SE 115 Introduction to Programming I

Lab No:	08
Topic:	Introduction to Classes II (Examples)

Scenario 1

Your company runs a futuristic robot factory, and you are asked to build a small software system to manage robots and their tasks.

A. Create a class named Robot.

- Add the following **private** data members (choose appropriate data types):
  - `modelName`
  - `batteryLevel`
  - `status` (example: "Active")
- Provide a parameterized constructor that initializes all data members.
- Write getter and setter methods for each data member.
- Add a method: `displayRobotInfo()` which prints the robot's model name, battery level, and current status.
- Create a separate driver class `FactoryDemo` with a `main` method:
  - Create at least two Robot objects.
  - Display their information using `displayRobotInfo()`.

B. Add operational methods to the Robot class.

- Add a method: `isBatteryEnough(int requiredAmount)` that returns **true** if the robot's batteryLevel is greater than or equal to requiredAmount, otherwise returns **false**.
- Add a method: `consumeBattery(int amount)` that decreases the battery level by the given amount.
- Add a method: `chargeBattery(int amount)` that increases the battery level by the given amount. If charging causes batteryLevel to reach at least 100, change the robot's status to "Active"; otherwise, keep the status as "Charging".
- Update the `FactoryDemo`:
  - Ask the user for an amount of battery to consume.
  - Have the robot attempt to consume that amount and observe whether the operation succeeded or failed.

C. Create a new class named Task.

- Private data members:
  - `taskName`
  - `energyCost` (amount of battery required)
- Provide a **parameterized constructor** to initialize both fields.
- Add getters for both fields.
- Update `FactoryDemo`:
  - Create one or more Task objects.
  - This time, instead of asking the user for an energy cost, check whether the robot has enough energy using the energy cost of the task objects you created.

D. Add task-processing capabilities.

- In the `Task` class:
  - Add a method `describeTask()` that prints the task name and its energy cost.
- In the `Robot` class:
  - Add a method: `performTask(Task t)` that:
    - Prints a message that the robot is attempting the task.
    - Checks if the robot's current status is "Charging". If it is, print a failure message.
    - Checks if the robot has enough battery for the task (using `isBatteryEnough()`).
    - If enough battery:
      - Consumes the required battery,
      - Prints confirmation that the robot completed the task.
    - If not enough battery:
      - Prints a failure message and does not modify battery or status.
- Update `FactoryDemo`:
  - Attempt to process a task and display robot information before and after calling `performTask()`.

E. Create a new class named TaskBatch.

- The TaskBatch class should include the following private data members:
  - `Task[] tasks`
  - `int taskCount`
- Provide a no-argument constructor that:
  - Allocates the `tasks` array with length **5** (the maximum batch size),
  - Sets `taskCount = 0`
- Write a method named `addTask(Task t)` that adds the given Task object to the array *only if* :
  - `taskCount` is less than 5. If the batch already has 5 tasks, print an error message and do not add more.

- Write a method named `getTotalEnergyCost()`:
    - Returns the sum of the energy costs of all tasks currently stored in the batch. If the batch contains no tasks, print a failure message.
- Write a method named `getTasks()`:
    - Returns all the tasks that are currently stored in the batch. If the batch contains no tasks, print a failure message.
- Write a method named `printBatchInfo()`:
    - Prints the name and energy cost of each task in the batch.
    - If the batch contains no tasks, print an appropriate message.
- Update `FactoryDemo`:
    - Create several Task objects with different energy costs.
    - Create a TaskBatch object.
    - Add at least three tasks to the batch.
    - Display batch information using `printBatchInfo()`.
    - Display the total energy cost of the batch.

BONUS: Update the Robot class to support task batches:

Add a method named `performTaskBatch(TaskBatch batch)` that:

- If the batch contains no tasks, prints an error message and returns.
- Calculates the total energy cost of the batch.
- Checks if the robot has enough battery for *all* tasks:
    - If not, prints an error message and does not modify the robot.
- If enough battery:
    - Prints a message confirming the robot is processing the batch.
    - For each task in the batch:
        - Processes the task using the `performTask` method.
    - Prints the robot's updated information.
- Update `FactoryDemo`:
    - Attempt to process the batch using one of the robots' `performTaskBatch` method.
    - Print the robot's battery information before and after processing.