**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

Лабораторная работа №3. Вариант 1

Выполнил: студент группы БПИ2401

Исламов Эмин Маратович

Проверил: Харрасов Камиль Раисович

**Цель работы:**

Изучить возможности класса Object, методы equals(), hashCode(), toString(), а также освоить работу с хэш-таблицами — как собственной реализации, так и встроенного класса HashMap.

**Теоретическая** **часть**

Класс Object является базовым для всех классов в Java. Каждый объект наследует от него базовые методы, в том числе:

* toString() — возвращает строковое представление объекта.
* equals(Object obj) — сравнивает объекты (по умолчанию по ссылкам, но обычно переопределяют для сравнения по содержимому).
* hashCode() — возвращает хэш-код объекта; используется коллекциями, такими как HashMap и HashSet.
* getClass() — возвращает объект класса Class, описывающий тип объекта.
* clone() — создаёт копию объекта.
* wait(), notify(), notifyAll() — используются для синхронизации потоков.
* finalize() — вызывается перед удалением объекта сборщиком мусора (устаревший метод).

При переопределении методов equals() и hashCode() необходимо соблюдать контракт:

1. Если два объекта равны согласно equals(), их hashCode() должен быть одинаковым.
2. Одинаковый хэш-код не гарантирует равенство объектов.

Хэш-таблица — это структура данных, где каждый элемент хранится в виде пары “ключ-значение”.

Позиция элемента определяется с помощью хэш-функции, которая преобразует ключ в индекс.

При совпадении индексов (коллизии) используется метод цепочек — элементы одного индекса хранятся в связанном списке.

В Java хэш-таблицы реализованы классами HashMap и Hashtable.

Они позволяют быстро выполнять операции добавления, поиска и удаления по ключу.

**Ход работы**

**Задание 1. Реализация собственной хэш-таблицы**

Создан класс HashTable, реализующий хэш-таблицу методом цепочек.

Ключ и значение могут быть любого типа (используется дженерик <K, V>).

Реализованы методы:

* put(key, value) — вставка пары;
* get(key) — получение значения по ключу;
* remove(key) — удаление пары;
* size() — количество элементов;
* isEmpty() — проверка на пустоту.

**Код класса Student**

**public** **class** **Student** {

**private** String name;

**private** String surname;

**private** int age;

**private** double averageGrade;

**public** **Student**(String name, String surname, int age, double averageGrade) {

this.name = name;

this.surname = surname;

this.age = age;

this.averageGrade = averageGrade;

}

@Override

**public** boolean **equals**(Object obj) {

**if** (this == obj) **return** true;

**if** (obj == null || getClass() != obj.getClass()) **return** false;

Student other = (Student) obj;

**return** age == other.age &&

Double.compare(other.averageGrade, averageGrade) == 0 &&

name.equals(other.name) &&

surname.equals(other.surname);

}

@Override

**public** int **hashCode**() {

int result = name.hashCode();

result = 31 \* result + surname.hashCode();

result = 31 \* result + age;

long temp = Double.doubleToLongBits(averageGrade);

result = 31 \* result + (int) (temp ^ (temp >>> 32));

**return** result;

}

@Override

**public** String **toString**() {

**return** name + " " + surname + ", возраст: " + age + ", ср. балл: " + averageGrade;

}

}

**Код класса HashTable**

**import** java.util.LinkedList;

**public** **class** **HashTable**<K, V> {

**private** **static** **class** **Entry**<K, V> {

**private** K key;

**private** V value;

**public** **Entry**(K key, V value) {

this.key = key;

this.value = value;

}

**public** K **getKey**() { **return** key; }

**public** V **getValue**() { **return** value; }

**public** **void** **setValue**(V value) { this.value = value; }

@Override

**public** String **toString**() {

**return** key + " = " + value;

}

}

**private** LinkedList<Entry<K, V>>[] table;

**private** int size;

**public** **HashTable**(int capacity) {

table = **new** **LinkedList**[capacity];

size = 0;

}

**private** int **hash**(K key) {

**return** Math.abs(key.hashCode() % table.length);

}

**public** **void** **put**(K key, V value) {

int index = hash(key);

**if** (table[index] == null) {

table[index] = **new** **LinkedList**<>();

}

**for** (Entry<K, V> entry : table[index]) {

**if** (entry.getKey().equals(key)) {

entry.setValue(value);

**return**;

}

}

table[index].add(**new** **Entry**<>(key, value));

size++;

}

**public** V **get**(K key) {

int index = hash(key);

**if** (table[index] != null) {

**for** (Entry<K, V> entry : table[index]) {

**if** (entry.getKey().equals(key)) {

**return** entry.getValue();

}

}

}

**return** null;

}

**public** **void** **remove**(K key) {

int index = hash(key);

**if** (table[index] != null) {

**for** (Entry<K, V> entry : table[index]) {

**if** (entry.getKey().equals(key)) {

table[index].remove(entry);

size--;

**return**;

}

}

}

}

**public** int **size**() {

**return** size;

}

**public** boolean **isEmpty**() {

**return** size == 0;

}

**public** **void** **printTable**() {

**for** (int i = 0; i < table.length; i++) {

System.out.print("Index " + i + ": ");

**if** (table[i] != null) {

**for** (Entry<K, V> entry : table[i]) {

System.out.print(entry + " -> ");

}

System.out.println();

} **else** {

System.out.println("пусто");

}

}

}

}

**Демонстрация работы (Main.java)**

**public** **class** **Main** {

**public** **static** **void** **main**(String[] args) {

HashTable<String, Student> table = **new** **HashTable**<>(7);

table.put("A123", **new** **Student**("Иван", "Петров", 20, 4.3));

table.put("B456", **new** **Student**("Анна", "Иванова", 19, 4.8));

table.put("C789", **new** **Student**("Олег", "Сидоров", 21, 3.9));

System.out.println("После добавления студентов:");

table.printTable();

System.out.println("\nПоиск студента с зачеткой B456:");

System.out.println(table.get("B456"));

table.remove("A123");

System.out.println("\nПосле удаления студента A123:");

table.printTable();

System.out.println("\nКоличество студентов: " + table.size());

System.out.println("Пуста ли таблица? " + table.isEmpty());

}

}

**Задание 2. Работа со встроенным классом HashMap**

Для решения второй части лабораторной работы использовался встроенный класс HashMap.

**Код программы**

**import** java.util.HashMap;

**import** java.util.Map;

**public** **class** **Main** {

**public** **static** **void** **main**(String[] args) {

Map<String, Student> students = **new** **HashMap**<>();

students.put("A123", **new** **Student**("Иван", "Петров", 20, 4.3));

students.put("B456", **new** **Student**("Анна", "Иванова", 19, 4.8));

students.put("C789", **new** **Student**("Олег", "Сидоров", 21, 3.9));

System.out.println("Все студенты:");

**for** (Map.Entry<String, Student> entry : students.entrySet()) {

System.out.println("Зачетка: " + entry.getKey() + " — " + entry.getValue());

}

String searchKey = "B456";

System.out.println("\nПоиск студента с зачеткой " + searchKey + ":");

Student found = students.get(searchKey);

**if** (found != null)

System.out.println(found);

**else**

System.out.println("Студент не найден.");

String removeKey = "A123";

System.out.println("\nУдаляем студента с зачеткой " + removeKey);

students.remove(removeKey);

System.out.println("\nПосле удаления:");

**for** (Map.Entry<String, Student> entry : students.entrySet()) {

System.out.println("Зачетка: " + entry.getKey() + " — " + entry.getValue());

}

System.out.println("\nКоличество студентов: " + students.size());

System.out.println("Пуста ли таблица? " + students.isEmpty());

}

}

**Вывод**

В ходе лабораторной работы были изучены методы базового класса Object, а также реализована работа с хэш-таблицами двумя способами:

1. Собственная реализация методом цепочек.
2. Использование встроенного класса HashMap.

Были освоены методы equals() и hashCode(), обеспечивающие корректную работу объектов в коллекциях.

Практически продемонстрировано добавление, поиск и удаление данных в хэш-таблице.

Использование HashMap позволяет значительно упростить разработку и обеспечить эффективность при работе с большими объёмами данных.

**Ответы на контрольные вопросы**

1. Для чего нужен класс Object?

Он является базовым классом всех объектов в Java и содержит общие методы (equals, toString, hashCode и др.).

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

Чтобы сравнивать объекты по содержимому и корректно использовать их в хэш-таблицах.

1. Какие есть правила переопределения equals() и hashCode()?

Если два объекта равны по equals(), их hashCode() должен быть одинаковым.

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

Возвращает строковое представление объекта; переопределяется для удобного вывода информации.

1. Что делает метод finalize()? Почему он устарел?

Вызывается сборщиком мусора перед удалением объекта; устарел из-за непредсказуемости вызова.

1. Что такое коллизия?

Ситуация, когда разные ключи имеют одинаковый хэш-код.

1. Какие способы разрешения коллизий существуют?

Метод цепочек и метод открытой адресации.

1. Как хранятся данные в хэш-таблице?

В виде массива списков (цепочек), где каждая ячейка содержит пары “ключ-значение”.

1. Что происходит, если добавить элемент с одинаковым ключом?

Старое значение заменяется новым.

1. Что происходит, если у разных ключей одинаковый хэш-код?

Они попадают в один список (коллизию).

1. Как изменяется HashMap при достижении порогового значения?

Увеличиваетсяразмер массива и перераспределяются элементы (rehash).