



План занятия

1. Что такое методы?
2. Объявляем метод
3. Вызываем методы
4. Возвращаем значение из метода
5. Особенности оператора return
6. Параметры, аргументы и сигнатура метода
7. Метод main
8. Декомпозиция кода на методы
9. Декомпозируем финансовое приложение



Что такое методы?



Что такое методы?

В реальной жизни мы часто объединяем несколько действий в один процесс. Например, процесс «приготовить завтрак» состоит из того, чтобы пожарить омлет, сварить кофе, нарезать хлеб, а процесс «сходить в магазин» включает составление списка покупок, выбор продуктов, их оплату на кассе и множество других шагов. Представьте, если бы вместо короткой фразы «иду в магазин» пришлось бы снова и снова проговаривать всю цепочку предполагаемых действий. Это было бы мучительно.



Что такое методы?

Так и в коде любую последовательность действий можно объединить в один блок — метод. **Метод** — это набор команд, у которого есть понятное имя. Методы нужны, чтобы структурировать код и сделать его проще для понимания. Благодаря им можно не дублировать команды снова и снова, а вызвать их выполнение одной строкой. Это можно сделать в любой момент и сколько угодно раз.

Вы уже сталкивались со стандартными методами Java — например, чтобы напечатать строку, вы пользовались методом `println()`, а число из консоли считывали методом `nextInt()`. Кроме того, весь код на Java пишется внутри метода `main()` — главного метода программы, отвечающего за её запуск.



Зачем нужны методы?

- Методы нужны, чтобы избежать дублирования кода.
- Методы нужны, чтобы объединить в один блок несколько команд.
- Методы нужны, чтобы упростить чтение и понимание кода программы.



Что такое методы?

Узнать метод в программе можно по имени и круглым скобкам после него, например `println()`. Имя может быть любым, но лучше, чтобы оно соответствовало правилам, которые отражены в *Code Conventions* (англ. «соглашения по написанию кода»).



Что такое методы?

Имя метода рекомендуется начинать с глагола, потому что методы описывают набор действий. При его написании нужно использовать уже знакомый вам «верблюжий» стиль *lowerCamelCase* — первое слово со строчной, остальные с заглавной буквы без пробелов. В имени метода, как правило, отражено, какую задачу он решает. Например, логично, что метод `printMenu()` выводит меню, а метод `sayHello()` приветствует пользователя.



Что из перечисленного является методом?

- 1) ==
- 2) println()
- 3) getNumber
- 4) if()
- 5) double[]
- 6) showSchedule()
- 7) for()



Ответ

`println()`

Это метод для работы с потоком вывода, он содержит в себе команды «печать» и «переход на новую строку».

`showSchedule()`

Это метод — его можно узнать по понятному имени и круглым скобкам.



Что такое методы?

Выбирайте имена для методов так, чтобы они были лаконичными, но при этом оставались понятными. Не используйте слишком общие имена — `method()` или `makeAction()` — и не перестарайтесь в детализации. Метод, который здороваётся с пользователем, можно ведь назвать и так: `pleaseSayHelloToUserMyDearProgram()`. Смысл тот же, что и у метода `sayHello()`, но имя получилось слишком длинное — его сложно запомнить, а при использовании легко напутать.

Прежде чем перейти к объявлению и вызову методов, потренируемся их называть.



Выберите подходящее имя для метода, который находит самую большую трату в массиве недельных расходов.

- A. `maxExpence()`
- B. `checkAllExpensesAndFindTheBiggestOne()`
- C. `getmaxexpense()`
- D. `method2()`
- E. `expenses()`
- F. `getMaxExpense()`
- G. `getThingsDone()`

Ответ

- A. `maxExpense()` - Это удачное имя для переменной, в которой хранится размер самой большой траты. В имени метода лучше использовать глагол
- B. `checkAllExpensesAndFindTheBiggestOne()` - слишком длинный
- C. `getmaxexpense()` - не использован camelCase
- D. `method2()` - другой программист никогда не догадается, что делает такой метод
- E. `expenses()` - удачное имя для массива трат, но не для метода
- F. `getMaxExpense()` - Дословно такое имя можно перевести как «получить максимальную трату». Просто и понятно.
- G. `getThingsDone()` - другому программисту будет сложно понять, какую именно задачу он решает.



Объявляем метод

Объявляем метод

Создание метода начинается с его объявления. Научимся это делать. Вот так будет выглядеть объявление метода, который должен печатать приветствие:

```
// Объявили метод  
public static void sayHello() {  
    // Тут будет тело метода  
}
```

Объявляем метод

Сначала идут служебные слова **public** и **static**. Позже мы расскажем, что они означают. Пока вам нужно только запомнить, что их требуется указать при объявлении метода.

Далее следует ещё одно служебное слово **void** (англ. «пустой»). Оно означает, что метод выполняет действие, но не возвращает результат. Сравните, одно дело, если друг сказал вам «Привет!», и совсем другое, если он сказал «Держи яблоко!» и дал фрукт. В первом случае важно само действие — приветствие. Во втором случае результат — яблоко. То есть, можно сказать, что метод «Держи яблоко!» возвращает результат — конкретный фрукт, а метод «Привет!» — ничего не возвращает. В этом уроке мы работаем с методами, которые не возвращают значений.




sayHello — имя метода, по которому к нему можно будет обращаться. Имя придумывает разработчик. Круглые скобки после имени означают, что объявлен именно метод, а не переменная или массив.

Порядок объявления метода всегда один и тот же: сначала служебные слова **public** и **static**, затем **void** и только потом — имя метода с круглыми скобками. Если его нарушить — произойдёт ошибка.



```
public static void sayHello() {  
    // тело метода  
}
```



После имени, в фигурных скобках, идёт тело метода — набор команд, которые он будет выполнять. Реализовать метод — это значит перечислить в теле метода все необходимые команды. Можно оставить тело метода пустым, и тогда метод ничего не будет делать — однако в этом случае нет смысла его объявлять.

```
public static void sayHello() { // Объявили метод sayHello
    System.out.println("Привет!"); // Реализовали метод
}
```

Теперь метод `sayHello()` объявлен и реализован — он содержит одну команду, печать слова «Привет!».

Вопрос

Где при объявлении метода была допущена ошибка?

```
public void static sayGoodBye() {  
    System.out.println("Пока!");  
}
```

- A. В имени метода
- B. В теле метода
- C. В служебных словах



Ответ

С. В служебных словах

Порядок служебных слов неправильный: сначала должны быть **public** и **static**, а только потом **void**



Вопрос

Тело метода - это:

- A. Всё внутри фигурных скобок
- B. Всё внутри круглых скобок
- C. `System.out.println("Пока!");`
- D. `public void static sayGoodBye()`



Ответ

А. Всё внутри фигурных скобок

Все команды внутри фигурных скобок - это тело метода.



Вопрос

Что означает служебное слово **void**?

- A. Что метод `sayGoodBye()` пустой - он ничего не делает
- B. Что метод `sayGoodBye()` не возвращает никакого значения
- C. Что метод `sayGoodBye()` возвращает результат - команду "Пока!"




Ответ

В. Что метод `sayGoodBye()` не возвращает никакого значения

Служебное слово `void` показывает, что цель метода - только действие, без результата.

Тело метода может содержать любые команды: операции с переменными и массивами, условные выражения и циклы. К примеру, метод `printStudentsList()` сначала печатает число учеников в классе, а потом все их имена и фамилии.

```
public static void printStudentsList() {  
    String[] students = {"Иванов Алексей",  
        "Жумагулов Мирас",  
        "Смирнова Виктория",  
        "Абетаев Дамир",  
        "Асанова Гульжан",  
        "Токтаров Нуржан"};  
    int studentsCount = students.length;  
    System.out.println("Всего в классе " + studentsCount + " учеников:");  
  
    for (int i = 0; i < studentsCount; i++) {  
        System.out.println(students[i]);  
    }  
}
```



В одной программе может быть объявлено и реализовано любое количество методов. Важно запомнить, что внутри одного метода нельзя объявить и реализовать другой. Методы можно объявлять только последовательно, их блоки существуют в коде отдельно друг от друга:

```
public class Practice {  
    public static void main(String[] args) { // Блок метода #1  
    } // Блок #1 закончен, можно объявить новый метод  
  
    public static void sayHello() { // Блок метода #2  
        System.out.println("Привет!");  
    } // Блок #2 закончен, можно объявить новый метод  
  
    public static void sayGoodBye() { // Блок метода #3  
        System.out.println("Пока!");  
    } // Блок #3 закончен, можно объявить ещё один метод  
}
```



Задача

Вам нужно запрограммировать робота, который должен познакомиться с пользователем, поздороваться с ним в зависимости от времени суток и поздравить с успехами в программировании. Для этого нужно объявить и реализовать такие методы:

1. Метод `welcomeUserByName()` должен спрашивать у пользователя имя, а потом сообщать, что рад знакомству.

Задача

```
public class Practice {  
    public static void main(String[] args) {  
        System.out.println("Робот-помощник v1.0.");  
    }  
  
    ...{ // Объявите метод welcomeUserByName  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Как вас зовут?");  
        ... // Сохраните введённое пользователем имя в переменную name  
        System.out.println("Рад познакомиться, " + name + "!");  
    }  
}
```

Задача

2. Метод `sayHelloByTime()` должен спрашивать у пользователя время и в зависимости от ответа печатать приветствие:

- начиная с 22 часов вечера и до 6 часов утра не включительно — "Доброй ночи!";
- начиная с 6 до 12 не включительно — "Доброе утро!";
- начиная с 12 до 18 не включительно — "Добрый день!";
- начиная с 18 до 22 не включительно — "Добрый вечер!".

Объявите метод

Спросите у пользователя "Который час?" и сохраните ответ в переменную `currentHour`

В зависимости от времени предусмотрите печать приветствий



Задача

3. Третий метод `printSuccess()` должен печатать только одну строку — "У вас уже неплохо получается программировать!".



Решение

<https://gist.github.com/practicetasks/b817a964d5923dcad22fac5f1cf9d30c>



Вызываем методы

Вызываем методы

Вы уже знаете, как объявить и реализовать метод, но этого недостаточно, чтобы заложенные в нём команды начали выполняться.

```
public class Practice {  
    public static void main(String[] args) {  
    }  
  
    public static void sayHello() {  
        System.out.println("Привет!");  
    }  
}
```

Вызываем методы


Команды внутри метода начнут исполняться только после того, как метод будет вызван. Вызвать метод можно в любом месте кода, для этого нужно написать его имя и круглые скобки — `sayHello()`.

```
public class Practice {  
    public static void main(String[] args) {  
        sayHello(); // Вызвали метод sayHello  
    }  
  
    public static void sayHello() {  
        System.out.println("Привет!");  
    }  
}
```



Результат

Привет!



Сейчас метод `sayHello()` вызван из главного метода `main`, и при запуске кода он выполнится — приветствие будет напечатано. Можете убедиться в этом сами. Методы можно вызывать не только из `main`, но и друг из друга. Однако их выполнение начнётся только при вызове из `main`. Добавим в программу метод `printHeader()` и вызовем его из `sayHello()`.

```
public class Practice {  
    public static void main(String[] args) {  
        sayHello(); // Вызываем метод sayHello из main, чтобы при запуске он выполнился  
    }  
  
    public static void sayHello() {  
        printHeader(); // Вызываем метод printHeader  
        System.out.println("Привет!");  
    }  
  
    public static void printHeader() {  
        System.out.println("Сейчас будет приветствие:");  
    }  
}
```



Результат

Сейчас будет приветствие:
Привет!

Обратите внимание — строки будут напечатаны в порядке **вызова** методов, а не в порядке их объявления в программе.



Вызываем методы

Логика следующая: сначала запускается главный метод **main**. Далее из него вызывается метод **sayHello()**. Метод **sayHello()** выполняет первую команду — вызывает метод **printHeader()**. Метод **printHeader()** печатает строку «Сейчас будет приветствие:». После этого метод **sayHello()** выполняет вторую команду — печатает строку «Привет!».

Несмотря на то, что в примере метод **printHeader()** объявлен последним, он выполнит свою команду именно тогда, когда был вызван.

Задача

Объявите, реализуйте и вызовите метод `printCity()`. Он должен печатать, из какого вы города.

```
public class Practice {  
    public static void main(String[] args) {  
        ...  
    }  
  
    ... // Объявите метод printCity  
    ... // Метод должен выводить строку вида: Я из Астаны  
    ...  
}
```

Решение

```
public class Practice {  
    public static void main(String[] args) {  
        printCity();  
    }  
  
    public static void printCity(){  
        System.out.println("Я из Астаны");  
    }  
}
```


Задача 2

https://github.com/practicetasks/java_tasks/tree/main/methods/task_1

Ваш робот уже умеет знакомиться, здороваться в зависимости от времени суток и хвалить за успехи в программировании. Научите его ещё начинать общение с короткого приветствия и спрашивать у пользователя, из какого он города. Сделайте это с помощью методов `sayHello()` и `printCity()`. Результат должен получиться таким:

```
Привет!  
Который час?  
> ввод текущего часа  
Добрый день! (один из вариантов)  
Как вас зовут?  
> ввод имени  
Из какого вы города?  
> ввод города  
Рад познакомиться, <ваше имя> из <вашего города>!  
У вас уже неплохо получается программировать!
```



Решение

<https://gist.github.com/practicetasks/f2e6c092af4636832676e8833a8954e1>




Возвращаем значение из метода



Возвращаем значение из метода


Методы, с которыми вы работали до этого момента, только выполняли указанные в них команды. При их объявлении использовалось служебное слово **void** — это означало, что метод ничего не возвращает. Есть другая категория методов — они могут возвращать результат, какое-то значение. Это значение можно записать в переменную и в дальнейшем использовать в программе. Научимся это делать.




Вернёмся к примеру с приветствием и просьбой дать яблоко. В виде методов эти действия можно записать так:

```
public static void sayHello() {  
    System.out.println("Привет!");  
}  
  
public static String giveMeAnApple() {  
    return "Яблоко"; // Метод возвращает результат – строку "Яблоко"  
}
```

Метод `sayHello()` печатает «Привет!» и ничего не возвращает. Метод `giveMeAnApple()` ничего не печатает, но возвращает строку «Яблоко». Это происходит с помощью оператора `return` (англ. «возврат, вернуть»). Значение, которое указывается после `return`, называется **возвращаемым значением метода**.



То, что метод возвращает значение, нужно дополнительно отразить при его объявлении. Вместо **void** в методе **giveMeAnApple()** указан строковый тип данных — **String**. Это сигнализирует о том, что метод возвращает значения этого типа. Служебное слово **void** также является типом возвращаемого значения — оно обозначает его отсутствие. Вы можете объявить метод с любым типом возвращаемого значения. Главное, чтобы после оператора **return** в коде стояло значение такого же типа, что и в объявлении метода.




Возвращаемое значение метода можно сохранить в переменную. Тип этой переменной должен совпадать с типом возвращаемого значения. Значение возвращается через вызов метода. Например, метод `giveMeAnApple()` в результате своего выполнения возвращает строку «Яблоко», запишем её в переменную `apple` и напечатаем. В коде это будет выглядеть так:

```
public class Practice {  
    public static void main(String[] args) {  
        String apple = giveMeAnApple(); // Сохраняем возвращаемое значение метода в переменную  
        System.out.println(apple);  
    }  
  
    public static String giveMeAnApple() {  
        return "Яблоко"; // Метод возвращает результат – строку "Яблоко"  
    }  
}
```



Результат

Яблоко



С помощью методов можно возвращать также результаты вычислений. Например, напомним метод `sum()`, который будет возвращать сумму чисел 2 и 3. Тип такого результата будет `int`. Возвращаемое значение не обязательно записывать в переменную, можно сразу его напечатать, вызвав из метода `println`:

```
public class Practice {  
    public static void main(String[] args) {  
        System.out.println(sum());  
    }  
  
    public static int sum() {  
        return 2 + 3;  
    }  
}
```



Результат

5



Возвращаем значение из метода

Если тип значения после оператора **return** не будет совпадать с типом, указанным в объявлении метода — произойдёт ошибка. Убедитесь в этом сами: попробуйте поставить **String** перед методом **sum()**. Такая же ошибка произойдет, если при объявлении метода указать **void**, а потом в его теле попробовать вернуть значение.

Вопрос

Прочитайте код. Что будет напечатано в результате выполнения программы?

```
public class Practice {  
    public static void main(String[] args) {  
        double result = multiply();  
        System.out.println("Результат = " + result);  
    }  
  
    public static double multiply() {  
        return 3.0 * 6.0;  
    }  
}
```

- A. Результат = 3.0
- B. 18.0
- C. Результат = 6.0
- D. Результат = 18.0



Ответ

D. Результат = 18.0

Вопрос

А что будет напечатано после запуска такого кода?

```
public class Practice {  
    public static void main(String[] args) {  
        System.out.println(sayHello());  
    }  
  
    public static String sayHello() {  
        String name = "Пиксель";  
        return "Привет, " + name + "!";  
    }  
}
```

- A. Привет!
- B. Привет, name!
- C. Привет, Пиксель!
- D. Возникнет ошибка



Ответ

С. Привет, Пиксель!

Сначала произойдёт конкатенация имени персонажа с приветствием - и только потом возврат значения из метода.

Вопрос

Каким должен быть тип возвращаемого значения для метода `sayGoodBye()`.

```
public static ... sayGoodBye() {  
    return "Пока!";  
}  
}
```

- A. void
- B. String
- C. double
- D. int



Ответ

B. `String`

Метод возвращает строку, поэтому нужен `String`.

Вопрос

Ещё раз каким должен быть корректный тип возвращаемого значения.

```
public static ... sum() {  
    return 2.1 + 3.4;  
}
```

- A. void
- B. String
- C. double
- D. int



Ответ

C. double

Результатом сложения будет дробное число 5.5

Вопрос

Значение какого типа будет возвращать метод?

```
public static ... findMax() {  
    Scanner scanner = new Scanner(System.in);  
    int a = scanner.nextInt();  
    int b = scanner.nextInt();  
    int max;  
    if (a > b) {  
        max = a;  
    } else {  
        max = b;  
    }  
}
```



Варианты ответов

- A. void
- B. String
- C. double
- D. int



Ответ

A. `void`

Метод только вычисляет максимальное значение двух чисел, но не возвращает его.



Задача

Вам нужно дописать реализацию метода `findHighestGrossingFilm()`, который выбирает самый кассовый фильм из трёх предложенных вариантов — хитов Джеймса Кэмерона «Титаник» и «Аватар» и блокбастера Кристофера Нолана «Тёмный рыцарь». Названия фильмов и их сборы заранее сохранены в соответствующих переменных. Метод должен сравнить суммы сборов и вернуть значение — название фильма, заработавшего в прокате больше всего.

```
public class Practice {  
    public static void main(String[] args) {  
        // Ниже вызовите новый метод  
        String highestGrossingFilm = ...;  
        System.out.println("Самый кассовый фильм: " + highestGrossingFilm);  
    }  
  
    public static String findHighestGrossingFilm() {  
        String film1 = "Титаник";  
        int income1 = 2194;  
  
        String film2 = "Аватар";  
        int income2 = 2810;  
  
        String film3 = "Тёмный рыцарь";  
        int income3 = 1084;  
  
        // Допишите реализацию метода ниже  
        ...  
    }  
}
```


Решение

```
public static String findHighestGrossingFilm() {
    String film1 = "Титаник";
    int income1 = 2194;

    String film2 = "Аватар";
    int income2 = 2810;

    String film3 = "Тёмный рыцарь";
    int income3 = 1084;

    // Допишите реализацию метода ниже
    if (income1 > income2 && income1 > income3){
        return film1;
    } else if (income2 > income1 && income2 > income3){
        return film2;
    } else {
        return film3;
    }
}
```



Особенности оператора `return`

Особенности оператора `return`

С помощью оператора `return` можно не только возвращать значение из метода. Попробуем поставить `return` в конец метода типа `void`:

```
public static void sayHello() {  
    System.out.println("Привет!");  
}
```

```
public static void sayHello() {  
    System.out.println("Привет!");  
    return;  
}
```

// Эти два фрагмента кода делают одно и то же



Особенности оператора `return`

Методы идентичны и умеют делать одно и то же — печатать приветствие. Мы можем оставить в коде любой из них (два одинаковых метода в коде быть не может), и программа не сломается и не выдаст ошибку. Несмотря на то, что во второй метод добавлен оператор **`return`** — этот метод по-прежнему ничего не возвращает.



Особенности оператора `return`

Дело в том, что оператор `return` можно использовать внутри метода не только для возврата значения, но и для того, чтобы немедленно завершить его выполнение. Это может быть полезно в том случае, когда нужно завершить метод не на его последней строке, а где-то в середине, например внутри одного из блоков ветвления.

Рассмотрим такой пример. Внутри метода `example()` есть ветвление, которое зависит от значения переменной `command`. Блок `else` этого ветвления завершается оператором `return`:

```
public class Practice {
    public static void main(String[] args) {
        example();
    }

    public static void example() {
        int command = 1;
        if (command == 1) {
            System.out.println("Привет!");
        } else if (command == 0) {
            System.out.println("Выход");
            return; // Оператор return стоит в конце блока кода ветвления
                  // При command == 0 метод будет сразу завершён
        }
        // Эта строка будет выведена, если метод не завершился (command == 1)
        System.out.println("Напечатаем ещё одну строку");
    }
}
```



Результат

Привет!

Напечатаем ещё одну строку

Теперь меняем **command** на 0

```
public class Practice {  
    public static void main(String[] args) {  
        example();  
    }  
  
    public static void example() {  
        int command = 0;  
        if (command == 1) {  
            System.out.println("Привет!");  
        } else if (command == 0) {  
            System.out.println("Выход");  
            return; // Оператор return стоит в конце блока кода ветвления  
                  // При command == 0 метод будет сразу завершён  
        }  
        // Эта строка будет выведена, если метод не завершился (command == 1)  
        System.out.println("Напечатаем ещё одну строку");  
    }  
}
```




Результат

Выход




Особенности оператора `return`

Важно запомнить правило расположения **`return`**, когда он используется для прерывания работы метода. Нужно, чтобы оператор стоял либо в конце метода, либо в конце любого из **блоков кода** внутри метода — всегда перед закрывающей фигурной скобкой. Иначе код не запустится.

Особенности оператора return

Сравните два метода. В первом допущена ошибка — **return** стоит в середине блока кода. Во втором методе всё правильно:

```
public static void wrongExample() {  
    return; // return – не последняя команда, после него идёт вызов println  
    System.out.println("Такая программа работать не будет");  
}  
  
public static void correctExample() {  
    System.out.println("А такая – будет!");  
    return; // Здесь всё верно: return – последняя команда в блоке  
}
```



В методах, которые возвращают результат, можно несколько раз использовать оператор **return** и благодаря этому получать разные значения. Мы делали так, например, когда решали задачу с определением самого кассового фильма. Однако **return** всегда должен быть последним оператором в блоке кода:

```
public static void main(String[] args) {  
    System.out.println("Наибольшее из чисел = " + findMax());  
}  
  
public static int findMax() {  
    int a = 5;  
    int b = 3;  
    if (a > b) {  
        return a;  
    }  
    return b;  
}
```



Результат

Наибольшее из чисел = 5



Особенности оператора `return`


Важно не перепутать оператор `return` с оператором `break`. Если поставить `return` в блоке бесконечного цикла, он прервёт и выполнение цикла, и выполнение метода. Оператор `break`, в свою очередь, прерывает только выполнение цикла, и следующие после него команды в методе будут выполнены.

К примеру, сравните код двух методов, в каждом из которых есть бесконечный цикл.

В первом использован оператор **return** — одновременно с прерыванием цикла он завершает работу этого метода. Во втором оператор **break** — он останавливает цикл, но выполнение метода продолжается.

```
public static void returnExample() {
    boolean flag = true;
    while (flag) {
        System.out.println("Работаем в цикле!");
        return; // Метод завершён
    }
    System.out.println("Если вы видите эту строку, значит, return вы поменяли на break!");
}


public static void breakExample() {
    boolean flag = true;
    while (flag) {
        System.out.println("Работаем в цикле!");
        break; // Метод не завершён, завершено выполнение цикла
    }
    System.out.println("Если вы видите эту строку, значит, в этом методе оператор break!");
}
```



```
public class Practice {  
    public static void main(String[] args) {  
        returnExample();  
    }  
}
```

Результат:

Работаем в цикле!



```
public class Practice {  
    public static void main(String[] args) {  
        breakExample();  
    }  
}
```

Результат:

Работаем в цикле!

Если вы видите эту строку, значит, в этом методе оператор **break**!



Параметры, аргументы и сигнатура метода



Параметры, аргументы и сигнатура метода

При объявлении и вызове метода после его имени ставятся круглые скобки. До этого момента они всегда оставались пустыми. Однако скобки нужны методу не для красоты. Разберёмся, как их использовать.

Что такое параметры метода?

Вы уже не раз работали с методом `sayHello()` — он печатает приветствие. Дополним его так, чтобы приветствие стало адресным — поздороваемся, например, с котом Пикселем:

```
public class Practice {  
    public static void main(String[] args) {  
        sayHello();  
    }  
  
    public static void sayHello() {  
        System.out.println("Привет, Пиксель!");  
    }  
}
```



Результат

Привет, Пиксель!



Что такое параметры метода?

С Пикселем-то мы поздоровались, а с нашим старым другом, хомяком Байтом, нет. Можно, конечно, продублировать строку с приветствием и вписать в неё грызуна. Сделайте это — программа напечатает два приветствия подряд. Но что, если нам потребуется поздороваться с кем-то ещё или мы захотим поздороваться сначала с Байтом, а потом с Пикселем?

Что такое параметры метода?

Метод при этом не меняется — действие всегда одно и то же, однако требуется подставлять в него разные значения — имена тех, с кем нужно поздороваться. Было бы здорово получать их при вызове метода. Это можно сделать с помощью **параметров метода** — значений, которые может **принимать** метод. Они указываются в круглых скобках (да-да, для этого они и нужны) при его объявлении:

```
public static void sayHello(String username) { // У метода появился параметр
    System.out.println("Привет, " + username); // К параметру метода можно обратиться
}
```

Аргументы — конкретные значения параметров

Теперь метод `sayHello` принимает один параметр — имя того, с кем нужно поздороваться. У параметра есть имя — `username` и тип — `String`. Параметр очень похож на переменную: к нему тоже можно обратиться и получить его значение, но это разные понятия. При вызове метода можно присвоить параметру конкретное значение — это происходит также внутри круглых скобок:

```
public static void main(String[] args) {  
    sayHello("Пиксель"); // Передали в метод имя Пиксель  
    sayHello("хомяк Байт"); // Передали в метод имя и вид животного  
    sayHello("пользователь #1"); // Передали в метод слово пользователь и его номер  
    sayHello("здесь можно писать всё, что захочется!"); // Передали в метод предложение  
}  
  
// У метода объявляем параметр – строковую переменную username  
public static void sayHello(String username) {  
    System.out.println("Привет, " + username); // К параметру метода можно обратиться  
}
```




Аргументы — конкретные значения параметров

Конкретное значение, которое передаётся в метод при вызове, называется **аргументом метода**. То есть параметр — это указание на то, что метод принимает, а аргумент — это значение, которое в него передаётся снаружи. Это достаточно формальное разделение: на практике можно назвать параметры аргументами и наоборот, и вас поймут, но небольшая разница в этих понятиях всё же есть.




Аргументы — конкретные значения параметров

Благодаря тому, что у метода `sayHello` появился параметр, можно поздороваться и с Пикселем, и с Байтом, и со случайным пользователем и даже передать в метод целую фразу. Позапускайте код и передайте в качестве аргументов в приветствие любые другие значения.

Типы параметров и виды аргументов

Параметры метода не обязательно должны относиться к одному типу — они могут быть любых типов, включая массив. Если параметров несколько, то нужно перечислить их через запятую. К примеру, метод `printUserInfo()` печатает сразу имя, возраст и город пользователя — у него два параметра типа `String` и один типа `int`.

```
public static void main(String[] args) {  
    printUserInfo("Катя", 26, "Астана");  
    printUserInfo("Дамир", 40, "Алматы");  
}  
  
public static void printUserInfo(String username, int age, String city) {  
    System.out.println("Меня зовут " + username + ", я из города " + city + ", мне " + age + " лет.");  
}
```




А параметр метода `printNumbers()` — числовой массив `int[] numbers`. С помощью цикла `for` метод печатает его значения. Прежде чем вызвать такой метод, нужно не забыть присвоить всем элементам массива значения.

```
public static void main(String[] args) {  
    int[] numbers = {1, 4, 7, 9}; // Заполнили массив значениями  
    printNumbers(numbers); // Вызвали метод  
}  
  
public static void printNumbers(int[] numbers) { // Параметр метода – массив целых чисел  
    for (int i = 0; i < numbers.length; i++) {  
        System.out.println(numbers[i]);  
    }  
}
```

Параметры могут быть и у методов с **void**, и у методов, которые возвращают значения. Рассмотрим пример с методом **add()**, который возвращает сумму параметров **a** и **b**. В качестве аргументов в этот метод можно передать и целые числа, и арифметическое выражение и даже результат вызова метода. Вот как это выглядит в коде:

```
public static void main(String[] args) {  
    int result = add(2, 3); // Вызываем метод с аргументами 2 и 3, сохраняем результат в result  
    System.out.println("Тут всё просто, 2 + 3 = " + result); // Печатаем пример  
  
    // Второй раз вызываем метод  
    result = add(338 + 665, 551 + 779); // В качестве аргументов переданы суммы чисел  
    System.out.println("Считали-считали, и получилось " + result);  
  
    // Посчитаем то же самое, но с помощью вызова метода  
    result = add(add(338, 665), add(551, 779)); // Аргументы – возвращённые значения метода  
    System.out.println("Ничего не изменилось, здесь получается " + result);  
}  
  
public static int add(int a, int b) { // Метод принимает два параметра – два целых числа  
    return a + b;  
}
```



В переменную **result** записывается результат выполнения метода **add(int a, int b)**. Если аргументы метода — конкретные числа (как в первом случае — **add(2, 3)**), то сразу вызывается метод. Если в качестве аргумента передаётся арифметическое выражение **338 + 665** или результат работы другого метода **add(338, 665)** то программа сначала вычисляет значение — конкретное число, которое и становится аргументом. Когда аргументы вычислены, происходит вызов метода.

Вопрос

Чему будет равна переменная `result`?

```
public static void main(String[] args) {  
    int result = doubleIt(add(13 + 2, doubleIt(5)));  
}  
  
public static int add(int a, int b) {  
    return a + b;  
}  
  
public static int doubleIt(int number) {  
    return number * 2;  
}
```

A. 40

B. 60

C. 50

D. 45



Ответ

С. 50

Ошибки при работе с параметрами и аргументами

Если при вызове метода с параметрами не передать им значения, произойдёт ошибка.

```
public class Practice {  
    public static void main(String[] args) {  
        sayHello();  
    }  
  
    public static void sayHello(String username) {  
        System.out.println("Привет, " + username);  
    }  
}
```

Результат

```
java: method sayHello in class Practice cannot be applied to given types;  
  required: java.lang.String  
  found:    no arguments  
  reason: actual and formal argument lists differ in length
```

Метод **sayHello** не может быть запущен, потому что ему не передано конкретное значение переменной **username**. Важно запомнить, что если у метода есть параметры, то при его вызове надо обязательно указать аргументы.

С аргументами методов могут возникать и другие ошибки. Так, код не запустится, если передать неверное количество аргументов или передать аргумент неправильного типа. Например, метод ожидает аргумент типа **int**, а при вызове передаётся **String**:

```
public class Practice {  
    public static void main(String[] args) {  
        add(2, 3, 9); // Ошибка – передано больше аргументов, чем нужно  
        add("Привет", 3); // Другая ошибка – первый аргумент неверного типа  
    }  
  
    public static int add(int a, int b) { // Метод принимает два целочисленных параметра  
        return a + b;  
    }  
}
```

Результат

```
java: method add in class Practice cannot be applied to given types;  
  required: int,int  
  found:    int,int,int  
  reason: actual and formal argument lists differ in length
```

Сигнатура метода

Имя метода вместе с типами его параметров называется **сигнатурой метода**. В сигнатуру не входят служебные слова **public static**, тип возвращаемого результата и имена параметров. Два метода с одинаковой сигнатурой объявить нельзя.

```
public class Practice {  
    public static void main(String[] args) {  
        printUserInfo("Катя", 26); // Программа не поймёт, какой метод вызвать  
    }  
  
    public static void printUserInfo(String name, int age) {  
        System.out.println("Меня зовут " + name + ", мне " + age + " лет.");  
    }  
  
    public static void printUserInfo(String city, int age) {  
        System.out.println("Я из города " + city + ", мне " + age + " лет.");  
    }  
}
```

Результат

```
java: method printUserInfo(java.lang.String,int) is already defined in class Practice
```

Сигнатура у объявленных методов одинаковая — `printUserInfo(String, int)`. Из-за этого произошла ошибка: программа просто не понимает, какой из методов вызвать.



Сигнатура метода

Чтобы устранить ошибку, можно переименовать один из методов. Если имена методов хочется оставить неизменными, нужно поменять количество аргументов в одном из методов (например, добавить в первый метод дополнительный строковый аргумент, который будет принимать название города) или изменить тип одного из аргументов.

Вопрос

Что из приведённого является сигнатурой метода.

- A. `public static int add(int, int)`
- B. `add(int, int)`
- C. `int add(int, String)`
- D. `add`
- E. `(int, double, String)`



Ответ

B. `add(int, int)`

Сигнатура метода — это его имя вместе с набором параметров.

Задача

Метод `findMax()` сейчас может работать только с фиксированными значениями — 3 и 5. Исправьте код метода таким образом, чтобы он мог принимать два любых целых значения. Затем по аналогии вызовите метод, чтобы напечатать результат сравнения ещё двух пар чисел: 16 и 5, -1 и -7.

```
public static void main(String[] args) {  
    System.out.println("Наибольшее из чисел 3 и 5 = " + findMax());  
    // Напишите аналогичный вызов метода findMax для таких пар чисел: 16 и 5, -1 и -7  
    ...  
}  
  
public static int findMax() {  
    int a = 5;  
    int b = 3;  
    if (a > b) {  
        return a;  
    }  
    return b;  
}
```

Решение

```
public static void main(String[] args) {  
    System.out.println("Наибольшее из чисел 3 и 5 = " + findMax(3, 5));  
    System.out.println("Наибольшее из чисел 16 и 5 = " + findMax(16, 5));  
    System.out.println("Наибольшее из чисел -1 и -7 = " + findMax(-1, -7));  
}  
  
public static int findMax(int a, int b) {  
    if (a > b) {  
        return a;  
    }  
    return b;  
}
```

Задача 2

Объявите и реализуйте метод **findMaxExpense** — он должен находить в списке расходов за неделю самую большую трату и возвращать её значение. В качестве единственного параметра этого метода укажите массив расходов **expenses**.

```
public class Practice {
    public static void main(String[] args) {
        double[] expenses = {1772.5, 367.0, 120.6, 2150.2, 874.0, 1.0, 1459.4};
        double maxExpense = ... // Вызовите метод и присвойте maxExpense значение его результата
        System.out.println("Самая большая трата недели " + maxExpense);
    }

    ... // Объявите метод findMaxExpense
    ... // Реализуйте метод. Чтобы найти самую большую трату, воспользуйтесь циклом
    ... // Самую большую трату запишите в переменную maxExpense
}
```

Решение

```
public class Practice {
    public static void main(String[] args) {
        double[] expenses = {1772.5, 367.0, 120.6, 2150.2, 874.0, 1.0, 1459.4};
        double maxExpense = findMaxExpense(expenses);
        System.out.println("Самая большая трата недели " + maxExpense);
    }

    public static double findMaxExpense(double[] expenses){
        double maxExpense = expenses[0];
        for (int i = 1; i < expenses.length; i++) {
            if (expenses[i] > maxExpense){
                maxExpense = expenses[i];
            }
        }
        return maxExpense;
    }
}
```



Методы и видимость переменных



Методы и видимость переменных

В темах, посвящённых условным операторам и циклам, вы уже встречались с таким понятием, как области видимости переменных (переменные, объявленные в одном блоке кода, не видны в другом). **Методы** — это тоже отдельные блоки кода, их тело заключено в фигурные скобки так же, как тело циклов и ветвлений. Поэтому работа с методами в программе всегда строится с учётом видимости переменных. Изучим этот вопрос подробнее.

Методы и видимость переменных

Поскольку блоки кода методов ничем не отличаются от блоков циклов или ветвлений, переменные, объявленные в разных методах, находятся в разных областях видимости. Прочитайте такой код и определите, с кем поздоровается программа — с Байтом или Пикселем:

```
public class Practice {  
    public static void main(String[] args) {  
        String username = "Пиксель";  
        sayHello();  
    }  
  
    public static void sayHello() {  
        String username = "Байт";  
        System.out.println("Привет, " + username);  
    }  
}
```

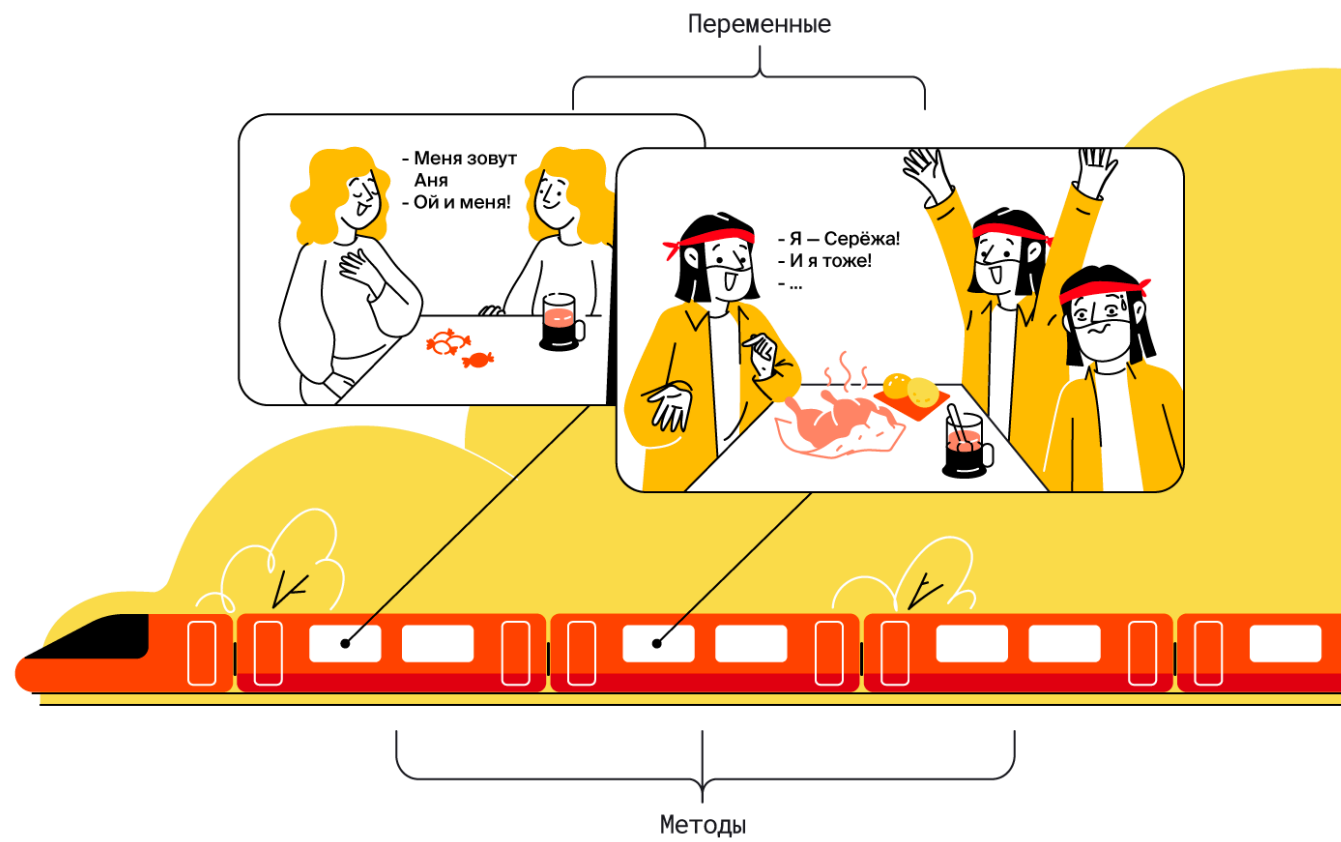



Результат

Привет, Байт

Программа поздоровается с хомяком Байтом, а не с котом Пикселем, как можно было бы подумать. Так происходит, потому что метод `sayHello()` берёт значение переменной `username` из своего блока кода — это "Байт". Переменная `username` со значением "Пиксель" объявлена внутри метода `main` — метод `sayHello()` её не видит.

Без ограничения видимости переменная, объявленная в одном методе, была бы доступна и в других. Это сильно усложнило бы написание программ: имя каждой переменной пришлось бы делать уникальным, а придумать имя для переменной не такая простая задача, как кажется, — на эту тему даже есть много мемов. Благодаря областям видимости можно использовать в разных методах переменные с одинаковыми именами — в примере с Пикселем и Байтом у нас две переменные `username`.



Задача


Исправьте предложенный код так, чтобы переменные **a** и **b** стали видны внутри метода `findMaxOf16And5()`. Решите эту задачу, не добавляя в метод параметры.

```
public class Practice {
    public static void main(String[] args) {
        int a = 16;
        int b = 5;
        System.out.println("Наибольшее из чисел 16 и 5 = " + findMaxOf16And5());
    }

    public static int findMaxOf16And5() {
        if (a > b) {
            return a;
        }
        return b;
    }
}
```

Решение

```
public class Practice {  
    public static void main(String[] args) {  
        System.out.println("Наибольшее из чисел 16 и 5 = " + findMaxOf16And5());  
    }  
  
    public static int findMaxOf16And5() {  
        int a = 16;  
        int b = 5;  
        if (a > b) {  
            return a;  
        }  
        return b;  
    }  
}
```



Имя для метода `findMaxOf16And5` выбрано не просто так — оно указывает, что метод работает только с числами **16** и **5**. Поэтому требовалось указать их в виде переменных внутри метода, а не в виде параметров. Если изменить метод так, чтобы он выбирал наибольшее значение из любой пары чисел, то тогда логичнее передавать их значения с помощью параметров.

Переделаем метод `findMaxOf16And5()` в метод `findMax()` с параметрами `int a` и `int b`. При вызове передадим в `findMax(int a, int b)` в качестве аргументов имена переменных, которые объявлены внутри метода `main`. Запустите код и убедитесь, что его результат будет таким же, как если бы мы объявили переменные в теле метода `findMax()`:

```
public class Practice {  
    public static void main(String[] args) {  
        int a = 16; // Переменные метода main  
        int b = 5;  
        // Передаём имена переменных метода main в качестве аргументов в метод findMax  
        System.out.println("Наибольшее из чисел 16 и 5 = " + findMax(a, b));  
    }  
  
    // Чтобы передать в метод аргументы, у него должны быть параметры  
    public static int findMax(int a, int b) {  
        if (a > b) {  
            return a;  
        }  
        return b;  
    }  
}
```



Результат

Наибольшее из чисел 16 и 5 = 16

Таким образом, если передать имена переменных методу в качестве аргументов — он сможет использовать их значения внутри своего блока. Внутри метода всегда видны его параметры.

Одному параметру можно передавать значения разных переменных.

Задача

Исправьте код таким образом, чтобы метод `sayHello()` принимал параметр — имя того, с кем нужно поздороваться. Затем поздоровайтесь с Пикселем и Байтом, используя переменные в `main`.

```
public class Practice {  
    public static void main(String[] args) {  
        String catName = "Пиксель";  
        String hamsterName = "Байт";  
        sayHello();  
    }  
  
    public static void sayHello() {  
        System.out.println("Привет, " + username);  
    }  
}
```


Решение

```
public class Practice {  
    public static void main(String[] args) {  
        String catName = "Пиксель";  
        String hamsterName = "Байт";  
        sayHello(catName);  
        sayHello(hamsterName);  
    }  
  
    public static void sayHello(String username) {  
        System.out.println("Привет, " + username);  
    }  
}
```

Когда в качестве аргумента используется имя переменной, в метод передаётся копия её значения, а не ссылка на него. Поэтому если изменить значение аргумента внутри метода, то с исходной переменной ничего не произойдёт — её значение останется прежним. Прочитайте следующий код и предположите, что будет напечатано в итоге. Затем проверьте себя.

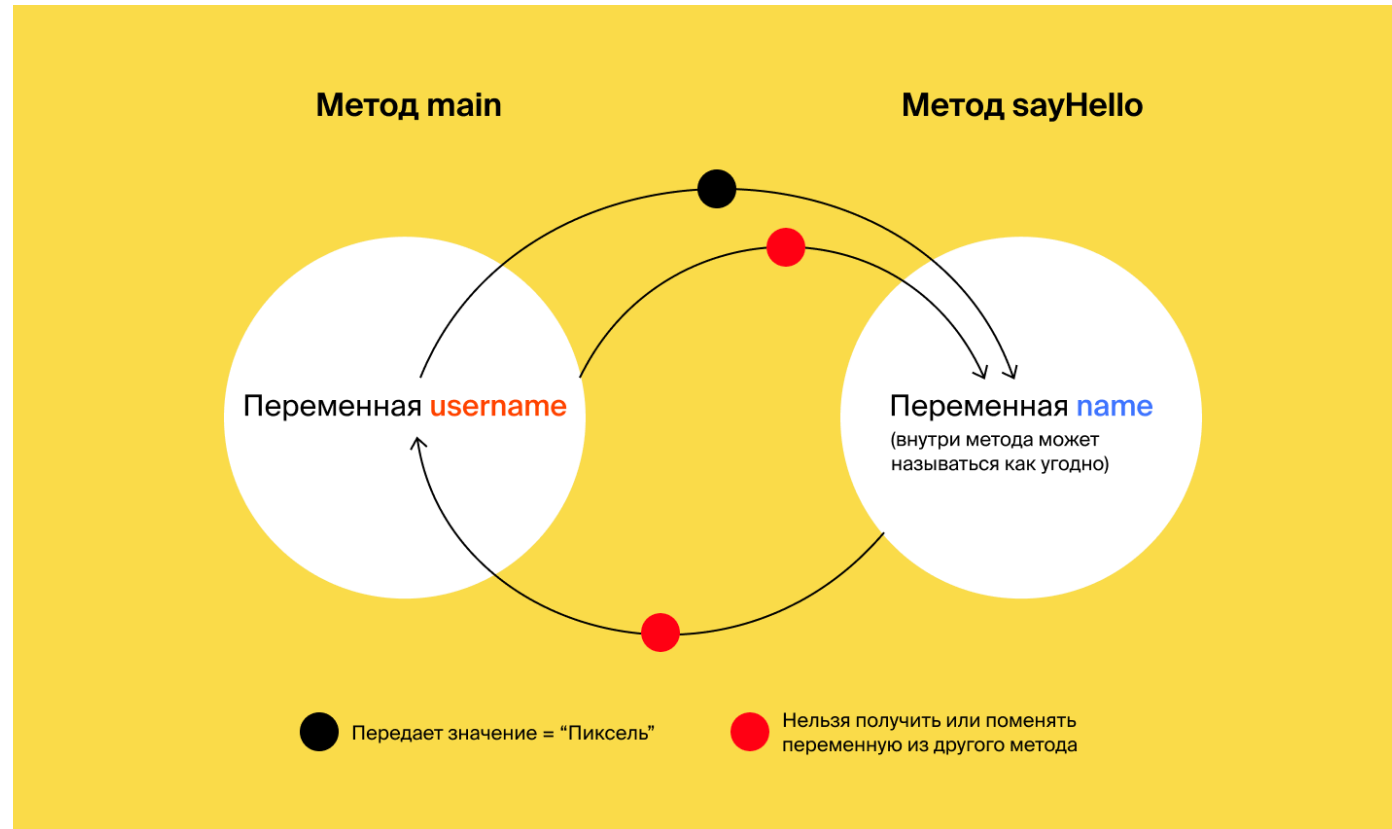
```
public class Practice {  
    public static void main(String[] args) {  
        String username = "Пиксель";  
        sayHello(username); // Здравуаемся из метода sayHello  
        System.out.println("Привет, " + username); // Здравуаемся из метода main  
    }  
  
    public static void sayHello(String name) {  
        name = "Байт"; // Меняем значение аргумента внутри метода  
        System.out.println("Привет, " + name);  
    }  
}
```

Результат

```
Привет, Байт  
Привет, Пиксель
```

Программа сначала поздоровается с Байтом, а потом с Пикселем. Сначала используется значение `username`, скопированное в аргумент и переданное в `sayHello(String name)` — там оно изменено на «Байт». С исходной переменной внутри `main(String[] args)` при этом ничего не произойдёт — её значение внутри главного метода останется прежним — «Пиксель». Поэтому второй раз программа поздоровается с котом.

В виде схемы это можно изобразить так. Область видимости переменной `username` — метод `main(String[] args)`. Область видимости переменной `name` — метод `sayHello(String name)`. Переменная `username` передала своё значение в качестве аргумента в метод `sayHello(String name)`. Получилось две переменных с одинаковым значением. Когда значение `name` меняется внутри метода `sayHello(String name)`, значение исходной переменной `username` в методе `main(String[] args)` остаётся прежним.



Задача

Исправьте ошибки в коде так, чтобы он заработал и поприветствовал Пикселем.

```
public class Practice {  
    public static void main(String[] args) {  
        String username = "Пиксель";  
        sayHello(name);  
    }  
  
    public static void sayHello(String name) {  
        System.out.println("Привет, " + username);  
    }  
}
```

Решение

```
public class Practice {  
    public static void main(String[] args) {  
        String username = "Пиксель";  
        sayHello(username);  
    }  
  
    public static void sayHello(String name) {  
        System.out.println("Привет, " + name);  
    }  
}
```



Задача 2

Кот Пиксель очень любит играть с мячиками — всего их у него 15. Днём он повсюду их прячет (число спрятанных мячиков считывается из консоли), но вечером всегда возвращает на место. Расставьте переменные **balls** и **hiddenBalls** в коде так, чтобы программа смогла правильно посчитать количество мячиков у Пикселя до начала игры, во время неё и в конце дня.

```

import java.util.Scanner;
public class Practice {

    public static void main(String[] args) {

        ... // Сохраните общее число мячиков Пикселя в переменной balls
        System.out.println("У Пикселя " + ...+" мячиков");

        ... // Поиграйте с Пикселем, вызвав метод playPixel

        // После игры Пиксель должен вернуть все мячики на место!
        System.out.println("Пиксель вернул все мячики");
        System.out.println("Их снова " + ...);
    }

    public static void playPixel(int balls) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Сколько мячиков он спрятал?");
        ... // Сохраните количество мячиков, которые спрятал Пиксель, в переменную hiddenBalls

        balls = ... // Посчитайте, сколько у Пикселя осталось мячиков
        System.out.println("Осталось " + ... );
    }
}

```


Решение

```
import java.util.Scanner;
public class Practice {
    public static void main(String[] args) {
        int balls = 15;
        System.out.println("У Пикселя " + balls + " мячиков");
        playPixel(balls);
        System.out.println("Пиксель вернул все мячики");
        System.out.println("Их снова " + balls);
    }

    public static void playPixel(int balls) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Сколько мячиков он спрятал?");
        int hiddenBalls = scanner.nextInt();
        balls -= hiddenBalls;
        System.out.println("Осталось " + balls );
    }
}
```

Задача 3

Вам нужно дописать код приложения, которое анализирует финансовые планы пользователя и помогает скорректировать их. Программа считывает из консоли значения зарплаты, планируемых трат на транспорт и еду, а также сумму возможных сбережений. Эти значения сохраняются в соответствующих переменных. Метод **correctExpenses** сравнивает планируемые траты с зарплатой, печатает размер излишка или недостатка средств и выдаёт рекомендации.

Объявите метод **correctExpenses**, который принимает значения переменных из главного метода в качестве аргументов. Дополните строки печати корректными значениями и исправьте ошибку в теле метода, связанную с видимостью одной из переменных.

https://github.com/practicetasks/java_tasks/tree/main/methods/task_2



Метод `main`

Метод `main`

Объявление метода `main` начинается со служебных слов `public` и `static`, а перед его именем стоит тип `void` — этот метод ничего не возвращает. Метод `main` всегда должен быть `void`. Если он будет возвращать значение другого типа, Java не сможет распознать его как главный метод программы и запустить.

Имя главного метода всегда неизменно — это обязательно `main`. Несмотря на то, что `main` — это основной метод, он не создаётся автоматически — каждый раз при создании программы его нужно объявлять, как и остальные методы.



Метод `main`

У метода `main` единственный параметр — массив строк `String[] args`. Имя `args` (сокр. от англ. *arguments* — «аргументы») является общепринятым. Но на самом деле оно может быть любым: программа запустится, даже если вы назовёте его `params`, `bams`, `parameters` или как-то ещё.

Массив `String[] args` содержит стартовый набор аргументов программы. Они нужны для того, чтобы управлять программой снаружи.



Метод `main`

Метод `main` мы не вызываем — это делает специальная программа для выполнения кода на Java, она называется **Java Virtual Machine** (англ. «виртуальная машина Java», сокр. JVM). Эта программа должна быть установлена на компьютере, на котором запускается код на Java.

Если использовать в `main` оператор `return`, то он завершит не только выполнение этого метода, но и всей программы.

К примеру, в этом коде **return** использован дважды: внутри главного метода и внутри метода **checkNumber**.

```
public class Practice {  
    public static void main(String[] args) {  
        int number = 15;  
        if (number >= 0) {  
            checkNumber(number);  
        } else {  
            System.out.println("Выбрано отрицательное значение! Конец программы!");  
            return;  
        }  
        System.out.println("Выбрано положительное значение или ноль! Конец программы!");  
    }  
  
    public static void checkNumber(int number) {  
        if (number > 0) {  
            System.out.println("Больше нуля.");  
        } else {  
            System.out.println("Ноль. Конец метода.");  
            return;  
        }  
        System.out.println("Конец метода.");  
    }  
}
```

Результат

Больше нуля.

Конец метода.

Выбрано положительное значение или ноль! Конец программы!

Если у **number** отрицательное значение — программа сразу завершится, так как именно после этого условия в главном методе стоит оператор **return**. Последняя строка в **main** напечатана не будет. Если у **number** нулевое значение, оператор **return** прервёт выполнение метода **checkNumber**. При этом программа продолжит работать и напечатает последнюю строку в методе **main**. При положительном значении **number** не будет задействован ни один из операторов **return**, и поэтому будет напечатано максимальное число строк — три.



Вопрос

Что из списка относится к особенностям метода **main**.

- A. С метода **main** начинается выполнение всей программы.
- B. Метод **main** создаётся автоматически при создании новой программы на Java.
- C. Метод **main** может обойтись без аргументов.
- D. Единственный параметр метода **main** — стартовый набор аргументов программы.
- E. Оператор **return** внутри метода **main** завершает выполнение всей программы.
- F. Оператор **return** внутри **main** завершает выполнение только метода, но не программы.
- G. Метод **main** может быть только **void**.
- H. Метод **main** может возвращать значения.



Ответ

- A. С метода **main** начинается выполнение всей программы.
- D. Единственный параметр метода **main** — стартовый набор аргументов программы.
- E. Оператор **return** внутри метода **main** завершает выполнение всей программы.
- G. Метод **main** может быть только **void**.



Декомпозиция кода на методы



Декомпозиция кода на методы

Теперь в вашем распоряжении есть все базовые знания о методах — вы умеете их объявлять и вызывать, возвращать из них значения и работать с их параметрами. С этими знаниями можно смело двигаться дальше. Научимся правильно разбивать код программы на методы.

Для начала изучите такой фрагмент кода и ответьте на вопрос о том, что делает метод

```
import java.util.Scanner;
public class Practice {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Как вас зовут?");
        String name = scanner.nextLine();
        sayHello(name);
    }

    public static void sayHello(String name) {

        System.out.println("Привет, " + name + "!");

        double[] expenses = {1012.50, 77.0, 810.6, 2150.20, 310.0, 486.0, 175.4};

        double maxExpense = 0;
        for (int i = 0; i < expenses.length; i++) {
            if (expenses[i] > maxExpense) {
                maxExpense = expenses[i];
            }
        }

        System.out.println("Самая большая трата за неделю составила " + maxExpense);
    }
}
```



Вопрос

Какие задачи решает метод `sayHello(String name)`?

- A. Находит самую большую трату недели.
- B. Узнаёт родной город пользователя.
- C. Печатает самую большую трату недели.
- D. Узнаёт имя пользователя.
- E. Приветствует пользователя по имени.
- F. Суммирует все траты недели.



Ответ

A. Находит самую большую трату недели.


Для этого в методе есть массив и цикл.

C. Печатает самую большую трату недели.

Метод умеет выводить в консоль самую большую сумму расходов.


E. Приветствует пользователя по имени.

Это метод делает в первую очередь.




Правильный ответ будет таким: «Метод `sayHello(String name)` приветствует пользователя по имени, находит максимальную трату за неделю и выводит её на экран». Объединить эти задачи было неудачным решением: в одном методе оказались два несвязанных набора команд: приветствие и работа с тратами.

Имя `sayHello` также выбрано неудачно — оно не соответствует содержанию метода. Впрочем, такому методу сложно придумать нормальное имя: если бы мы попытались отразить в названии всё, что он делает, то получилось бы что-то вроде `sayHelloAndSaveExpensesAndFindMaxExpense`.



Разобьём метод `sayHello(String name)` на два. Первый, `sayHello()`, будет отвечать за приветствие — перенесём в него ввод имени из `main(String[] args)`, чтобы собрать всё в одном месте.

```
// Метод отвечает за знакомство и приветствие  
public static void sayHello() {  
    Scanner scanner = new Scanner(System.in);  
    System.out.println("Как вас зовут?");  
    String name = scanner.nextLine();  
    System.out.println("Привет, " + name + "!");  
}
```




Второй метод, `findMaxExpense()`, будет анализировать траты — искать и возвращать максимальную сумму расходов. В `main(String[] args)` останется только вызов методов и вывод самой большой траты.

```
// Метод анализирует траты и возвращает наибольшую
public static double findMaxExpense() {
    double[] expenses = {1012.50, 77.0, 810.6, 2150.20, 310.0, 486.0, 175.4};
    double maxExpense = 0;
    for (int i = 0; i < expenses.length; i++) {
        if (expenses[i] > maxExpense) {
            maxExpense = expenses[i];
        }
    }
    return maxExpense;
}
```



В `main(String[] args)` останется только вызов методов и вывод самой большой траты.

```
// В main остались только вызовы методов  
public static void main(String[] args) {  
    sayHello();  
    System.out.println("Самая большая трата за неделю составила " + findMaxExpense());  
}
```



Такой код стало проще читать и воспринимать. Разделение больших сложных фрагментов кода на более мелкие и понятные называется **декомпозицией**. Декомпозиция нужна, если один метод решает сразу множество разноплановых задач или если код вообще не разбит на методы и с ним неудобно работать.

При декомпозиции кода принято придерживаться такого правила: **один метод — одна задача**. Если вы видите, что метод решает сразу несколько кардинально разных задач — самое время его декомпозировать.



Декомпозиция кода на методы

Структурирование кода значительно упрощает его чтение и понимание. Это увеличивает скорость написания программы и сокращает трудозатраты команды разработчиков.

Лайфхак на случай, если сомневаетесь, нужно ли декомпозировать метод. Задайте вопрос: «Какую задачу решает этот метод?». Если в ответе слишком часто употребляется союз «и» — метод точно можно разбить на несколько.



Декомпозиция кода на методы

Декомпозиция кода — отдельный этап работы. Всегда старайтесь продумывать структуру кода, прежде чем приступить к его написанию. Когда код написан, найдите время перечитать его, оценить и в случае необходимости дополнительно декомпозировать — это существенно повысит качество вашей работы.



Порядок действий при декомпозиции кода должен быть примерно таким:

- Сформулируйте, какие задачи должна решать программа. Для каждой задачи нужно предусмотреть свой метод.
- Если какой-то фрагмент кода дублируется несколько раз — это также повод завести для него отдельный метод.
- Проанализируйте, не возвращают ли методы какой-либо результат, работают ли они с внешними значениями. В зависимости от этого предусмотрите возврат значения или добавьте в метод параметры.
- Объявите и реализуйте все нужные методы. Проверьте их имена — они должны быть ёмкими и лаконичными, понятными другим разработчикам.

Вопрос

Прочитайте и проанализируйте код. На сколько методов его нужно декомпозировать?

https://github.com/practicetasks/java_tasks/tree/main/methods/decomposition_example

- A. 10
- B. 5
- C. 3
- D. 7
- E. 8
- F. Этот код нельзя декомпозировать — он не работает.



Ответ

10 - Нет, этот код не выполняет столько задач. Для печати цифрового меню достаточно одного метода.

5 - Нет. В этом коде семь разных задач — для каждой нужен свой метод.

3 - Нет. Если разбить этот код всего на три метода — их придётся декомпозировать повторно.

7 - Верно. Этот код можно декомпозировать на семь разных методов.

8 - Неверно. Этот код решает меньшее количество задач — прочитайте его внимательнее.

Этот код нельзя декомпозировать — он не работает. - Нет. В этом коде нет ошибок — его можно и нужно декомпозировать на отдельные методы.



Вопрос

Теперь, когда вы знаете, на сколько методов можно декомпозировать этот код, какие из этих методов имеют подходящие сигнатуры.

- A. `getMinExpenses(double[] expenses)`
- B. `singMummyTrollSongAboutGhosts(String ghosts)`
- C. `printMenu()`
- D. `findMaxResult()`
- E. `hamsterDiet(double weight, int carrot, int days)`
- F. `sayGoodBye(String lyrics)`



Ответ

- A. `getMinExpenses(double[] expenses)`
- C. `printMenu()`
- F. `sayGoodBye(String lyrics)`



Задача

Прочитайте код. Сейчас программа анализирует расходы на корм одновременно для двух питомцев: кота Пикселя и хомяка Байта — из-за этого в результатах печати можно запутаться. Декомпозируйте код — разбейте его отдельные методы: **sayHello**, **sayEnjoyMeal**, **findMaxExpense** и **findExpensesSum**. Методы с приветствием и пожеланием приятного аппетита должны быть типа **void**; методы, касающиеся анализа трат, должны возвращать значение.

Задача

У всех методов должны быть параметры. Внутри главного метода `main(String[] args)` должны остаться массивы с тратами, вызов методов и вывод результатов трат.

Вызовите методы так, чтобы сначала была напечатана информация про Пикселя, а потом про Байта. Порядок вывода такой: сперва приветствие, затем стоимость самого дорогого корма и общие траты на него и только потом пожелание приятного аппетита.

https://github.com/practicetasks/java_tasks/tree/main/methods/task_3



Решение

<https://gist.github.com/practicetasks/9f1e5a1e1302743346b5b43ac125a4ac>

Задача 2

Завершите декомпозицию финансового приложения — в методе `main(String[] args)` всё ещё сосредоточено слишком много задач. Вынесите обработку трат в отдельные методы: объявите, реализуйте и вызовите их.

https://github.com/practicetasks/java_tasks/tree/main/methods/task_4



Решение

<https://gist.github.com/practicetasks/39c00b599f77c8f4db063af0e1c806d9>