Name: Quang Vinh Le

Id: 104097488

1.How I completed my task.

-First, I create the 'LookCommandExecute' method to handle the execution of the 'look' command in the game. This method takes three parameters: 'Command look,' 'string input,' and 'Player player.'

```
1 reference
static void LookCommandExecute(Command look, string input, Player player)
{
    Console.WriteLine(look.Execute(player, input.Split()));

}
```

Its purpose is to invoke the 'Execute' method of the provided 'Command' object, 'look,' passing the 'player' and the split 'input' as arguments. This is where the logic for the 'look' command is implemented.

-In the 'Main' method, two variables, 'name' and 'desc,' are initialized to collect the player's name and description. Additionally, a 'help' variable is set to display useful commands to assist players in inputting commands.

```
0 references
static void Main(string[] args)
{
    string name, desc;
    string help = "Here are some useful commands:\nTo get item list:\n-look at me\n-look at bag\n\nTo get item description:\n-look at {item}\n-look at {item} in me\n-look at {item} in bag\n\nTo qui

    Console.WriteLine("Welcome to SwinAdventure...\n");



    Console.Write("Enter Player Name: ");
    name = Console.ReadLine();
    Console.Write("Enter Player Description: ");
    desc = Console.ReadLine();
    Player player = new Player(name, desc);
    Console.Write("\nEnter 'help' to show some useful commnads.");
```

-Next, I initialize items like a sword, touch, ruby, and a bag are created and placed in the player's inventory and the bag's inventory.

```
Item sword = new Item(new string[] { "sword" }, "a sword", "This is a sword");
Item touch = new Item(new string[] { "touch" }, "a touch", "This is a touch");
Item ruby = new Item(new string[] { "ruby" }, "a ruby", "This is a ruby");



Bag bag = new Bag(new string[] { $"bag" }, $"{player.Name}'s bag", $"This is {player.Name}'s bag");
player.Inventory.Put(sword);
player.Inventory.Put(touch);
player.Inventory.Put(bag);
bag.Inventory.Put(ruby);
```

-At the end, The 'LookCommand' object is set up for handling player input. The program enters a loop, continuously awaiting user input. If the user types 'quit,' the program exits the loop and terminates. Typing 'help' displays a list of available commands. For any other input, the

'LookCommandExecute' function is called to execute the entered command and display the result."

```
        Command look = new LookCommand();

        while (!false)
        {
            Console.Write("\nCommand: ");
            string _input = Console.ReadLine();
            if (_input == "help")
            {
                Console.Write(help);

            }
            else if (_input == "quit")
            {
                break;
            }
            else
            {
                LookCommandExecute(look, _input, player);
            }

        }

    }
}
```

2. How the OOP principles were used?

-The code uses the principles of abstraction, encapsulation, polymorphism, and inheritance to implement the player, items, and bag.

- **Abstraction**: The 'Player,' 'Item,' and 'Bag' classes serve as high-level abstractions, shielding users from the complexities of how players, items, and bags are implemented. This abstraction simplifies interactions, making it easy to work with these game entities.
- **Encapsulation**: The ‘Player’, ‘Item’, and ‘Bag’ classes encapsulate the player's name, description, and inventory, as well as the item's name, description, and aliases.
- **Polymorphism**: The ‘LookCommand’ class can be used to execute different types of look commands, such as looking at the player, looking at an item in the player's inventory, or looking at an item in the bag.
- **Inheritance**: The ‘Bag’ class inherits from the Item class, which means that it can be used like any other item in the game.