Individual Programming Assignment 1

The first programming assignment will be to program the game of **Zork**. Zork was one of the first interactive role-playing, text based adventure computer games. You will design a simpler variant of this game which will be able to read as input an XML file with complete information of a specific adventure and create the set of objects to interact with the player in that environment.

Example games that you can play can be found at http://thcnet.net/error/index.php and http://www.xs4all.nl/~pot/infocom/zork1.html. Since this is the first large C++ or Java program for many of you, you must write the program "from scratch."

- Do not download an open-source program and modify it for this assignment.
- This specification is given "as it is". If anything is incomplete, inconsistent, or incorrect, please explain your interpretation or improvement of the specification in a document that you turn in with your assignment.
- One of the goals of this assignment is to provide you with an opportunity to learn how to handle an imperfect specification, as all specifications are eventually found to be.
- Having said that, you may ask the course consultant about what things mean, and more importantly you may discuss *any part of this assignment with your classmates or others*, including tools, algorithms, what a pain it is, and so forth. *You must write your own code however -- code sharing is strictly forbidden*.

Commands in the game

- **n**, **s**, **e**, **w** - movement commands to put the player in a different room. If a room is bordered in the direction indicated, the description of the new room is to be printed to the screen. Otherwise print "Can't go that way."
- **i** - short for "inventory", lists all items in the player's inventory separated by commas (if more than one). If there are no items in the inventory, print "Inventory: empty".
- **take (item)** - changes item ownership from room or container to inventory. If successful print "Item (item) added to inventory". (Hint: this can be written as shortcut for put command)
- **open (container)** - prints contents of container in format "(container) contains (item), (item), ..." and makes those items available to pick up. If empty, should output "(container) is empty."
- **open exit** - if room is of type exit prints "Game Over" and gracefully ends the program.
- **read (item)** - prints writing on object if any available, else prints "Nothing written." if command is executed on an existing item in the player's inventory that does not contain writing.
- **drop (item)** - changes item ownership from inventory to present room and prints "(item) dropped."
- **put (item)** in (container) - changes item ownership from inventory to declared container if container is open and prints "Item (item) added to (container)."
- **turn on (item)** - activates item if in inventory printing "You activate the (item)." and executing commands in "turnon" element.
- **attack (creature) with (item)** - prints "You assault the (creature) with the (item)." and executes "attack" elements if item matches creature's "vulnerability" and existing conditions are met.

Behind the Scenes Commands

- **Add (object) to (room/container)** - creates instance of object with specific owner (does not work on inventory)
- **Delete (object)** - removes object references from game, but item can still be brought back into the

game through the 'Add' command. Rooms will have references to them as 'borders' removed, but there is no means at present for adding a room back in.
- **Update (object) to (status)** - creates new status for object to be checked by triggers
- **Game Over** - ends game with a declaration of "Victory!"

Order of Operations

When the user enters a command:

1. Check if command is overwritten by any triggers.
2. Execute command if not overwritten.
3. Check if effects of command activate a trigger.
4. If so, continue to check if new status effects activate additional triggers ad infinitum.

Objects

If element is followed by [ ] there may be multiple objects.

- Room - may contain the following elements: name, status, type, description, border[ ], container[ ], item[ ], creature[ ], trigger[ ]. Type is assumed 'regular' unless specified.
- Item - may contain name, status, description, writing, status, turn on, and trigger[ ]. If item has turn on element and "turn on" command is issued by user, action elements in 'turn on' are to be executed if any given conditions are met.
- Container - may contain name, status, description, accept[ ], item[ ], trigger[ ]. If an 'accept' element is added, only specific items may be put into the container and the container need not be opened to insert them (cannot be opened until one of those items is inserted, in fact).
- Creature - may contain name, status, description, vulnerability[ ], attack, trigger[ ]. If "attack" command is issued by user with creature's vulnerability, action elements in 'attack' are to be executed if any given conditions are met.

Special "Objects"

- Triggers - contains one or more condition including special conditions of type command (all of which need to be satisfied for the corresponding actions to take place), type, one or more print, and one or more action. Type is assumed to be 'single' (only used once) unless specified as 'permanent' (unlimited use). Order of execution is to output any 'print' action first, then other 'action' commands in the order given in the file.
- Possible conditions: owner, status
- Possible commands are any user command, recognized for the entire string (i.e. take sword). The trigger will pass the 'command' portion of of it's condition if there is no command element or if any one of the command element's contents are matched.
- Owner - will have object, has, and owner elements. Status - will have object and status elements.
- Also, context is very important. Only triggers in the present context of the game should be checked. This includes the inventory, the present room, and any items, containers, creatures, and items in containers in the present room. The actions those triggers perform, however, can act on any object in the game.

XML Formatting

Root element:

Each type of object will have elements associated with their given descriptions above with the addition of a 'name' element.

Triggers will have the additional complexity of containing a condition and action elements, with the condition having three additional elements to create an 'if' statement of the form "if (object) is/isn't in (owner)" with the is/isn't being determined by the element.

Grading (total 100 points)

I will ask you do demo these actions on your program during the time you set up to do this.

- 24: rooms - movement between entrance, regular room(s), and exit
- 24: items - take/drop, read, turn on, and add/remove
- 16: containers - take, put in, open, restrictions (accepts element)
- 8: creatures - attack and add/remove
- 28: triggers - permanence and activation with each other object type

NOTES

Commands are case sensitive.

Triggers always take precedence over default actions, and are only relevant in their given contexts (ie trigger associated with a room should not be tested if player is not in that room). When testing for triggers, test only triggers of objects in your present context (present room, objects in present room, objects in containers in present room, inventory), however, they can affect any object in the map.

Initial room will always be named "Entrance".

If an error message is not specified, the general error message is "Error".

Tips

A good structure might have classes corresponding to each object type, with all objects inheriting from a common class which can be searched for triggers

The program should be clearly divided between creation and play. For creation, you will find there are three basic XML parsing interfaces for Java. Introductions to XML parsing can be found here and here.

Check Blackboard for updates and don't be afraid to ask questions.

Sample Run Through

Here's a sample XML File: sample.txt.xml.

And the corresponding run with output: RunThroughResults.txt.

Note: in the RunThroughResults.txt file the ">" character is to depict input and should NOT be included in your project! It is for ease of reading only!!!

Sample pack for further testing: samplepack.zip.