**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

**ЛАБОРАТОРНАЯ РАБОТА №5**

по дисциплине: «Технологии разработки программного обеспечения»

на тему: «Принципы и технологии создания качественного кода в *Java*»

Выполнил: студент гр. ИТП-21

Бондарев Е.Ю

Принял: преподаватель-стажер

Малиновский И.Л.

Гомель 2023

**Цель работы:**

1. Создать новую ветку для рефакторинга в ранее созданном *git*-репозитории.

2. Дополнить разработанную в лабораторной работе 1 иерархию классом-репозиротием. Класс должен содержать коллекцию экземпляров иерархии, а также методы добавления, удаления и изменения элементов коллекции.

3. Отрефакторить код лабораторных работ 1 и 3.

4. Участки измененного кода снабдить комментариями.

5. Создать и одобрить пул-реквест на слияние ветки с изменениями с главной веткой.

6. Составить отчет о проделанной работе. Отчет должен содержать описание и обоснование всех изменений в коде.

**Решение:**

По условию задания в первой лабораторной работе был создан новый класс *DetailManager* для работы с коллекцией объектов типа *Detail*. В этом классе было создано поля для хранения списка деталей. Конструктор, в который передается список деталей. Были реализованы методы для добавления, удаления и изменения исходного списка. Были добавлены и изменены методы подсчёта общего вес деталей, содержащих одинаковую форму, вывода количество деталей и вывода деталей, которые отличается своей формой. Результат рефакторинга первой лабораторной работы указан в приложении A.

Далее были внесены изменения во вторую лабораторную работу ко всем методам были добавлены комментарии. Далее были созданы две новы ветки, которые содержали новый вариант первой и второй лабораторной работ. Пример *GitHub* репозитория после изменений указан на рисунке 1.

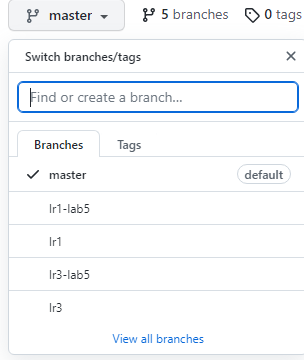


Рисунок 1 – Пример репозитория после выполнения лабораторной работы

**Вывод**: в ходе выполнения лабораторной работы был произведен рефакторинг кода лабораторных работ 1 и 3. К первой лабораторной работы был добавлен новый класс для работы со списком объектов. К первой и второй лабораторных работах были добавлены комментарии. Все изменения, которые были сделаны были загружены на *GitHub*.

**ПРИЛОЖЕНИЕ А**

Листинг класса *DetailManager*

import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
import java.util.stream.Collectors;  
  
*/\*\*  
  
 Класс для управления списком деталей.  
 \*/*public class DetailManager {  
  
 private List<Detail> details;  
  
 */\*\*  
  
 Конструктор класса DetailManager, инициализирующий пустой список деталей.  
 \*/* public DetailManager() {  
 details = new ArrayList<>();  
 }  
 */\*\*  
  
 Метод для добавления детали в список деталей.* ***@param*** *detail - деталь, которую нужно добавить.  
 \*/* public void addDetail(Detail detail) {  
 details.add(detail);  
 }  
 */\*\*  
  
 Метод для удаления детали из списка деталей.* ***@param*** *detail - деталь, которую нужно удалить.  
 \*/* public void removeDetail(Detail detail) {  
 details.remove(detail);  
 }  
 */\*\*  
 \* Метод для изменения детали в списке деталей.  
 \** ***@param*** *oldDetail - деталь, которую нужно изменить.  
 \** ***@param*** *newDetail - новая деталь, которой нужно заменить старую.  
 \** ***@return*** *true, если удалось изменить деталь, false - в противном случае.  
 \*/* public boolean updateDetail(Detail oldDetail, Detail newDetail) {  
 int index = details.indexOf(oldDetail);  
 if (index != -1) {  
 details.set(index, newDetail);  
 return true;  
 }  
 return false;  
 }  
  
 */\*\*  
  
 Метод для получения списка всех деталей.* ***@return*** *список всех деталей.  
 \*/* public List<Detail> getDetails() {  
 return details;  
 }  
 */\*\*  
  
 Метод для вывода общего веса деталей каждой формы.  
  
 Выводится вес каждой формы и суммарный вес всех деталей этой формы.  
 \*/* public void overallWeight() {  
 Map<String, Double> totalWeightsByShape = details.stream()  
 .collect(Collectors.*groupingBy*(Detail::getForm, Collectors.*summingDouble*(Detail::getWeight)));  
  
 totalWeightsByShape.forEach((form, totalWeight) -> {  
 System.*out*.println("Общий вес деталей формы " + form + " равняется: " + totalWeight + " kg");  
 });  
  
 System.*out*.println("\n");  
 }  
  
 */\*\*  
  
 Метод для вывода деталей, которые отличаются своей формой от всех остальных деталей.  
 \*/* public void findUniqueDetails() {  
 List<Detail> uniqueDetails = details.stream()  
 .filter(detail -> details.stream()  
 .filter(otherDetail -> !detail.equals(otherDetail))  
 .noneMatch(otherDetail -> detail.getForm().equals(otherDetail.getForm())))  
 .collect(Collectors.*toList*());  
  
 uniqueDetails.forEach(detail ->  
 System.*out*.println("Деталь, которая отличается своей формой: " + detail.getForm()));  
 }  
}