Mine Pump Control System

You have been tasked to develop a mine pump control system, designed to monitor and pump flood water out of mine shafts. As underground mining operations take place far below the water table, flooding into mine galleries and shafts is an ever-present danger. Excessive flooding is clearly a safety hazard for workers, but also has profitability implications ranging from equipment damage to productivity delays, to mine closures in extreme circumstances.

The system to be developed will be required to monitor the water level in a given mine shaft using two sensors. A high water sensor that measures the maximum acceptable level of flooding in a shaft before pumping begins, and a low water sensor, which measures the minimum level of acceptable flooding and pumping stops. These sensors are used to start a mine pump. When the flooding level exceeds the level determined by the high water sensor the pump is switches *on*. When the water has been pumped out and the minimum level of acceptable flooding has been reached, as measured by the low water sensor, the pump switches *off*.

In addition to flooding mining is often hindered by methane pockets, where gas seeps into the shafts and galleries triggering an evacuation. Again this is a safety hazard, the mining staff won't be able to breathe, and even more critically, operating equipment may generate sparks which will cause the methane to ignite. Therefore the system will include a methane sensor that will be used to trigger an evacuation alarm in the presence of dangerous levels of methane (measured in N parts per million), and also switch *off* the pump regardless of the current water level.

The system is used by two key roles. The first is the *Operator*. This role is required to log in to the system with a username and password. Following a successful login the operator is able to start or stop the pump if, and only if, the water level is between the high and low sensor limits. The second role is the *Supervisor*. A supervisor must verify their security credential as per the operator above. Following a successful login they are able to switch the pump on, or off at any time. For example a supervisor could run the pump after the flood level has dropped below the level set by the low water sensor. They could also switch the pump off if the water level goes over the maximum high level of flooding. In these cases the supervisors actions override the automatic behavior of the pump. A supervisor is required to reset the pump system in order to re-establish automatic behavior.

Finally to meet Federal monitoring standards a persistent log is required to capture the following events:

- ☒ Pump switched on by high water sensor
- ☒ Pump switched off by low water sensor
- ☒ Pump switched on or off by operator or supervisor
- ☒ Evacuation alarm triggered by methane sensor

☒ The reading of the methane sensor every 30 minutes

The reading of the methane sensor (for the last 24 hours) can be read by the operator. All readings (up to 30 days) can be read by the supervisor. The supervisor also has the capability to add a note to any specific log event that occurs within 24 hours.

Answer the following questions:

1. Abstract Factory Design Pattern:

After an analysis of the market your company has decided to provide device drivers for a range of different platforms, to enable your application to communicate with each type of sensor supported by the application (high water, low water, methane sensor). These are listed as follows:

1. Linux USB Driver
2. Linux Serial Port Driver
3. Linux Wireless Driver
4. Win32 USB Driver
5. Win32 Serial Port Driver

Questions:

a. Define a class diagram that shows the Abstract Factories and Product Hierarchies necessary to implement these variations for every type of Sensor supported. Make sure each factory is also defined as a *Singleton* (why? What would be the advantage of this?)

2. Composite Design Pattern:

Your company has decided to provide an environmental simulator used to simulate events that the system may receive in order to test installations and diagnose outstanding software or hardware problems. As part of this simulator you will be required to build an object structure used to model expressions in PSL, the proprietary *Pump Simulator Language*.

a. This language has the following primitives:
   i. High Water Event represents a maximum water level reading from a high water sensor
   ii. Low Water Event represents a minimum water level reading from a low water sensor
   iii. Methane Alarm represents a dangerous methane reading from the methane sensor
   iv. Supervisor Switches Pump On represents the supervisor switching the pump on manually
   v. Supervisor Switches Pump Off represents the supervisor switching the pump off manually
   vi. Run for N minutes runs the pump for N minutes, this must follow an event from (i)-(iv)
   vii. Reset resets all alarms, switches off the pump

b. These primitives can be combined into higher level constructs called workflows. A workflow is a composite that may hold one or more primitives, or other workflow objects.

Using the composite pattern, develop the following tactical solutions:
Questions:

a. Develop a class diagram showing the components of the language and how they interrelate.
b. Develop a number of sequence diagrams that show example PSL processing scenarios.

3. **Composite & Template method Design Pattern:**
In order to handle user interactions in the presentation layer your architecture team has decided to handle individual user actions in separate *action* objects. An action will receive a user request, unpack input variables, call the appropriate business logic tier interface(s), get back the results and send them as output back to the user.
   a. An Action object must support the following functionality:
      i. StartTransaction will start a transaction every time the action object is called
      ii. endTransaction will end (and commit) the transaction when output is returned to the user
      iii. onError will rollback the transaction if an error occurs
      iv. doWorkflow will do the necessary action
   b. These actions can be nested, i.e. it is possible to have an action object made up of smaller action object. These child actions are simply executed in sequence.

Using the composite and template method patterns do the following:
Questions:

a. Develop a class diagram showing the structure you have designed
b. Develop two sequence diagrams
   i. Show a normal transaction
   ii. Show a transaction where error (and rollback occurs)
c. Hint : For the superclass in the composite you will need an abstract class not an interface. Why?
d. For composite action objects how can I add logic to the execution of nested leaf or composite actions i.e. *if X then do action Y else action Z.*

4. Adapter Design Pattern:
Many pumps typically run hot and require airflow cooling systems to keep them within safe operating temperatures. These systems are expensive to operate and so should run in step with the operation of the pump.
Questions:

a. Build a class diagram that provides a class/interface for the cooling system.This interface provides two methods:

i. Switch (on/off), turns the cooling system on or off based on the parameter, when switched off the pump will run for an additional 20 minutes in order to cool the pump down following operation.
ii. Emergency Off switches off the cooling system immediately

b. Build an adapter that will operate the pump and cooling system simultaneously
c. Model the dynamic behavior of the adapter using a sequence diagram, show scenarios that demonstrate both normal on/off operation and the behavior exhibited when the methane alarm is activated.

5. Observer Design Pattern:

A pump will generate a diagnostic message every 30 seconds that reports the following information:

- The current temperature of the pump
- The amount of water pumped (in cubic cm) since the last reading
- The current time

This information is required to be sent to a user interface, a mail message must be sent to a dedicated email inbox, and it must be placed in a dedicated pump reading database.

Questions:

a. Make the appropriate changes to your pump class, use an observer pattern to model the scenario described above using a class diagram.Produce a sequence diagram to illustrate how this pattern will work when the pump generates a diagnostic message.

6. Visitor Design Pattern:

Each sensor and pump in the system will also be required to have the following two data attributes:

a. Last Serviced the calendar data it was last serviced on
b. Next Service Due the time by which it must be serviced (an offset of N months from the Last Serviced date)

In addition a pump will have the following information:

c. Pump Location (e.g. Shaft 6)
d. Logical Pump Name (e.g. Old Ironsides)

Using the visitor pattern you are required to build a simple reporting addition that will

Questions:

a. Generate a report showing all the sensor and pump information on request from a supervisor.

7. Architectural Patterns for architectural capabilities:

For each problem scenario given below, select one *or more* architectural patterns as the basis for your final system. For each pattern selected undertake the following:

Questions:

a. Using UML *packages*, partition your existing class diagram(s) to conform to the architectural configuration suggested by your chosen patterns.
b. What additional classes will your packages require in order to meet the requirements of this architecture? e.g.: How is distributed communication managed? How are common resources efficiently shared?

1. *Fault Detection*

   Your system relies heavily upon the correct operation of hardware, specifically pumps sensors and alarms. These needs to be closely monitored to ensure that they are all responsive during normal operations.

   Questions:

   a. Design a heartbeat monitor to examine each piece of hardware in your system, update your class diagrams with the new architecture, and provide a simple sequence diagram to indicate how this will operate.
   b. Design a ping monitor to examine each piece of hardware in your system, update your class diagrams with the new architecture, and provide a simple sequence diagram to indicate how this will operate.
   c. Design an exception class hierarchy that captures and propagates faults for all types of hardware in the system.
   d. Should these designs be generic (i.e., for all types of system hardware? Or specialized to meet individual operating requirements?)

2. *Fault Recovery*

   Sensors are typically inexpensive to replace but are prone to failure (Mean Time Between Failure 8000 hours), as a consequence multiple instances of a given server type can exist within a specific mine shaft.

   Questions:

   a. Based on one of the detection strategies you designed above, specify, using class and sequence diagrams build, recovery strategies using Shadowing, Voting and Replication based architectures.
   b. Report your findings.

3. *Performance and Concurrency*

   In order to partition and handle the incoming system events, refactor your architecture to include the presence of active objects. These will be processes/threads that respond to incoming events (such as interrupt driven sensors) or those that periodically poll resources (such as sensors that are polled for readings).

   Questions:

   a. Introduce active objects into your class diagram in order to handle the dynamic aspects of your system, e.g. sensor input, alarm triggering, pump actions, user input and so on.
   b. Assign each active object a relative priority based on the following levels:
      1. DAEMON run me in the background if there are spare CPU cycles.
      2. LOW run me if there are no higher priority events waiting.
      3. MEDIUM run me if there are no HIGH priority events waiting.
      4. HIGH run me first!
   c. What happens if there are multiple HIGH priority processes waiting, how should these be handled?

d. What arbitrates who should have priority? Introduce this to your architecture. What Priority should it run at? How should it work?

4. *Security*

The application provides a role based security model, verified by a simple user/password access control scheme.

Questions:

a. Add the following to your architecture:
    1. Classes to handle the login procedure, this will validate the user based on a supplied username/password stored in a database and assign them the appropriate role. Describe the dynamic behavior using a sequence diagram.
    2. Classes to administer the security database, creating, updating, editing usernames, passwords and role.
    3. Classes to encrypt / decrypt usernames and passwords using an encryption algorithm chosen by the customer from the following list (Blowfish, DES, Triple-DES).

5. *Validity/Testability*

As a safety-critical/mission-critical application your company has decided to provide architectural hooks into the system to facilitate programmatic testing. As a consequence you will be required to undertake the following:

a. Provide a class and illustrate sequence diagram to provide recording/playback capabilities for the following interface(s):
    1. Commands sent to the pump
    2. Information sent/polled by each sensor
    3. Commands sent to the Evacuation Alarm
b. Provide specialized interfaces for your basic abstractions that allow access to all private attributes for unit test purposes. Do not allow these to be accessed by normal clients ensuring system operation (***hint*** : use two UML interfaces, one for normal operation, one for test purposes only.)