CSC 232 – Object-Oriented Software Development
Project Checkpoint #3
Due: Wednesday, December 9th (by 11:59PM – Greencastle time)

In this checkpoint, you will be adding support for taking items out of a "**container item**" (i.e., an Item that is capable of storing Items … as well as putting items into a "**container item**" – think: desk, treasure chest, box, etc.). As a refresher of what you will be building in this checkpoint, open up the game of Zork and type the following commands:

1. Type **look** and hit Enter
   - Recall, this command prints out the Location your character is currently, its description, and a listing of the Item objects found at this Location
2. Type **go south** and hit Enter
3. Type **go east** and hit Enter
4. Type **open window** and hit Enter
5. Type **enter window** and hit Enter
   - You should now be in the Kitchen where your character can see a table with a brown bag and glass bottle. It is important to notice that the **glass bottle** is a "container" (i.e., "The glass bottle contains: A quantity of water"). Recall, a "container" item as a **special type of Item** that has everything that a regular Item object has, but also, it has the ability to store Items inside itself. Since a "container" is an item-that-contains-items, this means we can take an Item out of a "container" and add it to our inventory (backpack) …
6. Type **inventory** and hit Enter
   - Notice: Your character's inventory is currently empty
7. Type **open bottle** and hit Enter
   - You will **not** need to support opening container Items to take an Item out in your text-based adventure game – this is just how Zork chooses to implement their game
8. Type **take water from bottle** and hit Enter
   - This command takes two words as "parameters" (i.e., take ____ from ____) and causes the water to be removed from the container and added to our inventory. In this checkpoint, you will implement this feature in your text-based adventure game.
9. Type **inventory** and hit Enter
   - Notice: Your character's inventory now has the water (i.e., "a quantity of water") – not the glass bottle
10. Type **look** and hit Enter
    - Notice: The glass bottle still remains in the Location but since we have taken the water out, it no longer contains any Item(s) inside of itself.
11. Type **go west** and hit Enter
    - Your character should now be in the Living Room that contains Items such as a a rug, trophy case, lantern, and sword.
12. Type **take sword** and hit Enter
13. Type **inventory** and hit Enter
    - Your character's inventory should now contain the quantity of water and a sword
14. Type **open trophy case** and hit Enter
    - You will **not** need to support opening container Items to put an Item into the container in your text-based adventure game – this is just how Zork chooses to implement their game

15. Type **put sword in trophy case** and hit Enter
    - This command takes two words as "parameters" (i.e., put ___ in ____) and causes the sword to be removed from the character's inventory and added to the "container" (e.g., trophy case). In this checkpoint, you will implement this feature in your text-based adventure game.
16. Type **inventory** and hit Enter
    - Notice: Your character's inventory no longer has the sword
17. Finally, type **look** and hit Enter
    - Notice: The trophy case "container" now contains the sword inside ("… collection of treasures consists of: A sword")

Once you have a good grasp for how to interact with "container" Items, you (and your partner) can proceed in completing the following tasks for Checkpoint 03.

Task – Modify your Driver Class

You will need to modify your Driver class once more to support a range of commands related to acquiring and removing items to/from your character's inventory.

- ☐ Modify your **createWorld**( ) method in your Driver class to add at least **two** different ContainerItems (e.g., chest, desk, vault, etc.) spread out across your world's locations so that I may test your code. Each ContainerItem should have some starting Items stored inside of it for testing purposes. Note: If you put all of the ContainerItems in the starting location, I will deduct points if I have to manually add ContainerItems to other Locations in order to test your code. If you do not put starting Items inside of these ContainerItems, I will also deduct points if I have to manually add Items to them.
- ☐ The commands that your text-based adventure game **must** add support for are:
  - ☐ If the user types **examine** __CONTAINER__ , then your program should print the ContainerItem's name, type, description, and **names-only** of the items that it contains
  - ☐ If the user types **take** __NAME__ **from** __CONTAINER__ , then your program should remove the Item whose name is __NAME__ from the ContainerItem whose name is __CONTAINER__ at the current location and add the item to the character's inventory (e.g., **take key from chest**)
    - Recall: You should not use try-catch statements at all to handle unchecked exceptions – doing so will result in a loss of points
  - ☐ If the user types **put** __NAME__ **in** __CONTAINER__ , then your program should remove the Item whose name is __NAME__ from the character's inventory and add the item to the ContainerItem whose name is __CONTAINER__ at the current location (e.g., **put sword in vault**)
    - Recall: You should not use try-catch statements at all to handle unchecked exceptions – doing so will result in a loss of points

edit help!!

Important: This is the final checkpoint for your text-based adventure game. You should have addressed any issues listed in previous checkpoints to ensure that your program is working correctly. The grading rubric that I will be using for this checkpoint is provided below. Each test case/scenario will be checked on your program – if your program works correctly for the test case, it will receive 100% for that criteria, otherwise, it will receive 0%. The new functionality that you added for this checkpoint will be worth more weight than prior functionality but all will be assessed one final time.

You will want to test each of the commands listed on the next page to make sure that they behave correctly regardless of the user's input:

- ☐ The '**look**' command should work correctly at any Location (i.e., the look command should print the location's name, description, and the names-only of items that are at your character's current location)
- ☐ The '**examine**' command should (a) notify the user they need to type an item's name if it was omitted and (b) it should not cause the program to crash
- ☐ The '**examine**' command should notify the user that the item does not exist when an invalid Item name is typed (i.e., if I type 'examine shoe' and there is not a shoe item, then I should be alerted that 'shoe' does not exist)
- ☐ The '**examine**' command should work correctly if a valid Item name is typed by printing out information about that item (as required from Checkpoint 01)
- ☐ The '**examine**' command should work correctly if a valid Item name is typed (regardless of uppercase/lowercase letters)
- ☐ The '**go**' command should (a) notify the user that they need to type a direction name if it was omitted and (b) it should not cause the program to crash
- ☐ The '**go**' command should move the character if a valid direction name is provided and a location exists in that direction to move to
- ☐ The '**go**' command should (a) notify the user if a valid direction name was provided but there is no location to move to in that direction
- ☐ The '**go**' command should work correctly regardless of uppercase/lowercase letters
- ☐ The '**go**' command should notify the user if an invalid direction name is used (e.g., go backward)
- ☐ The '**take**' command should notify the user that they need to type an item's name if it was omitted and (b) it should not cause the program to crash
- ☐ The '**take**' command should work correctly if a valid item name was provided at the character's current location
- ☐ The '**take**' command should work correctly regardless of uppercase/lowercase letters
- ☐ The '**take**' command should notify the user that the item does not exist when an invalid Item name is typed (i.e., if I type 'take shoe' and there is not a shoe item at my character's current location)
- ☐ The '**drop**' command should notify the user that they need to type an item's name if it was omitted and (b) it should not cause the program to crash
- ☐ The '**drop** command should work correctly if a valid item name was provided in the character's inventory
- ☐ The '**drop**' command should work correctly regardless of uppercase/lowercase letters
- ☐ The '**drop**' command should notify the user that the item does not exist when an invalid Item name is typed (i.e., if I type 'drop shoe' and there is not a shoe item in my character's inventory)
- ☐ The '**inventory**' command should correctly display the names of item objects that I have taken
- ☐ The '**help**' command should properly display the list of commands currently supported
- ☐ The '**take** ___ **from** ____' command should work correctly if a valid Item and ContainerItem name were provided
- ☐ The '**take** ___ **from** ____' command should notify the user if an invalid ContainerItem name was provided (i.e., a ContainerItem that does not exist or an Item that is not a ContainerItem)
- ☐ The '**take** ___ **from** ____' command should notify the user if a valid ContainerItem name was provided however the Item name provided was not contained inside
- ☐ The '**put** ___ **in** ____' command should work correctly if a valid Item and ContainerItem name were provided
- ☐ The '**put** ___ **in** ____' command should notify the user if an invalid ContainerItem name was provided (i.e., a ContainerItem that does not exist or an Item that is not a ContainerItem)
- ☐ The '**put** ___ **in** ____' command should notify the user if a valid ContainerItem name was provided however the Item name provided was not in your character's inventory
- ☐ Any other command that is not supported, and your game should properly alert the user that the command is not supported