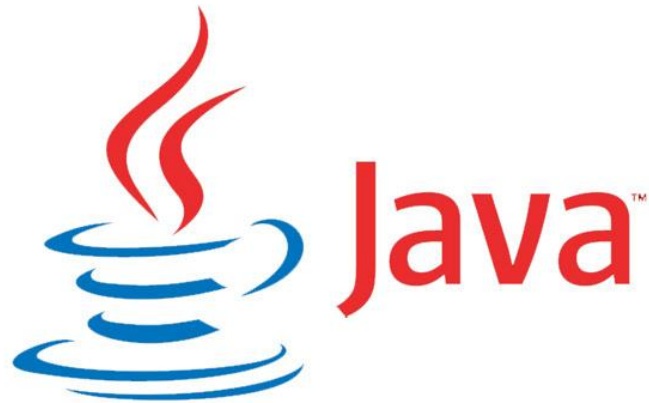


# Тема 5

# Классы и объекты



# Классы и объекты

Java является объектно-ориентированным языком, поэтому такие понятия как "**класс**" и "**объект**" играют в нем ключевую роль. Любую программу на Java можно представить как набор взаимодействующих между собой объектов.

Шаблоном или описанием объекта является класс, а объект представляет экземпляр этого класса. Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о машине – наличие колес, двигателя, кузова и т.д. Есть некоторый шаблон - этот шаблон можно назвать классом. Реально же существующий автомобиль (фактически экземпляр данного класса) является объектом этого класса.

# Классы и объекты



Класс определяется следующим способом:

```
class Person {  
  
}
```

# Классы и объекты



В данном случае класс называется Person. После названия класса идут фигурные скобки, между которыми помещается тело класса - то есть его поля и методы. Любой объект может обладать двумя основными характеристиками: состояние - некоторые данные, которые хранит объект, и поведение - действия, которые может совершать объект.

# Классы и объекты



Для хранения состояния объекта в классе применяются поля или переменные класса. Для определения поведения объекта в классе применяются методы. Например, класс Person, который представляет человека, мог бы иметь следующее определение:

```
class Person {  
    private int age;  
    private String name;  
  
    public void displayInfo () {  
        System.out.println("Person: age: " + this.age + " : " + "name: " + this.name);  
    }  
}
```

# Классы и объекты



В классе `Person` определены два поля: `name` представляет имя человека, а `age` - его возраст. И также определен метод `displayInfo`, который ничего не возвращает и просто выводит эти данные на консоль.



# Классы и объекты

Как правило, классы определяются в разных файлах. Класс представляет новый тип, поэтому мы можем определять переменные, которые представляют данный тип.

# Конструкторы

Кроме обычных методов классы могут определять специальные методы, которые называются конструкторами. Конструкторы вызываются при создании нового объекта данного класса.

Конструкторы выполняют инициализацию объекта. Если в классе не определено ни одного конструктора, то для этого класса автоматически создается конструктор без параметров.



# Конструкторы



```
class Person {  
    private int age;  
    private String name;  
    private static int counter;  
  
    public Person () {  
        System.out.println("Initialized constructor");  
        counter++;  
    }  
  
    public Person (int age1) {  
        age = age1;  
    }  
  
    public Person (int age2, String name) {  
        counter++;  
        age = age2;  
        this.name = name;  
    }  
}
```

# Конструкторы

Для создания объекта `Person` используется выражение **`new Person()`**. Оператор `new` выделяет память для объекта `Person`. И затем вызывается конструктор по умолчанию, который не принимает никаких параметров. В итоге после выполнения данного выражения в памяти будет выделен участок, где будут храниться все данные объекта `Person`. Если конструктор не инициализирует значения переменных объекта, то они получают значения по умолчанию. Для переменных **числовых типов** это число `0`, а для типа **`String`** и классов - это значение `null` (то есть фактически отсутствие значения).



# Ключевое слово `this`

Ключевое слово `this` представляет ссылку на текущий экземпляр класса. Через это ключевое слово мы можем обращаться к переменным, методам объекта, а также вызывать его конструкторы.

# Инициализаторы

Кроме конструктора начальную инициализацию объекта вполне можно было проводить с помощью инициализатора объекта. Инициализатор выполняется до любого конструктора. То есть в инициализатор мы можем поместить код, общий для всех объектов класса.

```
class Person {  
    private int age;  
    private String name;  
    private static int counter;  
  
    static {  
        System.out.println("Initialized static block");  
        counter = 2;  
    }  
  
    {  
        System.out.println("Initialized non static block");  
    }  
}
```



# Модификаторы доступа

Все члены класса в языке Java - поля и методы - имеют модификаторы доступа. В прошлых темах мы уже сталкивались с модификатором `public`. Модификаторы доступа позволяют задать допустимую область видимости для членов класса, то есть контекст, в котором можно употреблять данную переменную или метод.

# Модификаторы доступа



## В Java используются следующие модификаторы доступа:

- **public**: публичный, общедоступный класс или член класса. Поля и методы, объявленные с модификатором `public`, видны другим классам из текущего пакета и из внешних пакетов.
- **private**: закрытый класс или член класса, противоположность модификатору `public`. Закрытый класс или член класса доступен только из кода в том же классе.
- **protected**: такой класс или член класса доступен из любого места в текущем классе или пакете или в производных классах, даже если они находятся в других пакетах
- **Модификатор по умолчанию**. Отсутствие модификатора у поля или метода класса предполагает применение к нему модификатора по умолчанию. Такие поля или методы видны всем классам в текущем пакете.

# Перечисления

## Проход по всем элементам

Кроме отдельных примитивных типов данных и классов в Java есть такой тип как enum или перечисление.

Перечисления представляют набор логически связанных констант. Объявление перечисления происходит с помощью оператора enum, после которого идет название перечисления. Затем идет список элементов перечисления через запятую.

```
public enum CoffeeSize {  
    BIG, HUGE, OVERWHELMING  
}
```

# Перечисления

Перечисление фактически представляет новый тип, поэтому мы можем определить переменную данного типа и использовать ее:

```
public class CoffeeSizeDemo1 {  
    public static void main(String[] args) {  
        CoffeeSize coffeeSize = CoffeeSize.BIG;  
        if (coffeeSize == CoffeeSize.BIG) {  
            System.out.println(coffeeSize);  
        }  
    }  
}
```



# Методы перечислений

Каждое перечисление имеет статический метод `values()`. Он возвращает массив всех констант перечисления:

```
enum Type {  
    SCIENCE,  
    BELLETRE,  
    PHANTASY,  
    SCIENCE_FICTION  
}  
  
public class CoffeeSizeDemo1 {  
    public static void main(String[] args) {  
        Type[] types = Type.values();  
        for (Type s : types) {  
            System.out.println(s);  
        }  
    }  
}
```

# Методы перечислений

Метод `ordinal()` возвращает порядковый номер определенной константы (нумерация начинается с 0):

```
enum Type {  
    SCIENCE,  
    BELLETRE,  
    PHANTASY,  
    SCIENCE_FICTION  
}  
  
public class CoffeeSizeDemo1 {  
    public static void main(String[] args) {  
        System.out.println(Type.BELLETRE.ordinal()); //1  
    }  
}
```

# Конструкторы, поля и методы перечисления

Перечисления, как и обычные классы, могут определять конструкторы, поля и методы. Например:

```
public enum CoffeeSize4 {  
    BIG( ml: 100),  
    HUGE( ml: 150),  
    OVERWHELMING( ml: 200) {  
        @Override  
        public String getLidCode() { return "A"; }  
    };  
  
    private int ml;  
  
    CoffeeSize4(int ml) { this.ml = ml; }  
  
    public int getMl() { return ml; }  
  
    public String getLidCode() { return "B"; }  
}
```

# Методы

Если переменные и константы хранят некоторые значения, то методы содержат собой набор операторов, которые выполняют определенные действия.

Общее определение методов выглядит следующим образом:

```
[модификаторы] тип_возвращаемого_значения название_метода ([параметры]){  
    // тело метода  
}
```

Модификаторы и параметры необязательны.

# Методы

## Параметры переменной длины:

Метод может принимать параметры переменной длины одного типа. Например, нам надо передать в метод набор чисел и вычислить их сумму, но мы точно не знаем, сколько именно чисел будет передано - 3, 4, 5 или больше.

Параметры переменной длины позволяют решить эту задачу:

```
public class Application {
    public static void main(String[] args) {
        sum( ...nums: 1, 2, 3);           // 6
        sum( ...nums: 1, 2, 3, 4, 5);    // 15
        sum();                           // 0
    }

    static void sum(int ...nums){

        int result =0;
        for(int n: nums)
            result += n;
        System.out.println(result);
    }
}
```

# Методы

Методы могут возвращать некоторое значение. Для этого применяется оператор `return`.

**return** *возвращаемое\_значение;*

```
public class Application {
    public static void main(String[] args) {
        int x = sum( a: 1, b: 2, c: 3);
        int y = sum( a: 1, b: 4, c: 9);
        System.out.println(x); // 6
        System.out.println(y); // 14
    }

    static int sum(int a, int b, int c) {
        return a + b + c;
    }
}
```

No unused imports found

# Методы

После оператора `return` указывается возвращаемое значение, которое является результатом метода. Это может быть литеральное значение, значение переменной или какого-то сложного выражения. В методе в качестве типа возвращаемого значения вместо `void` используется любой другой тип. В данном случае метод `sum` возвращает значение типа `int`, поэтому этот тип указывается перед названием метода. Причем если в качестве возвращаемого типа для метода определен любой другой, отличный от `void`, то метод обязательно должен использовать оператор `return` для возвращения значения. При этом возвращаемое значение всегда должно иметь тот же тип, что значится в определении функции. И если функция возвращает значение типа `int`, то после оператора `return` стоит целочисленное значение, которое является объектом типа `int`. Как в данном случае это сумма значений параметров метода.

# Методы

Метод может использовать несколько вызовов оператора `return` для возварачения разных значений в зависимости от некоторых условий:

```
public class Application {
    public static void main(String[] args) {
        System.out.println(daytime( hour: 7)); // Good morning
        System.out.println(daytime( hour: 13)); // Good after noon
        System.out.println(daytime( hour: 18)); // Good evening
        System.out.println(daytime( hour: 2)); // Good night
    }
    static String daytime(int hour){
        if (hour >24 || hour < 0)
            return "Invalid data";
        else if(hour > 21 || hour < 6)
            return "Good night";
        else if(hour >= 15)
            return "Good evening";
        else if(hour >= 11)
            return "Good after noon";
        else
            return "Good morning";
    }
}
```



# Методы

## Выход из метода

Оператор `return` применяется для возвращения значения из метода, но и для выхода из метода. В подобном качестве оператор `return` применяется в методах, которые ничего не возвращают, то есть имеют тип `void`