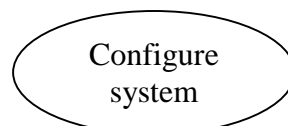
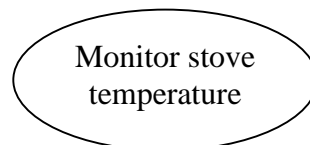
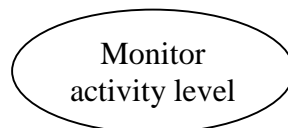
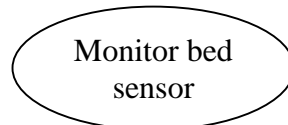


4. Consider the requirements for the eldercare monitoring system below. Draw a Use Case diagram according to the specifications listed, using associations between use cases where appropriate. (20 points)

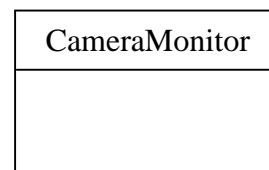
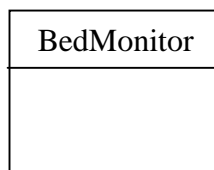
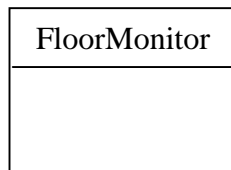
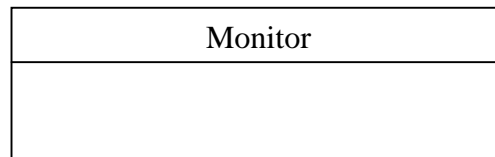
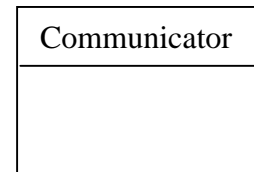
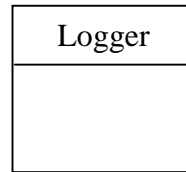
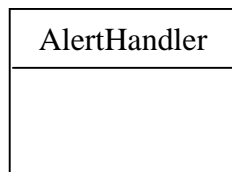
The following Use Cases are included in the system. Actors are shown in italics.

1. **Monitor for falls** – Inputs data from the *floor sensors* and the *cameras*.
2. **Fall alert** – Handles exception cases in the event of a fall detection; if a fall is detected, a page is sent to the *pager* and the alert is sent to the **Log Event** use case for logging.
3. **Log event** – Outputs a record of the event to an internal database.
4. **Monitor bed sensor** – Inputs data from the *bed sensor* and sends events to the **Log Event** use case (for pulse, respiration & bed restlessness).
5. **Monitor activity level** – Inputs data from the *motion sensors* and sends events to the **Log Event** use case.
6. **Monitor stove temperature** – Inputs data from the *stove temperature sensor* and the *motion sensors*.
7. **Stove on alert** – Handles exception cases in the event that the stove is on but no one is in the kitchen; if an alert condition is detected, a page is sent to the *pager* and the alert is sent to the **Log Event** use case.
8. **Configure system** – Handles the system configuration, receiving inputs from a *system administrator*.



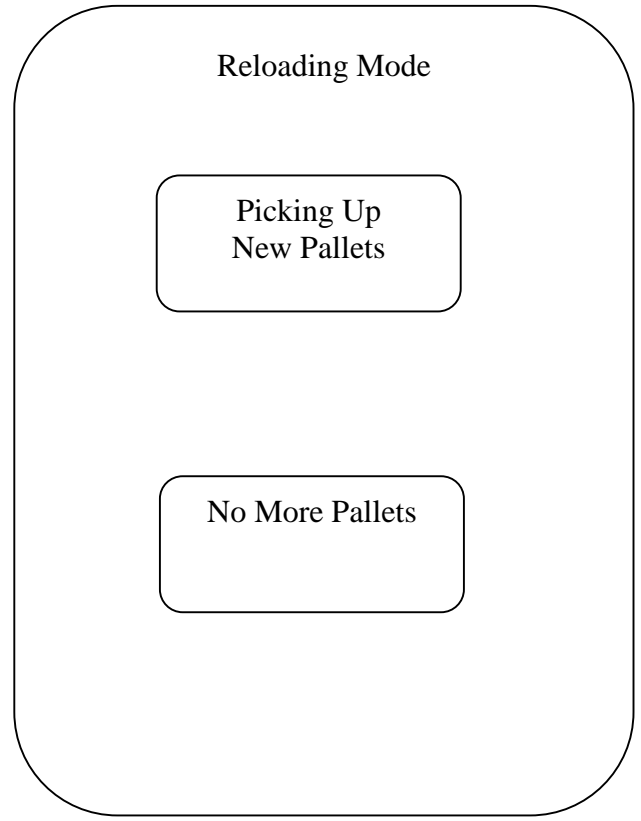
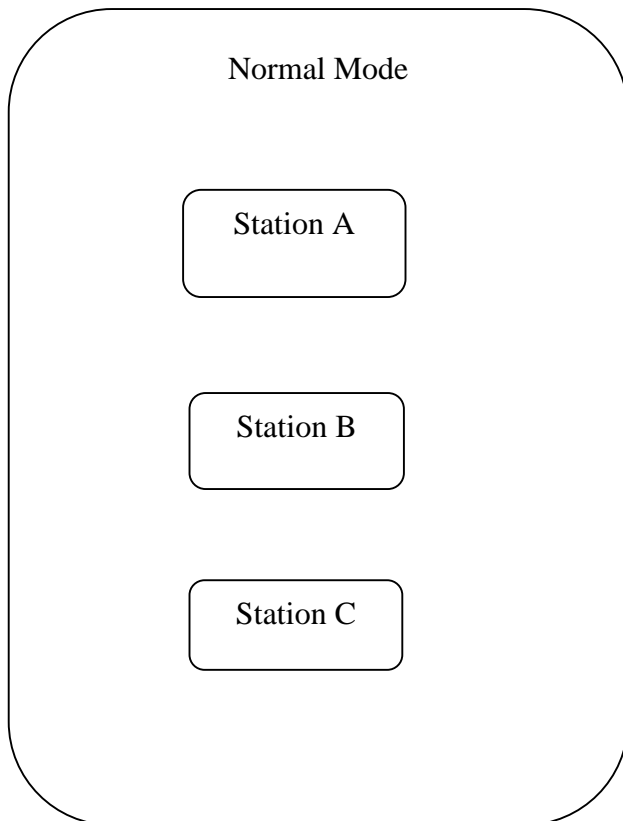
5. Given the start of a UML class diagram below, finish the diagram according to the description. Note that there are 10 classes. (15 points)

- The **Monitor** class is a parent class for the **FloorMonitor**, **BedMonitor**, and **CameraMonitor** classes.
- The **Monitor** class contains the following component classes: **ConfigHandler** (one instance to one Monitor), **AlertHandler** (one or more instance to one Monitor), **Logger** (one instance to one Monitor) and **Communicator** (one-three instances to zero or more Monitors). The **Monitor** class is responsible for creating and deleting the **ConfigHandler** and the **AlertHandler**.
- There is also a **Timer** class, which sends messages to the one or more of the **Communicator** classes and zero or more of the **CameraMonitor** classes.
- Finally, a **DBHandler** class receives messages from the **Logger** class (one **Logger** to one **DBHandler**).



6. Draw a **UML Statechart for a mobile robot controller** designed for use in a factory. Using the states given, devise appropriate **events, guards, transitions, actions, and default states** to implement the system described below. State any assumptions you have made. (30 points)

- There are 2 superstates as shown: **Normal Mode** and **Reloading Mode**.
- The robot's function is to carry pallets from the loading dock to the workstations. The robot can carry 100 pallets but can only pick up or deposit 10 pallets at a time.
- In the **Normal Mode** state, the robot moves from **Station A** to **Station B** to **Station C** and back to **Station A**. While at each station, it moves new pallets from its transport to the working area for that station. The robot can deposit 10 pallets at a time. When the station has 20 pallets, the robot moves on to the next station.
- When the robot runs out of pallets, it transitions to the **Picking Up New Pallets** state in the **Reloading Mode**, where it moves to the loading dock and loads up 100 pallets, and then returns to the **Normal Mode**, remembering where it should go next in the sequence, based on the situation when it ran out of pallets.
- If the loading dock runs out of pallets, the robot transitions to the **No More Pallets** state, where it terminates.



7. Consider the C program below with a Main Loop and an Interrupt Service Routine. **What are the values of x, y, and z (in hex) after 3 interrupts?** Show your work. (15 points)

```

main ()
{
    volatile int x = 0x00;
    volatile int y = 0x01;
    volatile int z = 0x11;

    while (1)          /* loop forever */
    {
        /* do stuff */
    }
}

interrupt void isr_timer()
{
    x++;

    y = (x + y) & 0x0f;

    z = (z << 1) | x;
}

```

	Initial value	After 1 interrupt	After 2 interrupts	After 3 interrupts
x				
y				
z				