

# Объектная модель в языке Java

# ООП

Java является объектно-ориентированным языком программирования

ООП — методология программирования, основанная на представлении программного продукта в виде совокупности объектов, каждый из которых является экземпляром конкретного класса. ООП использует в качестве базовых элементов взаимодействие объектов.

Объект — именованная модель реальной сущности, обладающая конкретными значениями свойств и проявляющая свое поведение.

В применении к объектно-ориентированным языкам программирования понятия объекта и класса конкретизируются.

Объект — обладающий именем набор данных (полей и свойств объекта), физически находящихся в памяти компьютера, и методов, имеющих доступ к ним. Имя используется для работы с полями и методами объекта. Любой объект относится к определенному классу. В классе дается обобщенное описание некоторого набора родственных объектов.

Объект — конкретный экземпляр класса.

Объектно-ориентированное программирование основано на принципах:

- инкапсуляции;
- наследования;
- полиморфизма, в частности, «позднего связывания».

# Объявление класса

Синтаксис объявления класса:

```
[спецификаторы] class ИмяКласса [extends СуперКласс] [implements список_интерфейсов] {  
/* определение класса */  
}
```

Спецификаторы доступа к  
классу:

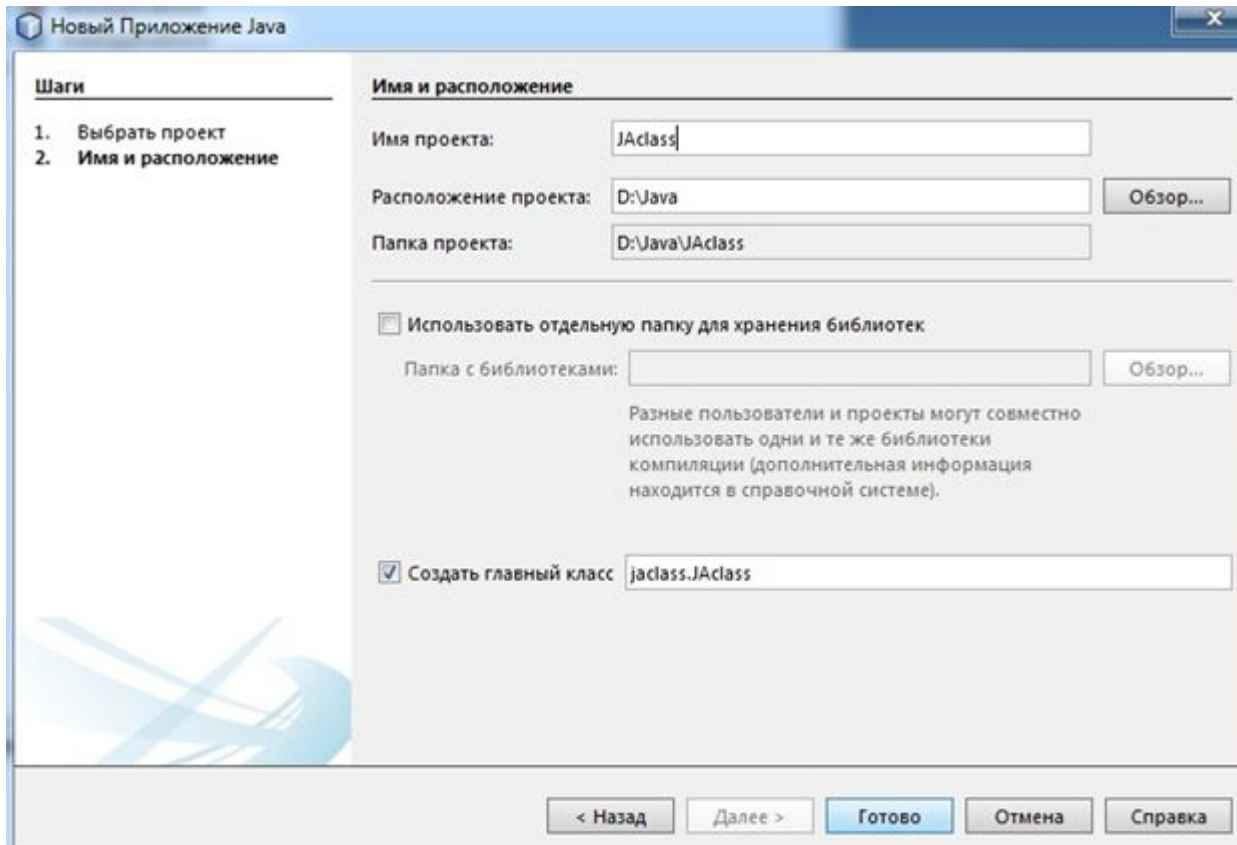
- **public**
- **final**
- **abstract**
- **friendly**

Спецификаторы доступа к методам и свойствам  
класса:

- **public**
- **private**
- **protected**

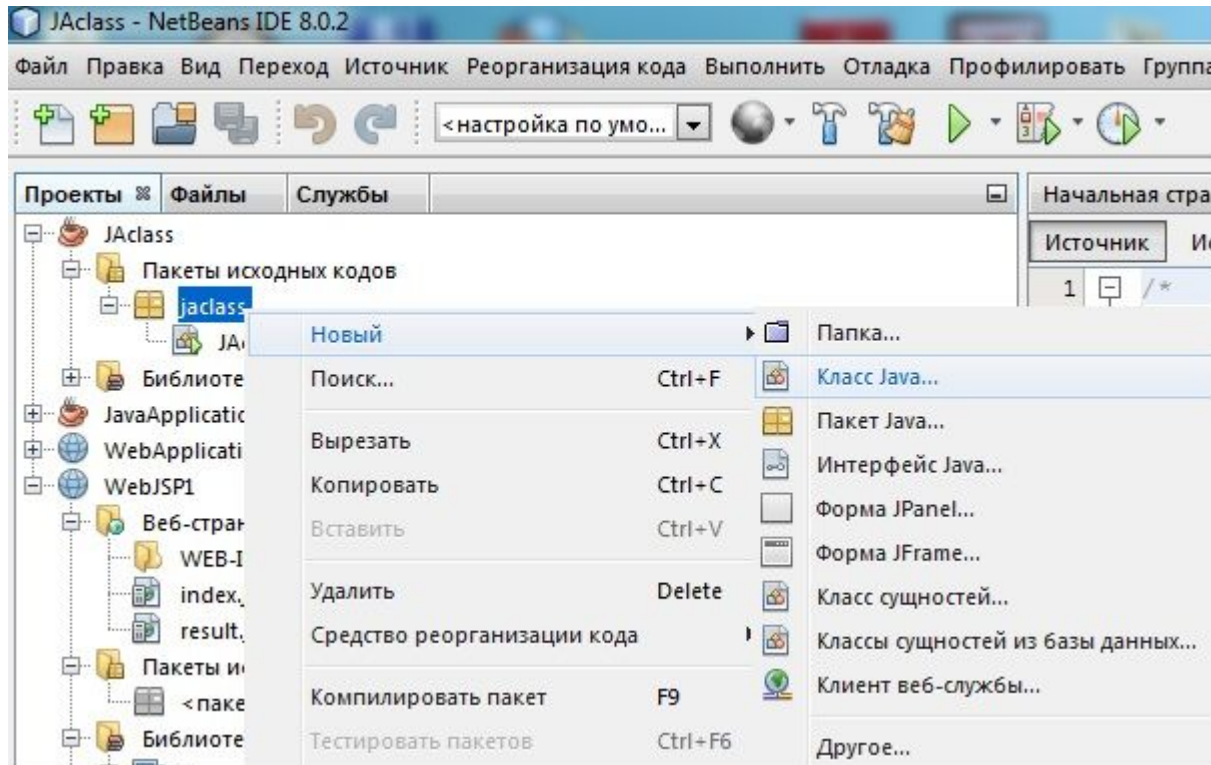
# Пример для работы с классами

## 1. Создаем проект



# Пример для работы с классами

## 2. Добавляем класс Triangle



## 3. Добавляем поля `private double x1,x2,x3,y1,y2,y3`

# Пример для работы с классами

## 4. Реорганизуем код

```
6 package jclass;
7
8 /**
9  *
10  * @author admin
11  */
12 public class Triangle {
13     private double x1, x2, x3, y1, y2, y3;
14
15 }
16
```

- Переход
  - Показать документацию Java ALT+F1
  - Найти случаи использования ALT+F7
  - Иерархия вызова
- Вставка кода... ALT+Insert
- Исправить операторы импорта Ctrl+Shift+I
- Средство реорганизации кода
- Формат ALT+Shift+F
- Выполнить файл Shift+F6
- Отладка файла Ctrl+Shift+F5
- Тестировать файлов Ctrl+F6
- Отладка файла теста Ctrl+Shift+F6
- Метод выполнения специализированного теста
- Метод отладки специализированного теста
- Запустить в методе
- Создать наблюдение...
- Переключить точку останова
- Профилирование
- Вырезать
- Копировать
- Вставить
- Свертывание кода
- Выбрать в проектах

- Переименовать... Ctrl+R
- Переместить... Ctrl+M
- Копировать...
- Безопасное удаление... Alt+Delete
- Встроить...
- Изменить параметры метода...
- Вытянуть вверх...
- Вытолкнуть вниз...
- Извлечь интерфейс...
- Извлечь родительский класс...
- Использовать родительский тип во всех возможных случаях...
- Ввести
- Переместить внутренний на внешний уровень...
- Превратить анонимный класс во внутренний...
- Инкапсулировать поля...
- Заменить конструктор на фабрику...
- Заменить конструктор на компоновщик...
- Инвертировать логическое значение...
- Проверить и преобразовать...

### Инкапсулировать поля

Список полей для инкапсулирования:

Поле	Создать метод получения	Создать метод установки
x1 : double	<input checked="" type="checkbox"/> getX1	<input checked="" type="checkbox"/> setX1
x2 : double	<input checked="" type="checkbox"/> getX2	<input checked="" type="checkbox"/> setX2
x3 : double	<input checked="" type="checkbox"/> getX3	<input checked="" type="checkbox"/> setX3
y1 : double	<input checked="" type="checkbox"/> getY1	<input checked="" type="checkbox"/> setY1
y2 : double	<input checked="" type="checkbox"/> getY2	<input checked="" type="checkbox"/> setY2

Точка вставки: Значение по умолчанию

Сортировать по: Пары "метод получения/метод установки"

Документация Javadoc: Создание комментариев по умолчанию

Видимость полей: private

Видимость аксессоров: public

Использовать аксессоры даже для доступных полей (пока не доступно)

Генерировать поддержку изменения свойства

Дать поддержку изменений с правом вето

Выбрать все  
Снять выделение  
Выбрать методы получения  
Выбрать методы установки

Предварительный просмотр Средство реорганизации кода Отмена Справка

# Пример для работы с классами

Класс Triangle:

```
package jclass;
```

```
/**
 *
 * @author admin
 */
public class Triangle {
    private double x1,x2,x3,y1,y2,y3;

    /**
     * @return the x1
     */
    public double getX1() {
        return x1;
    }

    /**
     * @param x1 the x1 to set
     */
    public void setX1(double x1) {
        this.x1 = x1;
    }
}
```

```
    * @return the x2
    */
    public double getX2() {
        return x2;
    }

    /**
     * @param x2 the x2 to set
     */
    public void setX2(double x2) {
        this.x2 = x2;
    }

    /**
     * @return the x3
     */
    public double getX3() {
        return x3;
    }

    /**
     * @param x3 the x3 to set
     */
    public void setX3(double x3) {
        this.x3 = x3;
    }
}
```

# Пример для работы с классами

```
/**  
 * @return the y1  
 */  
public double getY1() {  
    return y1;  
}
```

```
/**  
 * @param y1 the y1 to set  
 */  
public void setY1(double y1) {  
    this.y1 = y1;  
}
```

```
/**  
 * @return the y2  
 */  
public double getY2() {  
    return y2;  
}
```

```
/**  
 * @param y2 the y2 to set  
 */  
public void setY2(double y2) {  
    this.y2 = y2;  
}
```

```
/**  
 * @return the y3  
 */  
public double getY3() {  
    return y3;  
}
```

```
/**  
 * @param y3 the y3 to set  
 */  
public void setY3(double y3) {  
    this.y3 = y3;  
}
```



# Пример для работы с классами

```
private double distance(double x1, double y1, double x2, double y2){
    return Math.pow(Math.pow(x2-x1,2)+Math.pow(y2-y1, 2), 0.5);
}

public double perimeter(){
    return distance(x1, y1, x2, y2)+distance(x2, y2, x3, y3)+distance(x3, y3, x1, y1);
}

public double area(){
    double p=perimeter()*0.5;
    return Math.pow(p*(p-distance(x1, y1, x2, y2))*(p-distance(x2, y2, x3,
y3))*(p-distance(x3, y3, x1, y1)), 0.5);
}
}
```

# Пример для работы с классами

Класс JAclass:

```
package jaclass;
import java.util.Scanner;
/**
 *
 * @author admin
 */
public class JAclass {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Triangle tr=new Triangle();
        try{
            Scanner con = new Scanner(System.in);
            System.out.println("Введите координаты первой точки");
            System.out.println("x1:");
            tr.setX1(con.nextDouble());
            System.out.println("y1:");
            tr.setY1(con.nextDouble());
```

```
            System.out.println("Введите координаты второй точки");
                System.out.println("x2:");
                tr.setX2(con.nextDouble());
                System.out.println("y2:");
                tr.setY2(con.nextDouble());
                System.out.println("Введите координаты первой точки");
                System.out.println("x3:");
                tr.setX3(con.nextDouble());
                System.out.println("y3:");
                tr.setY3(con.nextDouble());
                System.out.format("Периметр треугольника=%10.1f%n",tr.perimeter());
                System.out.format("Площадь треугольника=%10.1f%n",tr.area());
                con.close();
            }catch(Exception e){
                System.err.println("Ошибка" + e.toString());
            }
        }
    }
}
```

## Оператор if-else

Оператор if – else имеет следующий синтаксис:

```
if(условие) {
```

```
...
```

```
}
```

```
else {
```

```
...
```

```
}
```

## Оператор switch

Оператор switch имеет следующий синтаксис:

```
switch(целочисленное выражение или enum){
```

```
  case метка1: оператор1,оператор2,...,break;
```

```
  case метка2: оператор1,оператор2,...,break;
```

```
...
```

```
  case меткаN: оператор1,оператор2,...,break;
```

```
  default: оператор1,оператор2,...;
```

```
}
```

В Java 7 в switch можно использовать класс String

Пример:

```
public String getNameOfDay(int N) {  
    String typeOfDay="";  
    switch (N) {  
        case 1: typeOfDay = "Monday"; break;  
        case 2: typeOfDay = "Thursday"; break;  
        case 3: typeOfDay = "Wednesday"; break;  
        case 4: typeOfDay = "Thursday"; break;  
        case 5: typeOfDay = "Friday"; break;  
        case 6: typeOfDay = "Saturday"; break;  
        case 7: typeOfDay = "Sunday"; break;  
        default: throw new IllegalArgumentException("Invalid day of  
                                                    the week: " + N);  
    }  
    return typeOfDay;  
}
```

# Циклы в Java.

1. Цикл while.

Цикл while имеет вид

```
while (условие) {
```

```
...
```

```
};
```

2. Цикл do – while.

Цикл do - while имеет вид

```
do{
```

```
...
```

```
} while (условие);
```

3. Цикл for

Цикл for имеет вид

```
for(инициализация; условие; приращение){
```

```
...
```

```
}
```

#### 4. Цикл for-each.

Цикл for-each развитие цикла for. Пример:

```
int[] a={1,2,3,4};  
int sum=0;  
for(int value:a){  
    sum+=value;  
}
```

ЭТОТ КОД ЭКВИВАЛЕНТЕН КОДУ:

```
int[] a={1,2,3,4};  
int sum=0;  
int value=0;  
for(int i=0;i<a.length;i++){  
    value=a[i];  
    sum+=value;  
}
```