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

по дисциплине «Информационные технологии и программирование на языке Java»

Выполнил: студент группы БВТ2403

Подлуцкий Никита Сергеевич

Руководитель: _____

Москва, 2025

2. Цель работы

Цель работы — изучить принципы работы хэш-таблиц в Java. В ходе работы нужно было разобраться с ключевой ролью методов hashCode() и equals(), реализовать собственную простую хэш-таблицу с использованием метода цепочек, а также научиться применять стандартный класс HashMap для решения практических задач.

3. Индивидуальное задание

Работа состояла из двух основных частей:

1. Создание собственного класса HashTable. Требовалось реализовать универсальную (с помощью дженериков) хэш-таблицу, основанную на массиве связных списков (LinkedList) для разрешения коллизий. Необходимо было написать методы для основных
   операций: put (добавление/обновление), get (получение), remove (удаление), а также size и isEmpty.

2. Использование встроенного класса HashMap. На основе варианта №1 («Информация о студентах») нужно было продемонстрировать работу со стандартной хэш-таблицей Java. Для этого требовалось создать класс Student для хранения данных и использовать HashMap для добавления, поиска и удаления студентов по номеру зачетной книжки.

4. Основная часть

В первой части был разработан собственный дженерик-класс HashTable<K, V>.

- Структура: В основе класса лежит массив, каждый элемент которого является связным списком LinkedList. Такая структура реализует метод цепочек для обработки коллизий, когда несколько ключей имеют одинаковый хэш-код.

- Хранение данных: Для хранения пар "ключ-значение" внутри цепочек был создан вложенный статический класс Entry<K, V>.

- Логика работы: При добавлении элемента его индекс в массиве вычисляется на основе key.hashCode(). Если в ячейке уже есть элементы, новый просто добавляется в конец списка. Методы get и remove находят нужную ячейку по хэш-коду, а затем перебирают элементы в LinkedList, чтобы найти запись с нужным ключом.

Во второй части задания использовался стандартный класс java.util.HashMap.

- Был создан простой класс Student для инкапсуляции данных (имя, фамилия, возраст, средний балл).

- Для хранения базы студентов была создана карта HashMap<Integer, Student>, где ключом выступал целочисленный номер зачетной книжки, а значением — объект класса Student.

- Была продемонстрирована работа с основными методами HashMap: put() для добавления новых студентов, get() для их поиска по номеру зачетки и remove() для удаления.

5. Заключение

В ходе выполнения этой лабораторной я на практике разобрался, как устроены и работают хэш-таблицы. Реализация собственного класса помогла понять механику разрешения коллизий методом цепочек и важность правильного использования hashCode() и equals().

Работа с HashMap показала, насколько удобно и эффективно использовать готовые коллекции из стандартной библиотеки Java. Стало очевидно, что для большинства реальных задач лучше использовать проверенные и оптимизированные классы, а не писать свои с нуля. Полученные навыки будут полезны для дальнейшей работы со структурами данных в Java.

Контрольные вопросы

1. Для чего нужен класс Object?
Класс Object — это "прародитель" абсолютно всех классов в Java. Любой класс, который мы создаем, неявно наследуется от Object. Благодаря этому у каждого объекта в Java по умолчанию есть базовые методы, такие как equals(), hashCode() и toString().

2. Для чего нужно переопределять методы equals() и hashCode()?
Это критически важно для корректной работы коллекций,
особенно HashMap и HashSet.

> hashCode() вычисляет "адрес" ячейки (бакета), в которую будет помещен объект.
>
> equals() используется для того, чтобы найти нужный объект среди других в той же ячейке, если произошла коллизия (когда у разных объектов одинаковый хэш-код). Если не переопределить их для своих классов (как Student), то HashMap не сможет правильно находить объекты.

3. Какие есть правила переопределения методов equals() и hashCode()?
Главное правило (контракт): если два объекта равны по equals(), то у них обязан быть одинаковый hashCode().
Обратное неверно: если у объектов одинаковый hashCode(), они не обязательно равны по equals() — это и есть коллизия.

4. Что делает метод toString()? Почему его часто переопределяют?
Метод toString() возвращает строковое представление объекта. По умолчанию он выводит что-то нечитаемое, вроде com.example.Student@1f32e575 (имя класса и его хэш-код). Его переопределяют, чтобы при выводе объекта на печать получать осмысленную информацию, как мы сделали в классе Student, чтобы видеть имя, фамилию и т.д.

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

Метод finalize() вызывался сборщиком мусора перед тем, как удалить объект из памяти. Его считают устаревшим и не рекомендуют использовать, потому что нет никаких гарантий, когда и будет ли он вообще вызван. Это очень ненадежный механизм.

6. Что такое коллизия?
Коллизия в хэш-таблице — это ситуация, когда два разных ключа получают одинаковый хэш-код и, соответственно, претендуют на одну и ту же ячейку в массиве.

7. Какие есть способы разрешения коллизий?
Основные способы:

1. Метод цепочек (chaining): В каждой ячейке массива хранится связный список (LinkedList) всех элементов, попавших в нее. Именно этот метод мы реализовывали в Задании 1.

2. Открытая адресация (open addressing): Если ячейка занята, ищется следующая свободная ячейка по определенному правилу (например, просто следующая по порядку).

8. Как хранятся данные в хэш-таблице?
Данные хранятся в виде пар "ключ-значение" в массиве ячеек (бакетов). Индекс ячейки для каждой пары определяется хэш-кодом ее ключа. Если происходит коллизия, для хранения нескольких пар в одной ячейке используется дополнительная структура, например, связный список.

9. Что происходит, если в хэш-таблицу добавить элемент с одинаковым значением ключа?
Старое значение, связанное с этим ключом, заменяется (перезаписывается) новым. Количество элементов в таблице при этом не меняется.

10. Что происходит, если в хэш-таблицу добавить элемент с таким же хэш-кодом ключа, но разными исходными значениями (ключами)?
Это и есть коллизия. Новый элемент (пара "ключ-значение") будет добавлен в ту же самую ячейку, где уже лежит другой элемент. В нашей реализации он просто добавится в конец LinkedList'a в этой ячейке.

11. Как изменяется HashMap при достижении порогового значения?
Когда количество элементов в HashMap превышает пороговое значение (capacity * loadFactor, где loadFactor обычно 0.75), происходит перестройка (resizing). HashMap создает новый, более вместительный внутренний массив (обычно в два раза больше), и все существующие элементы заново распределяются по этому новому массиву согласно их хэш-кодам. Это делается для того, чтобы цепочки не становились слишком длинными и производительность не падала.