CSCI 345
Assignment 4
Due May 29, 2013

Introduction

For this assignment you are to add features to the adventure engine from Assignment 3.

Problem Statement

For this version of your adventure engine, you are to add the following three features:

1. Change `messageOut` from PrintStream to a new class.
2. Add GameItems to the game.
3. Add Events to the command interpreter.

In addition, I want you to update the UML diagram you did in Assignment 2 to reflect the class structure you have implemented.

New HardCodedGame

I am providing you with a new HardCodedGame class. This implements the game I demonstrated in class. This new class is an expansion of the prior HardCodedGame class that uses all the features that are described above. In order to test intermediate stages of your work, I suggest that you comment out the new features in HardCodedGame until you are ready to implement the corresponding features. I have provided comments in HardCodedGame as to which sections are related to which new features.

Change messageOut

The `messageOut` global is currently a PrintStream. Create a new class for messageOut. This class should work like a PrintStream, except that long messages are broken at spaces into multiple lines.

The constructor for this class should include a parameter specifying the maximum length of a line of output. You can assume that this number is greater than 40 and less than 200.

This new class should ensure that no output line is longer than the specified maximum line length characters. Messages should be divided at spaces between words so that no more than the given maximum number of characters are output on a single line. Finally, a new line character ('\n') in an output message indicates the current line should end and a new line started. This is designed to allow multiple messages to be output and appear on output as a single message. See the example below where room descriptions and game item descriptions are concatenated.

The class must provide the following methods (these are used by HardCodedGame.java):

- `print(String s)` – add s to the output
- `println(String s)` – add s to the output followed by a new line ('\n')

- `println()` – same as println(""), that is, just add the new line
- `printf(String format, Object... args)` – This is the same idea as the printf for PrintStreams. The static method String.format should be used to perform printf style formatting and return the result as a String. Add the resulting string to the output. (Read the Java documentation for String.format at http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#format%28java.lang.String,%20java.lang.Object...%29 .)

  Useful Java hint: The following works:

```
void printf(String format, Object... args) {
    print(String.format(format, args));
}
```

Here are some additional functions you might need as part of your implementation:

- `flush()` – This flushes any residual saved output to the underlying PrintStream. Note that flush should not output a line ending character.
- `freshline()` – This positions the output at the beginning of a line. (If the output is already at the beginning of a line, this does nothing.)
- `resetline()` – This resets the class to assume that output is positioned at the beginning of a new line. (This can be used after receiving user input.)

Here is one way to implement this:

1. The constructor for the new class should take two arguments:
   - A PrintStream which is where output from the buffer should be sent (see next item).
   - An int which is the maximum line length. Once set, the maximum line length does not change.
2. As part of the constructor for your command interpreter, you should construct an instance of your new class and assign that instance to the messageOut attribute in Game Globals, replacing the current assignment of the parameter out to messageOut. The constructor for your new class should use the out parameter for its output stream and 72 for the maximum line length.
3. When data is output, observe the following rules:
   - Lines should be broken at spaces. Do not break a line in the middle of a word.
   - The length of an output line should not exceed the maximum line length unless there is a single word of output which exceeds the maximum line length. If there is a single word exceeding the length, output just that word.
   - Don't output extra spaces. In particular, make sure that when you break a line that there are no extra spaces at the end of the line and that the beginning of the new line doesn't start with a space.
   - If there is a new line ('\n') in the buffer, that ends a line. The remainder of the buffer is the start of a new line.

- If the output ends with non-white space, that output needs to be retained pending the next output. This is needed because the following output might extend the length of the final word or add punctuation to that word.
- Don't output the new line ('\n') character to the underlying Printstream. When outputting data, use either a println function on the underlying PrintStream or the %n formatting code in printf to indicate an end of line. (This has to do to with the differences between Windows and other operating systems with respect to line ending characters. You can find out what the line ending sequence for the system you are running on with the call System.getProperty("line.separator").)

4. Here's a proposed approach to implementing the class:
    - The class should contain a buffer which is an instance of StringBuilder. (Read the Java documentation on StringBuilder at http://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html.) Whenever print, println, or printf is called, the output string it is added to the buffer.
    - After every print, println or printf, scan the buffer:
        - If a newline is encountered, output a new line (see comment above) and reset the internal status to the beginning of a new line.
        - If any other whitespace (see Character.isWhitespace) is encountered, set a flag indicating that whitespace is needed to separate the prior output from the next output.
        - If a non-whitespace is encountered, scan forward until whitespace or an end of line is found.
        - If whitespace is found, the text just scanned is a word to be output. Output the word. If the flag indicating a whitespace separator is set, a space is needed before the word. If the word, with a space if needed, would overrun the end of line, then start a new line (and remember to reset the whitespace separator flag).
        - If the end of buffer is found, leave the final word in the buffer to be output following a subsequent command.

Add GameItems

The purpose of this change is to add items to the game that the player can pick up, move around, and manipulate in other ways. Here is what you need to do to implement this change:

1. Add a new class GameItem. GameItem has the following attributes:
    - A name which is a String.
    - A VocabItem giving the words which can be used to identify the item.
    - Three descriptions, all of which are Strings:
        - A short name which is the inventory description, for example, “gold coin”.
        - A “here is” description which is used together with the description of a location, for example, “There is a gold coin here.”

        - An "examine" description which is used to describe the item in more detail, for example, "This is a beautiful gold US double eagle dated 1900."

2. Add an interface IGameItem. IGameItem has the following four methods:
    - `boolean match(IWord w)` – return true if the Word w is in the VocabItem for the GameItem
    - `String getInventoryDesc()`, `String getHereIsDesc`, and `String getLongDesc()` – return the corresponding description for the GameItem
3. GameItem must implement IGameItem.
4. Add an additional method `IGameItem makeGameItem(String name, IVocabItem vocab, String inventoryDesc, String hereIsDesc, String longDesc)` and a corresponding method in your GameBuilder class. This method should construct a new GameItem and return it, similar to the other "make" methods in IBuilder.
5. Add a static attribute `allItems` to Game Globals which implements the interface `Collection<IGameItem>` and contains all the GameItems. Make sure that GameItems get added to `allItems` when `makeGameItem` is called.
6. Add Containers. Players and Rooms are Containers. Containers must implement the following methods:
    - `void addItem(IGameItem item)` – add the given item to the container
    - `void removeItem(IGameItem item)` – remove the given item from the container
    - `boolean contains(IGameItem item)` – return true if the given item is in the container
    - `Collection<IGameItem> getContents()` – return a collection of all the items that are in the container

    These methods also need to be part of the IPlayer and IRoom interfaces. The easiest way to accomplish this is to create an IContainer interface and have IPlayer and IRoom interfaces extend the IContainer interface.
7. Finally, modify the output from Player.lookAround (and, if needed, the output from the Player arriving in a room) so that it outputs the description of the room and the "here is" description for any items that are found in the room.

Add Events

1. There are four kinds of events:
    - INIT which is signaled only at the start of the game,
    - MOVE which is signaled when thePlayer moves,
    - COMMAND which is signaled each time an Action is executed by the command interpreter, and
    - EXIT which is signaled when the game is to be exited.
2. For this version of the game, you should provide an Event class which is an Enumeration. This enumeration should have four instances: INIT, MOVE, COMMAND, and EXIT.

Aside: Ordinarily, you would have an Event class with subclasses that provided specific information about each event. However, for this example, no additional information is required so the enumeration type is sufficient.

3. Your ICommandInterp interface, and, of course, you command interpreter class needs a method:

```
public void queueEvent(Event e);
```

   This method will be called when an event (e) is to be added to the command interpreters event queue.

4. You should modify your run method in the command interpreter to queue an INIT event before doing any other processing.

5. You should modify your moveOnPath method in the Player class to queue a MOVE event when the player's location changes. You should also modify this so that the Player's lookAround method is not automatically called. The event handler for the MOVE event will do that.

6. HardCodedGame.java defines handlers for all four types of events. You can check to code there to see what happens with each event.HardCoded Game calls the following method:

```
void makeHandler(Event event, HandlerMethod handler);
```

   This method will need to create a Handler object (similar to the Action object). I have provided the code for the HandlerMethod interface (similar to the ActionMethod interface).

   This method needs to be added to both the IBuilder inteface and your Builder class.

   Similar to Actions, one of the side effects of a call to makeHandler needs to be to add the created Handler object to some sort of allHandlers attribute that is available to the command interpreter.

7. The run method in the command interpreter needs to check after each command to see if there are any events. If there are events, the events are processed one at a time. All handlers for that Event are called with the event in question. Here's some pseudo-code for the new version of the run method:

```
public void runEvents() {
      while (! eventQueue.isEmpty()) {
            Event e = eventQueue.remove(0);
            for (Handler h : allHandlers) {
                  if (<<h handles the event e>>) {
                        h.doHandler(e);
                  }
            }
      }
}

public void run() throws IOException {
      queueEvent(Event.INIT);
      while (!exit) {
            runEvents();
```

```
                if (exit) {
                    break;
                }
                runOneCommand();
            }
        }
```

Example

Here's some sample output from my version of the program. (I've provided my version of the program as Assign4.jar. You can run it using `java -jar Assign4.jar`.) This version is slightly different from the one I showed in class. It counts both the number of commands issued and the number of moves made.

```
Welcome to your adventure. Have a good game.
You are on a balcony facing west, overlooking a beautiful garden. The
only exit from the balcony is behind you.
? go in
You are in the north end of the Big Room. The room extends south from
here. There is an exit to the outside to the west. There is a gold coin
here.
? go south
You are in the south end of the Big Room. The room extends north from
here. There is a door in the east wall.
? go east
You have entered the ballroom. The room has a high ceiling with two
magnificent crystal chandeliers which illuminate the room. The floor is
a polished wooden parquet with flowers inlaid around the edge. The north
wall is lined with mirrors while the south wall has large windows which
look out on a beautiful garden. There is a door in the west wall of the
room. There is a piece of paper here.
? get paper
You're now carrying a piece of paper.
? go west
You are in the south end of the Big Room. The room extends north from
here. There is a door in the east wall.
? go north
You are in the north end of the Big Room. The room extends south from
here. There is an exit to the outside to the west. There is a gold coin
here.
? get coin
You're now carrying a gold coin.
? look around
You are in the north end of the Big Room. The room extends south from
here. There is an exit to the outside to the west.
? go south
You are in the south end of the Big Room. The room extends north from
here. There is a door in the east wall.
? inventory
You are carrying a gold coin, a piece of paper.
? examine coin
```

```
It's a US golden double eagle.
? drop coin
You've dropped a gold coin.
? examine paper
It's a piece of paper with some writing on it.
? read message
The message says, "Enjoy your game."
? inventory
You are carrying a piece of paper.
? drop paper
You've dropped a piece of paper.
? inventory
You are not carrying anything.
? magic
You are in the magic workshop. There are no doors in any of the walls.
? magic
You are in the south end of the Big Room. The room extends north from
here. There is a door in the east wall. There is a piece of paper here.
There is a gold coin here.
? quit
You used 19 commands and made 8 moves.
Hope you enjoyed your game. Come back and play again.
```

What to Submit

This assignment is due by midnight, May 29.

You are to submit three things:

1. The source for your solution. If you are using Eclipse, you can zip the contents of the `src` directory from your project.
2. A text file showing the results of your test(s) of the version of the game you are submitting.
3. A UML diagram showing the class structure that you implemented in your program.

Make a zip file with all three items and submit that zip file on moodle.

Grading

To Be Supplied.