

ПРИМИТИВНЫЕ ТИПЫ ДАННЫХ, УСЛОВНЫЕ ОПЕРАТОРЫ, ВЫХОД ЗА ГРАНИЦЫ ТИПОВ И ПОГРЕШНОСТЬ ВЫЧИСЛЕНИЙ



ОКСАНА МЕЛЬНИКОВА



ОКСАНА МЕЛЬНИКОВА

Software testing engineer





ПЛАН ЗАНЯТИЯ

1. [Примитивные типы](#)
2. [Литералы](#)
3. [Узкие места примитивных типов данных](#)
4. [Приведение типов](#)
5. [Условия и операторы сравнения](#)
6. [Итоги](#)



ПРИМИТИВНЫЕ ТИПЫ



ПРИМИТИВНЫЕ ТИПЫ

На прошлой лекции мы с вами поговорили о примитивных типах данных.

Что будем делать сегодня:

- будем практиковаться в их использовании;
- посмотрим на некоторые узкие места, которые вы должны знать как тестировщики.



ПРИМИТИВНЫЕ ТИПЫ

Вспомним, что к примитивным типам относятся:

- `boolean`;
- `byte`, `char`, `short`, `int`, `long`;
- `float`, `double`.



ЗАДАЧА

Наша задача: протестировать часть бонусной системы одного из онлайн-магазинов.

Работает она так:

- для **зарегистрированного** пользователя за покупку начисляется бонусный балл в размере **3% от суммы покупки**;
- для **незарегистрированного** пользователя за покупку начисляется бонусный балл в размере **1% от суммы покупки**;
- бонусный балл **не может превышать 500 за одну покупку**;
- бонусный балл измеряется в **целых числах** и **округляется в меньшую сторону**.



ЗАДАЧА

На примере этой задачи в течение всей лекции мы рассмотрим:

- примитивные типы;
- литералы;
- узкие места примитивных типов данных;
- приведение типов;
- условия и операторы сравнения.



ЗАДАЧА

Поскольку это всего лишь часть требований и планируется «усложнение» системы, мы решили сразу написать небольшую программу на Java, которая поможет нам и нашим коллегам **генерировать правильные ответы на тестовые данные**.



ОЦЕНКИ

Важно: вы всегда должны оценивать стоимость «автоматизации» и её целесообразность.

Как минимум, можно сосчитать количество запланированных/затраченных часов на автоматизацию * стоимость часа автоматизатора, и сравнить с тем, сколько времени будет сэкономлено.

И тут очень важен момент практики: если вы никогда не оценивали и никогда не автоматизировали, то без практики вы не научитесь этого делать.



ПЕРЕМЕННЫЕ

Создадим необходимые переменные для хранения входных данных и выберем для них типы.

Вопрос к аудитории: какие типы данных вы предлагаете выбрать для хранения:

- зарегистрирован пользователь или нет;
- стоимости покупки;
- % бонусного балла;
- итогового бонусного балла.

ПЕРЕМЕННЫЕ

Мы выбрали следующие `float` для хранения суммы и %, и `boolean` для информации о том, зарегистрирован пользователь или нет:

```
1  ► public class Main {  
2  ►   public static void main(String[] args) {  
3      boolean registered = true;  
4      float amount = 1000.60;  
5      float percent = 0.03;  
6  }  
7  }
```

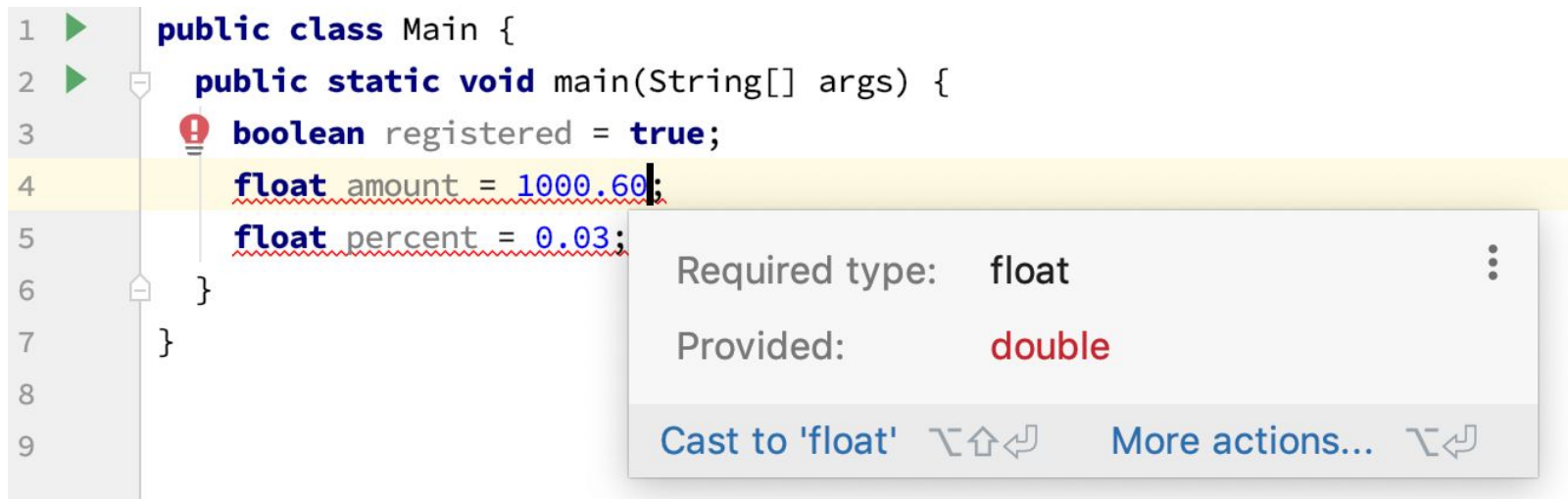
Кроме того, мы решили для начала автоматизировать кейс для зарегистрированного пользователя.

ПЕРЕМЕННЫЕ

IDEA сообщает нам о проблеме: **мы пытаемся положить в тип переменной float — тип double** (строки с ошибкой подчёркнуты красным). А double занимает больше места, чем float.*

Условно говоря, в float размером 4 байта мы пытаемся вложить double, который занимает 8 байт.

```
1  public class Main {  
2  public static void main(String[] args) {  
3      boolean registered = true;  
4      float amount = 1000.60;  
5      float percent = 0.03;  
6  }  
7  }  
8  
9
```



Примечание*: это упрощённое описание.

ПЕРЕМЕННЫЕ

Но почему IDEA (а заодно и Java) думает, что 1000.50 это double?

Ответ кроется в том, как Java обрабатывает значения (1000.50 и 0.03), записанные в коде.

```
1  public class Main {  
2      public static void main(String[] args) {  
3          boolean registered = true;  
4          float amount = 1000.60;  
5          float percent = 0.03;  
6      }  
7  }  
8  
9
```

Required type: float

Provided: double

Cast to 'float' More actions...



ЛИТЕРАЛЫ

ЛИТЕРАЛЫ

Литерал — это непосредственное значение, записанное в коде.

Например, в выражении: `float amount = 1000.50, 1000.50` — это литерал.

В Java к численным литералам применяются два ключевых правила:

- если литерал представляет из себя целое число, то он имеет тип `int`
- если литерал представляет из себя вещественное число (число с точкой), то он имеет тип `double`

Получается, что `1000.50` — это `double`. Именно об этом и сообщает IDEA.

ЛИТЕРАЛЫ

Q: хорошо, но если я всё-таки хочу `float`, `long` или другой тип данных?

A: для этого существует несколько возможностей:

1. Для `float` нужно указывать после литерала суффикс `f` или `F`.
2. Для `long` нужно указывать после литерала суффикс `L` или `l`.
3. Для `byte` и `short` ничего указывать не нужно, Java сама разберётся, если литерал уместится в границы типа.

ЛИТЕРАЛЫ

```
public class Main {  
    public static void main(String[] args) {  
        boolean registered = true;  
        float amount = 1000.60F;  
        float percent = 0.03F;  
  
        // умножаем  
        float bonus = amount * percent;  
        System.out.println(bonus);  
    }  
}
```

Выведенный результат: 30.017998

Q: Но это же неправильно! Должно быть: 30.018.



УЗКИЕ МЕСТА ПРИМИТИВНЫХ ТИПОВ ДАННЫХ



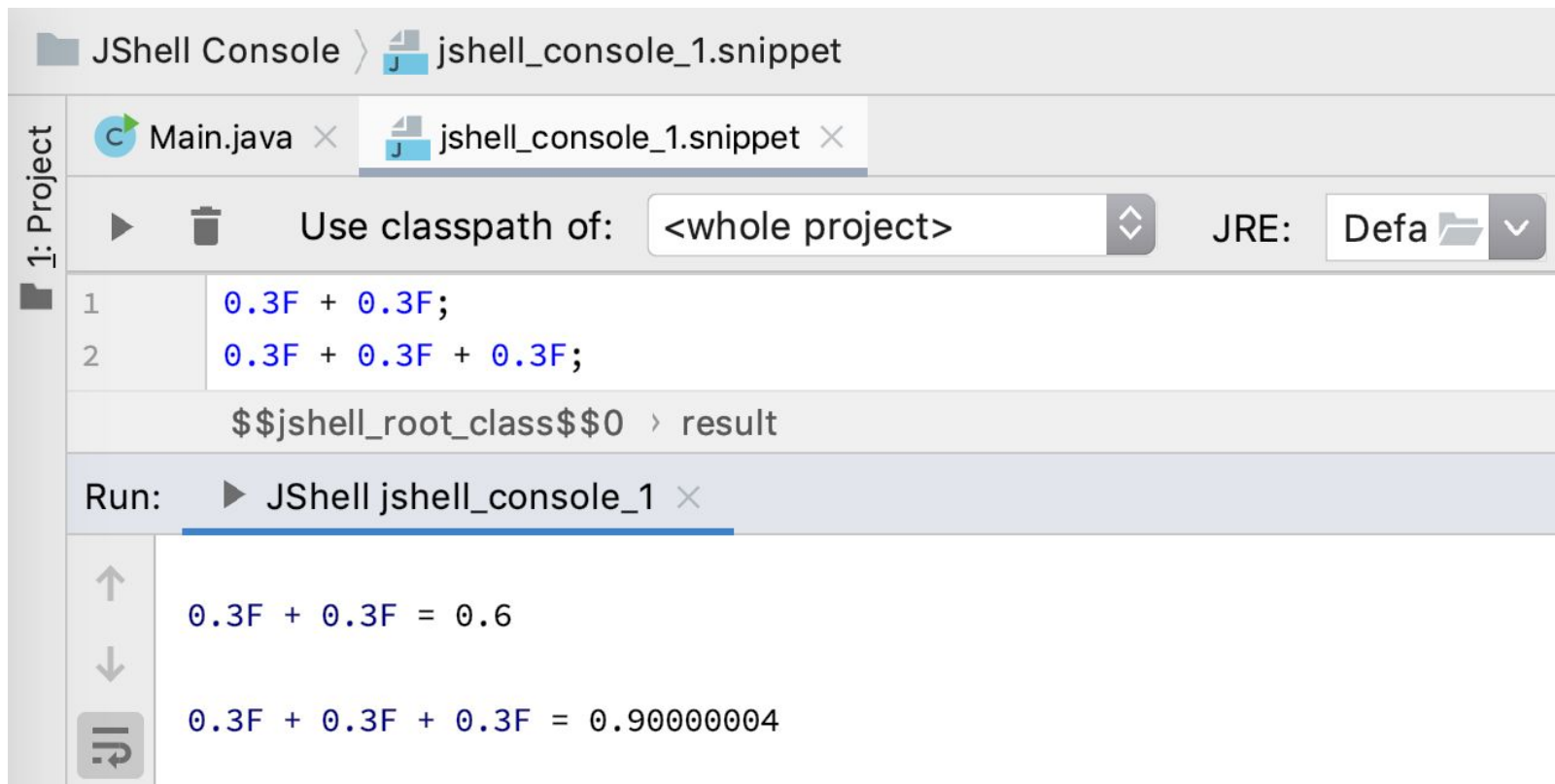
УЗКИЕ МЕСТА ПРИМИТИВНЫХ ТИПОВ ДАННЫХ

Это связано с особенностями хранения вещественных чисел в памяти компьютера: не все числа можно «аккуратно» и «точно» представить в памяти компьютера.

У целых чисел тоже есть свои особенности, которые мы тоже рассмотрим.

JSHELL CONSOLE

Для экспериментов воспользуемся JShell Console — инструментом, позволяющим выполнять кусочки кода без необходимости создавать полноценный проект:



ВЕЩЕСТВЕННЫЕ ЧИСЛА

Как вы видите, «точность» теряется.

Q: что же делать? Округлять?

A: если нужны точные вычисления, то нужно отказаться от вещественных чисел. Например, деньги в них не считают, и то, что мы выбрали `float` для хранения суммы и процента очень плохо.

Q: а в чём тогда считать?

A: либо в целых числах (например, с точностью до копеек), либо использовать специальные типы вроде `BigDecimal` (об этом позже).

ВЕЩЕСТВЕННЫЕ ЧИСЛА

Есть у вещественных чисел ещё одна особенность:

```
4 float before = 999_999_999_999F;  
5 float after = before - 1000F;  
6 before == after;
```

Run: JShell jshell_console_1 ×

field float before = 1.0E12

field float after = 1.0E12

before == after = true

1.0E12 — специальная запись (scientific notation), которая означает $1.0 * 10^{12}$

А оператор `==` проверяет на равенство два числа: `true` — равны, `false` — не равны.



ВЕЩЕСТВЕННЫЕ ЧИСЛА

Q: почему так?

A: потому что при хранении что очень больших, что очень маленьких чисел — есть определённая степень точности.

Q: а что с `double`?

A: с `double` будет всё то же самое, только на гораздо более маленьких или больших числах.

ЦЕЛЫЕ ЧИСЛА

Q: хорошо, но если мы будем считать в целых числах, то всё будет хорошо?

A: почти. Но если выйдем за границы типа, то уйдём в «минус» (и наоборот):

```
12 2_000_000_000 + 1_000_000_000;  
13 -2_000_000_000 - 1_000_000_000;
```

Run: ▶ JShell jshell_console_1 ✕

2_000_000_000 + 1_000_000_000 = -1294967296

-2_000_000_000 - 1_000_000_000 = 1294967296

ЦЕЛЫЕ ЧИСЛА

Это опять-таки связано с тем, что на хранение данных `int` всего 4 байта.

Один из бит отвечает за знак (0 — число положительное, 1 — отрицательное).

При этом при сложении больших чисел этот 0 или 1 просто «перетираются» (как при сложении столбиком) и знак числа меняется.

Поэтому очень важно знать границы типов и эти особенности.

ОБНОВЛЁННАЯ ВЕРСИЯ

```
public class Main {  
    public static void main(String[] args) {  
        boolean registered = true;  
        long amount = 100060;  
        int percent = 3;  
  
        long bonus = amount / 100 * percent / 100;  
        System.out.println(bonus);  
    }  
}
```

Выведенный результат: 30

Q: Но почему?



ПРИВЕДЕНИЕ ТИПОВ



ПРИВЕДЕНИЕ ТИПОВ

В Java строго соблюдается типизация. Поэтому если в выражении с арифметическими операторами участвуют только целые числа, то и результат будет целое число.

Поэтому мы и получаем целое число (т.к. в представлении целых чисел дробной части просто не существует).

Q: т.е. дробная часть округляется?

A: нет, ничего не округляется, дробной части просто нет.

ПРИВЕДЕНИЕ ТИПОВ

Это порождает ряд интересных эффектов, например:

- `long bonus = amount / 100 * percent / 100;` будет 30
- `long bonus = percent / 100 * amount / 100;` будет 0

Во втором случае 0 будет потому, что `3/100` (целочисленно) будет 0.

Поэтому правильнее было бы записать вот так:

```
long bonus = amount * percent / 100 / 100;
```

ПРИВЕДЕНИЕ ТИПОВ

Другой вопрос, если в выражении встречаются разные типы данных, то используются чёткие правила для выведения типа всего выражения (т.к. результатом выражения может быть только одно значение):

1. Если один из операндов `double`, то всё выражение `double`.
2. Если один из операндов `float`, то всё выражение `float`.
3. Если один из операндов `long`, то всё выражение `long`.
4. Всё остальное — `int`.

Таким образом, например, `long + float` будет `float`.

ПРИВЕДЕНИЕ ТИПОВ

Q: стоп-стоп, но как ведь `long` — 8 байт, а `float` — всего 4. Как это возможно?

A: дело в том, что `float` хранит степень, из-за этого может уместить всё, что хранит `long`, естественно, при этом «точность» может пострадать (забудет малые числа).

Отсюда очень простой вывод: желательно без лишней необходимости не смешивать целые и вещественные числа.

ПРИВЕДЕНИЕ ТИПОВ

Q: хорошо, а что если нужно сделать из float int?

A: для этого есть специальный оператор cast:

```
int result = (int)99.99F;
```

При этом произойдёт принудительное преобразование из float в int.

Естественно, вместо int можно подставить любой тип.

ПРИВЕДЕНИЕ ТИПОВ

Вопрос к аудитории: как вы думаете, какой результат будет у этого выражения?

```
short result = (short) 35_000;
```

ПРИВЕДЕНИЕ ТИПОВ

Вопрос к аудитории: как вы думаете, какой результат будет у этого выражения?

```
short result = (short) 35_000;
```

Ответ: результат будет отрицательный, т.к. 35_000 выходит за границы short.

Точный результат: -30_536 (но его запоминать не нужно)



ПРИВЕДЕНИЕ ТИПОВ

Q: такие проблемы с типами только в Java?

A: нет, в большинстве языков программирования.



ЗАДАЧА

Хорошо, мы научились считать бонусы для зарегистрированного пользователя.

Осталось решить, как работать с незарегистрированным и добавить лимит на бонусы.



УСЛОВИЯ И ОПЕРАТОРЫ СРАВНЕНИЯ

УСЛОВИЯ

В зависимости от того, зарегистрирован пользователь или нет, логика нашей программы должна изменяться.

В Java для этого есть конструкция `if`. Существует она в двух формах:

```
if (expression) {  
    // код внутри фигурных скобок  
    // выполняется только если expression == true  
}
```

```
if (expression) {  
    // код внутри фигурных скобок  
    // выполняется только если expression == true  
} else {  
    // код внутри фигурных скобок  
    // выполняется только если expression == false  
}
```

УСЛОВИЯ

Попробуем переделать нашу программу (пока без лимита):

```
1  ▶ public class Main {  
2  ▶  public static void main(String[] args) {  
3      boolean registered = true;  
4      if (registered) {  
5          int percent = 3;  
6      } else {  
7          int percent = 1;  
8      }  
9      ! long amount = 100060;  
10     long bonus = amount * percent / 100 / 100;  
11     System.out.println(bonus);  
12 }  
13 }  
14
```

Cannot resolve symbol 'percent'

Create local variable 'percent'

More actions...

ОБЛАСТЬ ВИДИМОСТИ

{ } определяют блок кода, ограничивающий область видимости переменных.

Область видимости переменной — это область, в которой переменная доступна по имени.

Это значит, что переменная не доступна за пределами области видимости, в которой она объявлена.

На это и ругается IDEA — переменная `percent` не может быть найдена, так как находится вне области видимости..

ОБЛАСТЬ ВИДИМОСТИ

А вот так проблем уже нет, т.к. сама переменная объявлена в «объемлющем» блоке (мы добавили `int percent` в 4 строке):

```
1  ► public class Main {  
2  ►   public static void main(String[] args) {  
3      boolean registered = true;  
4  →   int percent;  
5      if (registered) {  
6          percent = 3;  
7      } else {  
8          percent = 1;  
9      }  
10     long amount = 100060;  
11     long bonus = amount * percent / 100 / 100;  
12     System.out.println(bonus);  
13   }  
14 }
```

Таким образом, переменные становятся видны с той строки, где они объявлены и в том блоке, в котором объявлены (включая вложенные блоки).

ОБЛАСТЬ ВИДИМОСТИ

Но не наоборот: переменные объявленные во вложенных блоках не видны
ВО ВНЕШНИХ:

```
1  public class Main {  
2  public static void main(String[] args) {  
3      boolean registered = true;  
4      if (registered) {  
5          int percent = 3;  
6      } else {  
7          int percent = 1;  
8      }  
9      long amount = 100060;  
10     long bonus = amount * percent / 100 / 100;  
11     System.out.println(bonus);  
12 }  
13 }  
14 }
```

Cannot resolve symbol 'percent'

Create local variable 'percent'

More actions...



ОБЪЯВЛЕНИЕ ПЕРЕМЕННОЙ

Q: а почему объявление `amount` перенесли вниз? Удобнее же было всё хранить наверху.

A: это относится к стилю кодирования (написанию кода). Обычно стараются объявлять переменные как можно ближе к месту использования, чтобы не терять контекст.

Об этом мы поговорим отдельно (о стилях кодирования), когда будем рассматривать инструменты статического анализа кода (Checkstyle).



ОБЪЯВЛЕНИЕ ПЕРЕМЕННОЙ

Q: почему IDEA некоторые выражения подсвечивает жёлтым?

A: так она предлагает «упростить» или «улучшить» некоторый код.

Но с этим нужно быть аккуратным, т.к. в нашем случае, если мы это сделаем, она просто заменит `if` на `int percent = 3;`, т.к. будет считать, что `registered` всегда `true`.

ОПЕРАТОРЫ СРАВНЕНИЯ

Теперь, когда мы знаем как использовать конструкцию `if`, осталось только добавить лимит. В этом нам помогут операторы сравнения:

- `==` — проверка на равенство (не используйте с вещественными числами);
- `!=` — проверка на неравенство;
- `<` — меньше;
- `<=` — меньше или равно;
- `>` — больше;
- `>=` — больше или равно.

Важно: операторы сравнения всегда возвращают `boolean`.

Вам, как тестировщикам стоит помнить, что именно эти операторы определяют границы, соответственно, создают ошибки граничных значений.

ЛИМИТ

```
1  ► public class Main {
2  ►   public static void main(String[] args) {
3      boolean registered = true;
4      int percent;
5      if (registered) {
6          percent = 3;
7      } else {
8          percent = 1;
9      }
10     long amount = 100060;
11     long bonus = amount * percent / 100 / 100;
12     long limit = 500;
13     if (bonus > limit) {
14         bonus = limit;
15     }
16     System.out.println(bonus);
17 }
18 }
```

Обратите внимание на 14-ую строку: если условие срабатывает, то мы просто заменяем значение, хранящееся в `bonus` на значение, хранящееся в `limit`.

ТЕРНАРНЫЙ ОПЕРАТОР

Иногда для сокращения кода используют тернарный оператор.
Выглядит он следующим образом:

```
expression ? if-true-value : if-false-value
```


ТЕРНАРНЫЙ ОПЕРАТОР

В нашем случае мы можем использовать его следующим образом:

```
1  ► public class Main {
2  ►   public static void main(String[] args) {
3      boolean registered = true;
4  →   int percent = registered ? 3 : 1;
5      long amount = 100060;
6      long bonus = amount * percent / 100 / 100;
7      long limit = 500;
8      if (bonus > limit) {
9          bonus = limit;
10     }
11     System.out.println(bonus);
12 }
13 }
```

Ключевое: тернарный оператор должен улучшать читабельность кода, а не ухудшать её.



ПОСЛЕДНИЙ ШТРИХ

В целом, приложение уже достаточно хорошо выглядит, но, возможно, для большей удобочитаемости стоит переименовать `amount` в `amountInKopecks`.



ИТОГИ



ИТОГИ

Мы сегодня обсудили достаточно много вопросов:

1. Примитивные типы и особенности их работы в Java.
2. Правила приведения типов.
3. Условную конструкцию `if` и тернарный оператор.



ИТОГИ

Ключевое: всё, что мы делали, пока выглядит довольно неудобно: приходится каждый раз запускать вручную и смотреть результат в консоли.

Пока мы делаем какие-то утилиты (вспомогательные программы) для себя — это нормально.

Но как только мы начнём сами писать приложения — так уже не пойдёт.

Поэтому на следующей лекции мы научимся писать автотесты на наш код.



ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задаем в чате Slack!
- Задачи можно сдавать по частям.
- Зачет по домашней работе проставляется после того, как приняты **все задачи**.



Задавайте вопросы и напишите отзыв о лекции!

ОКСАНА МЕЛЬНИКОВА

 Оксана Мельникова