

Конспект спринта №2

Типы данных

Типы данных

Приведение типов

Ссылочные типы (классы)

Классы-обёртки

Методы

Передача по ссылке

Операции с примитивами

Арифметические операции

Порядок арифметических операций

Логические операции

Списки и хэш-таблицы

Списки

Методы списков

Хэш-таблицы

Методы хэш-таблиц

Java относится к **типизированным** языкам программирования с **сильной**, или **строгой типизацией**. Тип переменной определяет не только множество значений, которые могут быть ей присвоены, но и операции, которые можно с ней осуществлять. Например, в Java можно поделить одно число на другое, но нельзя поделить строку на строку.

Java **ещё и язык с явной статической типизацией**. Явная типизация означает, что типы задаёт сам разработчик при написании кода, а статическая — что это происходит до того, как программа запущена.



Важно не забывать, что тип данных в Java указывается не только у переменных, но и у возвращаемых значений метода, переменных итерирования в циклах, параметров метода и конструктора объектов.

Все типы в Java делятся на два вида: **примитивы** (или **примитивные типы**) и **классы** (их ещё называют **ссылочными типами**). Названия классов пишутся с заглавной буквы: `String` или `Hamster`. Примитивы же — со строчной, например `int` и `double`. Они хранят конкретное значение, например число или символ. Размер примитивов в памяти компьютера фиксирован.

Целые числа

Название	Примечания	Диапазон значений	Размер переменной
<code>byte</code>	(от англ. <i>bite</i> — «кусок», «откусанная часть»)	от -128 до 127	8 бит
<code>short</code>	(англ. <i>short</i> — «короткий»)	-32 768 до 32 767	16 бит
<code>int</code>	(от англ. <i>integer</i> — «целочисленный») Самый распространённый тип целых чисел.	от -2 147 483 648 до 2 147 483 647	32 бита
<code>long</code>	(англ. <i>long</i> — «длинный»)	от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	64 бита

Числа с плавающей точкой

Название	Примечание	Диапазон значений	Размер переменной
<code>float</code>	(англ. «плавающий», подразумевается «плавающая точка»)	от $1.4 * 10^{-45}$ до $3.4 * 10^{38}$ (как положительные, так и отрицательные числа)	32 бита
<code>double</code>	(англ. «двойной» — размер переменной в два раза больше <code>float</code> , чем обеспечивается «двойная» точность)	от $4.9 * 10^{-324}$ до $4.9 * 10^{308}$ (как положительные, так и отрицательные числа)	64 бита

Ошибка при отсутствии значения у переменной — это стандартное поведение программы на Java. Однако **если не проинициализировать значения полей внутри класса, такой ошибки не произойдёт**. Полям будет присвоено **значение по умолчанию** в зависимости от их типа.

У целочисленных типов — это `0`, у дробных — `0.0`, у символьного это `\u0000` (символ, который обозначает пустоту или «ничего»), у логического типа —

`false`. У ссылочных типов значение по умолчанию — `null` (англ. «ноль»). Оно означает, что переменная ссылается «в никуда».



Обратите внимание, использовать значения по умолчанию нужно очень аккуратно! Когда нет явной инициализации, нельзя отследить, как в переменную попало «нулевое» значение.

Приведение типов

Приведение типов может быть автоматическим, когда программисту ничего не нужно делать, и явным, когда разработчик самостоятельно преобразует один тип в другой.

Автоматическое (неявное) приведение типов работает, когда типы данных с меньшим диапазоном нужно привести к типам с большим диапазоном.

Подобное преобразование называется **расширяющим приведением типа**.

Оно возможно для любых числовых типов. Например, `short` можно расширить до `long`, `float` до `double`, а любое целое число до дробного.

Java не позволяет автоматически провести **сужающее приведение типов**. Но можно привести типы самостоятельно. В круглых скобках перед значением не того типа нужно указать нужный (`byte`). Это называется **явным приведением типов**. Так можно привести любой числовой тип с большим диапазоном к типу с меньшим, но будьте осторожны — помните о рисках потерять значение.

Ссылочные типы (классы)

К ссылочным типам относятся строки `String`, массивы (хоть они и не всегда пишутся с заглавной буквы), классы стандартной библиотеки, такие как `Scanner` и `Random`, а также те классы, которые разработчики сами создают при написании программ.

Классы отличаются от примитивов, конечно, не только тем, что пишутся с заглавной буквы. Первое и самое главное отличие — примитивов всегда ровно восемь, классов же может быть столько, сколько потребуется. В больших программах их тысячи.

Переменные примитивных типов хранят в себе конкретное значение, а переменные классов — ссылку на него. Именно поэтому классы иначе называют ссылочными типами.

При создании переменной примитивного типа под неё всегда выделяется один и тот же объём памяти вашего компьютера, а вот размер объекта класса не фиксирован.

Классы-обёртки

Классы-обёртки — специальные классы из стандартной библиотеки Java, призванные расширить функционал и возможности использования примитивных типов.

В отличие от примитивов классы-обёртки:

- Хранят не значение, а ссылку на него.
- Не имеют фиксированного размера в памяти компьютера.
- В качестве значения по умолчанию возвращают `null`.
- Обладают своими методами.

Имена классов-обёрток являются производными от названий примитивов и пишутся в коде с заглавной буквы. Диапазон значений класса-обёртки такой же, как и у соответствующего ему примитива.

Методы

`parse[примитив]()` — метод, позволяющий преобразовывать строки в свой тип. Есть у всех классов-обёрток, кроме `Character`.

`max()` и `min()` — находит максимальное и минимальное значение из двух вариантов. Вызываются также с помощью имени класса-обёртки и точечной нотации. В качестве аргументов в методы нужно передавать значения соответствующего класса или его примитива. Нет у `Byte` и `Short`.

`[имя примитива, к которому нужно привести]value()` — приведение переменных классов-обёрток к примитивам.

Передача по ссылке

Переменные примитивного типа хранят в себе непосредственно сами значения. Поэтому, когда вы используете такую переменную в качестве аргумента, её содержимое копируется в метод.

В качестве аргумента в метод можно передать как примитив, так и объект класса. В первом случае состоится передача **по значению**, а во втором случае — **по ссылке**.

Передача по ссылке работает так: в метод передаётся не значение, а ссылка на него, и переменная (объект, на который указывает ссылка) при этом не дублируется. Метод переходит по ссылке и меняет значение в первоисточнике. Старое значение не сохраняется.



При передаче в метод класса-обёртки нужно быть очень внимательным! Из-за автоматических процессов упаковки и распаковки в примитив и обратно можно получить совсем не тот результат, который ожидается.

Упаковка происходит каждый раз, когда переменной класса-обёртки передаётся значение соответствующего ему примитивного типа. Обратный процесс по приведению класса-обёртки к примитиву называется ***unboxing***, или **распаковкой типов**. Он также происходит автоматически. Есть только одно исключение — если в переменной класса-обёртки хранится значение `null`, при распаковке Java выдаст ошибку.

Операции с примитивами

Типы данных

Приведение типов

Ссылочные типы (классы)

Классы-обёртки

Методы

Передача по ссылке

Операции с примитивами

Арифметические операции

Порядок арифметических операций

Логические операции

Списки и хэш-таблицы

Списки

Методы списков

Хэш-таблицы

Методы хэш-таблиц

Арифметические операции

Все арифметические операции можно производить как с целыми, так и с дробными числами. При операциях с дробными типами в результате получается дробное число, при работе с целочисленными типами — результат тоже будет целочисленным.

Когда в арифметическом выражении есть дробное число, то результат всегда будет дробным, так как значение типа `double` или `float` нельзя сохранить в переменной целого типа `int` или `long`.

Арифметические операторы

Аа Название операции	≡ Пример
<u>Сложение</u>	<code>a = b + c;</code>
<u>Вычитание</u>	<code>a = b - c;</code>
<u>Умножение</u>	<code>a = b * c;</code>
<u>Деление</u>	<code>a = b / c;</code>
<u>Остаток от деления</u>	<code>a = b % c;</code>

Операторы, применяемые к той же переменной

Аа Название операции	≡ Пример	≡ Короткая запись
<u>Сложение</u>	<code>a = a + b;</code>	<code>a += b;</code> (для увеличения на единицу — <code>a++;</code> или <code>++a;</code>)
<u>Вычитание</u>	<code>a = a - b;</code>	<code>a -= b;</code> (для уменьшения на единицу — <code>a--;</code> или <code>--a;</code>)
<u>Умножение</u>	<code>a = a * b;</code>	<code>a *= b;</code>
<u>Деление</u>	<code>a = a / b;</code>	<code>a /= b;</code>
<u>Остаток от деления</u>	<code>a = a % b;</code>	<code>a %= b;</code>

Порядок арифметических операций

Порядок выполнения арифметических операций в программировании основывается на математических правилах.

1. Сначала выполняются умножение и деление;
2. Затем — сложение и вычитание;

3. При наличии скобок действия в них выполняются первыми;
4. Считаём всегда слева направо.

Нужно также учитывать порядок выполнения операторов инкремента и декремента.

1. Первыми отработают операторы префиксного инкремента и декремента — `++a` и `--a`;
2. Затем — операторы умножения, деления и остатка от деления: `*`, `/` и `%`;
3. Третьи на очереди — операторы сложения и вычитания `+` и `-`;
4. Последними выполняются операции с постфиксным инкрементом и декрементом: `a++` и `a--`.

Операции с постфиксным инкрементом и декрементом выполняются только после того, как все другие завершены. Поэтому они не влияют на общий результат выражения, а меняют только значение своей переменной.



Если в арифметическом выражении использованы операции инкремента и декремента, то важно учитывать, что они всегда меняют исходное значение переменной. Эти операции сами по себе являются законченными выражениями и для программы нет разницы, стоят они отдельно или внутри каких-либо вычислений.

Логические операции

Утверждение

В переменную типа `boolean` чаще всего записывается некое утверждение. Оно может быть либо истинным — тогда переменная принимает значение `true`, либо ложным — значение переменной будет `false`. На основе утверждения можно составить логическую конструкцию — ветвление.

Отрицание

Чтобы изменить значение логического выражения на противоположное, нужно использовать **отрицание**. Оно обозначается **оператором** `!` перед утверждением. Отрицание можно применить к любому логическому выражению.

Но не всегда стоит увлекаться отрицанием. Иногда достаточно просто поменять знак. Логические выражения `temperature <= 10` и `!(temperature > 10)` равнозначны

между собой, но первое воспринимать проще. Аналогично с отрицанием равенства: `!(code == 9999)` можно заменить на более удобную запись с оператором неравенства `code != 9999`

Чтобы проверить несколько условий, в программировании используются **логические операторы И и ИЛИ**.

Логические операции

Аа Название	≡ Описание	≡ Обозначение в коде	≡ Пример использования
<u>Отрицание</u>	Изменяет логическое выражение на противоположное: <code>true</code> на <code>false</code> , <code>false</code> на <code>true</code>	<code>!</code>	<code>!isRainy</code>
<u>Логическое ИЛИ</u>	Истинно, если истинна хотя бы одна часть выражения	<code> </code>	<code>isEven isPositive</code>
<u>Логическое И</u>	Истинно, если истинны все части выражения	<code>&&</code>	<code>isEven && isPositive</code>

При комбинировании разных логических операций в одном выражении или условии нужно учитывать приоритет их выполнения в коде:

1. Сначала всегда выполняется отрицание `!`.
2. Логическое умножение предшествует сложению — поэтому логическое И `&&` в приоритете.
3. Логическое ИЛИ `||` при наличии других операций выполняется последним.

Логические операции так же, как и арифметические, выполняются слева направо. Операции в скобках вычисляются в первую очередь — поэтому скобки всегда помогут добиться нужного порядка действий в выражении.

Списки и хэш-таблицы

Типы данных

[Приведение типов](#)

[Ссылочные типы \(классы\)](#)

[Классы-обёртки](#)

[Методы](#)

[Передача по ссылке](#)

[Операции с примитивами](#)

[Арифметические операции](#)

[Порядок арифметических операций](#)

[Логические операции](#)

[Списки и хэш-таблицы](#)

[Списки](#)

[Методы списков](#)

[Хэш-таблицы](#)

[Методы хэш-таблиц](#)

Списки


Список — это структура данных, которая так же, как и массив, хранит элементы одного типа. Отличие списка в том, что при заполнении его размер автоматически увеличивается.

Для списков в Java есть специальный класс — `ArrayList`, который является частью стандартной библиотеки Java. Класс `ArrayList` называется в Java **обобщением**, или **дженериком** (англ. *generics*). Это означает, что он умеет работать с объектами разных типов.

```
ArrayList<String> names = new ArrayList<>(); // Список имён
ArrayList<Integer> counts = new ArrayList<>(); // Список целых чисел
ArrayList<Hamster> hamsters = new ArrayList<>(); // Список хомяков
ArrayList<ArrayList<Double>> matrix = new ArrayList<>(); // Вы в Матрице!
```

1. Для списков обязательно следует импортировать пакет `java.util.ArrayList`.
2. При создании списка можно указать размер, но это не обязательно.
3. При создании списка в угловых скобках необходимо указать, какой тип данных он будет хранить.

	ДА	НЕТ	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>



Методы списков

Вызываются с помощью имени списка-объекта и точечной нотации. Параметр `E e` означает, что метод принимает одно значение любого из типов.

Метод `add(E e)` — добавить значения в список. В отличие от массивов, индекс элемента можно не указывать, в этом случае новый элемент добавляется в конец списка.

Метод `add(int index, E e)` — добавить значение на конкретную позицию.

Индексация в списках так же, как и в массивах, начинается с нуля. Если по этому индексу уже хранится какой-то элемент — он станет следующим, а новый займёт его позицию.

Метод `get(int index)` — получить элемент из списка.

Метод `size()` — узнать количество элементов списка. Размер списка равен количеству сохранённых в него значений.

Метод `remove(int index)` — удалить элемент из списка по индексу.

Метод `remove(Object o)` — удалить элемент из списка по значению.

Метод `clear()` — удалить все элементы списка.

Метод `isEmpty()` — проверить, есть ли в списке элементы. Возвращает значение булева типа: `true` — если список пустой, `false` — если в нём содержится хотя бы один элемент.

Метод `contains(E e)` — проверить, добавлен ли элемент в список. Результат вызова можно сохранить в булеву переменную или сразу подставить в условие ветвления.



Важно помнить, что индекс не может быть больше длины списка. Иначе произойдёт исключение `IndexOutOfBoundsException` — ошибка из-за выхода за границы списка

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.

Другой вариант — печать через сокращённый `for`:

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

В цикле всего два параметра: переменная `exp` с типом `Double` (совпадает с типом элементов) и список для обхода — `expenses`. Запись читается так: «Для каждого элемента типа `Double` в списке `expenses` выполнить код в теле цикла».

Хэш-таблицы

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. 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.

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:

```
import java.util.HashMap;  
HashMap<String, Double> planetsWeight;
```

Для создания хеш-таблицы, как и любого объекта, нужно вызвать конструктор с помощью слова `new`.

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

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.

Методы хэш-таблиц

Имена и функционал методов хэш-таблиц частично совпадают с методами списков. Все методы хеш-таблиц можно найти в [официальной документации о HashMap](#).

Метод `put(K key, V value)` — добавить элементы. Принимает любые типы. В качестве первого аргумента передаётся ключ, а в качестве второго — значение.

В хеш-таблицу можно помещать любые ссылочные типы, в том числе список

`ArrayList<Double>%`:

```
HashMap<String, ArrayList<Double>> menu = new HashMap<>(); //Создаём хеш-таблицу
ArrayList<Double> mohitoPrice = new ArrayList<>(); //Создаём список с ценами для коктейля
```

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.

Ещё методы:

`get(Object key)` — получить элементы из хеш-таблицы,

`remove(Object key)` — удалить элемент по ключу,

`clear()` — очистить хеш-таблицу полностью,

`containsKey(Object key)` — проверить наличие ключа,

`containsValue(Object value)` — проверяет наличие значения.

Если попробовать извлечь элемент по ключу, которого нет в хеш-таблице, Java вернёт `null` — нужного значения нет.

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.

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

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

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