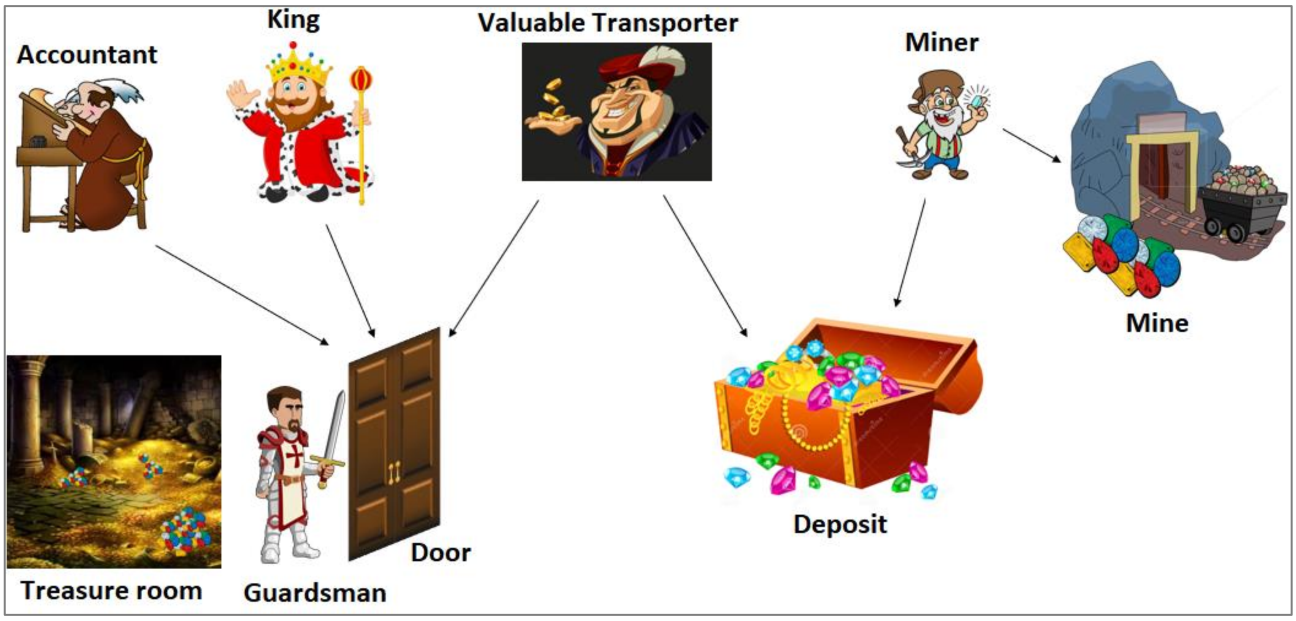
Assignment 4, SDJ2
(Singleton, Proxy, Adapter, JUnit, Producer-Consumer, Readers-Writers)

The assignment:

You must design and implement the Kingdom simulation shown below. The ultimate goal is to get valuables, so the king can throw parties.



The King has a Treasure room (lower left corner) with the door guarded buy a Guardsman. The valuables in the Treasure room comes from a Mine (upper right corner) with Miners transporting their findings to a Deposit (lower right corner). Occasionally, Valuable Transporters are moving valuables from Deposit to the Treasure room and Accountants are counting all valuables in the Treasure room. When the King feels like partying, he takes valuables from the Treasure room if there are enough for a party.

Requirements

- Threads to simulate the actors: King, Accountant, ValuableTransporter and ValuablesMiner.
- Singleton to log any action, e.g. when an actor waits or perform a job.
- A class to create valuables, e.g. Diamond, GoldNugget, Jewel, Ruby, WoodenCoin, etc, for the Miner.
- JUnit testing an ArrayList to be used in the Deposit.
    - Documentation: http://ict-engineering.dk/javadoc/MyArrayList/
    - Jar file: http://ict-engineering.dk/jar/MyArrayList-0.1.jar
- Adapter for the ArrayList in the Deposit. The Deposit is a blocking queue, you should use the provided ArrayList as the basis for this blocking queue.
- Producer-Consumer for the Deposit with the Miners and Valuables Transporters being producers and consumers, respectively. I.e. Miners insert valuables into the Deposit, and Valuable Transporters take them out.

- Readers-Writers for the Treasure room and Guardsman.
- Proxy between the Treasure room and the Guardsman.

This is quite a long list and therefore you should break it up into parts, finishing one part at the time.
One way of working could be:

1. JUnit testing the ArrayList is independent of the rest and could be done at any time. There are some errors, but they should not influence in your use of it. See if you can find the errors, but don't fix them.
2. Singleton could be first step. Create an Archive class (as singleton) which has a log method with a simple printout to console or file.
3. The Mine. This class has a method, which can return a Valuable. Randomly figure out which type of Valuable is returned, and make a few Valuable subclasses with value and name fields.
4. Adapter for a blocking queue Deposit delegating to the ArrayList given. I.e. use the ArrayList methods to implement a blocking queue, methods like get, remove, add, size are needed.
5. Producer-Consumer creating two Runnable classes Miner and ValuableTransporter and the Deposit as the shared resource, i.e. blocking queue. Remember to use the Singleton to print out when a Miner or ValuableTransporter are waiting and working (in class Deposit). *Note: before creating a Treasure room, just let the ValuableTransporter "throw away the valuables" clearing his list.*
    - The Miner could have a while(true) loop, in which he will get a (maybe random) valuable from the Mine, and insert this into the Deposit (the blocking queue). Thus, making the Miner the producer. Afterwards, the Miner needs rest (thread should sleep for a couple of milliseconds), and then get back to work (because of the while(true)).
    - There is quite a distance from the Deposit to the Treasure Room, so a Valuables Transporter does not want to transport only one valuable at a time. Therefore, the behaviour of the Valuables Transporter must be ***strictly*** as follows, in a while(true) loop:
        1. Generate a random number, e.g. between 50 and 200.
        2. He will then, a number of times, get the next valuable from the Deposit, and put in a local list (his valuable transport bag). This will continue, until he has a list of valuables with a total value equal to or more than the original target number.
        3. Now clear the List used to contain the valuables (currently we have no place to put them, this will come later. Print out something like "throws away valuables")
        4. Sleep for a little while
        5. Start over from step 1
6. Test this part in a main method with a Deposit, a couple of Miners and a couple of ValuableTransporters. Inspect the print outs (have them in many places to be able to closely follow the behaviours), do they behave as expected?
7. Readers-Writers and Proxy
    - Create an interface TreasureRoomDoor to be implemented by the Guardsman with methods to acquire and release read and write access.
    - Include in the interface methods to add (write), retrieve (write) and look at valuables (read), this is for the accountants, so they can count the valuables. Maybe it's just a method which returns the total value, you decide.
    - Create class TreasureRoom. Use a list as the instance variable, e.g. the ArrayList given. Implement interface TreasureRoomDoor. Whenever someone wishes to access the TreasureRoom, just allow it. The Guardsman will be responsible for the readers-writers stuff.

- Implement class TreasureRoomGuardsman. The guardsman is your readers-writers manager, which keeps track of who can access the treasure room. You decide on the readers-writers strategy.
- Now, the TreasureRoom and Guardsman share an interface. If you just use the treasure room, anyone can get access all the time at any time. If you use the Guardsman as a proxy, he controls the access with a readers-writers strategy, and then delegates to the TreasureRoom. I.e. TreasureRoom is real subject, Guardsman is proxy.
- Modify the ValuableTransporter class to add the valuables retrieved from the Deposit, thus ValuableTransporter being a 'writer' (you could use sleep to simulate that it takes time to add valuables to the Treasure room on at a time).
- Implement the Accountant as a Runnable class. The Accountant is a "reader" class with a while(true) loop in the run method.
    1. He will acquire read access
    2. Count the total sum of the valuables worth in the TreasureRoom (it may include a sleep to simulate it takes time to count the valuables)
    3. Print the total sum out (using the Singleton class)
    4. Release read access
    5. Sleep for a little while
    6. Start over. Counting is life.
- Implement the King as a Runnable class being a 'writer' wanting to take out valuables from the TreasureRoom in order to throw a party. The behaviour of the King is strictly as follows:
    1. Similar to the ValuableTransporter, the King will generate a random number, e.g. 50-150, to pay for the next party.
    2. He will acquire write access
    3. Retrieve the valuables one at a time. If the target worth cannot be met, he will cancel the party, and put the valuables back (again, it could include a short sleep to simulate it takes time to get the desired valuables)
    4. Release write access

    a. Alternatively, if the target is met, he will hold a party and "throw away" the valuables retrieved. Print stuff out.

    5. Sleep for a while
    6. Start over
- Test the full system in a main method with a Deposit, one King and a couple of each of the other actors: Miners, ValuableTransporters and Acountants

Deadline

See itslearning

Format

May be done in groups. Hand-in as a single zip-file with

- Class diagram(s) (where the different patterns and other subjects are clearly identified, e.g. by color if possible)
- Source code for all Java classes
- Related resources if used

Evaluation

Your hand-in will be registered and counts for one of the exam requirements. No feedback will be given.