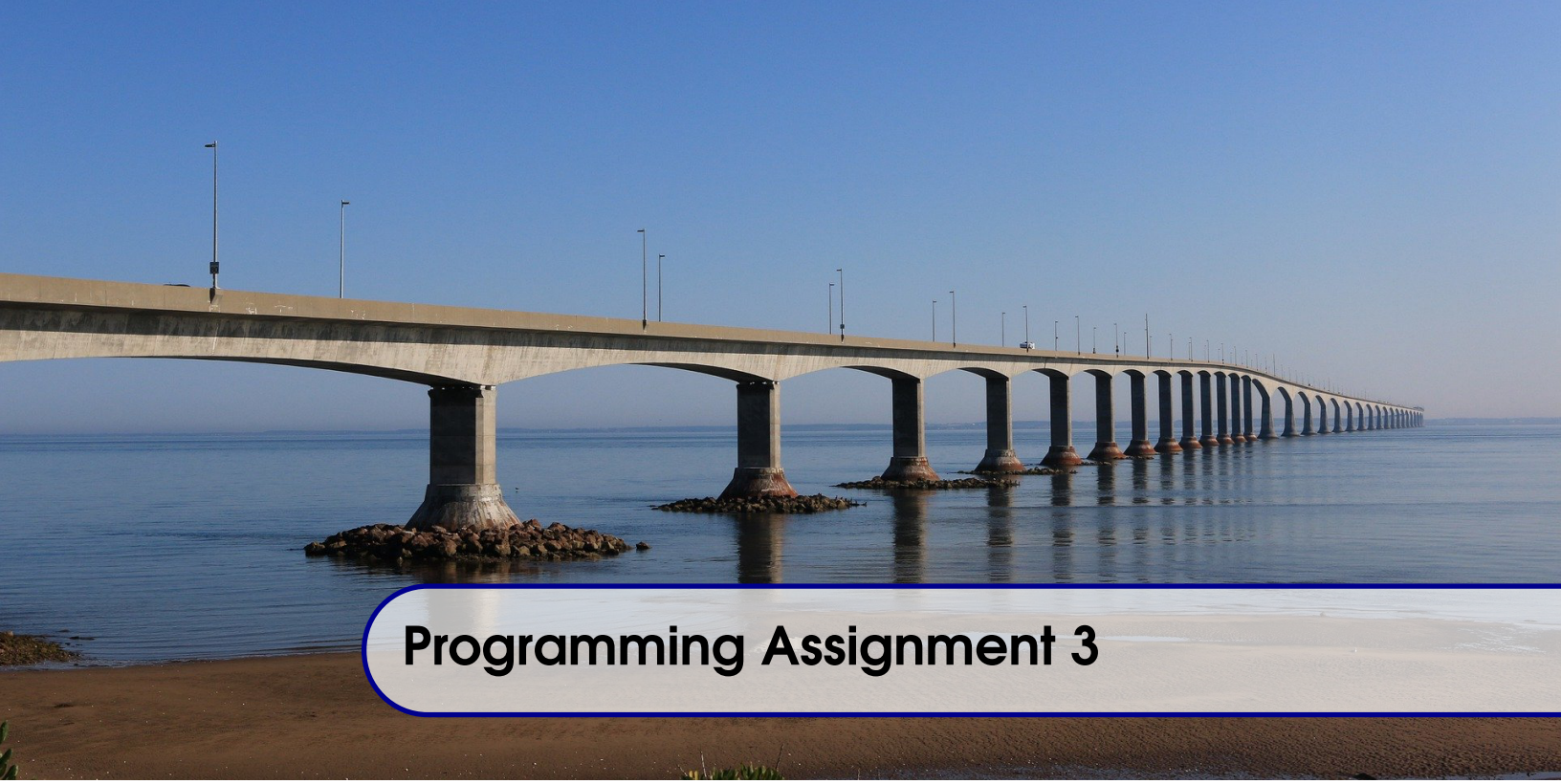

Programming Assignment 3

Assignment Objectives

By completing this assignment you are demonstrating your ability to:

- Create your own classes implementing a Java interface.
- Create your own classes by extending an existing abstract class.
- Read and write data from files.
- Create your own linked structure for representing complex data.
- Design and implemented fairly complex algorithms for manipulating the data.

Scenario: Mazes & Maps

In this and the next assignment we will be working with mazes and maps for representing and navigating through them. In these mazes each room has four walls: east, north, west and south. Each wall may or may not have a gate, and some gates will be locked. You would need a color key to unlock them. Some rooms will also contain such keys.

We will name each room using a string, which could simply be a letter. Different rooms are connected by gates, these are magical gates that have teleport portals in them, and thus, they can arbitrarily connect to each other. Each maze will also have an Entrance and an Exit gate.

There are two types of Mazes:

- Unidirectional Mazes: Portal Gates in this type of mazes are not symmetrical; which means that the south gate from Room A may take you to the north gate of room B, but the the north gate from room B may take you to a different location, e.g. east gate in room C. These mazes are much more difficult to navigate, and thus, we won't be focusing on them.
- Bidirectional Mazes: In these mazes gates have the exact opposite direction either way. This means that if the north gate of room A connects to the south gate of room B, then the south gate of room B will also connect to the north gate of room A. These rooms are easier to navigate

and also easier to describe: **you only need to describe the connection in one direction**. In this assignment we will be working with this type of maze.

Description of a maze

A maze can be described in a text file, with each line providing information of the connection between two rooms' gates, a key contained in the room or the location of the entrance or exit gate. For example, the following line indicates that the west gate of room A connects to the south gate of room B, and vice-versa:

A west <-> B south

This other line indicates that the north gate of room B connects to the south gate of room E, and vice-versa, but also indicates that a BLUE key is required to unlock the gate:

B north <BLUE> E south

A line like this one indicates that the 'Entrance' to the maze is through the south gate of room A.

A south <-> Entrance

And finally, a line like this one, indicates that room B contains a RED key:

B RED key

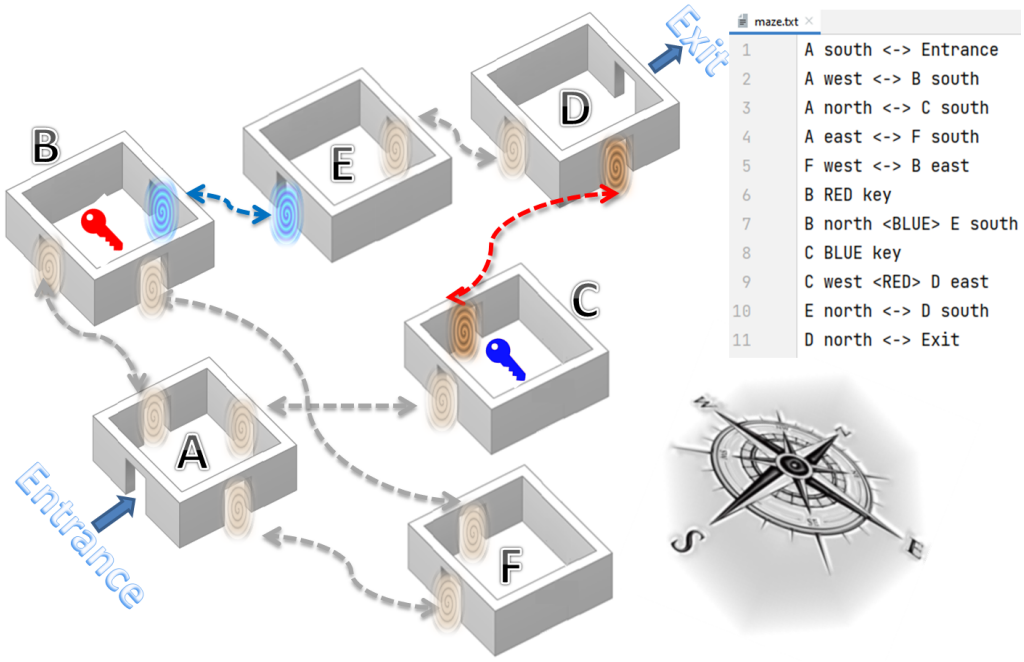

Figure 1: This picture shows a maze and its description in a text file.

The MazeRoom interface and the Action class

You will be provided with an interface for a Maze Room. This interface defines the following methods:

- *public String name()* - returns the name of the room.
- *public String roomKey()* - returns a string in capital letters indicating the color of the key (e.g. "BLUE") or an empty string ("") if no key is present.

MazeRoom interface also extends the generic *Iterable<Action>* interface. This means that for a class to implement the *MazeRoom* interface, it must also implement the *Iterator()* method, which must return an iterator; in this case an iterator of actions. For more information on the Iterable and Iterator interfaces refer to the documentation:
https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html
https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

The *Action* class represents a possible action (specifically a movement) that can be performed in a room. It contains public fields for the gate (String), the room that can be accessed through that gate (MazeRoom) and the required key to unlock the gate (String); empty if none. This class is also provided to you.

The MazeMap hierarchy

There is also a hierarchy of classes for representing maps of different types of mazes. At the top of the hierarchy there is an abstract *MazeMap* class (which you will also get). Concrete classes for BiDirectional and Unidirectional Maze Maps should inherit from the MazeMap class. The MazeMap class contains the following fields and methods:

- *protected MazeRoom entrance* - contains a reference to a 'dummy' entrance node.
- *public void mazeReport()* - prints a report indicating the rooms and keys that are accessible from the entrance.

Problem 1: Implement a Room class

Implement a *Room* class that implements the *MazeRoom* interface. For this you will have to implement all the methods in the *MazeRoom* and the *Iterable<Action>* interfaces. Notice that to achieve this you will also have to create a *MazeRoomIterator* class that implements the *Iterator* interface.

Note that the *Room* class will have other methods than those required by the interfaces, e.g. to set the keys and Maze rooms accessible through each gate. The *Room* class will also require attributes to store this information (these attributes should be private).

To test Problem 1 you must provide a Driver class, name it 'DriverP1', that creates a Room named 'A' and sets rooms named 'W', 'N', 'S' and 'E' as the rooms accessed through the 'west', 'north', 'south' and 'east' gates respectively. The East gate should also require a 'RED' key to be unlocked. The main method should then print all this information using an enhanced for loop to iterate through the actions provided by room A's iterator. Figure 2 shows a screenshot of executing the driver. The order of the rooms will depend on how you implemented the iterator class.

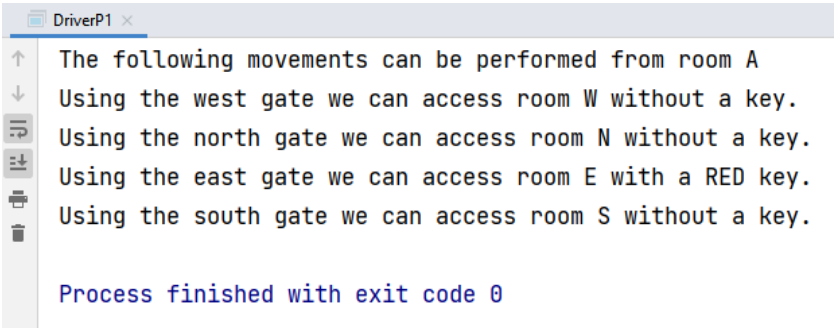

Figure 2: An example of executing the driver for Problem 1. The order of the rooms will depend on how you implemented the iterator class.

Problem 2: Implement the BidirectionalMap class

The second part of this assignment is to implement the *BidirectionalMap* class with just a constructor (in the next assignment we will create more methods for this class). The *BidirectionalMap* class must extend the *MazeMap* class and provide a constructor that creates a linked structure representing a Maze from a file describing the maze.

The linked structure should have a 'dummy' (*MazeRooms*) node representing the entrance to the maze. This node should be assigned to the corresponding variables defined in the superclass *MazeMap*.

For debugging purposes we encourage you to install the 'Java Visualizer' plugin for IntelliJ. This plugin facilitates the visualization of linked data structures when debugging. Figure 4 shows the linked structure of MazeRooms that represent the maze in Figure 1. You can see that the entrance node has all its gates pointing towards the entrance gate of the maze. This dummy node is added to facilitate the access and traversal of the data structure.

Important note, any information about rooms that can't be accessed from the entrance can be ignored when creating the linked data structure. For example, the 'maze.txt' file provided has an extra line (compared to the one shown in Figure 1): *X south* <-> *Y north*, these two rooms (X and Y) shouldn't be part of the linked structure because they are not connected to the central component of the maze.

For Problem 2 you must provide a Driver class, name it 'DriverP2', which will create a maze map object from the provided 'maze.txt' file. The driver must then execute the *mazeReport()* method on the map. If you correctly created the maze then the output will be like the one shown in Figure 3. **Important note,** the output of a correctly constructed maze will coincide with the one in the figure, but correct output doesn't guarantee a correctly built maze structure. For ensuring the correctness of your code you should debug using Java Visualize plugin.

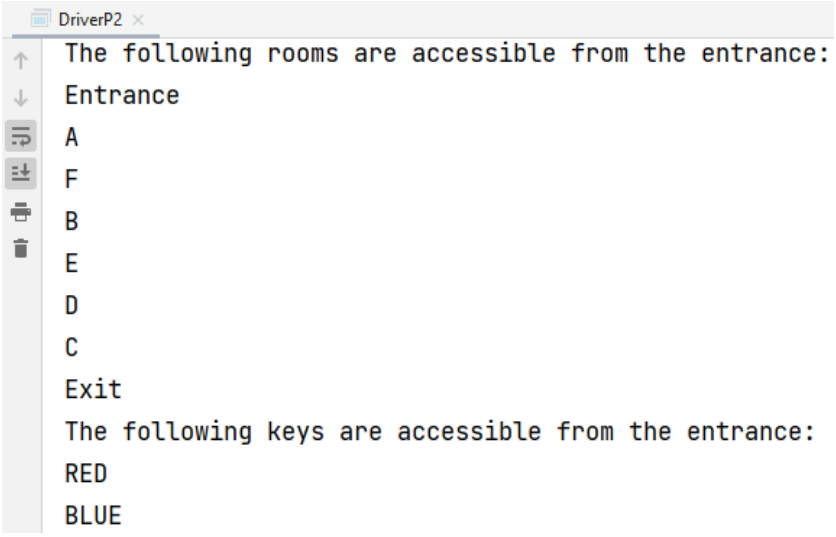

Figure 3: An example of executing the driver for Problem 1. The order of the rooms will depend on how you implemented the iterator class.

Academic Integrity Statement

For this, and all assignments in CS2910, you will include a text file in the project folder, at the same folder level as `src` and `data`, named `academic-integrity-statement.txt`. Within that file you should include the following text:

As a student in CS2910 at the University of Prince Edward Island, I am committed:

- To sustain my effort and engagement for the learning in this course.
- To ask questions about the purpose of or criteria for an assessment if I am uncertain.
- To ask questions about the rules surrounding the assessment if I am uncertain.
- To follow the rules for the assessment, including under conditions of no supervision.
- To treat my personal learning as valuable it its own right.

I submit this assessment, acknowledging the above commitments and that I have followed the rules outlined by my instructor and consistent with the Academic Regulation 20.

At the bottom of this text you should put your name, student number and the date of submission.

<STUDENT NAME>
<STUDENT NUMBER>
<ASSIGNMENT DUE DATE>

Please replace the items in angle brackets (<>) with appropriate information for your assignment.

Please note: We have not included this file in the package download. You must add this file yourself with the contents as specified above. If you do not submit this statement you will receive 0 on the assignment.

Submitting your assignment

Your submission will consist of one zip file containing solutions to all problems (see below). This file will be submitted via a link provided on our Moodle page just below the assignment description. This file must be uploaded by 5pm (Moodle time) on the due date in order to be accepted. You can submit your solution any number of times prior to the cutoff time (each upload overwrites the previous one). Therefore, you can (and should) practice uploading your solution and verifying that it has arrived correctly.

What's in your zip file?

A zip file is a compressed file that can contain any number of folders and files. Name your zip file using this format.

Example:
Submission file: `asnX_studentnum.zip`
Where `X` is the assignment number between 1 and 4, and `studentnum` is your student ID number.

Your zip file should contain an IntelliJ project and appropriate files for the assignment and your Academic Integrity Statement.

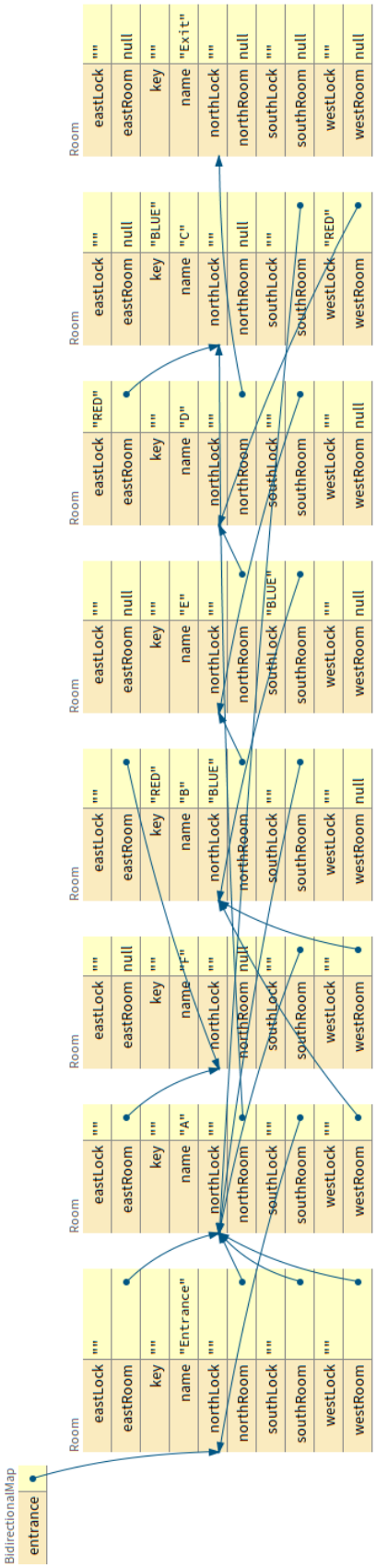

Figure 4: Linked structure visualized by the Java Visualizer tool for the map in Figure 1.