Project : a text-based adventure game

Consignes :

- Le projet est à effectuer par trinôme.

- Les livrables :
  — une archive nommée `nom1_nom2_nom3.zip`, contenant :
    — l'ensemble des fichiers source **organisé en packages**,
    — l'ensemble des fichiers de test (vous indiquerez de quels types de tests il s'agit),
    — un fichier `README.txt` indiquant comment installer et utiliser votre programme,
    — un rapport (nbPages<20), rédigé en français, dans lequel vous présenterez :
      — la partie documentation utilisateur indiquant comment utiliser votre jeu (version détaillée du README),
      — la partie documentation développeur, présentant la conception de votre projet. En particulier, vous présenterez le diagramme de classes commenté.
      — L'organisation du groupe et la répartition des tâches (qui a fait quoi)

  À déposer sur UPdago avant le vendredi 29/11/2024 11h00

- La soutenance **en anglais** :
  — Présentation du projet (10 minutes) : un diaporama détaillant les choix de conception, les spécificités de votre projet, l'organisation du groupe, un bilan fonctionnel et une brève démo.
  — questions (10 minutes)

The goal of this project is to design and implement a small adventure game in the style of Colossal Cave Adventure[1]. This game, created in the 70s, is considered the first interactive adventure game : the program describes a situation (i.e. by a text display) and the player indicates what he wants to do by entering some text. The program analyzes the sentence entered by the user and reacts by displaying the new situation, and so on. Colossal Cave Adventure takes place in a cave, but you can create and imagine your own universe.

The main classes :

— **Some Locations**
You need to design your playing field (i.e. the map of your adventure) by a set of Locations where the player can go (for example, if the game takes place in space, each location can correspond to a planet ; whereas if it takes place in a building, each location can correspond to a room). During the adventure, the player moves from one location to another by crossing exits.

1. Some information about this game are availaible here : http ://www.rickadams.org/adventure/ and http ://en.wikipedia.org/wiki/Colossal_Cave_Adventure

— **Some Exits**
Each location contains a set of exits, allowing the player to go from the current location to a nearby one. You have to design the map of your adventure. Of course, different kinds of exits may exist (some can be always opened, other may need a key or a secret code...). So, each exit knows its neighbor location. Otherwise said, the player can try to cross any exit, but this will not be done the same way if the exit is a simple door or a door which needs a key. Note also that the game handles one-way exits (i.e. a location A may contain an exit for going into location B, but nothing guarantees that once the player is in location B there will be an exit for going back to location A).
From a given location, it will therefore be possible to go to a neighboring location provided you are able to cross the corresponding exit. All the exits of a location will be stored as a dictionary `Map<String,Exit>` allowing to associate each neighboring location with its name. To go to a neighboring location B, you will therefore have to cross the associated exit using the command `GO name_of_location_B` (see Commands section).

— **Items, Bags and Characters**
In order to make your game funnier, each location can contain some items (food, safe, weapons, magic wand...) and some characters. The hero (see. next part) carries a back-pack (which is itself an item), which can contain a given volume of things that can be carried (some items, or somes characters). the hero may interract with some things (discuss or fight some characters, or maybe with some items...).

— **Last but not least : a hero !**
The game mainly consists in giving instructions to the hero of your adventure. So do not forget to establish its characteristics (e.g. life points, list of objects he owns...) which could/should be updated during the game.
The player will have to interact with the program through a console. Your program will therefore have to analyze the standard input stream. More precisely, the program will have to analyze word by word each line of text entered by the user. In order to do this, an instance of `Scanner` instantiated from `System.in` will be used.
The program must "understand" at least the commands described in the Commands section, but you can add any other command you consider necessary for the smooth running of your game.

— **You win ! vs Game Over !**
Of course, you have to establish a goal, a quest, anything you want which can allow you to decide whether the game is over or if the hero succeeds and the player wins the game.

— **Commands**
As said before, the player interacts with the game by entering a sequence of sentences. As soon as the `end_of_line` character is entered, the entire sentence is parsed. A well-formed sentence will be of the form `COMMAND [ARGS]`, where `COMMAND` is a game command, and `ARGS` is a list of (possibly empty) words.
Your game must handle (at the beginning) at least the commands `GO`, `HELP`, `LOOK`, `ATTACK`, `TAKE`, `USE`, `QUIT`. You are of course free to add as many commands as you wish. Moreover, during the course of the game, the list of commands the hero can perform will evolve. For example, the hero will be able to perform new actions (detect traps, heal himself...) or lose the ability to perform certain actions. Finally, your design should allow you to add new possible commands in a simple way (in the sense that it should involve as few modification of code as possible).

Here is a quick description of how the basic commands work in a console :

- `GO location` : the `GO` command is followed by the name of the neighbor location the player wants to go. In case the `location` exists and the exit can be crossed, the hero goes there, otherwise he stays in the same room. In each case, a display will indicate what happens.
- `HELP` : indicates the set of availaible commands.
- `LOOK [arguments]` : displays a description of the the current location if no argument is given. In case a list of arguments is provided, a display of all arguments that can be observed is given.
- `TAKE argument` : add (if possible) the argument to the hero's items
- `QUIT` : quit the game.
- `USE arg1 [arg2]` : uses the object arg1. In case a second argument is given, the first one is used with the second. For example, `use gun bullet` may load the gun, which can be used after that.

Possible Add ons

- Handling exceptions,
- Save/Load a game (sérialization)
- Use of a countdown (thread). You may find a relevant use (triggerring something, limited time to win the game...)