BioLab Management System

1. Описание предметной области

Вы создаете софт для автоматизированного лабораторного шкафа. В шкафу хранятся как приборы (микроскопы, лазеры), так и образцы (пробирки с реактивами). Система должна позволять:

- Хранить всё оборудование в одном месте
- Автоматически пересчитывать чистоту реактивов (калибровка)
- Удалять опасные объекты при аварии

2. Иерархия классов

1. **Абстрактный класс LaboratoryItem (инвентарная единица):**
    - *Это базовый стандарт для всего, что попадает в лабораторию. У любой вещи (будь то колба с вирусом или дорогой микроскоп) должен быть серийный номер и уровень опасности, чтобы хаб мог работать с ними как с единым списком предметов*
    - Поля:
        - String serialNumber (серийный номер)
        - int biohazardLevel (уровень биологической опасности) - от 0 до 10
        - Integer sessionID
2. **Класс ResearchSample<T extends Number> (биологический образец/реактив):**
    - *Это конкретная пробирка с каким-то раствором или чем-то химическим. Это единственный объект, над которым проводятся химические вычисления*
    - Наследует LaboratoryItem
    - Поля:
        - String materialName (название биоматериала)
        - T purity (показатель чистоты, любой тип, наследующийся от Number) – может измеряться целым числом (95%), дробью (0.95), вашим классом Fraction и другими
3. **Класс MedicalDevice (медицинский прибор/оборудование):**
    - *Собственно, оборудование (центрифуга, лазер и т.п.)*
    - Наследует LaboratoryItem
    - Поля:
        - String deviceName (название прибора)
        - double precision (погрешность измерения)

4. **Класс ResearchHub<S extends LaboratoryItem> (исследовательский центр)**:
    - *Контейнер, который хранит все в одном массиве*
    - Параметризованный класс, ограниченный типом LaboratoryItem
    - Поля:
        - storage (внутреннее хранилище)
        - int count (текущее количество объектов)
    - Правила управления памятью:
        - Хранение осуществляется во внутреннем массиве
        - Начальный размер массива — 2 элемента
        - При заполнении текущего массива его емкость должна автоматически увеличиваться ровно в 2 раза
    - Функциональные методы:
        - void register(S item) – добавление объекта в систему
        - S release(int index) – извлечение объекта по индексу с удалением его из системы. После удаления пустые ячейки в массиве должны быть исключены
        - int indexOf(S item) – поиск позиции конкретного объекта в системе
        - void clearByHazardLevel(int limit) – удаление всех объектов, уровень опасности которых превышает заданное значение

3. Калибровка

В классе ResearchHub должны быть реализованы следующие операции:

- **Метод adjustPurity(Number offset, Number multiplier)**
    - Должен обновить поле purity у всех соответствующих объектов по формуле: (текущее значение + offset) * multiplier
    - Метод обязан корректно обрабатывать входные параметры типов Integer, Double и Fraction (из предыдущей работы)
- **Метод double getAveragePurity()** – расчет среднего значения чистоты всех образцов в хабе

4. Модуль верификации

Каждому исследованию присваивается уникальный идентификатор Integer sessionID

- **Метод boolean checkSession(S item, Integer sessionID)** – проверяет соответствие переданного идентификатора идентификатору объекта

- **Тестирование**: в классе Main необходимо реализовать сценарии проверки для ID со значениями **100** и **250.** Результаты работы и их причины должны быть обоснованы в комментариях к коду

5. Статические инструменты

Создать класс LabUtils со следующими статическими методами:

- Поиск объекта с максимальным показателем biohazardLevel в переданном массиве
- Метод для случайного перемешивания элементов в массиве любого типа
- Метод сравнения двух объектов ResearchHub на предмет идентичности их типов-параметров
- Метод генерации отчета: вывод списка всех объектов в формате [Класс] ID: X | Имя: Y

6. Сценарии тестирования для Main

Чтобы работа была принята, ваш класс Main должен последовательно воспроизводить следующие 5 сценариев. Результаты выполнения каждого шага должны выводиться в консоль. Каждый сценарий - отдельный метод в main

Сценарий №1

1. Создайте хаб, предназначенный только для образцов: ResearchHub<ResearchSample<Double>>
2. Попробуйте добавить в него объект типа MedicalDevice
3. Что видим? Опишите комментарием около вызова этого сценария и попробуйте пояснить почему именно так

Сценарий №2

1. Создайте универсальный хаб ResearchHub<LaboratoryItem>
2. Добавьте в него 3 любых объекта (например, два образца и один прибор)
3. Выведите текущее количество элементов (count)
4. Вызовите release(1) (удаление второго элемента)
5. Выведите в консоль полный отчет по хабу (список оставшихся элементов с их индексами)

Сценарий №3

1. Добавьте в хаб три образца:
    - ResearchSample<Double> (чистота 0.5)
    - ResearchSample<Integer> (чистота 50)
    - ResearchSample<Fraction> (чистота 1/2)
2. Вызовите метод adjustPurity(10, 0.5)

3. Выведите среднюю чистоту через getAveragePurity()
4. **Задание:** Проверьте полученное число на адекватность. В комментариях укажите, возникли ли сложности при работе с разными типами данных внутри одного метода

Сценарий №4

1. Создайте два объекта. Присвойте обоим sessionID = 100. Вызовите checkSession(item, 100) и выведите результат (true/false) в консоль
2. Создайте еще два объекта. Присвойте обоим sessionID = 250. Вызовите checkSession(item, 250) и выведите результат в консоль
3. **Задание:** В методе checkSession обязательно используйте оператор ==. В комментариях к Main дайте развернутое пояснение, почему результаты в этих двух случаях именно такие

Сценарий №5

1. Сформируйте массив из 5 предметов со следующими уровнями опасности (biohazardLevel): [2, 9, 4, 10, 3]
2. Используя LabUtils.findMaxHazard, найдите и выведите самый опасный объект из этого массива
3. Перенесите все 5 предметов в ResearchHub
4. Выполните команду clearByHazardLevel(5)
5. С помощью LabUtils.generateReport выведите итоговый список предметов, оставшихся в хабе
6. Создайте второй хаб с другим типом параметров. Используя LabUtils.compareHubTypes, сравните их и выведите результат