МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математическая кибернетика и информационные технологии

Отчет по лабораторной работе №3

Тема: Класс Object. Работа с хэш-таблицами

Выполнил: студент группы БПИ2401
Мещеряков Кирилл

Руководитель: Харрасов Камиль Раисович

Москва, 2025

Цель работы:

Целью данной лабораторной работы является изучение основных методов базового класса Object в Java и принципов их переопределения, а также освоение работы с хэш-таблицами.
В рамках работы реализуется собственная хэш-таблица методом цепочек и демонстрируется использование встроенного класса HashMap.

Задачи:

1. Изучить назначение и особенности методов класса Object: toString(), equals(), hashCode(), getClass().
2. Реализовать класс HashTable, использующий метод цепочек для хранения пар «ключ — значение».
3. Реализовать операции:
    - добавления put(key, value)
    - поиска get(key)
    - удаления remove(key)
    - проверки размера и пустоты таблицы (size(), isEmpty()).
4. Создать класс Product с переопределёнными методами equals() и hashCode().
5. На примере класса Warehouse показать использование встроенной HashMap для учёта товаров на складе.

Индивидуальное задание:

Реализовать хэш-таблицу для хранения пар ключ-значение без использования дженериков.
Также реализовать систему учёта продуктов, где ключом является **штрихкод**, а значением — объект класса **Product**, содержащий информацию о товаре.

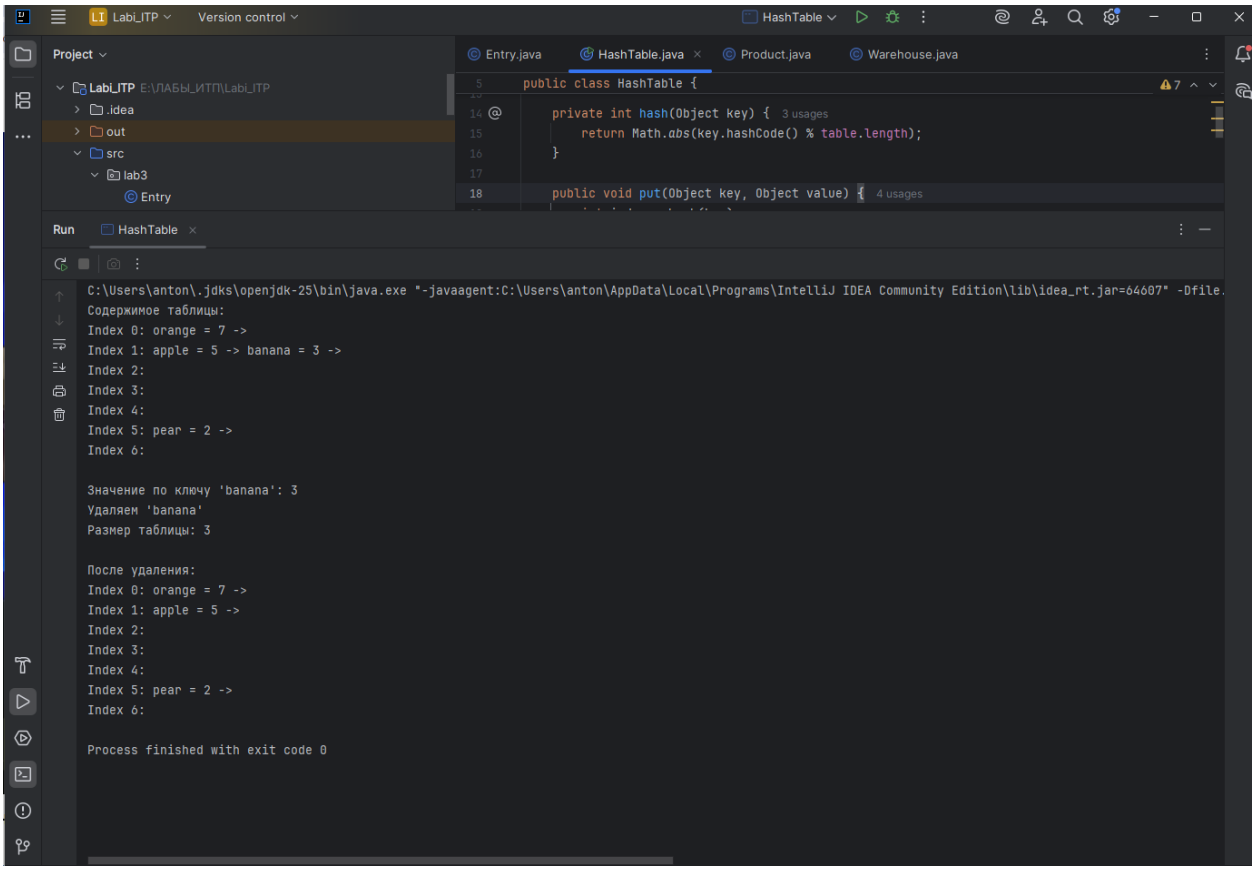
Ход выполнения:

1. Реализация собственной хэш-таблицы (HashTable)

Создан класс HashTable, который хранит массив списков (LinkedList). Каждый элемент массива — это список объектов класса Entry, представляющих пару (ключ, значение).

Методы класса:

- put(Object key, Object value) — добавление новой пары;
- get(Object key) — поиск значения по ключу;
- remove(Object key) — удаление пары;
- size() — количество элементов;
- isEmpty() — проверка пустоты таблицы;
- printTable() — отображение содержимого таблицы.



2. Реализация класса Product

Класс Product содержит поля:

String name;

double price;

int quantity;

Методы equals() и hashCode() переопределены для корректного сравнения и хэширования товаров по их содержимому.
Метод toString() возвращает человекочитаемое описание товара.

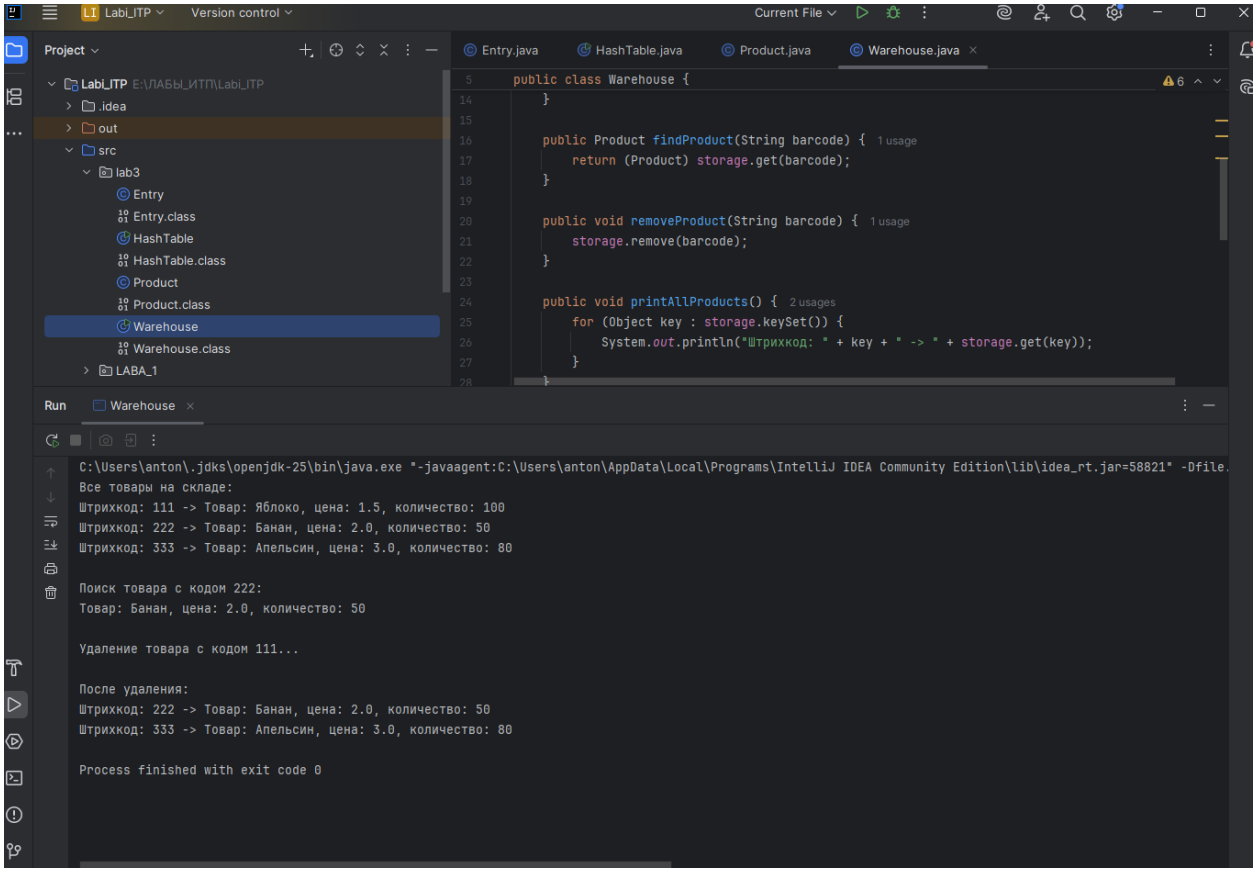
3. Работа с встроенным классом HashMap (Warehouse)

Создан класс Warehouse, который хранит товары по их штрихкоду в коллекции HashMap.

Реализованы методы:

- addProduct(String code, Product product) — добавление товара;
- findProduct(String code) — поиск по штрихкоду;
- removeProduct(String code) — удаление товара;
- printAllProducts() — вывод всего содержимого склада.

Пример работы программы:



Результат выполнения программы:

- Реализована собственная хэш-таблица без дженериков.
- Переопределены методы equals() и hashCode().
- Продемонстрирована работа с HashMap на примере учёта товаров.
- Проверена корректность операций добавления, поиска и удаления.

Заключение:

В ходе выполнения лабораторной работы были:

- Изучены ключевые методы класса Object и принципы их переопределения;
- Реализована собственная структура данных — хэш-таблица методом цепочек;
- Освоена работа с коллекцией HashMap на примере хранения объектов пользовательского класса;
- Получены практические навыки работы с хэшированием и разрешением коллизий.

Работа выполнена успешно, цели и задачи достигнуты.

Ссылка на GitHub-репозиторий:

https://github.com/M1ke0-0/ITiP/tree/main/Laba_3

Ответы на вопросы:

1. Для чего нужен класс Object?

Класс Object — это базовый класс для всех классов в Java.
Он обеспечивает каждый объект набором универсальных методов (таких как equals(), hashCode(), toString(), getClass() и др.), которые определяют базовое поведение объектов.
Все классы неявно наследуют Object, даже если это не указано в коде.

2. Для чего нужно переопределять методы equals() и hashCode()?

Эти методы переопределяют, чтобы сравнение объектов происходило **по содержимому**, а не по ссылке на память.
Без переопределения Java считает равными только те объекты, которые физически являются одной и той же ссылкой.
Переопределённые методы позволяют корректно работать с коллекциями, основанными на хэшировании (например, HashMap, HashSet).

3. Какие есть правила переопределения методов equals() и hashCode()?

1. Если два объекта равны по equals(), то их hashCode() должен быть одинаковым.
2. Если два объекта имеют одинаковый hashCode(), они могут быть не равны по equals().
3. Метод equals() должен быть:

- **рефлексивным** (объект равен сам себе),
- **симметричным** (если A равно B, то и B равно A),
- **транзитивным** (если A равно B, и B равно C, то A равно C),
- **неизменным** (результат не должен меняться, пока поля объекта не изменились),
- сравнение с null всегда должно возвращать false.

4. Что делает метод toString()? Почему его часто переопределяют?

Метод toString() возвращает строковое представление объекта.
По умолчанию он выводит имя класса и технический хэш-код.
Его часто переопределяют, чтобы получить более понятное и информативное описание объекта (например, данные о товаре, пользователе, координатах и т.д.), что упрощает отладку и вывод информации пользователю.

5. Что делает метод finalize()? Почему его использование считается устаревшим (deprecated)?

Метод finalize() вызывается перед удалением объекта сборщиком мусора для освобождения ресурсов.
Однако он **не гарантирует**, когда именно будет вызван, и может мешать нормальной работе сборщика мусора.
Из-за непредсказуемости и низкой эффективности его использование признано устаревшим.
Современная альтернатива — конструкция try-with-resources или интерфейс AutoCloseable.

6. Что такое коллизия?

Коллизия — это ситуация, когда два разных ключа дают одинаковый хэш-код и попадают в одну и ту же ячейку хэш-таблицы.
Коллизии неизбежны, поэтому важно уметь их правильно обрабатывать.

7. Какие есть способы разрешения коллизий?

1. **Метод цепочек (chaining)** — каждая ячейка таблицы содержит список всех элементов, попавших в этот индекс.
2. **Открытая адресация (open addressing)** — при коллизии ищется следующая свободная ячейка в таблице.

3. **Двойное хэширование (double hashing)** — используется вторичная хэш-функция для поиска нового индекса.

8. Как хранятся данные в хэш-таблице?

Данные хранятся в виде пар «ключ → значение».
Ключ преобразуется в индекс массива с помощью хэш-функции.
В ячейке по этому индексу может храниться либо один элемент, либо цепочка (список) элементов, если произошли коллизии.

9. Что происходит, если в хэш-таблицу добавить элемент с одинаковым значением ключа?

Если ключ уже существует, старое значение заменяется новым.
При этом количество элементов не увеличивается, потому что ключ остаётся тем же.

10. Что происходит, если в хэш-таблицу добавить элемент с таким же хэш-кодом ключа, но разными исходными значениями?

Происходит коллизия — новый элемент добавляется в ту же ячейку, что и предыдущий, но хранится отдельно (например, в виде узла списка).
При поиске или удалении таблица проверяет все элементы в этой цепочке по методу equals().

11. Как изменяется HashMap при достижении порогового значения?

У HashMap есть два параметра:

- **вместимость (capacity)** — размер внутреннего массива;
- **коэффициент загрузки (load factor)** — допустимая заполненность (по умолчанию 0.75).

Когда количество элементов превышает capacity * loadFactor,
HashMap **автоматически увеличивает размер массива в 2 раза** и перераспределяет все элементы по новым индексам — этот процесс называется **rehashing**.