RPG On Watch:
Using the Observer Design Pattern

Objectives

- Implement a set of classes and interfaces that can represent different RPG (Role-Playing Game) characters and their behaviors when encountering various situations when on watch or sleeping in the wilderness.
- Implementation should utilize the Observer Design Pattern as discussed in class
- Test that the design pattern is functioning properly through assembling an adventuring party and subjecting them to a night of encounters in the wilderness.

Background

You have been asked to expand your work on the fantasy RPG video game. At this point, you have created four fantasy RPG character types: Knight, Barbarian, Ranger and Wizard. Now, you are tasked with modeling a dangerous night in the wilderness for your party. Each party member has to take two turns watching for danger during the night. Those party members who are asleep are oblivious to any dangers and each party member that is awake will have a different attitude towards the subject of the encounter. You decide to continue to use inheritance, interfaces, polymorphism and established software design patterns to make your code easier to read, use, and maintain. In particular, you decide to use the new Observer design pattern you learned in your computer science class to allow characters to subscribe for notifications of encounters that are taking place.

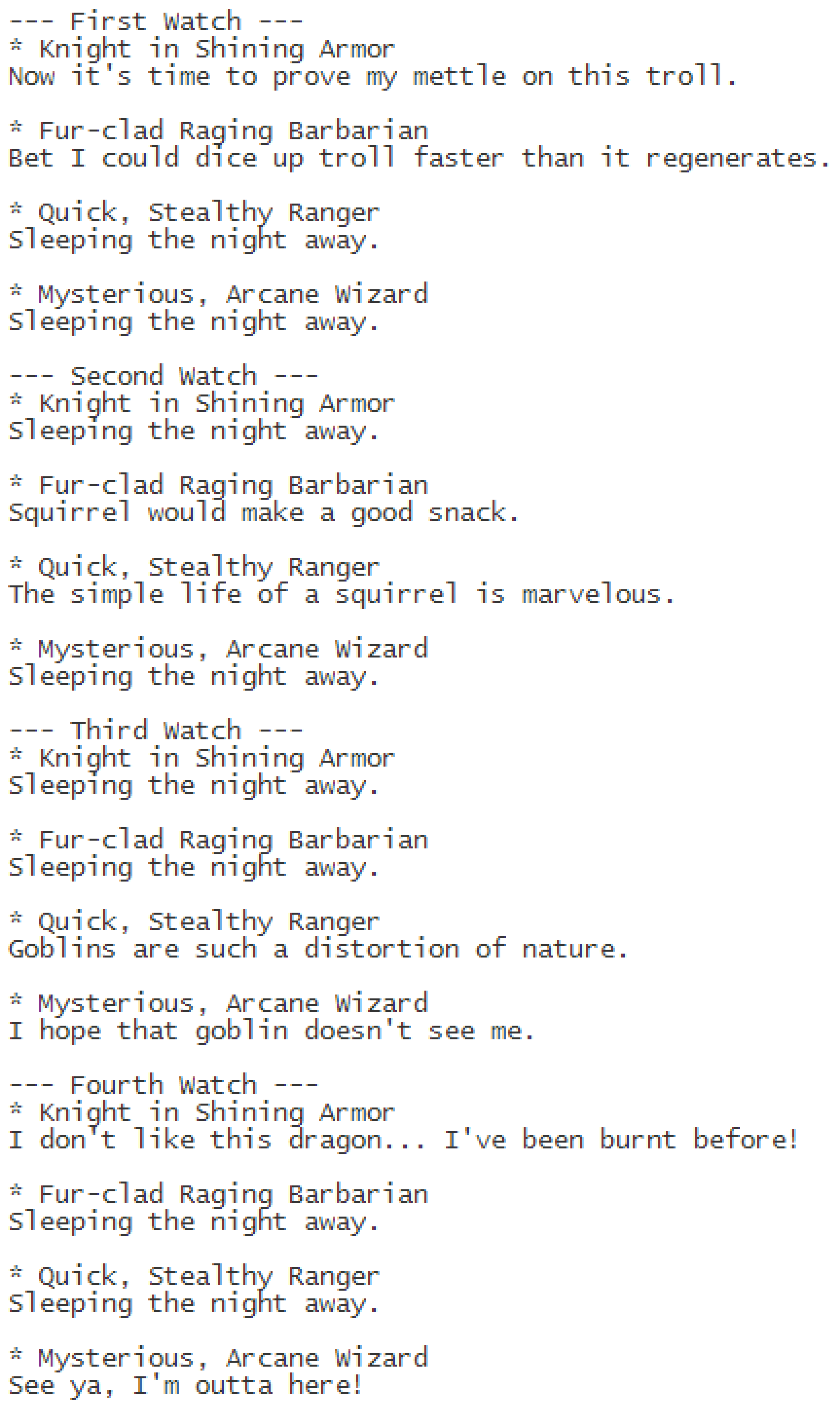
You have the basic classes for your four fantasy RPG character types: Knight, Barbarian, Ranger and Wizard. Each inherits from a base Character class that has been supplied with some method stubs and the declaration of the interface contracts: `OnWatch` & `DisplayAttitude`. The `OnWatch` interface has also been supplied and represents the “Observer” interface of the pattern. The `DisplayAttitude` interface will be used to print the current character attitude to the console. The `LongRest` interface represents the “Subject” interface of the pattern and is implemented by the `NightInTheWilderness` class.

Ultimately, you have been given the `Adventure` class to test your implementation. There is no reason to modify this class at all. You will know you have completed a successful implementation when the application runs successfully and produces the output given to you.

Tasks

1. Look over all the starter files carefully and make sure you understand how each class, interface and enumeration relates to the Observer pattern as a whole.
2. You will **not** be modifying the `Adventure` class at all, though any errors observed will help you troubleshoot your other classes.
3. Start by implementing the `OnWatch` and `DisplayAttitude` interfaces. Similar to the in-class example, you will want to store the String containing the attitude and the reference to the `LongRest` object in class member variables.
4. The `Character.goToSleep()` method should unsubscribe the character from the `LongRest` “notifications” and set the attitude to the String “Sleeping the night away.”
5. The `Character.stayAwakeForWatch()` method should subscribe the character to the `LongRest` “notifications”.

6. Each character will implement the `observeEncounter()` method differently because each will have their own unique attitude (as represented by a String) depending upon what is encountered. The following are the attitude values:
    - Knight
        - Nothing – “Enjoying some quiet time polishing my sword.”
        - Squirrel – “I hope the squirrel doesn't encounter an owl.”
        - Goblin – “I bet I could scare the goblin away with a creepy noise.”
        - Troll – “Now it's time to prove my mettle on this troll.”
        - Dragon – “I don't like this dragon... I've been burnt before!”
    - Barbarian
        - Nothing – “Staring into the fire... feeling hungry.”
        - Squirrel – “Squirrel would make a good snack.”
        - Goblin – “I could slice that goblin in two with one blow.”
        - Troll – “Bet I could dice up troll faster than it regenerates.”
        - Dragon – “Wonder how bad the dragon's fire hurts!”
    - Ranger
        - Nothing – “The stars are beautiful.”
        - Squirrel – “The simple life of a squirrel is marvelous.”
        - Goblin – “Goblins are such a distortion of nature.”
        - Troll – “I hate trolls! Time to light up my arrows.”
        - Dragon – “My arrows cannot penetrate the dragon scales!”
    - Wizard
        - Nothing – “Reading my spell book.”
        - Squirrel – “I could transform that squirrel into a raven.”
        - Goblin – “I hope that goblin doesn't see me.”
        - Troll – “Oh no, a troll! I wish I had prepared my fireball spell!”
        - Dragon – “See ya, I'm outta here!”
7. The `display()` method should print the current attitude of the character to the console.
8. After implementing the character functionality, it is time to work on the `NightInTheWilderness` class and implement the `LongRest` interface.
    - Similar to the in-class example, you will want to store the current `Encounter` state and the `ArrayList` of “observers”.
    - The `stayAwake()` method should add the character to the “observers” list if the character is not already in the list
    - The `goToSleep()` method should remove the character from the “observers” list.
    - The `setEncounter()` method should store the current `Encounter` and publish a notification to each of the observers.
    - The `somethingHappened()` method publishes a notification to each of the observers.
9. Compile and run the `Adventure` program and confirm that you get the following results:



```
--- First Watch ---
* Knight in Shining Armor
Now it's time to prove my mettle on this troll.

* Fur-clad Raging Barbarian
Bet I could dice up troll faster than it regenerates.

* Quick, Stealthy Ranger
Sleeping the night away.

* Mysterious, Arcane Wizard
Sleeping the night away.

--- Second Watch ---
* Knight in Shining Armor
Sleeping the night away.

* Fur-clad Raging Barbarian
Squirrel would make a good snack.

* Quick, Stealthy Ranger
The simple life of a squirrel is marvelous.

* Mysterious, Arcane Wizard
Sleeping the night away.

--- Third Watch ---
* Knight in Shining Armor
Sleeping the night away.

* Fur-clad Raging Barbarian
Sleeping the night away.

* Quick, Stealthy Ranger
Goblins are such a distortion of nature.

* Mysterious, Arcane Wizard
I hope that goblin doesn't see me.

--- Fourth Watch ---
* Knight in Shining Armor
I don't like this dragon... I've been burnt before!

* Fur-clad Raging Barbarian
Sleeping the night away.

* Quick, Stealthy Ranger
Sleeping the night away.

* Mysterious, Arcane Wizard
See ya, I'm outta here!
```

10. Document your project:

- Write a plain-text README file, called README, following the format described below.
- Comment your code appropriately; assume someone may be looking at your code and not know what you are trying to accomplish.

Documentation: README

Update the plain text file called README and write your name and class section on the top.

1. Fill out the **Overview** section
   Concisely explain what the program does. If this exceeds a couple of sentences, you are going too far. I do not want you to just cut and paste, but paraphrase what is stated in the project specification.
2. Fill out the **Compiling and Using** section
   This section should tell the user how to compile and run your code. It is also appropriate to instruct the user how to use your code. Does your program require user input? If so, what does your user need to know about it to use it as quickly as possible?
3. Fill out the **Discussion** section
   Think about and answer the following reflection questions as completely as you can. You may not have an “earth-shattering” reflection response to each one but you should be as thoughtful as you can.

   Reflections...
   - ➢ What problems did you have? What went well?
   - ➢ What process did you go through to create the program?
   - ➢ What did you have to research and learn that was new?
   - ➢ What kinds of errors did you get? How did you fix them?
   - ➢ What parts of the project did you find challenging?
   - ➢ Is there anything that finally "clicked" for you in the process of working on this code?
   - ➢ Is there anything that you would change about the lab?
   - ➢ Can you apply what you learned in this lab to future projects?
4. Fill out the **Testing** section
   You are expected to test your projects before submitting them for final grading. Pretend that your instructor is your manager or customer at work. How did you ensure that you are delivering a working solution to the requirements?

Grading

Points will be awarded according to the following breakdown:

Tasks	**Points**
Documentation: README file, comments	10
Observer pattern functionality within the classes	30
Quality - code formatting, naming conventions, style, etc.	10

Required Files

Submit the following files within a zip file named: **Lab03_RPGOnWatch_*LastName*_*FirstName*.zip**:

- `README`
- `Adventure.java`
- `Barbarian.java`
- `Character.java`
- `DisplayAttitude.java`
- `Encounter.java`
- `Knight.java`
- `LongRest.java`
- `NightInTheWilderness.java`
- `OnWatch.java`
- `Ranger.java`
- `Wizard.java`