

Лабораторная работа №21 ООП в Java.

Объектно-ориентированное программирование в Java похоже на ООП в C++

Нет нужды в объявлении класса и его методов в одном файле, а реализации – в другом, все осуществляется в рамках одного файла.

Например, опишем класс, представляющий автомобиль:

```
public class Auto {
    private String firm; //создаем закрытый член нашего класса с названием фирмы автомобиля
    private int maxSpeed; // закрытый член класса, содержащий максимальную скорость

    public void setFirm(String firma){ //открытая функция (метод класса) для задания
        firm=firma; //значения фирмы автомобиля
    }

    public void setMaxSpeed(int speed){ //открытая функция (метод класса) для задания
        maxSpeed=speed; //значения максимальной скорости автомобиля
    }

    public int getMaxSpeed(){ //открытая функция (метод класса) для вывода значения
        return maxSpeed; //максимальной скорости
    }

    public String getFirm(){ //открытая функция (метод класса) для вывода значения
        return firm; //заданной фирмы
    }

    public Auto(){ // конструктор класса (без параметров)
        firm="Без названия";
        maxSpeed=0;
    }
    public Auto(String firma, int speed){ //конструктор класса (с параметрами)
        firm=firma;
        maxSpeed=speed;
    }
}
```

Чтобы использовать созданный класс, лучше написать другой класс для тестирования нашего класса (в том же проекте):

```
public class test {
    public static void main(String[] args) {
        Auto myAuto1=new Auto("Ford",180); // создаем объект типа нашего класса
        System.out.println(myAuto1.getFirm()+" "+myAuto1.getMaxSpeed()); //вывод сведений в
    } // консоль
}
```

Или, с вводом данных с клавиатуры:

```
import java.util.Scanner; //подключаем класс для ввода данных с клавиатуры в консоли

public class test {
    public static void main(String[] args) {
        Auto myAuto1=new Auto(); //создаем объект типа нашего класса
        Scanner in = new Scanner(System.in); //создаем сканер для ввода данных из консоли
        System.out.print("Введите фирму: ");
        String nazv=in.next(); //считываем название из консоли !!!только 1 слово
        //т.к. in.next() считывает только символы до пробела, остальные символы отправляет
        //следующему оператору, связанному с консольным вводом
        myAuto1.setFirm(nazv); //задаем значение для параметра нашего класса
        System.out.print("Введите максимальную скорость: ");
        int s=in.nextInt();
        myAuto1.setMaxSpeed(s);
        System.out.println(myAuto1.getFirm()+" "+myAuto1.getMaxSpeed());
    }
}
```

Теперь на основе созданного класса легко создать классы-наследники – класс Car (легковая) и класс Truck (грузовая):

```
public class Car extends Auto{//файл Car.java
    private String model;
    private int numDoors;
    private Boolean fullTime; //полный привод

    public Car(){
        super(); // вызываем конструктор класса-родителя без параметров (см. класс Auto)
        model=""; // добавляем инициализацию новых членов
        numDoors=4;
        fullTime=false;
    }
}
```

```

public Car(String firma, int speed, String name, int n, Boolean f){
    super(firma,speed); вызываем конструктор класса-родителя с параметрами (см. класс Auto)
    model=name; // добавляем инициализацию новых членов
    numDoors=n;
    fullTime=f;
}

public void setModel(String name){
    model=name;
}
public String getModel(){
    return model;
}

public void setNumDoors(int n){
    numDoors=n;
}
public int getNumDoors(){
    return numDoors;
}

public void setFullTime(Boolean b){
    fullTime=b;
}
public Boolean isFullTime(){
    return fullTime;
}
public String toString(){
    return getFirm()+" "+getMaxSpeed()+" "+model+" "+numDoors+" "+fullTime;
}
}

```

//файл Truck.java

```

import java.util.Scanner;

public class Truck extends Auto{
    private String model;
    private int power;
    private Boolean trailer; //с прицепом или без

    public Truck(){
        super();
        model="";
        power=0;
        trailer=false;
    }

    public Truck(String firma, int speed, String name, int n, Boolean f){
        super(firma,speed);
        model=name;
        power=n;
        trailer=f;
    }

    public void setModel(String name){
        model=name;
    }
    public String getModel(){
        return model;
    }

    public void setPower(int n){
        power=n;
    }
    public int getPower(){
        return power;
    }

    public void setTrailer(Boolean b){
        trailer=b;
    }
    public Boolean isTrailer(){
        return trailer;
    }

    public void setAllInfo(){

```

```

        Scanner in = new Scanner(System.in);
        System.out.print("Введите фирму-производитель грузового авто: ");
        String nazv=in.next(); //метод next() позволяет вводить строки, но без пробелов
        setFirm(nazv);
        System.out.print("Введите максимальную скорость грузового авто: ");
        int s=in.nextInt();
        setMaxSpeed(s);
        System.out.print("Введите модель грузового авто: ");
        model=in.next();
        System.out.print("Введите мощность грузового авто: ");
        power=in.nextInt();
        System.out.print("Введите признак прицепа грузового авто (true/false): ");
        trailer=in.nextBoolean();
        System.out.println();
    }
    public String toString(){
        return "\n\tГрузовик"+ "\n\t" + "Фирма: " + getFirm() + "\n\t" + "Максимальная скорость: "
        + getMaxSpeed() + "\n\t" + "Модель: " + model + "\n\t" + "Мощность: " + power + "\n\t" + "Признак прицепа: "
        + trailer + "\n";
    }
}

```

Пример с использованием созданных классов (измененный test.java):

```

import java.util.Scanner;

public class test {

    public static void main(String[] args) {
        Auto myAuto1=new Auto();
        Scanner in = new Scanner(System.in);
        System.out.print("Введите фирму: ");
        String nazv=in.next();
        myAuto1.setFirm(nazv);
        System.out.print("Введите максимальную скорость: ");
        int s=in.nextInt();
        myAuto1.setMaxSpeed(s);
        System.out.println("Какой-то автомобиль: " + myAuto1.getFirm() + " " + myAuto1.getMaxSpeed());
        System.out.println();

        Car myCar1=new Car("Ford", 200, "Mustang", 2, false);
        Car myCar2=new Car();
        System.out.print("Введите фирму-производитель легкового авто: ");
        nazv=in.next();
        myCar2.setFirm(nazv);
        System.out.print("Введите максимальную скорость легкового авто: ");
        s=in.nextInt();
        myCar2.setMaxSpeed(s);
        System.out.print("Введите модель легкового авто: ");
        nazv=in.next();
        myCar2.setModel(nazv);
        System.out.print("Введите кол-во дверей легкового авто: ");
        s=in.nextInt();
        myCar2.setNumDoors(s);
        System.out.print("Введите признак полного привода легкового авто (true/false): ");
        Boolean f=in.nextBoolean();
        System.out.println();
        System.out.println("Первый легковой автомобиль: " + myCar1.toString());
        System.out.println("Второй легковой автомобиль: " + myCar2.toString());

        Truck myTruck=new Truck();
        myTruck.setAllInfo();
        System.out.println(myTruck.toString());
    }
}

```

Пример работы с программой:

Введите фирму: Lada
 Введите максимальную скорость: 130
 Какой-то автомобиль: Lada 130

Введите фирму-производитель легкового авто: Nissan
 Введите максимальную скорость легкового авто: 230
 Введите модель легкового авто: Patrol
 Введите кол-во дверей легкового авто: 5
 Введите признак полного привода легкового авто (true/false): true

```

Первый легковой автомобиль: Ford 200 Mustang 2 false
Второй легковой автомобиль: Nissan 230 Patrol 5 false
Введите фирму-производитель грузового авто: Kamaz
Введите максимальную скорость грузового авто: 180
Введите модель грузового авто: Masters
Введите мощность грузового авто: 400
Введите признак прицепа грузового авто (true/false): false

```

Грузовик:

```

Фирма: Kamaz
Максимальная скорость: 180
Модель: Masters
Мощность: 400
Признак прицепа: false

```

Мы можем использовать созданные классы не только в наследовании, но и для агрегации (когда один класс содержит в себе в качестве членов объекты других классов). Например, создадим класс Garage (Гараж), описывающий кол-во и состав машин в гараже:

```

import java.util.ArrayList; //нужно для работы с классом ArrayList

public class GarageCar {

    private ArrayList<Auto> masCar=new ArrayList<Auto>(); //массив с машинами

    public void addCar(Auto m){ //метод для добавления машины в гараж
        masCar.add(m);
    }

    public GarageCar () {

    }

    public Boolean findCar(Auto m){ //для выяснения - есть ли машина m в гараже
        return masCar.contains(m);
    }

    public GarageCar(ArrayList< Auto> n){ //конструктор для внесения существующего списка машин
        //в гараж
        masCar=n;
    }

    public void printGarage() { //для вывода на экран списка машин в гараже
        System.out.println("В гараже: ");
        for (Auto a:masCar){ //
            System.out.println("\t"+a.toString());
        }
    }
}

```

В этом примере использован объект типа ArrayList. Этот класс предназначен для работы с массивами объектов одного типа. Т.е. по сути это другой способ представления классического массива, но с удобными методами класса ArrayList. Синтаксис объявления и создания объекта типа ArrayList:

```
ArrayList<тип данных в массиве> имя_массива=new ArrayList<тип данных в массиве>();
```

После этого объекты в ArrayList можно добавлять при помощи метода add(элемент массива).

Некоторые другие полезные методы класса ArrayList (после двоеточия указан тип возвращаемого функцией значения, void – нет возвращаемого значения):

```

contains(m): Boolean – возвращает true, если содержится элемент m, иначе false
add (int index, m) : void – добавляет элемент m в конкретную позицию index
clear() : void – удаляет все элементы из списка
get(int index) : Тип_элементов_массива – возвращает элемент, стоящий на позиции index
indexOf(m) : int – возвращает номер позиции элемента m, если есть одинаковые элементы, то первую позицию
isEmpty() : boolean – возвращает true, если массив пустой, иначе false
lastIndexOf(m) : int – возвращает номер последней позиции элемента m
remove(int index) : Тип_элементов_массива – удаляет элемент с индексом index, остальные элементы сдвигаются на позицию, сам удаленный элемент возвращается функцией
remove(m) : boolean – удаляет элемент m, возвращает true, если элемент был в списке
set(int index, m) : Тип_элементов_массива – заменяет элемент на позиции index на новый элемент m, старый элемент возвращается функцией
size() : int – кол-во элементов, содержащихся в массиве
subList(int i1, int i2) : List<Тип_элементов_массива> – выдает список элементов от элемента с номером i1 до номера i2
toArray() : Object[] – преобразует в обычный массив

```

В нашем примере создается массив объектов типа Auto, т.е. мы сможем в нем хранить как элементы типа Car, так и элементы типа Truck.

Если посмотреть на цикл for, использованный в методе printGarage(), то можно увидеть, что он используется не совсем обычно:

```
for (Auto a:masCar){
    System.out.println("\t"+a.toString());
}
```

Этот цикл носит название «для каждого» (foreach). В качестве параметра цикла выступает переменная a типа Auto, при этом указывается, что переменная a будет каждую итерацию цикла заменяться элементом из masCar, который является представителем класса ArrayList, т.е. для каждого элемента из созданного массива будет выполняться действие в теле цикла. Этот цикл удобно использовать, если мы не хотим зависеть от размеров массива (класса ArrayList).

Напишем программу с использованием созданного гаража:

```
public class testGarage {

    public static void main(String[] args) {
        GarageCar myGarage=new GarageCar(); //создаем новый гараж
        Car myCar1=new Car("Ford", 200,"Mustang",2,false); //создаем легковую машину
        myGarage.addCar(myCar1); // добавляем ее в гараж
        myGarage.addCar(new Car("LADA", 140, "Kalina", 4, false)); //добавляем еще одну машину
        Truck myTruck=new Truck("Dove",160,"DTS",700,true); //создаем грузовик
        myGarage.addCar(myTruck); //добавляем его в гараж
        myGarage.printGarage(); //выводи на экран содержимое гаража
        if (myGarage.findCar(myCar1)) { //ищем машину
            System.out.println("Да");
        }
        else {
            System.out.println("Нет");
        }
    }
}
```

Пример работы программы:

В гараже:

```
Ford 200 Mustang 2 false
LADA 140 Kalina 4 false
```

```
Грузовик
Фирма: Dove
Максимальная скорость: 160
Модель: DTS
Мощность: 700
Признак прицепа: true
```

Да

Есть еще один интересный аспект при работе с классом ArrayList. Можно узнать класс объекта, который является текущим при обработке в цикле, для этого используется оператор instanceof – оператор сравнения на принадлежность к определенному классу или типу, т.е. можно написать

```
for (Auto a:masCar){
    if (a instanceof Car) {System.out.println("Это легковая машина");}
```

и текст *Это легковая машина* будет выведен столько раз, сколько содержится объектов типа Car в нашем массиве, то же самое можно сделать и для объектов типа Truck. Оператор instanceof понадобится для решения задач.

Задачи:

I. Обязательная задача для всех:

Добавить к гаражу возможность удаления из него машины, а к классу Auto добавить поле с гос. номером.

II. По вариантам (создать классы, в них предусмотреть различные члены классов и методы для работы):

1. Базовый класс – учащийся. Производные – школьник и студент. Создать класс Конференция, который может содержать оба вида учащихся. Предусмотреть метод подсчета участников конференции отдельно по школьникам и по студентам (использовать оператор instanceof).
2. Базовый класс – работник. Производные – работник на почасовой оплате и на окладе. Создать класс Предприятие, который может содержать оба вида работников. Предусмотреть метод подсчета работников отдельно на почасовой оплате и на окладе (использовать оператор instanceof).
3. Базовый класс – компьютер. Производные – ноутбук и смартфон. Создать класс РемонтСервис, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно ремонтируемых ноутбуков и смартфонов (использовать оператор instanceof).

4. Базовый класс – печатные издания. Производные – книги и журналы. Создать класс КнижныйМагазин, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно книг и журналов (использовать оператор `instanceof`).
5. Базовый класс – помещения. Производные – квартира и офис. Создать класс Дом, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно квартир и офисов (использовать оператор `instanceof`).
6. Базовый класс – файл. Производные – звуковой файл и видео-файл. Создать класс Каталог, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно звуковых и видео-файлов (использовать оператор `instanceof`).
7. Базовый класс – летательный аппарат. Производные – самолет и вертолет. Создать класс Авиакомпания, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно самолетов и вертолетов (использовать оператор `instanceof`).
8. Базовый класс – соревнование. Производные – командные соревнования и личные. Создать класс Чемпионат, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно командных соревнований и личных (использовать оператор `instanceof`).
9. Базовый класс – мебель. Производные – диван и шкаф. Создать класс Комната, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно диванов и шкафов (использовать оператор `instanceof`).
10. Базовый класс – оружие. Производные – огнестрельное и холодное. Создать класс ОружейнаяПалата, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно огнестрельного и холодного оружия (использовать оператор `instanceof`).
11. Базовый класс – оргтехника. Производные – принтер и сканер. Создать класс Офис, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно принтеров и сканеров (использовать оператор `instanceof`).
12. Базовый класс – СМИ. Производные – телеканал и газета. Создать класс Холдинг, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно телеканалов и газет (использовать оператор `instanceof`).

Лабораторная работа №22 Документирование кода на языке Java с использованием утилиты `javadoc`.

В среду программирования `javac` встроена поддержка утилиты `javadoc` для языка Java. Утилита `javadoc` входит в состав JDK – Java Development Kit (комплект разработчика приложений на языке Java). Она позволяет генерировать html-страницы с документацией созданных классов, если код этих классов размечен специальными комментариями. Эти комментарии должны начинаться со знаков `/**`, а заканчиваться `*/`

Внутри спецкомментариев допустимы следующие специальные дескрипторы:

Список дескрипторов <i>Javadoc</i>		
Дескриптор	Описание	Применим к
<code>@author</code>	Автор	класс, интерфейс
<code>@version</code>	Версия. Не более одного дескриптора на класс	класс, интерфейс
<code>@since</code>	Указывает, с какой версии доступно	класс, интерфейс, поле, метод
<code>@see</code>	Ссылка на другое место в документации	класс, интерфейс, поле, метод
<code>@param</code>	Входной параметр метода	метод
<code>@return</code>	Описание возвращаемого значения	метод
<code>@exception</code> <i>имякласса</i> <i>описание</i> <code>@throws</code> <i>имякласса</i> <i>описание</i>	Описание исключения, которое может быть обработано внутри метода	метод
<code>@deprecated</code>	Описание устаревших блоков кода	метод
<code>{@link reference}</code>	Ссылка	класс, интерфейс, поле, метод
<code>{@value}</code>	Описание значения переменной	статичное поле

Например, ранее созданный класс Auto можно было бы задокументировать следующим образом:

```
/**
 * Класс Автомобиль - базовый класс для объектов транспорта
 * @author Слива М.В.
 */
public class Auto {
    /**Поле для хранения названия фирмы автомобиля */
    private String firm;

    /**Поле для хранения максимальной скорости автомобиля */
    private int maxSpeed;

    /**
     * Устанавливает значение поля {@link Auto#firm}
     * @param firma - название фирмы автомобиля */
    public void setFirm(String firma){
        firm=firma;
    }

    /**
     * Устанавливает значение поля {@link Auto#maxSpeed}
     * @param speed - значение максимальной скорости автомобиля */
    public void setMaxSpeed(int speed) {
        maxSpeed=speed;
    }

    /**
     * Возвращает значение поля {@link Auto#maxSpeed}
     * @return целое значение максимальной скорости автомобиля */
    public int getMaxSpeed() {
        return maxSpeed;
    }

    /**
     * Возвращает значение поля {@link Auto#firm}
     * @return строку с названием фирмы автомобиля */
    public String getFirm(){
        return firm;
    }

    /**
     * Создает автомобиль с фирмой "Без названия" и максимальной скоростью, равной 0*/
    public Auto(){
        firm="Без названия";
        maxSpeed=0;
    }

    /**
     * Создает автомобиль с заданными значениями фирмы и максимальной скорости
     * @param firma - название фирмы автомобиля
     * @param speed - значение максимальной скорости автомобиля*/
    public Auto(String firma, int speed){
        firm=firma;
        maxSpeed=speed;
    }
}
```

Созданные описания можно предварительно посмотреть во вкладке Javadoc нижней панели среды

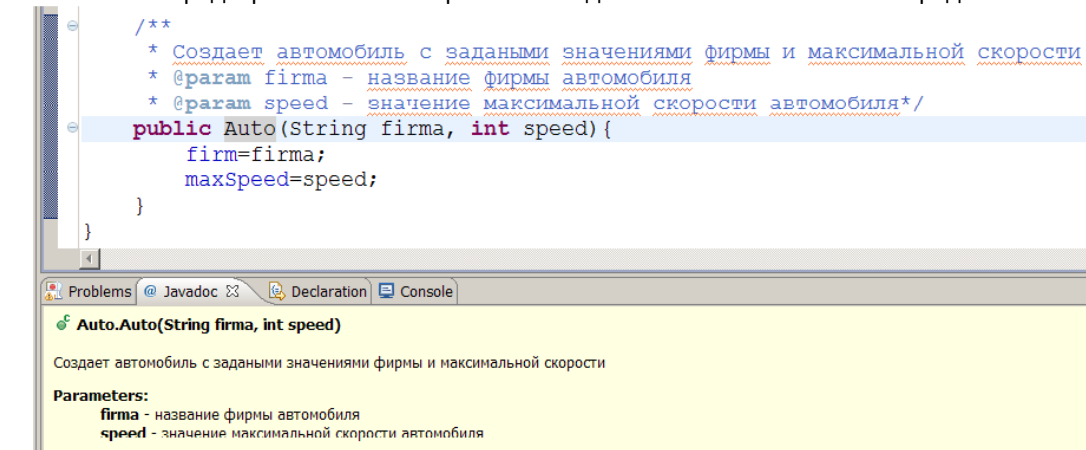


Рис. 1. Предварительный просмотр созданной документации.

Для этого нужно просто установить курсор на названии описанного метода, поля или класса.

Чтобы сгенерировать документацию по классу в виде html-страниц, нужно выбрать команды меню Project\Generate Javadoc. Появится диалоговое окно с параметрами (рис. 2), в котором нужно выбрать в своем проекте классы, для которых нужно сгенерировать документацию, выбрать путь к утилите Javadoc (как правило, это C:\Program Files\Java\jdk1.6.0_27\bin\javadoc.exe) и путь для сохранения документации (по умолчанию, это текущий проект). И выбрать **Private** в пункте **Create Javadoc for members with visibility**. Это позволит в документации осуществлять переход к полям класса, которые являются private.

После этого, нажав несколько раз Next и в конце Finish, можно получить набор html-страниц с документацией созданного класса. Каталог с ней будет примерно следующего вида (рис. 3):

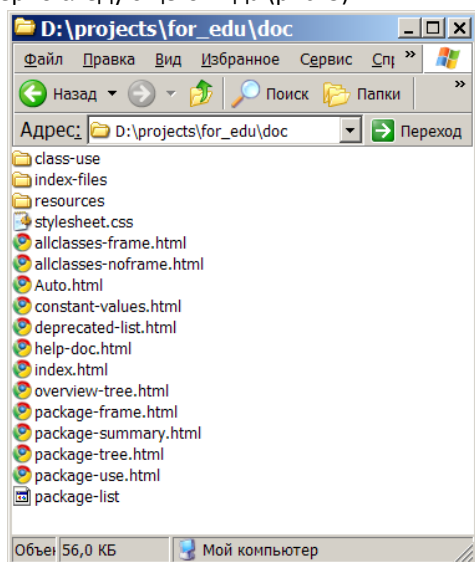


Рис. 3. Каталог с созданной документацией.

И, запустив файл index.html, в браузере можно увидеть документацию стандартного для Java вида (рис. 4):

Class Auto

java.lang.Object
└ Auto

```
public class Auto
extends java.lang.Object
```

Класс Автомобиль - базовый класс для объектов транспорта

Author:
Слива М.В.

Constructor Summary

Auto ()
Создает автомобиль с фирмой "Без названия" и максимальной скоростью, равной 0

Auto (java.lang.String firma, int speed)
Создает автомобиль с заданными значениями фирмы и максимальной скорости

Method Summary

java.lang.String	getFirm ()	Возвращает значение поля firm
int	getMaxSpeed ()	Возвращает значение поля maxSpeed
void	setFirm (java.lang.String firma)	Устанавливает значение поля firm
void	setMaxSpeed (int speed)	Устанавливает значение поля maxSpeed

Рис. 4. Документация в сгенерированном файле.

Задание:

Сделать документацию по всем созданным ранее классам.

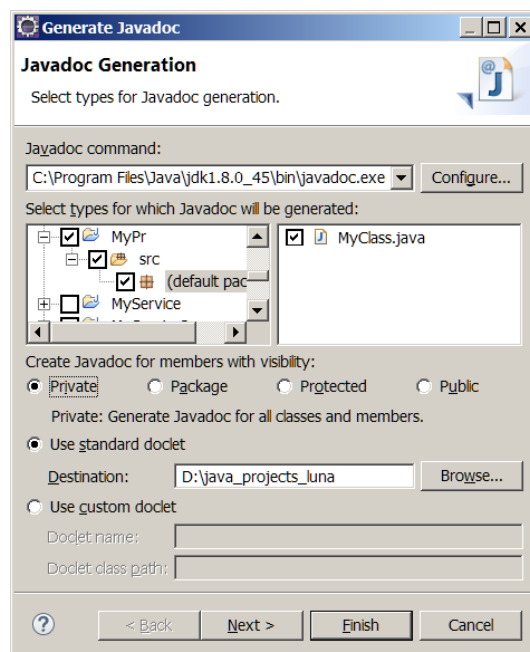


Рис. 2. Диалоговое окно Project\Generate Javadoc.

JavaDoc в IntelliJ

- 1) Используйте View / Quick Documentation или соответствующую комбинацию клавиш (по умолчанию: Ctrl + Q на Windows/Linux и Ctrl + J на macOS или F1 в последних версиях IDE).
- 2) Также можно включить автоматическое всплывающее окно JavaDoc при явном (вызываемом ярлыком) завершении кода в Settings | Editor | General | Code completion (документация Autopopup).
- 3) Перейдите к File/Settings , Editor, нажмите на General . Прокрутите вниз, затем ☐ Show quick documentation on mouse move