Team: Bo Cao
Fernando Nobre
Zhongzhi Zhang

Title: Robot Path Learning

1. What features were implemented?

The following use cases were implemented and tested:

Use Case	**Description**
UR-001	A visitor can register to become a regular user.
UR-002	User can log in to the system.
UR-003	User can logout of the system.
UR-004	A user views his profile.
UR-005	A user will be directed to the new map page when he/she logs into the system.
UR-007	User can select and view one of the previous paths.
UR-008	User can let the robot move along the path he created before and draw the trace behind it.
UR-009	A User can move the robot to a position on the map by clicking on the map.
UR-010	A User can add an obstacle to the map by clicking on the map.
UR-011	A User can remove an obstacle to the map by clicking on the map.
UR-012	A user can save path's he has created.
UR-013	A user can cancel path's he is creating.
UR-014	An administrator can view regular users' profiles.
UR-017	An external system can retrieve all the maps and paths created by all the users.

In addition, mapping classes into database tables was implemented by Hibernate instead of writing sql code manually.

The Singleton design pattern mentioned in Part 2 was implemented, only one object of Robot was allowed to be instantiated.

2. Which features were not implemented from Part 2?

The following use cases were not implemented:

Use Case	**Description**
UR-006	A User can see all the path's he has created.
UR-015	An administrator can designate a regular user to be an administrator.
UR-016	An administrator can terminate a regular user's account.

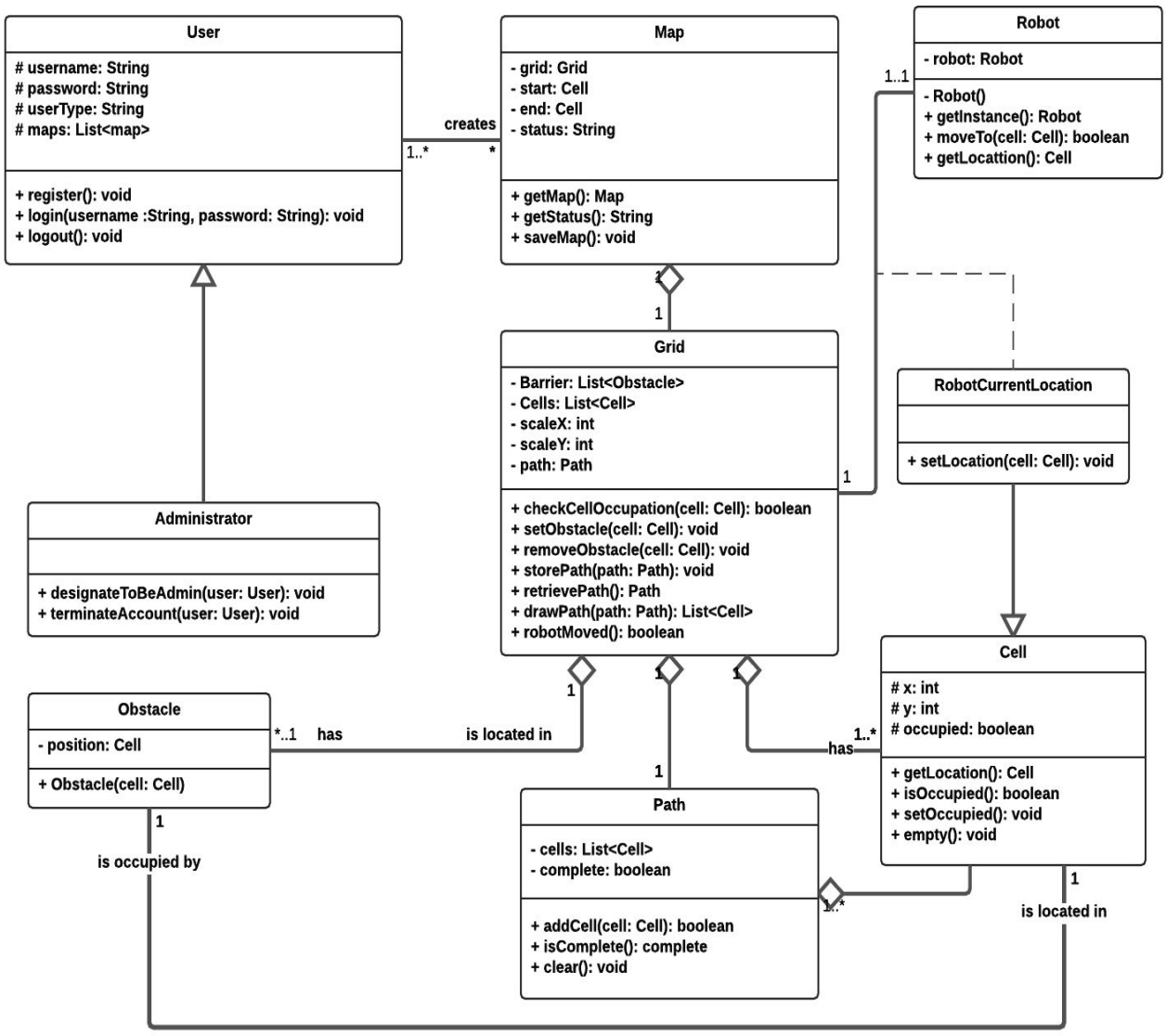
3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

During the development stage we had not changed the class diagram.

With this well-designed class diagram we avoided internal design defects from the design analysis stage. Our project was quickly developed based on it. Classes code was quickly generated according the diagram, which saved a large amount of time.

Defects from the early stages could be amplified in the following stages. However, in the implementation stage, since no errors related to class design occurred, we did not need to redesign our class diagram, or implement the new class diagram later, which could have been a great cost of time and efforts. The robust class diagram increased our development efficiency without reverting to previous development stages.

Part 2 Class Diagram



4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?

A few design patterns were used for the final implementation:

Design Pattern	**Implementation**

Observer	Used to observe mouse clicks and keyboard commands on the map to control the robot.
Decorator	Used for applying a "master layout" theme to the views. Can be easily used to customize the views for each type of client device (mobile/desktop/tablet, etc)
Singleton	Used in conjunction with Hibernate: The "SessionFactory" is only instantiated once per application.
Service Layer and Dependency Injection (IoC)	A service layer was created to decouple the business logic from the controller. Dependency Injection was used to configure the service layer class, which is @autowired into the client classes that need to use it.

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

The thorough analysis of requirements and use cases saves a lot of development time. In the requirements we brainstormed and went depth into the problems of the system. During the design analysis stage we carefully drew all design diagrams and checked the relations between them. Although this process did not go very quickly, but helped to avoid consuming much time for reverting back to previous stages. Correct sequence diagrams directly helped coding and testing. When errors occurred we checked if the code represents the corresponding diagrams. Sticking to these design diagrams helps to minimize the errors in the following stages.

Having a working knowledge of several design patterns was very helpful in creating useful and maintainable code. One of the best examples was Decorator design pattern we used, which made our representation layer easy to be organized and maintained without many duplicate code of displaying the same graphs.

We learned how to deal with changes in the requirements (such as removing requirements from the implementation phase) and also the importance of a collaboration in all phases of the project. We first divided our tasks to each team member while having the same system architecture and design pattern, making us easy to merge our implementations with the solid design from previous stages.