

## Модуль 4. Алгоритмы и структуры данных

### Тема 4.7. Хеш-таблица и функция хеширования

2 часа

#### Оглавление

4.7. Хеш-таблица и функция хеширования .....	2
4.7.1. Метод hashCode() .....	2
Упражнение 4.7.1 .....	5
4.7.2. Семейства контейнеров – Collections и Map .....	8
Контейнер Set .....	8
Контейнер HashSet .....	9
Контейнер TreeSet .....	10
Упражнение 4.7.2 .....	10
Задание 4.7.1.....	13
Задание 4.7.2.....	13
Ссылки .....	14
Благодарности .....	14

## 4.7. Хеш-таблица и функция хеширования

Современные программные системы обрабатывают огромные объемы текстовой информации. Мы каждый день используем системы интернет-поиска, пользуемся паролями. Все эти действия зачастую реализуются с использованием такого понятия как “хеширование”.

Хеширование применяют, когда нужно преобразовать некоторую совокупность связанных данных в число. Это преобразование называется хеш-функцией, а результат хеш-кодом. С помощью хеш-функций получают не просто произвольное число, а такое, которое зависит от содержания данных, причем если данные одинаковые, то и число должно получаться равным.

В качестве примера применения хеш-функций для поиска можно привести поиск в словаре. Поиск строкового значения в словаре - процесс для компьютера ресурсоемкий: нужно сравнивать каждый символ искомого слова в худшем переборном случае с каждым символом всех слов в словаре. Другое дело - сравнивать числа. Таким образом хеш-функция значительно ускоряет поиск.

Хеширование также широко применяется в криптографии. Более подробно с криптографическими хеш-функциями мы познакомимся в Модуле 6.

### 4.7.1. Метод hashCode()

Как мы с вами уже знаем, Object - это базовый класс для всех остальных объектов в Java. Каждый класс наследуется от Object. Соответственно все классы наследуют методы класса Object, среди которых наиболее известен метод **hashCode()**.

Метод hashCode() преобразует любой объект в хеш-код, который и возвращает в виде числа типа int.

Что же такое хеш-код? Хеш-код — это **битовая строка фиксированной длины, полученная из массива произвольной длины**.

Рассмотрим пример:

```
public class Main {  
    public static void main(String[] args) {  
        Object object = new Object();  
        int hCode = object.hashCode();  
        System.out.println(hCode);  
    }  
}
```

В результате выполнения программы будет выведено целое число. Это число и есть наша битовая строка фиксированной длины. В Java она представлена в виде числа примитивного типа int. Как следствие, хеш-код ограничен диапазоном значений типа int.

В качестве массива произвольной длины в примере выступает объект типа Object, который передается в метод hashCode() в качестве параметра.

Главные свойства хеш-кода :

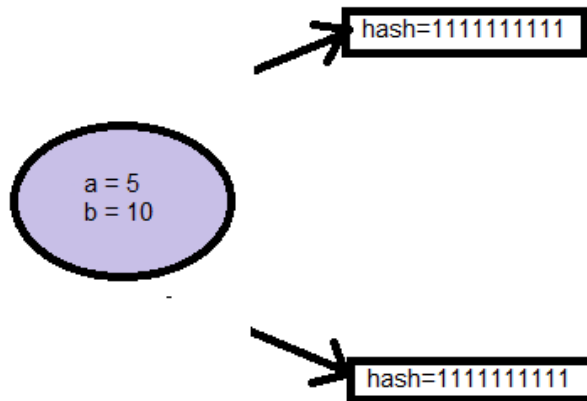
- если хеш-коды разные, то они получены из разных входных объектов;
- если объекты одинаковы, то их хеш-коды всегда одинаковы, однако обратное неверно.

Каким бы не был большим диапазон int, понятно, что он не бесконечен и существует вероятность того, что хеш-коды разных объектов могут совпасть. Ситуация, когда у **разных** объектов

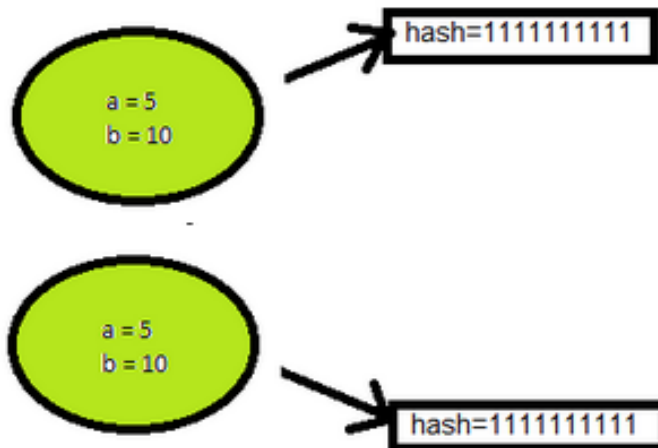
**одинаковые** хеш-коды называется — **коллизией**. Вероятность возникновения коллизии зависит от используемого алгоритма генерации хеш-кода.

Подведем промежуточный итог:

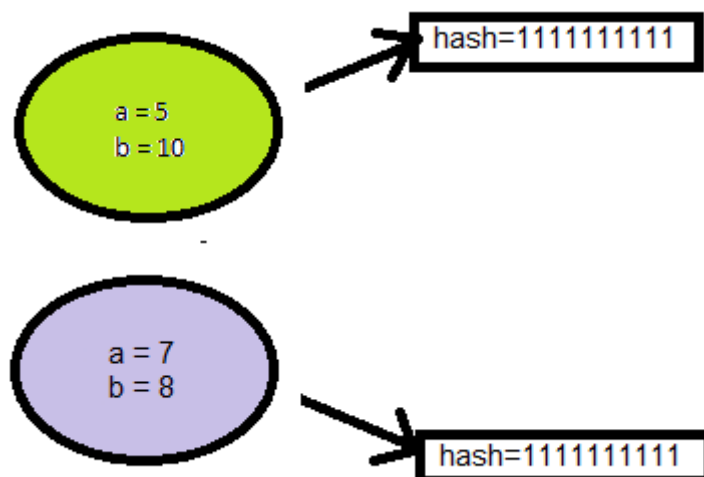
- для одного и того же объекта хеш-коды всегда одинаковы:



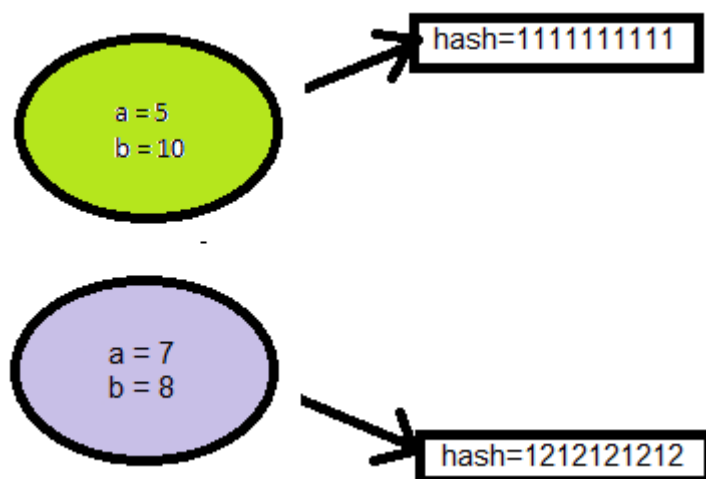
- для одинаковых и хеши одинаковы:



- если хеш-коды равны, то входные объекты не всегда равны (коллизия):



- если хеш-коды разные, то и объекты разные:



Для чего все это? Дело в том, что универсальной хеш-функции для написания метода `hashCode()` не существует: алгоритм его вычисления должен основываться на семантике объектов класса, для которого этот метод необходимо перегрузить. Руководящие принципы написания `hashCode()`:

- а) возвращение одного и того же значения для одного и того же объекта при каждом вызове;
- б) вычисление хеш-кода должно основываться исключительно на содержимом объекта, но не на его случайной уникальной информации (например, адресе в памяти);
- в) множество значений, генерируемое функцией `hashCode()` на множестве объектов контейнера (какой-либо структуры данных), должно быть распределено как можно равномернее, чтобы каждому значению соответствовало примерно равное количество объектов контейнера.
- г) хорошая хеш-функция должна стремиться сократить вероятность появления коллизий.

Приведем наиболее распространенный **рецепт для получения хешкода** переменной `value`:

1. Присвойте результирующей переменной (`result`) некоторое ненулевое простое число (например, 17)
2. В зависимости от типа поля `value` вычислите выражение по одному из 7-ми правил:

№	Тип/значение поля <code>value</code>	Вычисляемое выражение
1	<code>boolean</code>	<code>value ? 0 : 1</code>
2	<code>byte</code> , <code>char</code> , <code>short</code> , <code>int</code>	<code>(int)value</code>
3	<code>long</code>	<code>(int)(value - (value &gt;&gt;&gt; 32))</code>
4	<code>float</code>	<code>Float.floatToIntBits(value)</code>
5	<code>double</code>	<code>Double.doubleToLongBits(value)</code> а затем к результату применить преобразование, как для типа <code>long</code>
6	ссылка на объект	вызвать метод <code>hashCode()</code> этого объекта
7	ссылка на объект и <code>value</code> равно <code>null</code>	0

3. Объедините полученные в п. 2 значения, вычисляя последовательно для каждого поля выражение:

$$\text{result} = 37 * \text{result} + \text{value}$$

Если поле является массивом, примените правило 2 для каждого элемента массива. Проверьте, что равные объекты всех типов будут возвращать одинаковый hashCode, рассчитанный по этим правилам.


## Упражнение 4.7.1

Напишем программу, которая генерирует hashCode по вышеуказанным правилам:

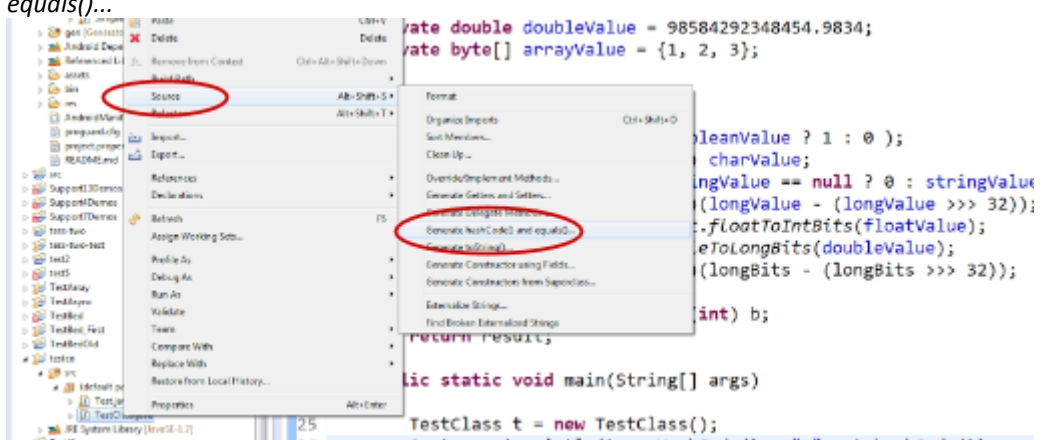
```
public class TestClass {
    private boolean booleanValue = true;
    private char charValue = 'd';
    private String stringValue = "TestClass";
    private long longValue = 348292458494983001;
    private float floatValue = 345832400.93f;
    private double doubleValue = 98584292348454.9834;
    private byte[] arrayValue = {1, 2, 3};

    @Override
    public int hashCode() {
        int result = 17;
        result = 37 * result + (booleanValue ? 1 : 0);
        result = 37 * result + (int) charValue;
        result = 37 * result + (stringValue == null ? 0 : stringValue.hashCode());
        result = 37 * result + (int)(longValue - (longValue >> 32));
        result = 37 * result + Float.floatToIntBits(floatValue);
        long longBits = Double.doubleToLongBits(doubleValue);
        result = 37 * result + (int)(longBits - (longBits >> 32));
        for( byte b : arrayValue )
            result = 37 * result + (int) b;

        return result;
    }
}
```

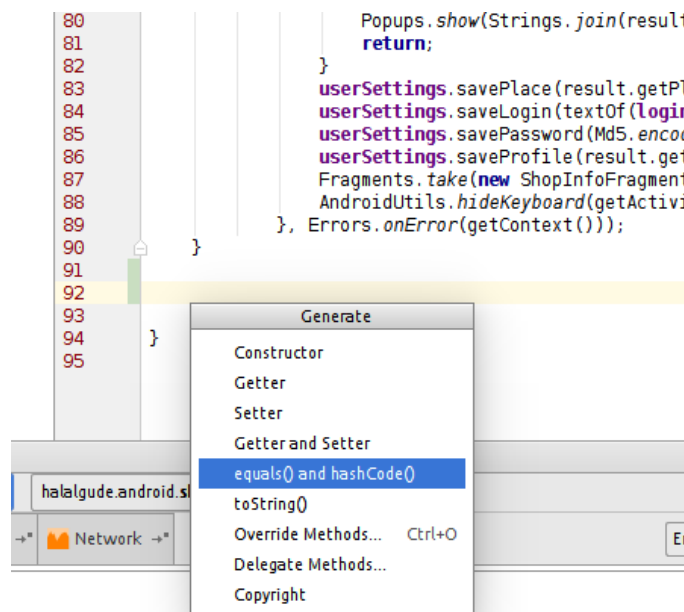


В Eclipse генерацию хешкода по этим правилам для своего объекта можно сделать автоматически: для класса в Package Explorer выбрать Source - Generate hashCode() and equals()...

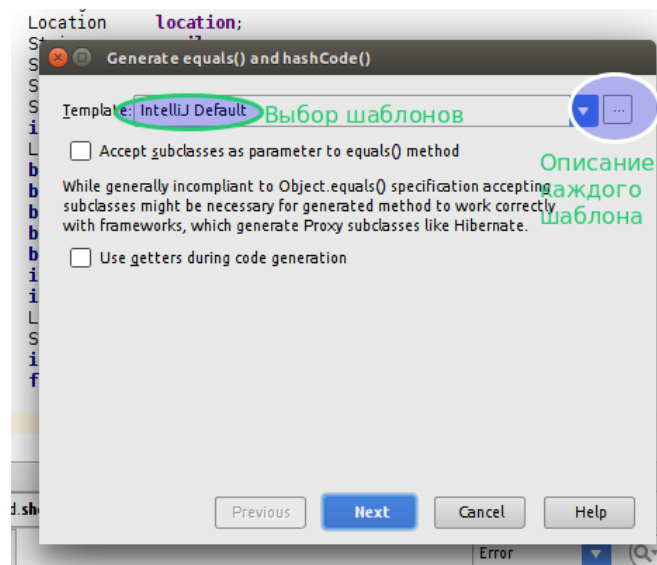




В Android Studio генерацию хешкода по этим правилам для своего объекта также можно сделать автоматически: комбинация клавиш **Alt+Insert**, в открывшемся контекстном меню выбираем **equals()** and **hashCode()**:



Далее среда предложит шаблоны, по которым можно сгенерировать эти методы, выбор полей, которые будут включены в эти методы.



Существует достаточно много алгоритмов для получения хеш-кодов. К примеру для данных строкового типа в статье Arash Partow "General Purpose Hash Function Algorithms" приведены восемь вариантов 32-битных хеш-функций:

- *rs* — простая хеш-функция из книги Роберта Седжвика 'Фундаментальные алгоритмы на C'
- *js* — побитовая хеш-функция от Justin Sobel
- *pjw* — алгоритм, основанный на работе Peter J. Weinberger
- *bkdr* — хеш-функция из книги Брайана Кернигана и Денниса Ритчи 'Язык программирования C'
- *sdbm* — специальный алгоритм, используемый в проекте SDBM

- *djb* — алгоритм, разработанный профессором Daniel J. Bernstein
- *dek* — алгоритм, предложенный Дональдом Кнутом в книге 'Искусство программирования'
- *ap* — алгоритм, разработанный Arash Partow

Еще пять вариантов:

- *faq6* — номер 6 из FAQ Боба Дженкинса
- *lookip3* — автор Боб Дженкинс
- *ly* — предложен Леонидом Юрьевым (конгруэнтный генератор)
- *rot13* — простой алгоритм с циклическим сдвигом, от Сергея Вакуленко
- *crc32* — стандартная контрольная сумма с полиномом  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

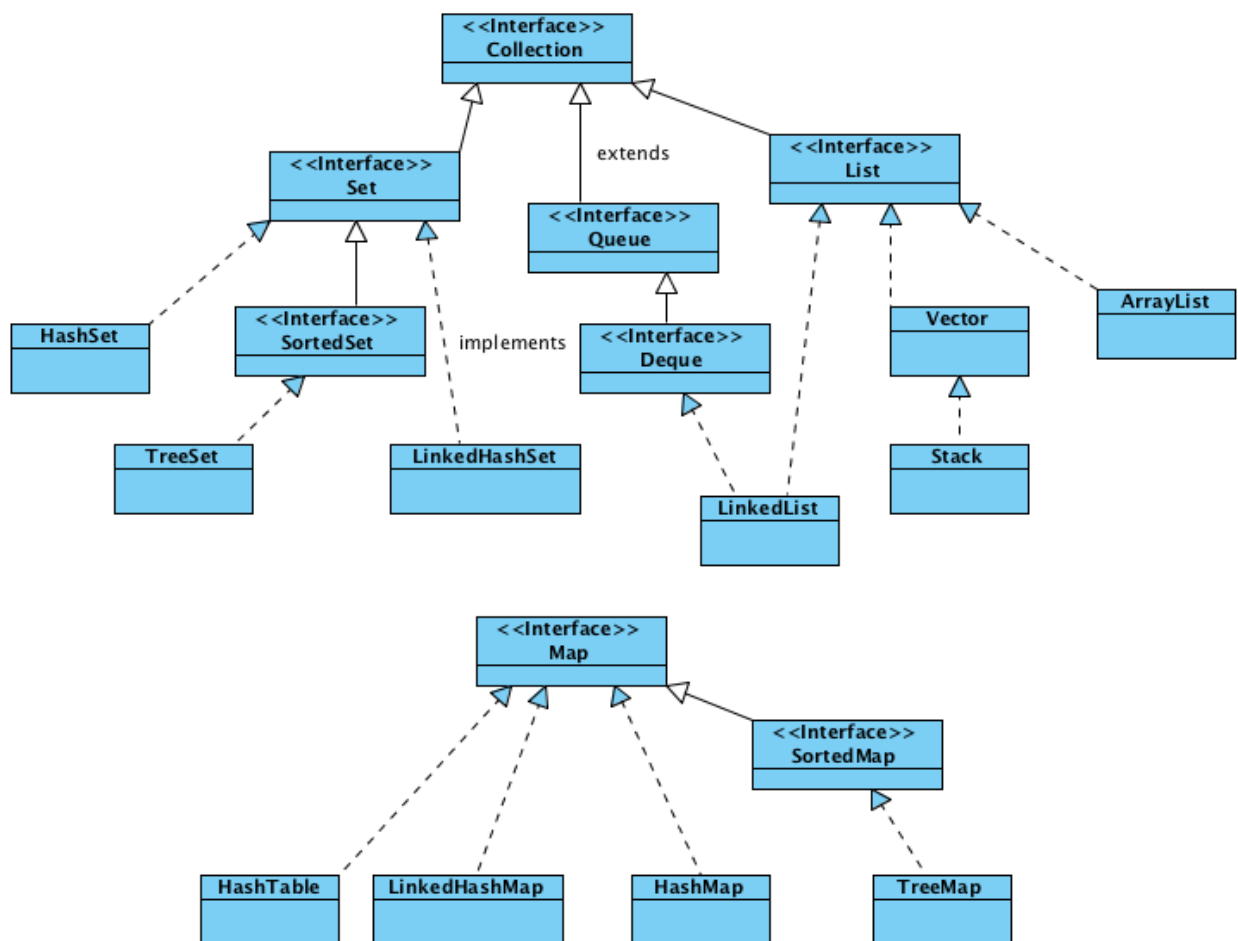
Подробнее об эффективности разных хешей можно почитать в сети. Ссылка в конце материала.

Стандартный (автоматически-генерируемый) способ генерации хешкода не всегда хорошо подходит. Ниже приведен код одного из алгоритмов вычисления хеш-кода для строк:

```
import java.util.ArrayList;

public class MyElems {
    public static void main(String[] str){
        ArrayList<String>list = new ArrayList<String>();
        list.add("One");
        list.add("Two");
        list.add("Three");
        list.add("Four");
        list.add("Five");
        list.add("Six");
        list.add("Seven");
        for(String bfr:list)
            System.out.println(hashCode(bfr));
    }
    public static int hashCode(String str) {
        int hash = 0;
        for (int i = 0; i < str.length(); i++) {
            hash = Math.abs(hash * 1664525 + str.charAt(i) + 1013904223);
        }
        return hash;
    }
}
```

## 4.7.2. Семейства контейнеров – Collections и Map



Java Collection Framework — иерархия интерфейсов и их реализаций, которая позволяет разработчику пользоваться большим количеством готовых структур данных.

Существует два семейства контейнеров:

1. Collection (коллекция) - группа отдельных элементов, сформированная по специальным правилам:
  - Set (множество) не позволяет хранить повторяющиеся элементы;
  - List (список) хранит элементы в порядке вставки;
  - Queue (очередь) выдает элементы в порядке очереди.
2. Map (карта, ассоциативные контейнеры): набор пар объектов "ключ-значение". Хранимые объекты извлекаются по ключу.

Мы уже познакомились с некоторыми из контейнеров Collection: с интерфейсами List и Queue. Рассмотрим более подробно контейнер Set.

### Контейнер Set

**Set** - описывает неупорядоченную коллекцию, не содержащую повторяющихся элементов. Это соответствует математическому понятию множества (set).

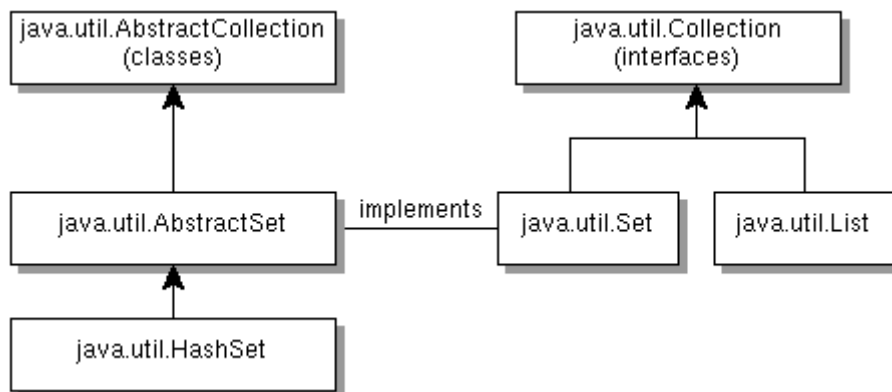
**HashSet** — реализация интерфейса Set, базирующаяся на HashMap. Внутри использует объект HashMap для хранения данных. В качестве ключа используется добавляемый элемент, а в качестве значения — объект-пустышка (new Object()). Из-за особенностей реализации порядок элементов не гарантируется при добавлении.





Метод `compareTo(Object obj)` возвращает отрицательное *integer*, ноль или положительное *integer*, когда текущее значение меньше чем, равно, или больше чем полученный объект.

### Контейнер HashSet



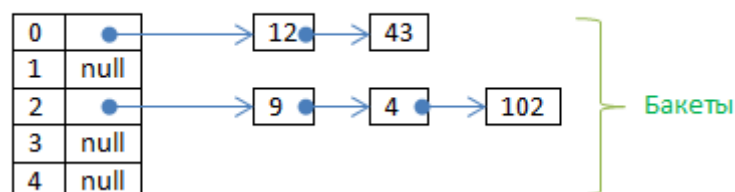
**HashSet** - коллекция, не позволяющая хранить одинаковые объекты (как и любой Set). HashSet инкапсулирует в себе объект **HashMap** (то-есть использует для хранения хеш-таблицу).

Хеш-таблица хранит информацию, используя механизм хеширования: для данных вычисляется хеш-код и он потом используется в качестве ключа, с которым ассоциируются эти данные. Преобразование ключа в хеш-код выполняется автоматически - ваш код не может напрямую индексировать хеш-таблицу.

Хеш-таблица - это обычный массив, но значения в него попадают через хеш-функцию. Как это происходит?

Рассмотрим упрощенный пример. Представим, что наша хеш-таблица имеет 5 ячеек, в нее мы помещаем поочередно некие данные (на рисунке это числа):

#### Хеш-таблица



Каждый раз для помещаемых данных вычисляется хеш-код, затем из него вычисляется остаток от деления на 5 и мы получаем значения в диапазоне от 0 до 4 - индекс ячейки массива, куда мы должны поместить значение. В ячейке массива хранится ссылка на список (bucket, бакет), в котором и сохраняются все значения, которые получили одинаковый индекс. Т.е. Хеш таблица - это массив списков.

На картинке мы видим, что у нас получилось неравномерное распределение значений по бакетам. Очевидно, что в общем случае лучше, если бакетов много и значения по ним раскладываются равномерно, тогда поиск и размещение значений в хеш-таблицу будет очень быстрым. Хорошая хеш-функция та, которая порождает хорошие ключи для распределения элементов по бакетам. Последнее требование минимизирует коллизии и предотвращает случай, когда элементы данных с близкими значениями попадают только в одну часть таблицы.

Хеш-таблицы часто применяются в базах данных, и, особенно, в языковых процессорах типа компиляторов и ассемблеров, где они обслуживают таблицы идентификаторов. В таких приложениях, таблица - наилучшая структура данных.

Хеш-таблица обеспечивает константное время выполнения методов `add()`, `contains()`, `remove()` и `size()` даже для больших наборов.

Методы:

- `public Iterator iterator()`
- `public int size()`
- `public boolean isEmpty()`
- `public boolean contains(Object o)`
- `public boolean add(Object o)`
- `public boolean addAll(Collection c)`
- `public Object[] toArray()`
- `public boolean remove(Object o)`
- `public boolean removeAll(Collection c)`
- `public boolean retainAll(Collection c)` - (`retain` — сохранить). Выполняет операцию "пересечение множеств".
- `public void clear()`
- `public Object clone()`



Если Вы хотите использовать **HashSet** для хранения объектов СВОИХ классов, то вы ДОЛЖНЫ переопределить методы `hashCode()` и `equals()`, иначе два логически-одинаковых объекта будут считаться разными, так как при добавлении элемента в коллекцию будет вызываться метод `hashCode()` класса `Object`, который скорее-всего вернет разный хеш-код для ваших объектов.

### Контейнер TreeSet

**TreeSet** — аналогично другим классам-реализациям интерфейса `Set` содержит в себе объект `NavigableMap`, что и обуславливает его поведение. Предоставляет возможность управлять порядком элементов в коллекции при помощи объекта `Comparator`, либо сохраняет элементы с использованием "natural ordering" (алфавитный порядок).

Если вы хотите использовать `TreeSet` для объектов собственного класса, то вам необходимо переопределить метод `compareTo(Object obj)` в этом классе: вы должны реализовать (implement) интерфейс `Comparable` и перегружать `compareTo(Object obj)` самостоятельно.

`TreeSet` - это реализация структуры данных "двоичное дерево поиска". Специальными алгоритмами обеспечивается быстрое размещение и поиск данных по дереву.

У многих молодых разработчиков возникает вопрос, когда использовать `HashSet`, а когда `TreeSet`. Рассмотренный ранее класс `HashSet` не гарантирует упорядоченности элементов, поскольку процесс хеширования сам по себе обычно не порождает сортированных наборов. Поэтому на практике, как правило, выбор зависит только от того, нужна ли сортированная последовательность. Если упорядоченность нужна, то используют `TreeSet`, в остальных случаях - `HashSet`.

### Упражнение 4.7.2

Рассмотрим пример, демонстрирующий отличие между контейнерами: заполнением `HashSet` и `TreeSet` одним и тем же набором строковых элементов в одной и той же последовательности; далее будем обходить их содержимые с помощью итератора и выведем на печать.

Реализуем пример в виде приложения под Android. В разметку добавим один EditText, два TextView и две кнопки. В EditText будем вводить слова, которые будем добавлять в контейнеры (TreeSet и HashSet) (по первой кнопке), по второй кнопке выведутся результаты на два TextView соответственно.

Разметка:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${relativePackage}.${activityClass}" >

    <Button
        android:id="@+id/butWrite"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/butPut"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="36dp"
        android:text="Написать" />

    <Button
        android:id="@+id/butPut"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="36dp"
        android:text="Ввести" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/butWrite" >

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="30dp"
            android:gravity="center_horizontal"
            android:orientation="vertical" >

            <TextView
                android:id="@+id/tsTv"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="TreeSet" />

            <TextView
                android:id="@+id/hsTv"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="HashSet" />
        </LinearLayout>
    </ScrollView>

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true"
android:ems="10" />
```

```
</RelativeLayout>
```

Код будет выглядеть так:

```
import java.util.HashSet;
import java.util.TreeSet;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity implements OnClickListener {
    TextView tvHs, tvTs;
    Button btnPut, btnWr;
    EditText editText;
    HashSet<String> myHashSet = new HashSet<String>();
    TreeSet<String> myTreeSet = new TreeSet<String>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btnPut = (Button) findViewById(R.id.butPut);
        btnWr = (Button) findViewById(R.id.butWrite);
        tvHs = (TextView) findViewById(R.id.hsTv);
        tvTs = (TextView) findViewById(R.id.tsTv);
        editText = (EditText) findViewById(R.id.editText1);

        btnPut.setOnClickListener(this);
        btnWr.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.butPut:
                myHashSet.add(editText.getText().toString());
                myTreeSet.add(editText.getText().toString());
                break;

            case R.id.butWrite:
                btnPut.setVisibility(View.INVISIBLE);
                //скрываем кнопку, используемую для размещения значений в контейнер
                for (String bfr : myHashSet) {
                    tvHs.append(bfr+"\n");
                }
                for (String bfr : myTreeSet) {
```

```
        tvTs.append(bfr+"\n");
    }
    break;
}
}
```

Протестируем приложение на следующих значениях:

*кошка, собака, птичка, обезьянка, лиса, медведь, волк.*

Результат получился следующий:

```
TreeSet
волк
кошка
лиса
медведь
обезьянка
птичка
собака
```

```
HashSet
собака
лиса
кошка
медведь
птичка
обезьянка
волк
```

Как можно заметить, TreeSet все разместил в алфавитном порядке.

HashSet, на первый взгляд, элементы разместил случайным образом. Однако, на самом деле **HashSet**, для эффективного размещения объектов в коллекции, использует метод `hashCode()` класса **Object**. В классах объектов, заносимых в **HashSet**, этот метод должен быть переопределен (`override`).

## Задание 4.7.1

Реализовать собственный класс `PairInteger` (пара целых чисел). Сделать так, чтобы его можно было использовать в `HashSet` (реализовать методы `hashCode` и `equals`) и в `TreeSet` (реализовать метод `compareTo`).

## Задание 4.7.2

Создать класс `Books` с полями `author`, `name`, `year`. Сделать так, чтобы его можно было использовать в классе `TreeSet`.

## Ссылки

1. Разбираемся с hashCode() и equals()  
<http://habrahabr.ru/post/168195/>
2. Коллекции и структуры данных:  
<http://www.quizful.net/post/Java-Collections>  
[habrahabr.ru/post/237043/](http://habrahabr.ru/post/237043/)
3. Эффективность хеш-кодов:  
<http://www.vak.ru/doku.php/proj/hash/efficiency>

## Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателям IT ШКОЛЫ SAMSUNG Маннапову Ильназ Магсумовичу, Ильину Владимиру Владимировичу.