Конспект спринта № 1

Методы/Декомпозиция

Из этой темы вы узнали, для чего используются методы и как с ними работать. Вот что важно запомнить:

Объявление метода

Объявить метод — значит написать «заготовку» того, как он будет выглядеть.

```
public static void sayHello() { // Объявили метод sayHello
```

`public` и `static` — служебные слова, которые нужно указывать при объявлении метода.

`sayHello` — имя метода, по которому к нему можно будет обратиться.

`()` — так Java понимает, что мы объявили именно метод, а не переменную.

`void` означает, что метод не возвращает никакого значения.

Вызов метода

Метод начнёт исполняться только после того, как будет **вызван**. Для того чтобы вызвать метод, необходимо указать его имя и круглые скобки после имени:

```
public class Practicum {
  public static void main(String[] args) {
    System.out.println("Начало программы");
    sayHello(); // Вызывов метода sayHello
    System.out.println("Конец программы");
  }

  public static void sayHello() { //объявление метода sayHello
      printHeader(); // Вызывов метода printHeader
      System.out.println("Привет!");
  }

  public static void printHeader() { // Объявление метода printHeader
    System.out.println("Сейчас будет приветствие...");
  }
}
```

Несмотря на то, что метод `printHeader()` объявлен последним, он выполнит описанные в нём команды именно тогда, когда будет вызван.

Инструкции выполняются в порядке вызова методов. Расположение самих методов в коде значения не имеет.

Возвращение значения из метода

Для получения результата выполнения метода в Java используется специальное слово `return` (англ. «возврат, вернуть»). После `return` нужно написать возвращаемое значение.

Мы должны явно сообщить программе, данные какого типа возвращает каждый метод `int sum()`. Если в качестве типа указать `void`, то метод не будет возвращать никакого результата.

```
public static int sum() {
  return 2 + 3; // Метод будет возвращать 5
}

public static int divide() {
  return  "Разделяй и властвуй"; // Так нельзя! В этом случае метод должен обязательно возвращать int
}
```

Тип возвращаемого значения должен соответствовать типу, указанному при объявлении метода.

Особенности return

Явно указывать слово `return` в методах, которые не возвращают результат, может быть полезно в том случае, если нужно завершить метод не на последней строке метода, а где-то в середине. `return` не только возвращает результат, но и немедленно завершает выполнение метода.

```
public class Practicum {

  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;
  }

}
```

Аналогично и в методах с результатом можно возвращать значение не в конце, а в середине выполнения метода.

> 💡 Оператор `return` всегда должен быть последней командой в блоке кода `{return;}`.

Если после `return` написать в том же блоке кода ещё какие-нибудь команды, то программа не будет работать. При запуске возникнет ошибка `java: unreachable statement` (англ. «недостижимый оператор»).

Аргументы метода

При объявлении метода в круглых скобках можно указать тип и имя переменных, которые будет принимать метод. Их называют **аргументы**, или **параметры** метода.

Для того чтобы передать в метод значение аргументов, их необходимо указать при вызове метода в круглых скобках через запятую `add(2, 3);`.

> 💡 Имя метода вместе с набором его аргументов называется **сигнатурой метода**. Два метода с одинаковой сигнатурой объявить нельзя — произойдёт ошибка. Служебные слова и тип возвращаемого результата в сигнатуру метода не входят.

В качестве аргумента метода можно передавать значения любого типа, в том числе массивы. Главное, чтобы тип передаваемого значения соответствовал типу аргумента в сигнатуре метода.

```
public class Practicum {

  public static void main(String[] args) {
    int[] numbers = {1, 4, 7, 9}; // Массив целых чисел
    String[] letters = {"A", "B", "C", "D"}; // Массив строк
    printArray(numbers, letters); // Сначала числовой массив, потом массив строк, согласно сигнатуре
  }

  public static void printArray(int[] numbers, String[] letters) { // Аргументы метода: массив чисел и массив строк
    for (int i = 0; i < numbers.length; i++) {
      System.out.println(numbers[i]);
    }
    for (int i = letters.length; i > 0; i--) {
      System.out.println(letters[i]);
    }
  }

}
```

Не соответствие параметров вызова метода сигнатуре могут привести к неисправности кода. Если передать неверное число аргументов в метод, передать аргумент неправильного типа (например, метод ожидает аргумент типа `int`, а при вызове туда передается `String`) или вообще не передать аргументы в метод, который их ожидает, произойдёт ошибка `"Метод не может быть применён к данным типам []"`.

Области видимости

Часть кода, в которой можно обратиться к конкретной переменной, называется её **областью видимости**. Вы уже сталкивались с ней в темах «Циклы» и «Условные операторы».

Метод может работать только с теми переменными, которые объявлены внутри него. Переменные, находящиеся в соседних методах, ему недоступны.

```
public class Practicum {

  public static void main(String[] args) {
      String username = "Пиксель";
      sayHello(username); // Передаём в метод имя Пиксель
  }

  public static void sayHello(String name) { // Метод принимает один параметр - строку name
      // System.out.println("Привет, " + username); // Так написать не можем - метод не видит переменную метода main
      System.out.println("Привет, " + name); // А так можем - переменная name в этом методе есть
  }

}
```

Чтобы переменная `username` стала видна в методе `sayHello(String name)`, её нужно передать в качестве аргумента.

Main

Метод `main(String[] args)` — это входная точка любой программы.

В метод `main` в качестве единственного параметра передаётся массив строк `String[] args` — это стартовый набор аргументов программы. В качестве результата **main** возвращает `void` .

Если указать в методе `main(String[] args)` оператор `return`, то он завершит не только выполнение этого метода, но и всей программы.

```
public static void main(String[] args) {
  boolean exit = true;
  if (exit) {
    System.out.println("Завершение всей программы.");
    return;
  }
  System.out.println("Эту строку увидим, если программа не будет завершена раньше.");
}

public static void example(boolean exit) {
  if (exit) {
    System.out.println("Завершение метода example.");
    return ;
  }
  System.out.println("Эту строку увидим, если метод не будет завершён раньше.");
}
```

Название главного метода должно быть обязательно `main`, а название параметра-массива может быть любым: `args`, `params` или каким-нибудь ещё. Общепринятым названием этого параметра является именно `args`, сокращённое от arguments (англ. «аргументы»).

У разработчиков есть свод договорённостей, которые регламентируют, как сделать код понятным для любого программиста. Эти договорённости зафиксированы в специальной документации. В том числе документ рекомендует:

1) начинать имя метода с глагола, потому что методы описывают набор действий,

2) использовать уже знакомый вам «верблюжий» стиль *lowerCamelCase.*

Методы часто используют для **декомпозиции —** разделения больших и малопонятных частей кода на отдельные чёткие задачи. Основные принципы декомпозиции:

1. Метод всегда должен выполнять только одну задачу. Если задач больше — метод нужно разбить.
2. Метод может вернуть только один результат или не возвращать его совсем.
3. Название метода должно быть ёмким и кратким.

Объекты и классы

В этой теме вы познакомились с понятием класс, научились создавать объекты класса и работать с полями объектов. Что важно запомнить:

Объявление класса

Создание класса начинается с его объявления. Для этого нужно указать служебные слова `public` и `class`. После них с заглавной буквы пишется имя класса: `Hamster`. Когда класс объявлен, можно приступать к его описанию. Внутри фигурных скобок нужно объявить поля (свойства объекта) и методы (функции) — это **тело класса**. Начинать принято с полей.

```
public class Hamster {

    // Поля
    String name = "Байт"; // Имя
    int age = 2; // Возраст
    String color = "Рыжий"; // Цвет
    int weight = 350; // Вес в граммах

    // Есть
    void eat(int foodWeight) {
        weight = weight + foodWeight; // Если покормить Байта, то он немного прибавит в весе
    }

    // Бегать в колесе
    void runInWheel() {
        System.out.println("Бегу-бегу-бегу!");
        weight = weight - 5; // Байт сбросит вес, бегая в колесе
    }

    // Прятать семечки
    void hideSeeds(int seedWeight) {
        weight = weight + seedWeight; // Семечки за щекой сделают Байта неповоротливым
        System.out.println("Зимой не заголодаю.");
  }
}
```

Каждый новый класс создаёт новый тип данных. Поэтому имя класса становится типом переменной. Для новой переменной `Hamster` мы придумали имя `bite`. По этому имени можно будет обращаться к объекту.

Для того чтобы присвоить переменной `bite` значение нового объекта, необходимо использовать конструктор. Без `new` не обойтись:

```
  public static void main(String[] args) {
       Hamster bite; // Объявили переменную с типом Hamster
       Hamster bite = new Hamster(); // Присвоили переменной значение нового объекта-хомяка
 }
```

Методы

Взаимодействие с новым объектом происходит через его методы. Например, `bite.runInWheel()`.

```
public class Practicum {
    public static void main(String[] args) {
      Hamster bite = new Hamster();
      System.out.println("Вес хомяка до пробежки: " + bite.weight);
      bite.runInWheel();
      System.out.println("Вес хомяка после пробежки: " + bite.weight);
    }
}
```

Обращение к полям и методам объекта происходит с помощью **точечной нотации** `bite.weight` , `bite.runInWheel();`.

Чтобы поменять базовое значение поля объекта, указанное в классе, нужно обратиться к нему и присвоить этому полю новое значение:

```
public class Practicum {
  public static void main(String[] args) {
    Hamster bite = new Hamster();

    System.out.println("Раньше хомяка звали: " + bite.name); // Выведем на экран изначальное имя
    bite.name = "Хомка"; // Присвоим полю name новое значение
    System.out.println("Теперь хомяка зовут: " + bite.name); // А теперь посмотрим, что получилось
  }
}
```

Внутри каждого объекта хранится своя копия значений полей. Если изменить свойства одного объекта, на остальные объекты класса это не повлияет — у них сохранятся базовые значения полей.

Конструкторы

Конструктор создаёт новые объекты класса. **Конструктор по умолчанию** `Hamster bite = new Hamster();` позволяет создавать объекты с заданными заранее значениями полей.

Для того чтобы создавать объекты с разными значениями полей, используются **конструкторы с параметрами**. Конструктор с параметрами объявляется внутри класса.

В одном классе может быть несколько конструкторов. Имена у них будут совпадать. Главное, чтобы отличались их сигнатуры — количество и типы параметров. Тогда программа сможет понять, какой именно конструктор вызвать.

```
public class Hamster {

  String name;   // Имя
  int age;       // Возраст
  int weight; // Вес в граммах
  String color;  // Цвет

  // Конструктор № 1 создаёт только рыжих хомяков
  Hamster(String hamsterName, int hamsterAge, int hamsterWeight) {
    name = hamsterName;
    age = hamsterAge;
    weight = hamsterWeight;
    color = "Рыжий"; // Закрепили за переменной постоянное значение
  }

  // Конструктор № 2 создаёт разных хомяков
  Hamster(String hamsterName, int hamsterAge, int hamsterWeight, String hamsterColor) {
    name = hamsterName;
    age = hamsterAge;
    weight = hamsterWeight;
    color = hamsterColor;
  }
}
```

Когда используется конструктор с параметрами, то значения полям присваиваются внутри него. При создании нового объекта после слов `new Hamster` внутри круглых скобок можно сразу указать нужные атрибуты объекта:

```
public class Practicum {
  public static void main(String[] args) {
    Hamster volt = new Hamster("Вольт", 2, 340, "Рыжий");
    Hamster bosya = new Hamster("Бося", 1, 300, "Чёрный");
  }
}
```

> 💡 Важная деталь: когда вы объявили в классе конструктор с параметрами, программа перестает добавлять конструктор по умолчанию. Если попробовать создать новый объект и не передать для него параметры — произойдёт ошибка.

Командная строка

Чтобы вам было удобнее взаимодействовать с командной строкой и запоминать новый материал, мы подготовили небольшую шпаргалку. В ней собраны не только все команды, о которых мы говорили в этой теме, но и их полезные вариации.

- `pwd` (от англ. *print working directory,* «покажи рабочую папку») — покажи, в какой я папке;
- `ls` (от англ. *list directory contents,* «список содержимого каталога») — покажи файлы в текущей папке;
- `cd first-project` (от англ. *change directory,* «сменить каталог») — перейди в папку `first-project`;
- `cd first-project/html` — перейди в папку html, находящуюся в папке `first-project`;
- `cd ..` — перейди на уровень выше, в родительскую папку;
- `cd ~` — перейди в домашнюю директорию (у нас это `/Users/Username`);
- `mkdir second-project` (от англ. *make directory,* «создать директорию») — в текущей папке создай папку с именем `second-project`;
- `rm about.html` (от англ. *remove,* «удалить») — удали файл `about.html`;
- `rmdir images` (от англ. *remove directory,* «удалить директорию») — удали папку `images`;
- `rm -r second-project` (от англ. *remove,* «удалить» + *recursive*, «рекурсивный») — рекурсивно удали папку `second-project` и всё, что она содержит;
- `touch index.html` (англ. *touch,* «коснуться») — создай файл `index.html` в текущей папке;
- `touch index.html style.css script.js` — если нужно создать сразу несколько файлов, напечатайте их имена в одну строку через пробел.

Чтобы не вводить название файла или папки полностью, можно набрать первые пару символов их имени и дважды нажать Tab. Если соответствующий файл или папка есть в текущей директории, командная строка допишет путь сама.

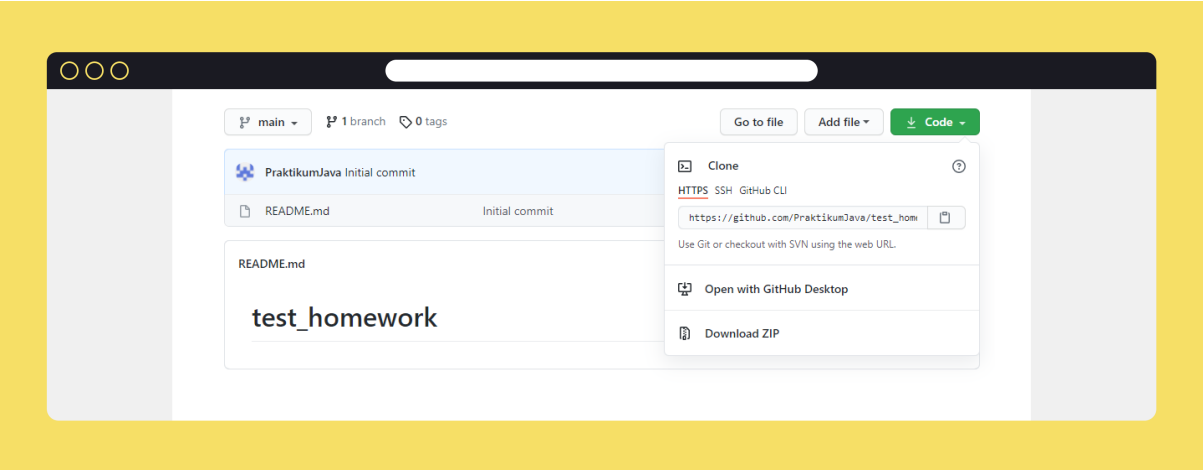
Например, находясь в папке `dev`, начните вводить `cd first` и дважды нажмите Tab. Если папка `first-project` есть внутри `dev`, командная строка автоматически подставит её имя. Останется только нажать Enter.

Работа с Git

Методы/Декомпозиция
Объявление метода

Клонирование репозитория

Для работы с репозиторием на своем компьютере его необходимо склонировать. На странице проекта GitHub нажмите кнопку Code в правой части страницы. Появится окно, из которого вы сможете скопировать адрес репозитория.



Этот адрес понадобится для клонирования. Теперь откройте терминал и перейдите в папку, куда собираетесь сохранить репозиторий. Введите команду `git clone` и адрес репозитория:

```
git clone https://github.com/ТУТ_ИМЯ_ВАШЕГО_АККАНТА/code-style-and-effective-work-in-ide-code-style.git
```

Создание коммита

Цикл создания любого коммита состоит из четырёх этапов:

- синхронизация локального и удалённого репозиториев;
- подготовка файла к коммиту;

- создание коммита;
- публикация коммита.

`git remote add origin https://github.com/YP/project.git` — привязка локального репозитория (текущей папки) к удалённому по URL.

`git push -u origin main` — залить все файлы из локального репозитория в удалённый.

`git add название_файла` — подготовка выбранного файла к коммиту.

`git add --all` — подготовка всех изменённых файлов к коммиту.

`git commit -m "Комментарий к коммиту."` — создание коммита.

`git push` — добавить изменения в удалённый репозиторий.

`git pull` — забрать изменения, сделанные другими разработчиками.

Внутри коммита

`git status` — проверить состояние файлов.

`git log` — посмотреть подробную историю коммитов.

`git log --oneline` — посмотреть короткий хеш коммитов.

`git reset HEAD` — вернуть проиндексированные файлы к состоянию при последнем коммите.

`git reset --hard b576d89` — удалить все незакоммиченные изменения из staged и рабочей зоны вплоть до указанного коммита.

`git diff` — посмотреть изменения в «рабочей зоне» — они маркируются как modifided, new или deleted.

`git diff --staged` — посмотреть изменения, добавленные в staged.

`git diff a9928ab 11bada1` — сравнить изменения двух коммитов.

`git commit --amend` — добавить изменения к последнему коммиту.

В зависимости от ОС для выхода из просмотра изменений может понадобиться нажать клавишу `Q` в английской раскладке.

Работа с ветками

`git branch название_ветки` — создать новую ветку.

`git checkout название_ветки` — переключиться в ветку.

`git checkout -b название_ветки` — создать ветку и сразу переключиться в неё.

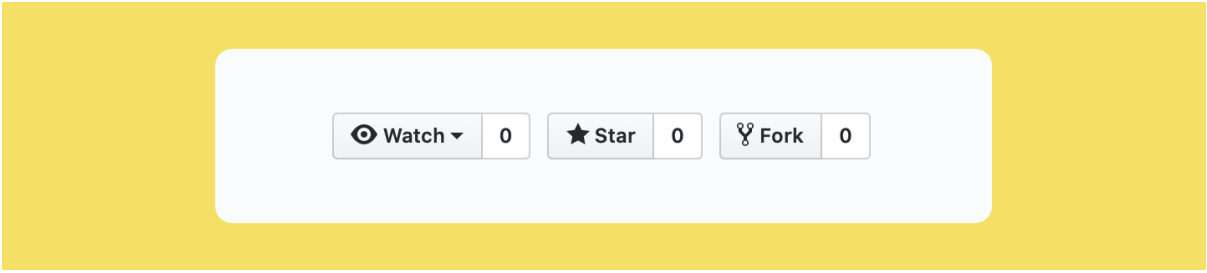
`git branch -D название_ветки` — удалить ветку.

`git merge название_ветки` — скопировать все изменения из ветки в ветку. Чтобы перенести изменения из ветки `develop` в ветку `main`, нужно находиться в ветке `main` и ввести `git merge develop`;

`git revert -m 1 хеш_коммита` — отмена коммита слияния веток. Опция `-m` со значением больше `0` указывает на основную ветку, которая будет сохранена.

Форк репозитория

Чтобы сделать свою копию репозитория, нажмите кнопку Fork в правом верхнем углу страницы репозитория. После этого проект скопируется в ваш аккаунт на GitHub. Возможность создания копии регулируется создателем репозитория.



JDK и среда разработки

Проверка установленного JDK

Чтобы убедиться, что системные переменные настроены правильно, нужно выполнить команды, приведённые ниже.

Если `JAVA_HOME` настроена, то первая команда распечатает её значение, а вторая покажет расположение папки, в которую установлена JDK. Если вместо результата будет пустая строка — значит, что-то настроено неправильно, обратитесь за помощью к наставнику.

- **Windows**

  Эти команды нужно выполнить в терминале Git Bash:

  ```
  echo $JAVA_HOME
  ```

  ```
  where java
  ```

  Пример выполнения:

  ```
  $ echo $JAVA_HOME
  C:\Program Files (x86)\Amazon Corretto\jdk11.0.10_9
  $ where java
  C:\Program Files (x86)\Amazon Corretto\jdk11.0.10_9\bin
  ```

- **Mac OS**

  ```
  echo $JAVA_HOME
  ```

  ```
  $(dirname $(readlink $(which javac)))/java_home
  ```

  Пример выполнения:

  ```
  $ echo $JAVA_HOME
  /Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home
  $ $(dirname $(readlink $(which javac)))/java_home
  /Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home
  ```

- **Linux**

```
echo $JAVA_HOME
```

```
dirname $(dirname $(readlink -f $(which java)))
```

Пример выполнения:

```
$ echo $JAVA_HOME
/usr/lib/jvm/java-11-amazon-corretto
$ dirname $(dirname $(readlink -f $(which java)))
/usr/lib/jvm/java-11-amazon-corretto
```

Проверка версии Java

Проверить, что на вашем компьютере установлена нужная версия Java, можно через консоль. Введите команду `java -version`.

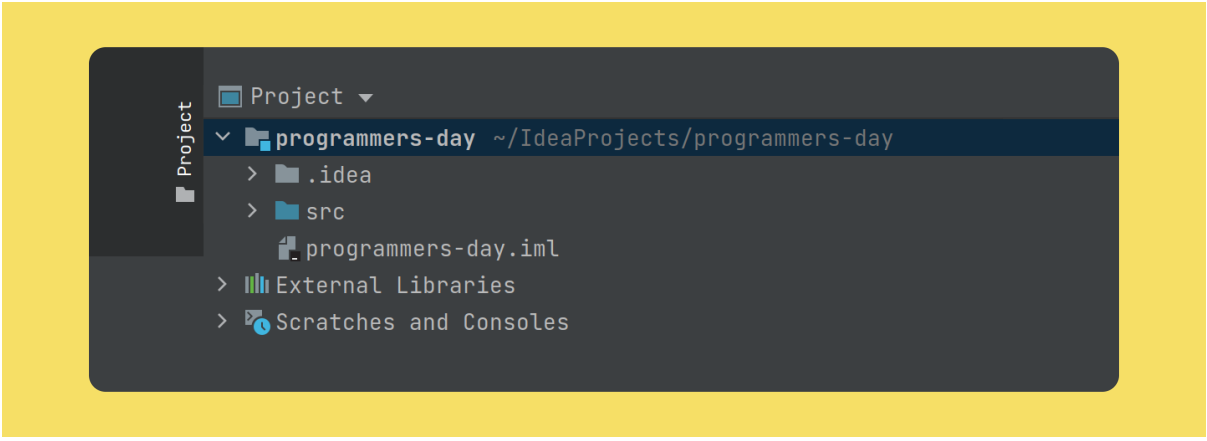
Если всё настроено правильно, результат выполнения будет выглядеть примерно так:

```
$ java -version
openjdk version "11.0.9.1" 2021-01-01 LTS
OpenJDK Runtime Environment Corretto-11.0.9.12.1 (build 11.0.9.1+12-LTS)
OpenJDK 64-Bit Server VM Corretto-11.0.9.12.1 (build 11.0.9.1+12-LTS, mixed mode)
```

В первой строке ответа указана версия и дата установки, в следующих двух — более подробная информация о JDK.

Структура проекта в IDEA

Чтобы увидеть структуру проекта в IDEA, нужно открыть вкладку Project.



Папки `.idea` и `src` появляются автоматически.

Все файлы, которые относятся к проекту, должны храниться в папке `src` (от англ. *source* — «исходники»). Это могут быть java-файлы, изображения, файлы настроек и прочие составляющие проекта.

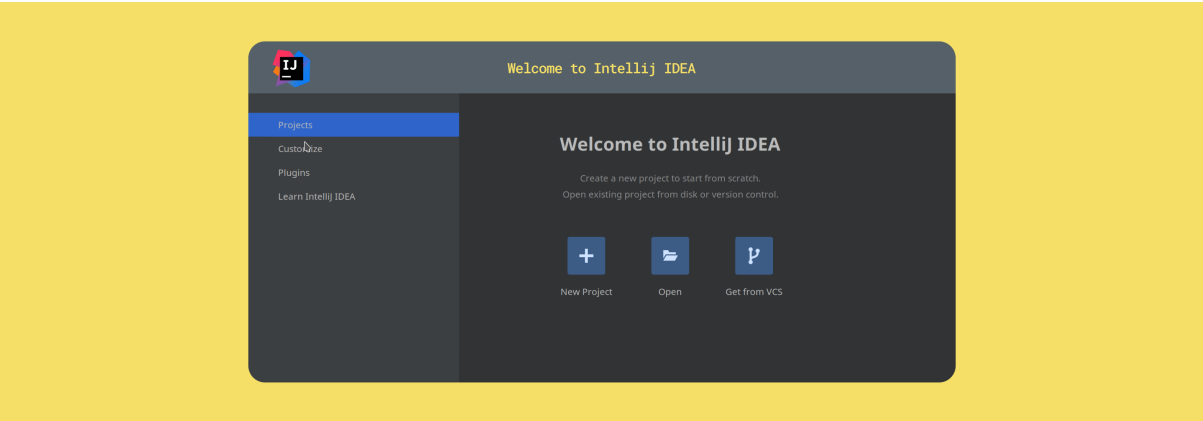
А папку `.idea` IntelliJ IDEA генерирует для себя, чтобы хранить в ней информацию, которая нужна для её корректной работы.

После запуска проекта в его структуру также добавится папка `out`. В неё попадут скомпилированные файлы, необходимые для выполнения программы.

Весь код проекта будет находиться в папке `src`. Откройте контекстное меню для этой папки и выберите **New → Java Class**, в новом окне введите имя класса — *Practicum* и нажмите **Enter**.

Импорт проекта в IDEA

Выберите пункт Open or Import в стартовом диалоге открытия IDEA.



В появившемся диалоговом окне открытия файлов выберите папку со склонированным репозиторием и нажмите кнопку Open. Обратите внимание, что открыть нужно весь каталог. После этого проект будет добавлен в IntelliJ IDEA.