CSC207 PROJECT PHASE 0
Group Name: Circus

1. **Domain**
   Warehouse logistics system

2. **Specification**
   Create a program with a user interface that allows users to load a new warehouse layout (made up of Tiles) given width and height specifications from the user. There are designated tiles for racks (of a given size storing the user's inventory) as well as receiving and shipping depots (for receiving new items and shipping items from the warehouse). Once this warehouse layout is loaded, users can create items to fill the inventory of their warehouse by providing item names, descriptions, and ids (if specified, otherwise the software will generate a random id for the new item). After the inventory is filled and loaded, the user will be able to view and access it's contents. Once both the warehouse layout and inventory are loaded, the program will visualize the user's warehouse (as a grid) as well as the location of all robots in the warehouse. This visualization should update their position in real-time as well as display routes they are currently undertaking.

   The user should be able to create orders to both add new items to the warehouse (increase inventory by adding this item to a rack), update existing items, and remove items from the warehouse using the interface. Note that each rack should only hold one item: it can hold a given quantity of that item, however it may not store more than one item type. Once the user creates an order for any of said reasons, it will be added to a queue (possibly behind other orders) visualized by the program so the user can track the progress of their order as it is processed. The user can make two possible orders: one that receives a new item in the designated depot and places it in the warehouse, and one that ships an item already in the warehouse via the shipping depot. Once the order reaches the front of the queue, a robot will be assigned to moving it to it's designated location (either placing the item in the warehouse or moving it out) and it will use a route-finding algorithm to reach the item, pick it up, and move it to its destination via the most efficient path. This robot must only travel in the user-designated paths, avoiding racks, robots, and other obstacles. After the robot completes it's route, the user's inventory should be updated (depending on whether the user removed or placed an item).

3. **Scenario Walk-Through**
   Suppose a user wishes to create a new warehouse layout and place an item into this warehouse efficiently using our program. Upon running the code, said user will be met with an interface (command shell) where they are able to input specifications (tile height and width of warehouse) to load and create a new warehouse layout. The command shell will visualize this warehouse for the user with a series of strings where '[]' represents a single tile. The user must first create an item they wish to store in the warehouse by providing a series of details including the item name, description, and id (optional). Once this item is created, the user can then create an order to add this item to the warehouse. Since this item is the first in the warehouse, we know the warehouse is not full and it will immediately be placed in the warehouse. The program will

calculate the most efficient route (path that will take the least steps) to move this item from the receiving depot tile to the nearest available rack. The rack will then update to reflect what item it is storing (since each rack can store only one item type) as well as the quantity of that item (which in this case would be one). The user will also be able to visualize and access the updated warehouse, which now contains one item.

4. Progress Report

For our project we chose to create a warehouse logistics system. Our specification outlines our goals for the finished program that will allow users to create and manage their warehouses of a specified size via a graphical user interface. This warehouse will be made up of tiles containing racks as well as receive/ship depots where the user is able to create orders to add/remove items from their warehouse. Robots in the warehouse will work through orders in a queue to move items from their source to their destination via routes calculated by a pathfinding algorithm which minimizes the number of "steps" each robot takes to reach the item and deliver it. All of these actions will be updated on the graphical user interface so the user can track the progress of their order in real time.

Our CRC model contains the classes that satisfy our specification. This includes classes for entities such as Item, Tile, Warehouse, and Rack as well as classes for use cases like Pathfinder, Robot and ItemScorer. We have an additional CRC card for our WarehouseController which is in charge of the warehouse as a whole and is the class the user interacts with (as opposed to use cases and entities). Our model also accounts for a basic command line interface which we plan to replace with a graphical user interface upon completion. The scenario walk-through uses most of the components of our CRC model in which a user creates a warehouse and is able to add an item to a given rack through a basic command shell.

Our current skeleton program allows a given user to complete the scenario outlined in the walkthrough. It consists of five packages: warehouse, containing classes for the basic warehouse layout; query, containing interfaces for AndQuery (a query that chains other queries together); pathfinding, containing classes that assist in calculating the most efficient routes for robots in the warehouse using a graph; inventory, consisting of entities InventoryCatalogue and Item; and application, which contains classes for the command line interface, as well as the Distributable and Receivable interfaces and "Main". We have also created tests for some of the central classes located in the tests folder.

Thus far our skeleton code enables a user to complete the walk-through scenario outlined above. These classes follow the dependency rule: in that there is solely dependence on the adjacent layer (from outer to inner) with the assistance of dependency inversion. As a result, it will be far easier to implement the rest of our program without dependency complications. One question our group has been struggling with in regards to software design is which design pattern to use for the architecture of the pathfinder code.

Throughout phase 0 each group member has worked on different aspects of the skeleton code and design plan. Shon worked on designing the architecture of the program, and implemented the basics of the shell application. He also worked on designing and implementing a query system for the items, which will allow our agents to retrieve items from the warehouse based on generic queries. Artem worked on designing the architecture of the program, and implemented the basic tests for a variety of classes. In addition, he worked on some shell

commands. Fiony worked on designing the architecture of the program, implemented a few tests for warehouseController, and helped out on other classes when needed. Audrey worked on designing the architecture of the program, as well as the specification, CRC cards, scenario walk-through, and progress report. Vic participated in discussions of the architecture of the code, and assisted with the scenario walk-through and the progress report. Liam worked on designing the architecture of the program and implementing the beginning of the A* algorithm that will come into use in the future of our project. Furthermore, he worked on some of the basics of our architecture such as the warehouse package and additionally implemented some tests for the Rack class and also some commands for the shell application.