CSC 220 – Project #4: Adventure Begins

Due: December 5th, 11:59pm

No late submissions after the deadline

Total Points:
90 + 10 (peer eval.) = 100 pts

PROJECT DESCRIPTION

To create a text-based adventure game. The player navigates a world of interconnected rooms, picks up items, and interacts with simple characters or objects. The game world will be loaded from text files, and the player will have the ability to save and load their progress.

A sample run in shown at the end of the project details.

LEARNING OBJECTIVES:

- **Inheritance:** Model the game's entities using a concrete base class and specialized subclasses.
- **Interfaces:** Create a common action (like `Interactable`) that different objects can perform in their own unique way.
- **File I/O:** Read files to build the game world and write to a file to save player progress. You will be give some sample CSV files.
- **Collections:** Use `ArrayLists` to manage rooms, inventory, and game objects.
- **Logic:** Implement a game loop and a simple command parser.

IMPLEMENTATION DETAILS

1. Classes Needed

- **`GameEntity` (Concrete Base Class):** This is the parent of all the other classes in the game.
  - **Data members:** `String name`, `String description`.
  - **Methods:** `getName()`, `getDescription()`.
- **`Item` (extends `GameEntity`):** A `GameEntity` that can be picked up.
  - **Data members:** `double weight`
- **`Weapon` (extends `Item`):** An `Item` that can be used in combat.
  - **Data members:** `int damage`.
- **`Key` (extends `Item`):** An `Item` that unlocks something.
  - **Data members:** `int keyID` (a unique number, e.g., `101`).
- **`Potion` (extends `Item`):** An `Item` that can be consumed.
  - **Data members:** `int healAmount`.

- `Creature` **(extends `GameEntity`):** A `GameEntity` that is alive.
    - **Data members:** `int health`, `Room currentRoom`.
- `Player` **(extends `Creature`):** The user-controlled `Creature`.
    - **Data members:** `ArrayList<Item> inventory`.
    - **Methods:** `addItem(Item item)`, `dropItem(Item item)`, `showInventory()`.
- `Monster` **(extends `Creature`):** A non-player `Creature`.
    - **Data members:** `int damage`, `Item loot`.

2. The Key Interface

- `Interactable` **(Interface):** Defines anything the player can use or interact with.
    - **Methods:** `void interact(Player player)`
    - **Classes that implement Interactable:**
        - `Monster`**:** The `interact` method starts combat.
        - `Potion`**:** The `interact` method heals the player and removes the potion from their inventory.

3. The Game Classes (No inheritance needed)

- `Room`**:** The main building block of the world.
    - **Data members:** `int roomID`, `String name`, `String description`, `HashMap<String, Integer> exits`, `ArrayList<Item> itemsInRoom`, `ArrayList<Monster> monstersInRoom`.
    - **Note:** The `exits` HashMap maps a direction (e.g., `"north"`) to another `roomID` (e.g., `5`).
- `Main`**:** This class contains the `main` method, the main game loop, and the command parser.

4. Directory Structure*

* Some of these files are given to you in the project folder with detailed instructions TO DO as comments.

A sample project file structure is show below. If using VS Code, you can create a new *project* by selection *New File* -> *New Java Project…* -> *No Build Tool*. Also check out official documentation here: https://code.visualstudio.com/docs/java/java-tutorial

```
TextAdventure/
├── src/
│   ├── main/
│   │   ├── Game.java
│   │   ├── WorldLoader.java
│   │   ├── SaveManager.java
│   │   ├── interfaces/
│   │   │   └── Interactable.java
│   │   └── models/
│   │         ├── GameEntity.java
│   │         ├── Room.java
│   │         ├── Creature.java
│   │         ├── Player.java
│   │         ├── Monster.java
│   │         ├── Item.java
│   │         ├── Weapon.java
│   │         ├── Key.java
│   │         └── Potion.java
└── world/
    ├── rooms.csv
    └── objects.csv
```

A key part of this project is **File I/O**. Your game world isn't "hard-coded" in Java. Instead, it's loaded from the `world/`folder. This is a powerful concept: **it separates your game's *engine* (the Java code) from your game's *content* (the CSV files).**

Here's the step-by-step story of how the data flows.

4.1. The Big Picture: From Text to Objects

1. You run `Game.java`.
2. The `Game` class's constructor immediately creates a `WorldLoader`.
3. The `WorldLoader` reads `rooms.csv` and `objects.csv` files given to you. You should replace the contents with your own spin to the game.
4. It builds all the `Room`, `Item`, and `Monster` objects in memory.
5. It gives the *complete, populated world* (as a `HashMap`) back to the `Game` class.
6. The `Game` class creates the `Player`, places them in the starting room, and begins the game loop.

4.2. The Data Files

4.2.1. **`world/rooms.csv`**

This file is the **blueprint of your world**. It defines the layout. The first row is the header (columns) followed by the values in each row for the various rooms.

```
id,name,description,north,south,east,west

1,"Forest Clearing","...",2,-1,3,-1

2,"Deep Woods","...",-1,1,-1,-1
```

- **How it's used:** The `WorldLoader.loadRooms()` method reads this file *first*.
- For each line, it creates one **`new Room(...)`** object.

- It stores all these `Room` objects in a `HashMap<Integer, Room>`. This `HashMap` acts like a directory for your game, letting you quickly find a room by its ID (e.g., `allRooms.get(3)`).

4.2.2. world/objects.csv

This file contains the "stuff" that populates the game world.

```
type,name,description,location_id,value

KEY,"Rusty Key","...",1,101

MONSTER,"Goblin","...",3,5
```

- **How it's used:** The `WorldLoader.loadObjects()` method reads this file *second* (this is crucial—rooms must exist before you can put things in them!).
- For each line, it performs this logic:
  1. **Reads the `type` ("KEY", "MONSTER", etc.).** This is where inheritance is key! A `switch` statement decides *which* subclass to create (`new Key(...)`, `new Monster(...)`, etc.).
  2. **Reads the `location_id` (e.g., `1` or `3`).**
  3. It uses this ID to look up the correct `Room` from the `HashMap` created in the first step (e.g., `allRooms.get(3)`).
  4. It then *adds* the newly created object (the `Goblin`) to that `Room` object (e.g., `targetRoom.addMonster(goblin)`).

THE JAVA CLASSES EXPLAINED

- **`WorldLoader.java`**

This class connects the files to the code. This file is given to you. Its only job is to read the CSVs and build the object world. It doesn't know *how* to play the game; it just builds the game board.

- **`Game.java`**

- It holds the `main` method and the **game loop**.
- It gets the `HashMap<Integer, Room>` from the `WorldLoader` and stores it.
- It holds the `Player` object.
- Please refer to comments labeled "TO-DO" in the file for detailed instructions.

- **`Room.java`**

- This class is mostly a data container.
- It holds its own info (name, description) and its exits.
- Most importantly, it holds an `ArrayList<Item>` and an `ArrayList<Monster>` for all the objects currently in that room.

- When a player types `"get key"`, your `Game.java` code will tell the `currentRoom` to `removeItem("key")`. If the room finds the key, it gives the `Item` object to the `Game`, which then gives it to the `Player`'s inventory.

▪ **Interface:** `Interactable` is the key interface. For example:
  - A `Monster` is `Interactable` (you "attack" it). A `Potion` is `Interactable` (you "drink" it). Your `handleInteract` command will check if an object `instanceof Interactable` and, if so, call its `.interact(player)` method.

▪ **`SaveManager.java`**

- This class is the *reverse* of the `WorldLoader`.
- When the user types `"save"`, you need to get the `Player` object and write its most important data (e.g., current health, current room ID, and a list of all item names in their inventory) to a new file, like `savegame.txt`.
- The `loadGame` method will do the opposite: read `savegame.txt`, find the right `Room` from the `allRooms` map, and put the player in it.

Submission Requirements

- ***Draw and submit a UML diagram of all the classes* and *interface* used.** You can use free online tools such as Canva, app.diagrams.net, Lucidchart, Microsoft Visio etc.
- **One page description of your game setup explaining the Monsters, rooms, etc.**
- *Comment your code:* your name, name of the class and project no. at the beginning of program, explanations of blocks of code throughout your program.
- Submit all your code files. One group member can submit files on behalf of the team member but please make sure you list your team member names as a comment when submitting.
- You need to fill out the **Peer Evaluation form** to rate your group members and reflect on the project. This is an individual submission, and both the group members are required to submit their own form (**please do not share it with one another**). Each person should submit peer-evaluation for their team-members.

Please note: I reserve the right to adjust the grades for each individual in the group. That is, not all group members may have the same grade on the project.

Grading Criteria:

Category	Criteria for Success	Points
Inheritance	A clear, concrete base class is used (e.g., GameEntity) Multiple subclasses correctly extend the base class (e.g., Item, Creature) Subclasses add new, relevant properties or methods (e.g., Weapon adds damage.) Polymorphism is used effectively (e.g., an ArrayList<Item> holds Weapon and Key objects).	**20**

Category	Criteria for Success	Points
Interface	A well-defined interface is created (e.g., Interactable). The interface is implemented by at least **two** different classes (e.g., Monster, Potion). Each implementation provides unique logic for the interface method (e.g., Monster.interact() starts combat, Potion.interact() heals).	**16**
File I/O	**World Loading:** Game correctly reads and parses all data from .csv files (rooms, objects) to build the world at startup.**Save Game:** Program successfully **writes** the player's essential state (e.g., current room, health, inventory) to a savegame.txt file. **Load Game:** Program successfully **reads** the savegame.txt file to restore the player's state	**20**
Program Logic & Functionality	**Game Loop & Parser:** The game runs in a stable loop and does not crash on bad input. The parser correctly handles commands and arguments (e.g., go north, get rusty key). **Core Mechanics:** All main commands (look, go, get, drop, inventory) work as intended. **Interaction:** The interact command works polymorphically on different objects (e.g., attack goblin, use potion).	**24**
UML Diagram	A clear UML class diagram is submitted. Diagram correctly shows all classes, the interface, and their relationships (inheritance vs. implementation).Key properties and methods are listed for each class	**10**
Peer Eval.	Based on confidential feedback submitted by your teammates. Points reflect your individual contribution, reliability, and communication within the group.	**10**
TOTAL	--	**100**

NOTE: Code that doesn't compile or run: 0 pts

SAMPLE RUN

The following is a sample run of what the program execution may look like. It does not need to have the exact wording in your project.

```
Successfully loaded 4 rooms.
Loading game...
No save file found. Starting new game.

Welcome to the Text Adventure!
--- Forest Clearing ---
A quiet clearing. Paths lead north and east.
You see:
  - Rusty Key
  - Iron Sword
```

```
Exits are: north, east

> get iron sword
You picked up the Iron Sword.

> go north
You walk north.
--- Deep Woods ---
It is dark here. A path leads south.
You see:
  - Health Potion
Exits are: south

> get health potion
You picked up the Health Potion.

> go south
You walk south.
--- Forest Clearing ---
A quiet clearing. Paths lead north and east.
Exits are: north, east

> go east
You walk east.
--- Cave Entrance ---
A dark cave looms to the east. A path leads west.
DANGER! You see:
  - Goblin
Exits are: east, west

> attack goblin
You attack the Goblin!
You hit the Goblin for 5 damage.
The Goblin hits you for 5 damage.

> use health potion
You drank the Health Potion and healed 20 HP.

> i
You are carrying:
  - Iron Sword

> quit

Thanks for playing!
```