



Хеш таблицы

Составитель: Рощупкин Александр



Хеширующие таблицы

Определение

- * **Хеш-таблица (hash table)** – структура данных для хранения пар «ключ – значение»
- * Доступ к элементам осуществляется по ключу (key)
- * Ключи могут быть любыми объектами - строками, числами и т.д.
- * Хеш-таблицы позволяют в среднем за время $O(1)$ выполнять добавление, поиски и удаление элементов

Пример аутентификации

- * Необходимо написать класс, который определяет правильность введения пары – логин, пароль

Принцип работы хеш таблицы

- * Обычные массивы хороши тем, что у них быстрый доступ по ключу (индексу)
- * Недостаток массивов – ключами (индексами) могут быть только целые числа
- * Хеш таблица позволяет использовать массив для хранения объектов и в качестве ключей (индексов) использовать любые типы данных
- * Для преобразования объекта в индекс используется принцип хеширования

Понятие хеширования

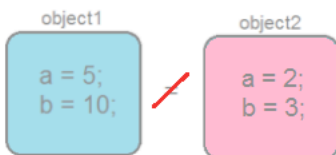
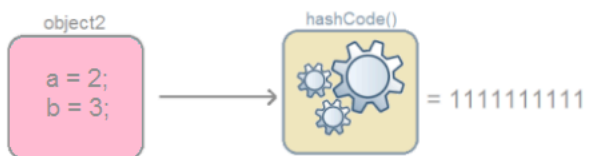
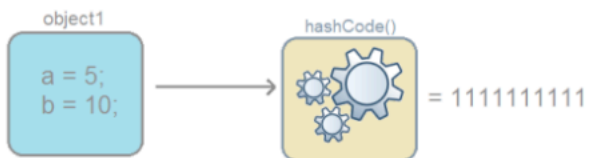
- * **Хеширование** – алгоритм преобразующий набор данных любой длины в набор данных фиксированной длины
- * **Хеш функция** – это метод, преобразующий ключ (объект) в номер элемента массива
- * **Хеш код** – результат работы хеш функции

Вычисление хеш кода

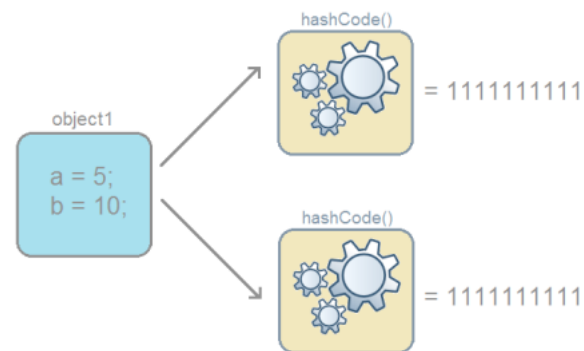
- * Вычислением хеш кода для любого объекта занимается метод `hashCode()` находящийся в классе `Object`
- * Значение хеш кода должно зависеть от значения всех данных объекта
- * К примеру можно вычислить циклическую сумму целочисленных значений всех полей

Значения хеш кода

- * Для одного и того же объекта значение хеш-кода должно быть всегда одинаково



1111111111 = 1111111111



1111111111 = 1111111111

- * Если объекты одинаковые, то и хеш-коды одинаковые

Метод hashCode из Object

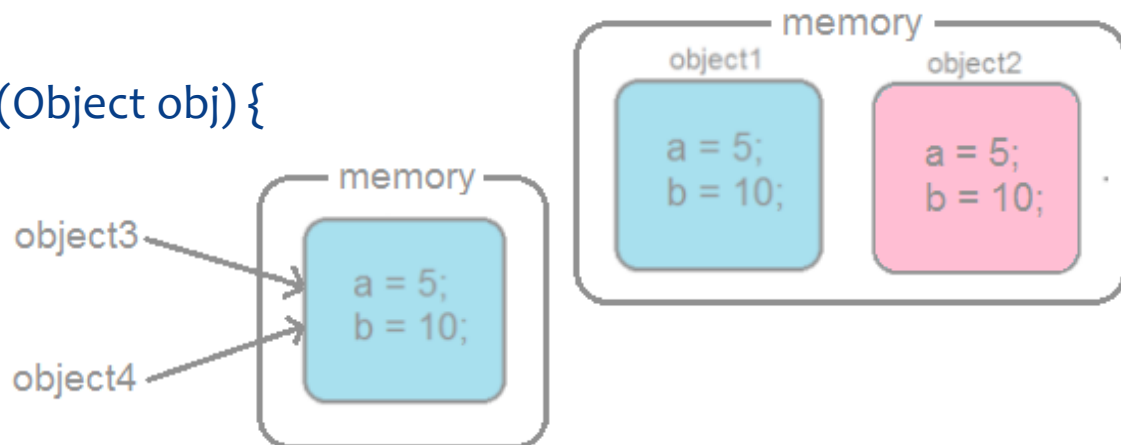
```
public native int hashCode();
```

- * Ключевое слово `native` означает, что реализация данного метода выполнена на другом языке, например на C, C++ или ассемблере. Конкретный `native int hashCode()` реализован на C++
- * Для вычисления хеш-кода используется [Park-Miller RNG](#) алгоритм, в основе которого лежит генерация случайных чисел
- * Хеш-код для объекта вычисляется при первом вызове и затем помещается в объекта и возвращается при каждом вызове метода `hashCode()`

Метод equals

- * Для проверки эквивалентности в классе Object существует метод equals(), который сравнивает содержимое объектов и выводит значение типа boolean true, если содержимое эквивалентно, и false — если нет
- * Не переопределённый метод equals сравнивает значение ссылок

```
public boolean equals(Object obj) {  
    return (this == obj)  
}
```



Контракт equals и hashCode

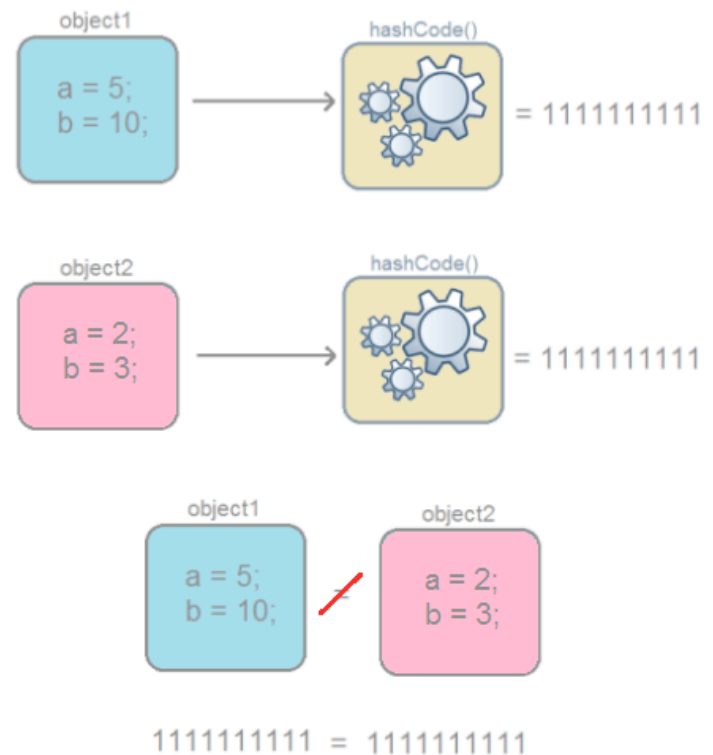
- * Эквивалентность и хеш-код тесно связаны между собой, поскольку хеш-код вычисляется на основании содержимого объекта (значения полей) и если у двух объектов одного и того же класса содержимое одинаковое, то и хеш-коды должны быть одинаковые

`object1.equals(object2)` // должно быть *true*

`object1.hashCode() == object2.hashCode()` // должно быть *true*

Коллизии

- * Ситуация при которой у разных объектов хеш коды одинаковы называется коллизией
- * Методы решения коллизий:
 - * Открытая адресация
 - * Метод цепочек



Линейное зондирование

- * Метод открытой адресации состоит в том, чтобы, пользуясь каким-либо алгоритмом, обеспечивающим перебор элементов таблицы, просматривать их в поисках свободного места для новой записи
- * Линейное опробование сводится к последовательному перебору элементов таблицы с некоторым фиксированным шагом

$$a = h(\text{key}) + c * i$$

- * При достижении конца таблицы осуществляется переход в начало

Домашнее задание

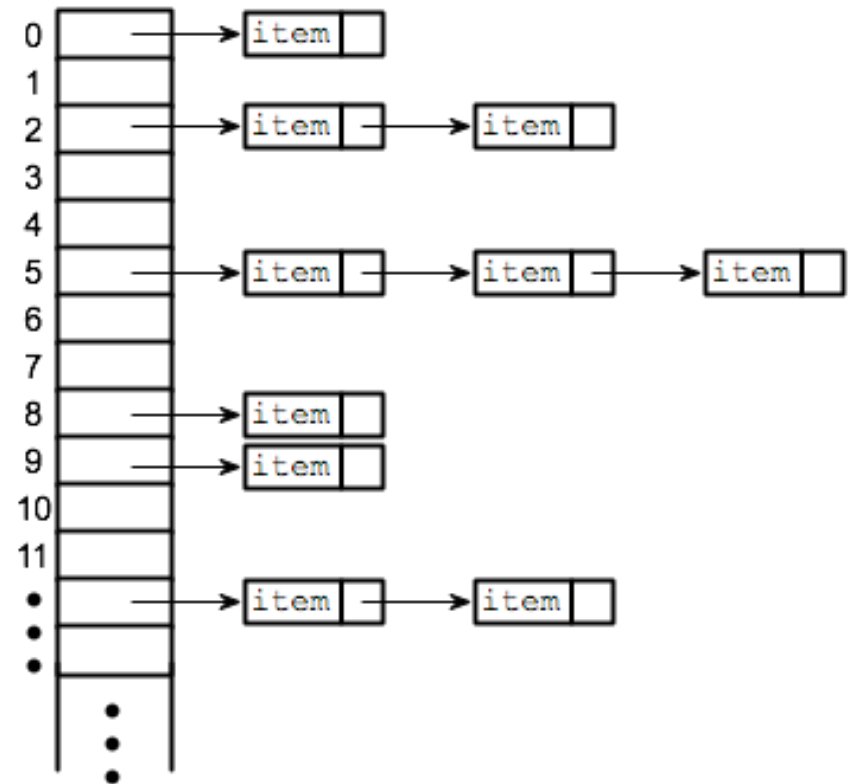
- * Написать собственную реализацию хеш-таблицы с решением коллизий методом линейного зондирования. Реализовать методы put, get, delete
- * Добавить в реализацию хеш-таблицы поддержку дженериков

Неправильный хеш код

- * Постоянное число
- * Случайное число

Метод цепочек

- * Объекты с одинаковыми хеш-кодами помещаются в одну ячейку (корзину) в виде связанного списка и затем сравниваются методом equals



Load factor

- * Load factor - коэффициент заполнения хеш-таблицы
- * Отношение числа n хранимых элементов в хеш-таблице к размеру h массива
- * По коэффициенту заполнения можно принимать решение об изменении размера хеш-таблицы
- * Для изменения размера нужно:
 - * Создать новый массив
 - * Перекопировать элементы с повторным хешированием

Правильный хеш-код

- * **Равномерность (uniform distribution)** – хеш-функция должна равномерно заполнять индексы массива возвращаемыми номерами
- * Желательно, чтобы все хэш-коды формировались с одинаковой равномерно распределенной вероятностью