



# План занятия

1. Список vs массив
2. Методы списков. Часть 1. Добавляем значения и вычисляем размер
3. Методы списков. Часть 2. Поиск и удаление элементов
4. Добавляем списки в финансовое приложение. Тренажёры
5. Что такое хэш таблица?
6. Добавляем элементы в хэш таблицу
7. Операции с хэш таблицами




# Список vs массив



## Список vs массив

Для хранения нескольких значений в программировании используются структуры данных. Одна из них вам уже знакома — массив. К примеру, массив **expenses** в финансовом приложении хранит траты за неделю.

```
double[] expenses = new double[7];
```



В **expenses** можно сохранить только семь значений — это очень мало! Современные приложения содержат данные не за неделю или месяц, а за несколько лет. Возникает вопрос: какого размера тогда должен быть массив. Можно, конечно, создать его заведомо большим — например, в тысячу элементов:

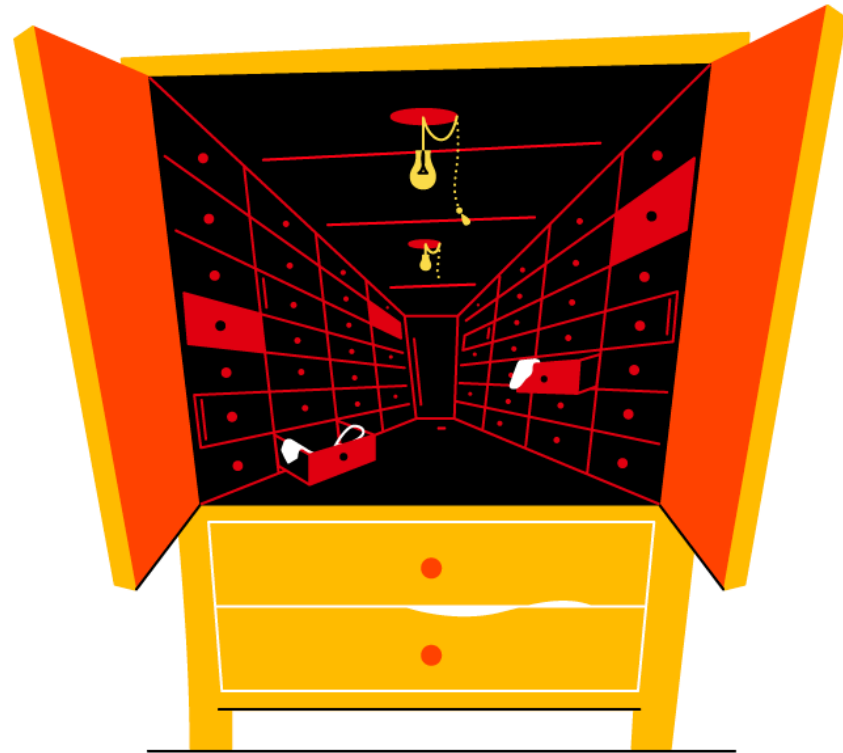
```
double[] expenses = new double[1000];
```

Но в какой-то момент и такого размера может не хватить. Эта проблема решается с помощью другой структуры данных — **списка** (англ. “list”).

Список так же, как и массив, хранит элементы одного типа. Отличие списка в том, что при заполнении его размер автоматически увеличивается. Можно представить, что массив — это обычный шкаф с фиксированным количеством ящиков, а список — это шкаф, к которому применено расширяющее заклятие — совсем как в книгах Джоан Роулинг о Гарри Поттере.



массив



список

# Создание списков

Один из базовых классов для списков в Java — **ArrayList** (англ. «список на основе массива»). Этот класс так же, как и уже известный вам **Scanner**, является частью стандартной библиотеки Java. Прежде чем создавать в коде новый список, **ArrayList** необходимо импортировать.

```
import java.util.ArrayList; // Импортировали класс ArrayList

public class Practice {
    public static void main(String[] args) {
        // Теперь можно создать список
    }
}
```

## Создание списков


При объявлении списка нужно указать класс, объекты которого он будет содержать. Для этого используются угловые <> скобки. Список может работать только со ссылочными типами. Примитивные типы, например, **double**, нужно заменить на обёртку — класс **Double**. Поэтому объявление списка расходов будет выглядеть так:

```
ArrayList<Double> expenses;
```

# Создание списков

Теперь нужно создать новый объект. При этом нужно поставить и угловые, и круглые скобки. В круглых скобках можно указать начальный размер, но это не обязательно.

```
ArrayList<Double> expenses = new ArrayList<>();
```

 Когда есть хотя бы приблизительное понимание, сколько элементов должен хранить список, то лучше указывать его размер. Это нужно для того, чтобы **ArrayList** не выполнял дорогостоящие операции расширения каждый раз, когда в него добавляется новый элемент. Но для начала мы будем работать со списками без указания размера.



# Что можно хранить в списке

Класс `ArrayList` называется в Java **обобщением**, или **дженериком** (англ. “generics”). Это означает, что он умеет работать с объектами разных типов. Например:

```
ArrayList<String> names = new ArrayList<>(); // Список имён  
ArrayList<Integer> counts = new ArrayList<>(); // Список целых чисел  
ArrayList<Hamster> hamsters = new ArrayList<>(); // Список хомяков
```

## Что можно хранить в списке

Можно даже создать список для хранения списков:

```
ArrayList<ArrayList<Double>> matrix = new ArrayList<>(); // Вы в Матрице!
```

# Задача

```
... // импортируйте пакет ArrayList
public class Practice {
    public static void main(String[] args) {
        ... // создайте список studentNames со строками
    }
}
```

- Для импорта класса **ArrayList** нужна команда **import java.util.ArrayList**.
- Тип данных в списках указывается в угловых скобках — **ArrayList<String>**.
- Новый объект-список создаётся с помощью оператора **new** и сочетания угловых и круглых скобок — **new ArrayList<>()**.

# Решение

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<String> studentNames = new ArrayList<>();
    }
}
```



## Что выбрать

И в массивах, и в списках можно хранить огромное количество значений. Однако при создании массива нужно заранее задать его размер и потом изменить его невозможно, можно только создать новый массив большего размера. При создании списка размер указывать необязательно.

В массивах можно хранить и примитивные, и ссылочные типы, а в списках — только классы. Для хранения примитивных типов в списках используются классы-обёртки.

Список так же, как и массив, хранит элементы одного типа. Отличие списка в том, что при заполнении его размер автоматически увеличивается. Можно представить, что массив — это обычный шкаф с фиксированным количеством ящиков, а список — это шкаф, к которому применено расширяющее заклятие — совсем как в книгах Джоан Роулинг о Гарри Поттере.


	<div><div>ДА</div><div>НЕТ</div></div>		Списки	Массивы
Используются для хранения множества элементов одного типа			●	●
Умеют самостоятельно изменять размер при добавлении элементов			●	●
Работают с примитивными типами			●	●
Работают с классами			●	●
Содержат много удобных методов			●	только свойство <code>length</code>





## Что выбрать

Благодаря тому, что списки умеют автоматически увеличиваться и имеют ряд удобных методов, в программах их можно встретить чаще. Массивы лучше использовать тогда, когда число элементов известно заранее.



# Методы списков. Часть 1.

## Добавляем значения и вычисляем размер





## Добавляем значения и вычисляем размер

Взаимодействие со списками в коде происходит через методы. С их помощью можно добавлять элементы в список, определять его длину, получать из списка значения, а также делать многое другое. Это методы класса **ArrayList** — и разработчику важно уметь с ними работать.

## Метод `add(E e)`

Чтобы добавить значения в список, нужен метод `add(E e)` (от англ. *add* — «добавлять»). Параметр `E e` означает, что метод принимает одно значение любого из типов. То есть можно передать строку — `add("Пиксель")`, число — `add(17)` (упаковка примитива в обёртку произойдёт автоматически) или любой объект — `add(hamster)`. Метод `add(E e)` вызывается с помощью имени списка-объекта и точечной нотации:

```
ArrayList<Double> expenses = new ArrayList<>(); // создали список трат
expenses.add(32.2); // добавили трату в список
```



## Метод `add(E e)`

В отличие от массивов, индекс элемента можно не указывать — в этом случае новый элемент добавляется в конец списка.

Если нужно добавить значение на конкретную позицию — в начало или середину списка, то можно использовать метод `add(int index, E e)`. У него есть второй параметр — `index`. Индексация в списках так же, как и в массивах, начинается с нуля:

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        expenses.add(32.2);
        expenses.add(0, 55.6); // добавили трату в начало списка
    }
}
```

Если по этому индексу уже хранится какой-то элемент — он станет следующим, а новый займёт его позицию. Список **expenses** получился таким: 55.6, 32.2. Важно помнить, что индекс не может быть больше длины списка. Иначе произойдёт исключение **IndexOutOfBoundsException** — ошибка из-за выхода за границы списка:

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        // В списке нет элементов
        expenses.add(10, 67.5); // Здесь произойдёт ошибка
    }
}
```



# Результат

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 10, Size: 0  
    at java.base/java.util.ArrayList.rangeCheckForAdd(ArrayList.java:756)  
    at java.base/java.util.ArrayList.add(ArrayList.java:481)  
    at Practice.main(Practice.java:7)
```

В список можно добавить значения только того типа, который указан при его объявлении:

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        expenses.add(32.2);
        expenses.add("150 тенге"); // Здесь произойдёт ошибка
    }
}
```




# Результат

```
java: incompatible types: java.lang.String cannot be converted to java.lang.Double
```

Ошибка возникает из-за несовместимости типов — **String** не может быть конвертирован в **Double**. Мы уже рассказывали об этом в теме о примитивах — автомобиль не может ездить по железной дороге, а поезд по автомагистрали.





Проверить, что элементы занесены в список, можно, напечатав его. Это легко сделать с помощью метода `println()`. Раскроем секрет: в качестве аргумента он принимает значения любых типов — число, строку или даже список.

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        expenses.add(120.43);
        expenses.add(290.34);
        expenses.add(420.1);
        System.out.println(expenses);
    }
}
```



# Результат

```
[120.43, 290.34, 420.1]
```

Все элементы списка напечатаны в квадратных скобках — попробуйте добавить ещё один на любую позицию и перезапустите код.

# Задача

Допишите код: импортируйте и объявите список, а затем заполните его элементами. Добавьте имя «Айгерим» после имени «Нурсая» (не меняя строки кода местами). Результат напечатайте.

```
...; // Импортируйте список

public class Practice {
    public static void main(String[] args) {
        ... femaleNames = ...; // Создайте список
        ...("Нурсая");
        ...("Мария");
        ...("Анна");
        ...(...); // Добавьте имя "Айгерим"
        ... // Распечатайте список
    }
}
```

# Решение

```
import java.util.ArrayList;


public class Practice {
    public static void main(String[] args) {
        ArrayList<String> femaleNames = new ArrayList<>();
        femaleNames.add("Нурсая");
        femaleNames.add("Мария");
        femaleNames.add("Анна");
        femaleNames.add(1, "Айгерим");
        System.out.println(femaleNames);
    }
}
```

## Метод `get(int index)`

Чтобы получить элемент из списка, нужно воспользоваться методом `get(int index)` (от англ. *get* — «получить»). В качестве аргумента передаётся индекс:

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        expenses.add(120.3); // добавили трату, её индекс 0
        expenses.add(1356.43); // добавили трату, её индекс 1
        double myExpense = expenses.get(0); // извлекли трату 0 из списка
        System.out.println(myExpense); // напечатали полученную трату
    }
}
```



Из списка так же, как и из массива, нельзя извлечь несуществующий элемент. Если попробовать это сделать — программа выдаст ошибку. Например:

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>(); // создали список расходов
        expenses.get(10); // пытаемся получить трату под индексом 10
    }
}
```

# Результат

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index 10 out of bounds for length 0
    at java.base/
jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:100)
    at java.base/
jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:106)
    at java.base/
jdk.internal.util.Preconditions.checkIndex(Preconditions.java:302)
    at java.base/java.util.Objects.checkIndex(Objects.java:385)
    at java.base/java.util.ArrayList.get(ArrayList.java:427)
    at Practice.main(Practice.java:6)
```

Сообщение в консоли означает, что нельзя получить элемент с индексом 10 из пустого списка.



## Метод `size()`

Узнать количество элементов можно с помощью метода **`size()`** (англ. «размер»).



Размер списка равен количеству сохранённых в него значений. При добавлении новых он автоматически увеличивается.

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        System.out.println("Размер списка: " + expenses.size() + " значений.");
        expenses.add(120.43);
        System.out.println("Размер списка: " + expenses.size() + " значение.");
        expenses.add(290.5);
        System.out.println("Размер списка: " + expenses.size() + " значения.");
        expenses.add(420.49);
        System.out.println("Размер списка: " + expenses.size() + " значения.");
    }
}
```



# Результат

Размер списка: 0 значений.  
Размер списка: 1 значение.  
Размер списка: 2 значения.  
Размер списка: 3 значения.

Сначала размер равен нулю. При добавлении каждой новой траты он увеличивается на одно значение.

Метод `size()` в отличие от свойства массива `length` отражает актуальное число элементов в списке, поэтому при добавлении или удалении значений его результат будет меняться. Размер массива, наоборот, задаётся при создании, поэтому результат обращения к свойству `length` будет всегда один и тот же.

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        double[] expenses = new double[10];
        System.out.println("Размер массива: " + expenses.length + " элементов.");
        expenses[0] = 120;
        expenses[1] = 290;
        System.out.println("Размер массива: " + expenses.length + " элементов.");
        expenses[2] = 420;
        System.out.println("Размер массива: " + expenses.length + " элементов.");
    }
}
```



# Результат

Размер массива: 10 элементов.  
Размер массива: 10 элементов.  
Размер массива: 10 элементов.

Сколько бы ни было элементов в массиве, **length** всегда возвращает размер, заданный при его объявлении.



## Печать списка с помощью методов `get()` и `size()`

Напечатать список можно не только с помощью `println()`, но и с помощью цикла `for` и сочетания методов `get()` и `size()`.

Цикл должен начинаться с нуля, а число повторений не должно превышать количество значений в списке. Например, напечатаем траты из списка **expenses**:

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>(); // создали список
        expenses.add(120.47); // добавили элемент
        expenses.add(290.24); // добавили элемент
        expenses.add(420.78); // добавили элемент

        System.out.println("Сейчас в списке: ");
        for (int i = 0; i < expenses.size(); i++) { // i строго меньше размера списка
            // печатаем траты и их индексы в списке
            System.out.println("Трата " + i + ": " + expenses.get(i) + " тенге.");
        }
    }
}
```



# Результат

Сейчас в списке:

Трата 0: 120.47 тенге.

Трата 1: 290.24 тенге.

Трата 2: 420.78 тенге.



## Печать списка с помощью методов `get()` и `size()`

Метод `get()` даёт возможность получить элементы. Метод `size()` задаёт условие работы цикла. Переменная итерирования `i` при этом должна быть строго меньше размера списка. Если поставить знак `<=`, то Java выдаст ошибку исключения — цикл будет пытаться напечатать несуществующий элемент.



# Задача

Отредактируйте код, который ищет максимальное значение в массиве. Измените массив на список, заполните его элементами и получите с помощью цикла **for** самое большое значение.

- Замените объявление массива на объявление списка **`ArrayList<Double> expenses = new ArrayList<>();`**.
- Добавьте элементы в список с помощью метода **`add()`** — например, **`expenses.add(92.3)`**.
- В условии цикла используйте метод **`size()`** — переменная итерирования должна быть строго меньше длины списка.
- Чтобы получить элемент списка, задействуйте метод **`get()`**. Передайте в него **`i`** в качестве аргумента.

```
public class Practice {  
    public static void main(String[] args) {  
        double[] temperatures = new double[4];  
        temperatures[0] = 92.3;  
        temperatures[1] = 12.4;  
        temperatures[2] = 74.1;  
        temperatures[3] = 45.0;  
  
        double max = 0;  
        for (int i = 0; i < temperatures.length; i++) {  
            if (temperatures[i] > max) {  
                max = temperatures[i];  
            }  
        }  
        System.out.println("Самая высокая температура: " + max + " °C.");  
    }  
}
```



# Решение

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> temperatures = new ArrayList<>();
        temperatures.add(92.3);
        temperatures.add(12.4);
        temperatures.add(74.1);
        temperatures.add(45.0);

        double max = 0;
        for (int i = 0; i < temperatures.size(); i++) {
            if (temperatures.get(i) > max) {
                max = temperatures.get(i);
            }
        }
        System.out.println("Самая высокая температура: " + max + " °C.");
    }
}
```

# Печать через сокращённый for

Лайфхак: можно обойтись и без использования методов **ArrayList** для печати списка — использовать сокращённую форму цикла **for**. Она не содержит переменной **i** и выглядит так:

```
for (Double exp : expenses) {  
    //тело цикла  
}
```



## Печать через сокращённый for

В цикле всего два параметра: переменная `exp` с типом `Double` (совпадает с типом элементов) и список для обхода — `expenses`. Имя переменной можно выбрать любое другое: `trata`, `pokupka` — от него работа цикла не зависит. Запись читается так: «Для каждого элемента типа `Double` в списке `expenses` выполнить код в теле цикла».

Например, ещё раз напечатаем список расходов:

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        expenses.add(120.23);
        expenses.add(290.32);
        expenses.add(420.03);

        System.out.println("Сейчас в списке: ");
        for (Double exp : expenses) {
            System.out.println("Трата на сумму " + exp + " тенге.");
        }
    }
}
```



# Результат

Сейчас в списке:

Трата на сумму 120.23 тенге.

Трата на сумму 290.32 тенге.

Трата на сумму 420.03 тенге





## Печать через сокращённый `for`

Сокращённая форма работает так же, как и полная — отображает все элементы из списка (или массива). Отличие заключается в отсутствии переменной `i` — из-за этого нельзя отобразить индекс элемента, как в прошлом примере. Однако если вам он не нужен, смело используйте сокращённый `for` — с ним код будет более простым и понятным.

# Задача

В списке **speeds** хранится информация о скоростях автомобиля на разных участках дороги. С помощью короткой формы цикла **for** и метода **size()** посчитайте среднюю скорость.

- Сокращённая запись цикла получится такой: **for (Integer speed : speeds).**
- В цикле нужно получить сумму всех элементов: **sum += speed;**
- Чтобы вычислить среднее значение скорости, нужно разделить полученную сумму на количество элементов в списке. Для этого понадобится вызвать метод **size()**.

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {

        ... speeds = ...; // Объявите список
        speeds.add(120);
        speeds.add(75);
        speeds.add(42);
        speeds.add(60);
        speeds.add(110);
        speeds.add(20);

        int sum = 0;
        for (... speed ...) {
            ...; // Сложите все значения в списке
        }

        int averageSpeed = ... // Вычислите среднюю скорость
        System.out.println("Средняя скорость равна " + ... + " км/ч");
    }
}
```



## Ожидаемый результат

Средняя скорость равна 71 км/ч



# Решение

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {

        ArrayList<Integer> speeds = new ArrayList<>(); // Объявите список
        speeds.add(120);
        speeds.add(75);
        speeds.add(42);
        speeds.add(60);
        speeds.add(110);
        speeds.add(20);

        int sum = 0;
        for (int speed : speeds) {
            sum += speed;
        }

        int averageSpeed = sum / speeds.size(); // Вычислите среднюю скорость
        System.out.println("Средняя скорость равна " + averageSpeed + " км/ч");
    }
}
```



## Задача 2

Вы уже немало знаете о Java и совсем скоро начнёте работать самостоятельно. Потренируйтесь это делать. Напишите от начала и до конца код небольшого приложения для маркетингового отдела зоопарка. Оно должно печатать список и количество животных, кормления которых смогут увидеть посетители, а также расписание сеансов.

Добавьте в список шиншиллу, крокодила, льва, медведя и слона — строго в таком порядке. При составлении расписания учитывайте, что первым кормят крокодила, затем слона, ещё через час — шиншиллу, в полдень — льва, а последним — медведя. Для печати животных используйте сокращённый цикл **for**.



# Ожидаемый результат

Сегодня в зоопарке можно увидеть кормления 5 животных:

Это будут:

Шиншилла

Крокодил

Лев

Медведь

Слон

Расписание кормлений:

В 9:00 – Крокодил

В 10:00 – Слон

В 11:00 – Шиншилла

В 12:00 – Лев

В 13:00 – Медведь



## Подсказки

- Начать нужно с импорта класса `ArrayList`, затем объявить класс `Practice` и метод `main`.
- Добавление элементов в список происходит с помощью вызова метода — `animals.add()`.
- Чтобы узнать количество животных, используйте метод `animals.size()`.
- Вариант цикла — `for (String animal : animals)`.
- Получить элементы можно с помощью метода `animals.get()` и индексов. Они должны идти в таком порядке: 1, 4, 0, 2, 3.
- Проверьте синтаксис: все ли скобки и точки с запятой на месте.



# Решение

```
import java.util.ArrayList;


public class Practice {
    public static void main(String[] args) {
        ArrayList<String> animals = new ArrayList<>();
        animals.add("Шиншилла");
        animals.add("Крокодил");
        animals.add("Лев");
        animals.add("Медведь");
        animals.add("Слон");
        System.out.println("Сегодня в зоопарке можно увидеть кормления " + animals.size() + "
животных:");
        System.out.println("Это будут:");
        for (String animal : animals) {
            System.out.println(animal);
        }
        System.out.println("Расписание кормлений:");
        System.out.println("В 9:00 - " + animals.get(1));
        System.out.println("В 10:00 - " + animals.get(4));
        System.out.println("В 11:00 - " + animals.get(0));
        System.out.println("В 12:00 - " + animals.get(2));
        System.out.println("В 13:00 - " + animals.get(3));
    }
}
```



# Решение 2

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<String> animals = new ArrayList<>();
        animals.add("Шиншилла");
        animals.add("Крокодил");
        animals.add("Лев");
        animals.add("Медведь");
        animals.add("Слон");
        System.out.println("Сегодня в зоопарке можно увидеть кормления " + animals.size() + "
животных:");
        System.out.println("Это будут:");
        for (String animal : animals) {
            System.out.println(animal);
        }
        System.out.println("Расписание кормлений:");
        int[] schedule = {1, 4, 0, 2, 3};
        int time = 9;
        for (int i : schedule) {
            System.out.println("В " + time + ":00 - " + animals.get(i));
            time++;
        }
    }
}
```



# Методы списков. Часть 2. Поиск и удаление элементов



## Поиск и удаление элементов

Продолжаем изучать методы класса `ArrayList`. С их помощью можно не только добавлять элементы, но и удалять их.

## Метод `remove(int index)`

Удалить элемент из списка можно с помощью метода `remove(int index)` (*remove* — англ. «удалить»). В качестве аргумента он, как и метод `get(int index)`, принимает индекс элемента:

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        expenses.add(120.23);
        expenses.add(290.32);
        expenses.add(420.03);
        System.out.println(expenses);
        expenses.remove(1); // удалили элемент под индексом 1
        System.out.println(expenses);
    }
}
```





## Результат

```
[120.23, 290.32, 420.03]  
[120.23, 420.03]
```

Элемент со значением 290.32 был удалён. Размер списка изменился. Вместо трёх трат в нём осталось две.

Можно также удалить элемент из списка не по индексу, а по значению. Для этого нужен метод `remove(Object o)`. Он принимает в качестве аргумента любой объект — при передаче примитивов произойдёт упаковка в класс-обёртку. Например:

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        expenses.add(120.23);
        expenses.add(290.32);
        expenses.add(420.03);
        expenses.remove(120.23);
        System.out.println(expenses);
    }
}
```



# Результат

[290.32, 420.03]

## Метод `clear()`

Метод `remove(int index)` стирает только один элемент из списка. Если нужно разом удалить все элементы, то нужен метод `clear()` (англ. «очистить»):

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        expenses.add(120.23);
        expenses.add(290.32);
        expenses.add(420.03);
        System.out.println("Элементов в списке - " + expenses.size());
        expenses.clear();
        System.out.println("Элементов в списке - " + expenses.size());
    }
}
```



# Результат

Элементов в списке - 3

Элементов в списке - 0

После того как список очищен, можно заново добавлять в него элементы.



## Метод `isEmpty()`

Проверить, есть ли в списке элементы, можно с помощью метода `isEmpty()` (англ. «пустой ли»).

Этот метод возвращает значение булева типа: **true** — если список пустой и **false** — если в нём содержится хотя бы один элемент.

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        expenses.add(120.23);
        expenses.add(290.32);
        expenses.add(420.03);
        if (expenses.isEmpty()) { // Проверяем, есть ли элементы в списке
            System.out.println("Нет сохранённых трат.");
        } else {
            System.out.println("Трат в списке - " + expenses.size());
        }
    }
}
```



# Результат

Трат в списке - 3

Попробуйте очистить список перед его проверкой и перезапустите код. В этом случае вы увидите сообщение «Нет сохранённых трат».





## Метод `contains(E e)`

Проверить, добавлен ли элемент в список, можно методом `contains(E e)` (англ. «содержит»). Он возвращает `true` или `false`.

В качестве аргумента передаётся элемент для поиска. Например, проверим, сохранена ли трата в 290.32 тенге:

```
import java.util.ArrayList;

public class Practice {
    public static void main(String[] args) {
        ArrayList<Double> expenses = new ArrayList<>();
        expenses.add(120.23);
        expenses.add(290.32);
        expenses.add(420.03);
        boolean isExp = expenses.contains(290.32); // Проверяем наличие элемента
        if (isExp == false) {
            expenses.add(1, 290.32); // Если элемента нет - его нужно добавить
            System.out.println("Трата " + expenses.get(1) + " тенге добавлена!");
        } else {
            System.out.println("Все расходы учтены!");
        }
    }
}
```



# Результат

Все расходы учтены!

Результат вызова метода `contains(E e)` можно сохранить в булеву переменную, как в примере, или сразу подставить в условие ветвления. Удалите элемент 290.32 из списка и перезапустите код — программа добавит его и напечатает другой вывод.



# Вопрос

Какие методы относятся к спискам, а какие нет.

- `add(E e)`
- `remove(int index)`
- `addElement(E e)`
- `next()`
- `clear()`
- `length`
- `contains(E e)`
- `delete(E e)`
- `size()`
- `println()`
- `get(int index)`
- `isEmpty()`

# ОТВЕТ

- `add(E e)`
- `remove(int index)`
- `addElement(E e)`
- `next()`
- `clear()`
- `length`
- `contains(E e)`
- `delete(E e)`
- `size()`
- `println()`
- `get(int index)`
- `isEmpty()`



# Задача


Усовершенствуйте приложение для зоопарка: добавьте возможность добавлять и удалять животных из списка, а также проверять, живёт ли указанное пользователем животное в зоопарке.

[https://github.com/practicetasks/java\\_tasks/tree/main/lists\\_and\\_hastables/task\\_1](https://github.com/practicetasks/java_tasks/tree/main/lists_and_hastables/task_1)



# Решение

<https://gist.github.com/practicetasks/a4c4e6b0aac8482adc9d0043ec749c1c>



Добавляем списки в финансовое приложение. Тренажёры





## Добавляем списки в финансовое приложение.

В ходе курса вы разработали своё первое приложение — финансовый помощник. Помощник умеет хранить траты пользователя по категориям, давать финансовые советы и конвертировать сбережения в различные валюты.

Часто разработчики пишут программу, запускают её и передают команде поддержки, а сами переходят к следующему проекту. Команда поддержки устраняет возможные недочёты в коде, вносит какие-либо другие исправления в приложение и следит за его работой.



## Добавляем списки в финансовое приложение.

Однако мир не стоит на месте. Иногда пользователям или заказчикам становится недостаточно текущих возможностей программы, и её приходится существенно дорабатывать. Так случилось и с вашим финансовым помощником.

Предлагаем вернуться в код финансового приложения, для хранения трат в приложении сейчас используется массив и это не очень удобно. Вам нужно провести рефакторинг с учётом новых знаний — сделайте так, чтобы траты хранились в списке вместо массива.



## Ранее написанный вами код

<https://gist.github.com/practicetasks/b26bf77cc7711c49df6ec43c08280e97>



## Вопрос

В вашей программе есть несколько классов. За что отвечает класс **Converter**?

- A. За вывод меню.
- B. За вывод сбережений в указанной валюте.
- C. За хранение информации о текущем балансе.
- D. За обработку различных команд в приложении.



## Ответ

- A. За вывод меню.
- B. За вывод сбережений в указанной валюте.
- C. За хранение информации о текущем балансе.
- D. За обработку различных команд в приложении.



# Вопрос

Соотнесите названия методов и задачу, которую выполняет каждый из них.

Вывод текущего баланса в указанной пользователем валюте

Добавление траты в приложение с разбивкой по категориям

Получение финансового совета

Вывод стоимости самой дорогой покупки

Вывод всех трат с указанием категории

**saveExpense**

**findMaxExpenseInCategory**

**convert**

**printAllExpensesByCategories**

**getAdvice**



## Ответ

Вывод текущего баланса в указанной пользователем валюте - **convert**

Добавление траты в приложение с разбивкой по категориям - **saveExpense**

Получение финансового совета - **getAdvice**

Вывод стоимости самой дорогой покупки - **findMaxExpenseInCategory**

Вывод всех трат с указанием категории - **printAllExpensesByCategories**



# Вопрос

За хранение какой информации отвечает поле класса **expensesByCategories**?

- A. Хранение текущего баланса пользователя
- B. Хранение всех трат с разбивкой по категориям
- C. Хранение информации о курсах валют для конвертации
- D. Хранение всех команд, доступных в приложении





## Ответ

- A. Хранение текущего баланса пользователя
- B. Хранение всех трат с разбивкой по категориям
- C. Хранение информации о курсах валют для конвертации
- D. Хранение всех команд, доступных в приложении



# Вопрос

Для чего используются циклы в методе `findMaxExpenseInCategory()`?

- A. Для итерации по значениям трат внутри категории
- B. Для итерации по названиям категорий
- C. Для итерации по спискам трат под категориями
- D. Для бесконечного запроса команды от пользователя



## Ответ

- A. Для итерации по значениям трат внутри категории
- B. Для итерации по названиям категорий
- C. Для итерации по спискам трат под категориями
- D. Для бесконечного запроса команды от пользователя



# Вопрос

В какие валюты приложение позволяет конвертировать баланс?

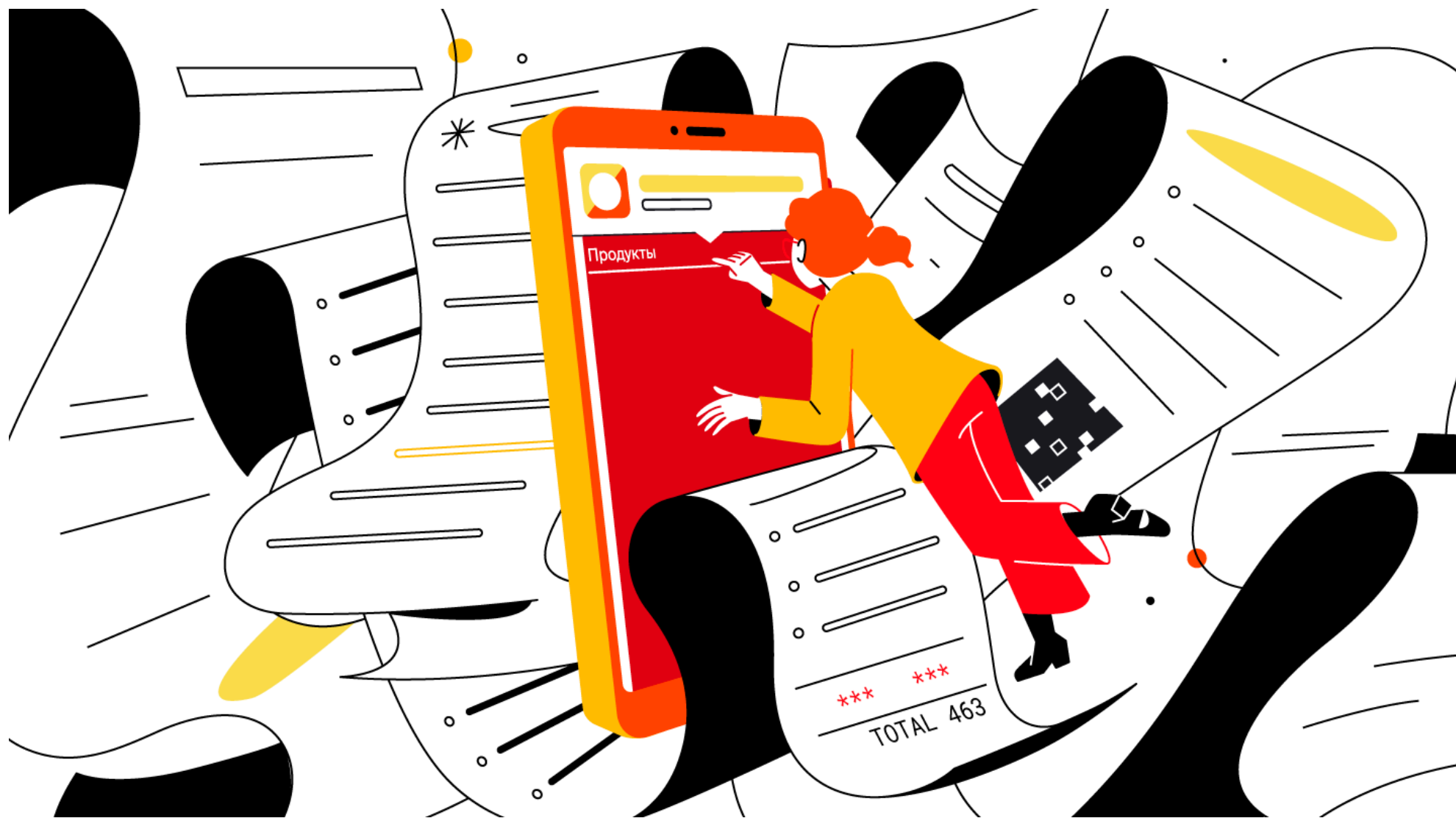
- A. рубли
- B. доллары
- C. иены
- D. франки
- E. евро



## Ответ

- A. рубли
- B. доллары
- C. иены
- D. франки
- E. евро

Замечательно! Вы вспомнили, за что отвечают разные части кода и хорошенько размялись перед тем, как приступить к рефакторингу.



# Задача

Перед вами код финансового приложения, который вы написали ранее. Для хранения трат в приложении сейчас используется массив и это не очень удобно. Вам нужно провести рефакторинг с учётом новых знаний — сделайте так, чтобы траты хранились в списке вместо массива.

[https://github.com/practicetasks/java\\_tasks/tree/main/lists\\_and\\_hastables/task\\_2](https://github.com/practicetasks/java_tasks/tree/main/lists_and_hastables/task_2)



# Решение

<https://gist.github.com/practicetasks/744c54bb84961e715c75c42574e7b088>





## Задача 2

Теперь, когда приложение работает со списками, можно расширить его функционал.

[https://github.com/practicetasks/java\\_tasks/tree/main/lists\\_and\\_hastables/task\\_3](https://github.com/practicetasks/java_tasks/tree/main/lists_and_hastables/task_3)



# Решение

<https://gist.github.com/practicetasks/2feaddba96da75d34c287fa6f9281511>



# Что такое хеш таблица



# Что такое хеш таблица

Помимо массивов и списков в Java есть ещё одна структура данных — **хеш-таблица** (англ. «Hash Map»). Её отличие в том, что вместо числового индекса используется **ключ**, и он может быть разных типов.

Вернёмся к аналогии со шкафом, где у каждого ящика есть свой номер. Совсем необязательно всегда помечать ящики цифрами.

Можно наклеить картинки или указать имена владельцев. Так, например, часто делают в детских садах — не нумеруют шкафчики, а пишут на дверцах имена детей. Ребёнок говорит, как его зовут, и воспитатель легко находит его вещи.



# Что такое хеш таблица

Хеш-таблица работает по принципу шкафчиков в детском саду, в неё сохраняются и значение — вещи ребенка, и ключ — имя малыша. Ключ — альтернатива индексу в массивах и списках. Он также уникален — не может быть двух одинаковых ключей, и по нему можно легко найти элемент. Однако у ключа есть пара отличий:

**1. Ключ может относиться к разным типам.**

Например, быть числом, символом, строкой или любым объектом. Для хранения значений примитивных типов используются классы-обёртки.

**2. Ключ должен указать сам разработчик.**

Ключ не присваивается автоматически при добавлении элемента, как индекс в массивах и списках.

# Списки и массивы

Индекс *		Значение
0	→	Красный грузовик
1	→	2 куклы и бусики
2	→	набор "Lego"
3	→	конфеты

\* Создаётся  
автоматически

# Хэш-таблица

Ключ *		Значение
Василий	→	Красный грузовик
Даша	→	2 куклы и бусики
Роман	→	набор "Lego"
Василина	→	конфеты

\* Добавляется программистом





## Что такое хеш таблица

Каждое значение в хеш-таблице всегда связано с определенным ключом. Не получится добавить значение и при этом не указать его ключ. Ключи используются для поиска и получения связанных с ними значений. Если ключ искомого значения неизвестен, то придется перебрать все значения, которые хранятся в хеш-таблице, чтобы найти нужное.



## Что такое хеш таблица

Какую структуру данных выбрать — зависит от конкретного случая. Преимущество хеш-таблицы в том, что она хранит пару ключ-значение. Это удобно, когда требуется быстро найти элемент не по номеру, а по имени, ассоциации или другому параметру.



Выберите наиболее подходящую структуру данных для каждого случая:

Меню ресторана с ценами

Фамилии посетителей выставки

Расходы за месяц по дням

Количество жителей в городах Казахстана

Продукты, которые нужно купить в магазине

Контакты в телефонной книге

Значения температуры за неделю

Перечень книг в библиотеке

Варианты: Хеш-таблица, Список, Массив



# Ответ

Меню ресторана с ценами - **Хеш-таблица**

Фамилии посетителей выставки - **Список**

Расходы за месяц по дням - **Массив**

Количество жителей в городах Казахстана - **Хеш-таблица**

Продукты, которые нужно купить в магазине - **Список**

Контакты в телефонной книге - **Хеш-таблица**

Значения температуры за неделю - **Массив**

Перечень книг в библиотеке - **Список**



# Класс HashMap

Класс хеш-таблиц — **HashMap** входит в стандартную библиотеку Java. Так же, как и класс **ArrayList**, его сначала нужно импортировать:

```
import java.util.HashMap;
```

## Класс HashMap

Теперь можно объявить хеш-таблицу. Как и списки, хеш-таблицы работают только со ссылочными типами, а для хранения примитивов используются классы-обёртки. В треугольных скобках указываются тип ключей и тип значений:

```
HashMap<String, String> officeTool;
```



## Класс HashMap

Здесь типы ключей и значений совпадают, но это необязательно. Можно создать хеш-таблицу, где они отличаются:

```
HashMap<String, Double> planetsWeight;
```



# Класс HashMap

Эта хеш-таблица предназначена для хранения данных о размерах планет: их названия будут строкового типа, а масса записана в виде дробных чисел. Так как хеш-таблицы могут работать только со ссылочными типами — используем для примитивного типа **double** класс-обёртку **Double**.



# Класс HashMap

Для создания хеш-таблицы, как и любого объекта, нужно вызвать конструктор с помощью слова **new**. При вызове конструктора класса-дженерика всегда ставятся угловые скобки. В них можно указать тип ключа и тип значения, такие же, как и при объявлении хеш-таблицы. Или можно оставить эти скобки пустыми. Закрывают конструкцию пустые круглые скобки.

```
HashMap<String, String> officeTool; // Объявили хеш-таблицу  
officeTool = new HashMap<String, String>(); // Создали объект  
officeTool = new HashMap<>(); // Так тоже можно
```

Запись с пустыми угловыми скобками короче и проще для восприятия, поэтому в дальнейшем мы будем чаще использовать её.



## Какие из этих утверждений верны о хеш-таблицах.

1. Для поиска элемента используется ключ.
2. У разных значений в хеш-таблице могут быть одинаковые ключи.
3. Ключ может быть любым примитивом.
4. Класс хеш-таблиц в стандартной библиотеке Java — HashMap.
5. Типы ключей и значений в хеш-таблице должны обязательно совпадать.
6. Ключ в хеш-таблице уникален.
7. По значению можно найти ключ.
8. Хеш-таблица хранит пары значений.



# Ответ

1. Для поиска элемента используется ключ.
2. У разных значений в хеш-таблице могут быть одинаковые ключи.
3. Ключ может быть любым примитивом.
4. Класс хеш-таблиц в стандартной библиотеке Java — `HashMap`.
5. Типы ключей и значений в хеш-таблице должны обязательно совпадать.
6. Ключ в хеш-таблице уникален.
7. По значению можно найти ключ.
8. Хеш-таблица хранит пары значений.



**Добавляем элементы в хеш-таблицу**



## Добавляем элементы в хеш-таблицу

После того как хеш-таблица создана, можно добавлять в неё элементы. Чтобы это сделать, нужно вызвать метод `put(K key, V value)`. Запись его параметров (`K key, V value`) означает, что метод принимает любые типы. В качестве первого аргумента передаётся ключ, а в качестве второго — значение.

Например, заполним хеш-таблицу инвентаризации офисной оргтехники:

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> officeTool = new HashMap<>();
        officeTool.put("S234", "Большой степлер");
        officeTool.put("P342", "Чёрно-белый принтер из коридора");
        officeTool.put("N845", "Острые ножницы");
        System.out.println(officeTool);
    }
}
```

## Результат

```
{P342=Чёрно-белый принтер из коридора, S234=Большой степлер, N845=Острые ножницы}
```

Для печати всех элементов хеш-таблицы также можно использовать метод `System.out.println()`.

После того как элемент добавлен, перезаписать его значение в хеш-таблице можно с помощью ключа:

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> officeTool = new HashMap<>();
        officeTool.put("P342", "Чёрно-белый принтер из коридора");
        System.out.println(officeTool);
        officeTool.put("P342", "Принтер из офиса 42");
        // Теперь ключу P342 соответствует значение "Принтер из офиса 42"
        System.out.println(officeTool);
    }
}
```





# Результат

{P342=Чёрно-белый принтер из коридора}  
{P342=Принтер из офиса 42}

# Задача

Напишите код: создайте хеш-таблицу **statesCapitals** и добавьте в неё страны с их столицами: Канада → Торонто, Ирак → Багдад, Австрия → Вена. Затем исправьте допущенную ошибку: столица Канады на самом деле Оттава. Хеш-таблицу распечатайте.

```
...  
// System.out.println(statesCapitals);  
...
```

## Решение

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> statesCapitals = new HashMap<>();
        statesCapitals.put("Канада", "Торонто");
        statesCapitals.put("Ирак", "Багдад");
        statesCapitals.put("Австрия", "Вена");
        statesCapitals.put("Канада", "Оттава");
        System.out.println(statesCapitals);
    }
}
```



# Хранение нескольких значений под одним ключом

В хеш-таблицу можно помещать любые ссылочные типы — разберём, насколько это может быть удобно. Допустим, вам нужно создать хеш-таблицу для записи цен в ресторанах: ключом должно стать название блюда, а значением — цена.

Сложность в том, что цена должна быть сразу в трёх валютах: в рублях, долларах и евро:

Ключ	Значение		
● Коктейль "Mojito"	350	15.50	13.20
● Тирамису	120	4.00	3.20
● Рамен	230	8.50	7.00
	₽	\$	€



# Хранение нескольких значений под одним ключом

Типом **Double** для хранения стоимости здесь не обойтись. А вот список **ArrayList<Double>** отлично справится с такой задачей. Код с добавлением элементов в хеш-таблицу с меню получится таким:

```
import java.util.ArrayList;
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, ArrayList<Double>> menu = new HashMap<>(); //создаём хеш-таблицу
        ArrayList<Double> mohitoPrice = new ArrayList<>(); //создаём список с ценами для коктейля
        mohitoPrice.add(350.0); //добавляем в список цену в рублях
        mohitoPrice.add(15.50); //добавляем в список цену в долларах
        mohitoPrice.add(13.20); //добавляем в список цену в евро
        menu.put("Коктейль Mojito", mohitoPrice); //добавляем коктейль и список ценами в хеш-таблицу

        ArrayList<Double> tiramisuPrice = new ArrayList<>(); //создаём список с ценами для тирамису
        tiramisuPrice.add(120.0);
        tiramisuPrice.add(4.00);
        tiramisuPrice.add(3.20);
        menu.put("Тирамису", tiramisuPrice); //добавляем тирамису и список с ценами в хеш-таблицу

        ArrayList<Double> ramenPrice = new ArrayList<>(); //создаём список с ценами для рамена
        ramenPrice.add(230.0);
        ramenPrice.add(8.50);
        ramenPrice.add(7.00);
        menu.put("Рамен", ramenPrice); //добавляем рамен и список с ценами в хеш-таблицу
        System.out.println(menu);
    }
}
```



## Результат

```
{Тирамису=[120.0, 4.0, 3.2], Рамен=[230.0, 8.5, 7.0],  
Коктейль Mojito=[350.0, 15.5, 13.2]}
```





# Хранение нескольких значений под одним ключом

Тип значений в списке внутри хеш-таблицы также может быть любой: **Integer**, **String** и даже **ArrayList**. Но лучше не увлекаться вложенностью, чтобы код оставался понятным и читаемым.

Работать с хеш-таблицами в коде не сложнее, чем со списками или массивами. Прежде чем вы подробнее познакомитесь с другими методами класса **HashMap** — немного практики.



# Задача

Напишите код хеш-таблицы государственных праздников **stateHolidays**, которая будет содержать месяцы и праздничные даты:



## Подсказки

- Импортируйте классы `HashMap` и `ArrayList`.
- Ключи хеш-таблицы `stateHolidays` должны быть строкового типа — `String`, а значения — списки из целых чисел `ArrayList<Integer>`.
- Создайте для каждого месяца свой список: `january`, `february`, `march`, `may`, `june`, `november`.
- Добавьте в списки элементы с помощью метода `add()`.
- Добавьте элементы в хеш-таблицу с помощью метода `put()`.



# Решение

```
import java.util.ArrayList;
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, ArrayList<Integer>> stateHolidays = new HashMap<>();
        ArrayList<Integer> january = new ArrayList<>();
        january.add(1);
        january.add(7);
        stateHolidays.put("Январь", january);
        ArrayList<Integer> march = new ArrayList<>();
        march.add(8);
        stateHolidays.put("Март", march);
        ArrayList<Integer> may = new ArrayList<>();
        may.add(1);
        may.add(9);
        stateHolidays.put("Май", may);
        System.out.println(stateHolidays);
    }
}
```



# Операции с хеш-таблицами

## Операции с хеш-таблицами

Помимо метода `put()` у класса хеш-таблиц есть и другие. Их имена и функционал частично совпадают с методами списков, поэтому их достаточно легко запомнить. Рассказывать обо всех методах хеш-таблиц мы не будем — остановимся только на самых распространённых.

Можно изучить этот вопрос подробнее, если обратиться [к официальной документации](#) от компании Oracle. Далее в уроках мы будем иногда ссылаться на документацию, чтобы после прохождения курса вы умели находить там нужную информацию самостоятельно.



## Получаем элементы из хеш-таблицы

Получить значение из хеш-таблицы легко, если известен ключ. Нужно вызвать метод `get(Object key)`. Сигнатура этого метода означает, что он принимает в качестве аргумента ключ любого ссылочного типа (при передаче примитива произойдёт автоматическая упаковка в класс-обёртку).



Извлечём значение по ключу:

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> officeTool = new HashMap<>();
        officeTool.put("S234", "Большой степлер");
        officeTool.put("P342", "Чёрно-белый принтер");
        officeTool.put("N845", "Острые ножницы");

        String tool = officeTool.get("N845");
        System.out.println(tool); // Получили "Острые ножницы"
    }
}
```



# Результат

Острые ножницы

В таблице хранятся названия стран и их столицы. Напечатайте названия столиц Аргентины и Норвегии.

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> countriesCapitals = new HashMap<>();
        countriesCapitals.put("Франция", "Париж");
        countriesCapitals.put("Аргентина", "Буэнос-Айрес");
        countriesCapitals.put("Россия", "Москва");
        countriesCapitals.put("Америка", "Вашингтон");
        countriesCapitals.put("Япония", "Токио");
        countriesCapitals.put("Норвегия", "Осло");

        System.out.println("Столица Аргентины: " + ...);
        System.out.println("Столица Норвегии: " + ...);

    }
}
```



# Решение

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> countriesCapitals = new HashMap<>();
        countriesCapitals.put("Франция", "Париж");
        countriesCapitals.put("Аргентина", "Буэнос-Айрес");
        countriesCapitals.put("Россия", "Москва");
        countriesCapitals.put("Америка", "Вашингтон");
        countriesCapitals.put("Япония", "Токио");
        countriesCapitals.put("Норвегия", "Осло");

        System.out.println("Столица Аргентины: " + countriesCapitals.get("Аргентина"));
        System.out.println("Столица Норвегии: " + countriesCapitals.get("Норвегия"));
    }
}
```



## Получаем элементы из хеш-таблицы

Если попробовать извлечь элемент по ключу, которого нет в хеш-таблице, Java вернёт **null** — нужного значения нет. Старайтесь избегать таких случаев, потому что может произойти ошибка **NullPointerException** (англ. «ошибка ссылки на **null**»).

Например, попробуйте напечатать список заказов клиента, которого нет в таблице **orders**:

```

import java.util.ArrayList;
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, ArrayList<Integer>> orders = new HashMap<>();

        ArrayList<Integer> ordersNum = new ArrayList<>(); // Создаём новый список
        ordersNum.add(1);
        ordersNum.add(5);
        ordersNum.add(6);
        orders.put("Иван И.", ordersNum); // Добавили имя клиента и список его заказов

        /* Для нового клиента новый список.
           Можно не создавать ещё одну переменную, а переиспользовать имеющуюся.*/
        ordersNum = new ArrayList<>();
        ordersNum.add(2);
        ordersNum.add(4);
        ordersNum.add(3);
        orders.put("Ольга С.", ordersNum);

        // Заводим переменную для списка заказов конкретного клиента
        ArrayList<Integer> customerOrders = orders.get("Костя Д.");
        // Печатаем номера заказов конкретного клиента
        for (int orderNum : customerOrders) {
            System.out.println("Заказ № " + orderNum);
        }
    }
}

```

## Результат

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke  
"java.util.ArrayList.iterator()" because "customerOrders" is null  
at Practice.main(Practice.java:25)
```

Так как метод **get** возвращает **null** (указанного ключа нет в таблице), при попытке запустить цикл возникает исключение. Исправьте ошибку — укажите ключ из таблицы, например, "Ольга С.". Тогда **customerOrders** получит ссылку на список, и программа работает корректно.





## Получаем элементы из хеш-таблицы

Если ключ неизвестен, но известно нужное значение, то можно получить его с помощью короткой формы цикла **for** и метода **values()** (англ. «значения»). Этот метод возвращает все значения, которые есть в таблице.

Тип переменной итерирования в цикле должен совпадать с типом значений в хеш-таблице, а её имя может быть любым. Найдём и выведем в консоль значение "Острые ножницы":

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> officeTool = new HashMap<>();
        officeTool.put("S234", "Большой степлер");
        officeTool.put("P342", "Чёрно-белый принтер");
        officeTool.put("N845", "Острые ножницы");

        for (String tool : officeTool.values()) {
            if (tool.equals("Острые ножницы")) {
                System.out.println(tool); // Получили "Острые ножницы"
            }
        }
    }
}
```



# Результат

## Острые ножницы

Значения хранятся в виде строк, поэтому переменная `tool` имеет тип `String`. Чтобы найти нужное значение, метод `values()` обходит всю хеш-таблицу. Удалите ветвление, оставьте печать `tool` и перезапустите код — в консоль будут выведены все значения.



# Задача

В таблице собраны имена клиентов зоомагазина и сумма, на которую они делали заказы. Посчитайте, сколько всего денег было потрачено на питомцев.

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, Double> orders = new HashMap<>();
        orders.put("Иван И.", 4345.5);
        orders.put("Ольга С.", 76564.43);
        orders.put("Александр Т.", 1234.86);
        orders.put("Александр Р.", 23432.87);
        orders.put("Екатерина О.", 1034753.6);
        orders.put("Ярослав В.", 450.0);

        ... // Объявите переменную, где будет сохранена общая сумма
        for (...) { // Пройдитесь в цикле по значениям
            ...
        }

        System.out.println("Всего было совершено заказов на сумму: " + ...);
    }
}
```



# Решение

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, Double> orders = new HashMap<>();
        orders.put("Иван И.", 4345.5);
        orders.put("Ольга С.", 76564.43);
        orders.put("Александр Т.", 1234.86);
        orders.put("Александр Р.", 23432.87);
        orders.put("Екатерина О.", 1034753.6);
        orders.put("Ярослав В.", 450.0);

        double total = 0; // Объявите переменную, где будет сохранена общая сумма
        for (Double value : orders.values()) {
            total += value;
        }

        System.out.println("Всего было совершено заказов на сумму: " + total);
    }
}
```

Получить все ключи хеш-таблицы тоже можно. Для этого нужно воспользоваться сокращённой формой цикла и методом `keySet()` (англ. «набор ключей»):

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> officeTool = new HashMap<>();
        officeTool.put("S234", "Большой степлер");
        officeTool.put("P342", "Чёрно-белый принтер");
        officeTool.put("N845", "Острые ножницы");

        for (String inventory : officeTool.keySet()) {
            System.out.println(inventory);
        }
    }
}
```





# Результат

P342

S234

N845

## Удаление элементов

В хеш-таблицах элемент удаляется так же, как и в списках — с помощью метода `remove(Object key)` (англ. «удалить»). Только в него нужно передать не индекс, а ключ элемента, который вы собираетесь удалить:

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> officeTool = new HashMap<>();
        officeTool.put("S234", "Большой степлер");
        officeTool.put("P342", "Чёрно-белый принтер");
        officeTool.put("N845", "Острые ножницы");

        officeTool.remove("P342");
        System.out.println(officeTool.get("P342"));
    }
}
```



## Результат

`null`

Программа напечатает `null` — элемент с ключом "P342" удалён из хеш-таблицы.

Метод `remove(Object key)` удаляет элементы по одному. Если потребуется очистить хеш-таблицу полностью, то нужен тот же метод, что и в списках — `clear()` (англ. «очистить»):

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> officeTool = new HashMap<>();
        officeTool.put("S234", "Большой степлер");
        officeTool.put("P342", "Чёрно-белый принтер");
        officeTool.put("N845", "Острые ножницы");

        officeTool.clear(); // в хеш-таблице больше нет элементов
        System.out.println(officeTool);
    }
}
```



# Результат





# Проверка наличия элемента

Выяснить, занесён ли элемент в хеш-таблицу, можно с помощью методов `containsKey(Object key)` (англ. «содержит ключ») — проверяет наличие ключа и `containsValue(Object value)` (англ. «содержит значение») — проверяет наличие значения. В качестве аргумента передаётся, соответственно, либо ключ, либо значение любых ссылочных типов.

Если поиск завершится результатом, метод вернёт **true**, в обратном случае — **false**:

```
import java.util.HashMap;

public class Practice {
    public static void main(String[] args) {
        HashMap<String, String> statescapitals = new HashMap<>();
        statescapitals.put("Россия", "Москва");
        statescapitals.put("Франция", "Париж");
        statescapitals.put("Италия", "Рим");

        System.out.println(statescapitals.containsKey("Италия")); // отобразит true
        System.out.println(statescapitals.containsKey("Германия")); // отобразит false

        System.out.println(statescapitals.containsValue("Париж")); // отобразит true
        System.out.println(statescapitals.containsValue("Пекин")); // отобразит false
    }
}
```



# Результат

```
true  
false  
true  
false
```





## Какие из этих утверждений верные?

- A. С помощью `System.out.println()` можно отобразить содержимое хеш-таблицы целиком.
- B. Цикл не рекомендуется использовать, так как есть метод `System.out.println()`.
- C. Если необходимо распечатать только значения из хеш-таблицы, нужно использовать метод `keySet()`.
- D. Обход хеш-таблицы можно выполнять только по ключам.
- E. Методы для удаления элементов у хеш-таблицы такие же, как и у списков.
- F. Метод `containsKey()` по ключу вернёт элемент хеш-таблицы.
- G. Используя ключ, можно получить связанное с ним значение или удалить элемент хеш-таблицы.

## Ответ

- A. С помощью `System.out.println()` можно отобразить содержимое хеш-таблицы целиком.
- B. Цикл не рекомендуется использовать, так как есть метод `System.out.println()`.
- C. Если необходимо распечатать только значения из хеш-таблицы, нужно использовать метод `keySet()`.
- D. Обход хеш-таблицы можно выполнять только по ключам.
- E. Методы для удаления элементов у хеш-таблицы такие же, как и у списков.
- F. Метод `containsKey()` по ключу вернёт элемент хеш-таблицы.
- G. Используя ключ, можно получить связанное с ним значение или удалить элемент хеш-таблицы.

# Задача

Допишите код приложения, которое позволяет пользователям узнать год основания легендарных музыкальных коллективов, а также сохранить информацию о своей любимой группе. Вам нужно объявить и проинициализировать хеш-таблицу, напечатать все хранящиеся в ней группы, а также дописать реализацию цифрового меню. Для считывания из консоли пользуйтесь методом `nextLine()`. В комментариях прекода оставлены фразы для печати и имена переменных.

[https://github.com/practicetasks/java\\_tasks/tree/main/lists\\_and\\_hastables/task\\_4](https://github.com/practicetasks/java_tasks/tree/main/lists_and_hastables/task_4)



# Решение

<https://gist.github.com/practicetasks/a2692f2cd8d6fb1bb0099bd982edabad>



## Задача 2

В хеш-таблице `customersOrders` класса `OrdersManager` собрана информация о клиентах зоомагазина (это ключи) и их заказах. Дополните код методов:

[https://github.com/practicetasks/java\\_tasks/tree/main/lists\\_and\\_hastables/task\\_5](https://github.com/practicetasks/java_tasks/tree/main/lists_and_hastables/task_5)



# Решение

<https://gist.github.com/practicetasks/452cbea6e0ef7b93d9e4ef8554611732>