Task 12 - Spike: Command Pattern

CORE SPIKE

Context: A text-based adventure game can create an immersive experience where a player feels like the game "understands" them. One part of this is a robust way to process user input (text commands) and turn them into game world actions. The skills used to create good text parsers and the management of commands are also very useful in many other game types and general software projects.

Knowledge/Skill Gap: The developer needs to know how to create, for a text-based adventure game, a text command parser that is robust and accepts typical human variations. It should include an effective way to specify, manage and extend commands.

Goals/Deliverables:
[CODE] + [SPIKE REPORT]

Building on the work of earlier Zorkish tasks and related spikes, create a modular and robust text command parser and command manager for the game suitable for the Zorkish: Phase 2 game. (Refer to the game specification document on the subject website for details).

NOTE: This time you ARE combining your earlier Zorkish-related spike code to create an early some-what working version of the Zorkish game. Still, try to stay focused on the spike "gap" – a working command manager.

The commands are limited (do not change the game world at all), but are the essential building blocks for the Zorkish command processing game engine. Your spike outcome code must be able to demonstrate the following:

1. The **loading of an adventure file** (text format) from the "Select Adventure" stage (state pattern).

2. The adventure file format needs to include support for game locations (earlier spike) and extended to support NEW **game entities** (rocks, flowers etc).
    - You will need to add these new "entities" to the world graph work you did earlier.
    - Commands will look at, but will not move or change, these entities.

3. A robust **command processor** supporting a minimum of the following commands
    - GO to change location. (Use an alias later to remap "MOVE" to GO.)
    - HELP to lists known commands and their syntax details.
    - INVENTORY to list what the player has.
    - LOOK, LOOK AT (but not LOOK IN yet) to show details of location or entities.
    - ALIAS to remap commands to alternative command strings.
    - DEBUG TREE to show debug-useful details of the game graph world and entities.

4. A **UML diagram** that matches your final command pattern-related classes. Include this in your spike report.

Recommendations:
- Start by create the new adventure file that contains the additional game entities. Update your adventure loading code) so that your game world (a graph) supports entities (like rocks, etc). Each location will need a collection of entities (even if it is empty).
- Research the "command pattern". You'll probably want Command subclass objects and a CommandManager instance that can "run()" (or "execute()") one of the known command objects.
- Avoid creating new Command instances every time a command is (re)run.
- If you use "new" to create objects, do you also "delete" them later? Check!!