Классы и объекты

Переменные и методы в классах

Переменные предназначены для того, чтобы хранить данные описываемого классом объекта. Так, если вам необходимо описать геометрическую точку у вас должны быть переменные, описывающие ее координаты. В классе это будет выглядеть следующим образом:

```
public class Point {
        int x;
        int y;
}
```

Таким образом, мы говорим, что в объектах нашего класса точка будет описываться двумя значениями: **x** и **y**.

Переменные, созданные в самом классе, называются полями или свойствами класса.

Если мы хотим, чтобы эти значения были доступны всем, перед их объявлением следует поставить слово **public**. Данное слово называют **модификатором доступа**.

```
public class Point {
        public int x;
        public int y;
}
```

Также можно сделать так, чтобы данные переменные были видны только внутри этого класса. В этом случае вместо public перед типом переменной ставится слово **private**.

Модификаторы доступа

Существует четыре основных модификатора доступа:

- **private**
- **public**
- **protected**
- **по умолчанию или package-private**

При создании классов рекомендуется, по возможности, закрывать доступ к переменным, то есть делать их private, а работу организовывать через методы.

Если переменная имеет тип доступа private, чтобы другие классы имели доступ к ее значению, делаются методы для назначения нового значения и его получения.

```
public class Point {
        private int x;
        public void setX(int x) {
```

```
        this.x = x;
    }
    public int getX() {
        return x;
    }
}
```

Методы в классах

Метод setX получает извне новое значение для x и заносит его в свойство класса. Метод getX возвращает значение x.

Для возвращения значения используется оператор **return**. После него ставится возвращаемое значение и точка с запятой.

Оператор return всегда прекращает работу метода. Если метод имеет тип результата void, после return не должно стоять возвращаемого значения, а сразу должна идти точка с запятой. Также для void метода, оператор return может отсутствовать. В этом случае метод завершается, когда выполнились все операции, которые находятся в его теле.

Создание классов и объектов

Ранее мы уже рассматривали общую структуру класса. Рассмотрим пример класса, описывающий лампочку. Наша лампа должна уметь включаться, выключаться, а также у нас должна быть возможность узнать, в каком она сейчас состоянии:

```
public class Lamp {
    private boolean isOn;
    public void on() {
        this.isOn = true;
    }
    public void off() {
        this.isOn = false;
    }
    public boolean getState() {
        return isOn;
    }
}
```

Таким образом, мы имеем свойство, хранящее состояние лампочки, и три метода позволяющие с ним работать. Методы on и off включают и выключают лампочку, а getState позволяет узнать, в каком она состоянии в данный момент.

Для того, чтобы начать использование этого класса, необходимо создать его объект. Например, это можно сделать внутри метода **main**. Сначала следует создать переменную объектного типа:

```
Lamp first;
```

Созданная переменная пока не содержит объекта. Переменные классов не содержат самих объектов, они служат для хранения адресов объектов в памяти. Изначально в переменной будет пустое значение, то есть **null**. И если вы попытаетесь у этой переменной вызвать методы класса, вы получите ошибку.

Чтобы создать сам объект, следует воспользоваться оператором **new** в таком виде:

```
first = new Lamp();
```

После этого объект будет создан и лампочку можно включить, например:

```
first.on();
```

Можно совместить объявление переменной и создание объекта:

```
Lamp first = new Lamp();
```

На один и тот же объект может ссылаться несколько переменных. Так вы можете записать:

```
Lamp first2 = first;
```

В результате у нас будет две переменных: **first** и **first2**, но они будут ссылаться на один и тот же объект.

Созданная нами лампочка будет существовать, пока есть хоть одна переменная, в которой хранится адрес этого объекта. После того, как все ссылки на объект исчезли, он будет уничтожен специальной службой, которая называется сборщиком мусора.

Конструкторы. Ссылка this

Для выполнения операций, необходимых для создания класса, в Java используются конструкторы. Конструктор – метод класса, вызываемый при создании объекта. Главные особенности объявления конструктора: он должен иметь имя, совпадающее с именем класса, и не должен иметь возвращаемого значения (даже void).

```
Модификатор_Доступа ИмяКласса() {
        ...
}
```

Например, мы хотим, чтобы каждая лампочка из предыдущего примера при создании была включена. Для этого в класс необходимо добавить метод вида:

```
public Lamp() {
        isOn = true;
```

}

В результате, когда мы будем использовать оператор **new** для создания нового объекта, будет вызываться конструктор и включать лампочку.

Конструкторы могут иметь входные параметры. В этом случае при создании объекта можно указать в скобках значения для этих параметров. Конструкторы также могут быть перегружены, то есть может быть несколько конструкторов с разными наборами параметров. Это же относится и к другим методам. Можно создать несколько вариантов методов с одинаковыми именами и разными исходными параметрами. Данный механизм называется перегрузкой метода (**overloading**). По умолчанию любой класс имеет пустой конструктор, даже если он не объявлен явно, но если мы его перегружаем, то чтобы его использовать, нам необходимо добавить его явно.

Например, нам необходимо создать класс, описывающий геометрическую точку с двумя координатами:

```
public class Point {
        private int x;
        private int y;
        public Point() {
                x = 1;
                y = 1;
        }
        public Point (int x1,int x2){
                x = x1;
                y = x2;
        }
}
```

Такую точку можно создать двумя способами:

```
Point p1 = new Point();
Point p2 = new Point(15,6);
```

В первой строке вызывается первый конструктор и создается точка с координатами 1 и 1, во втором – второй конструктор с координатами 15 и 6.

Следует учитывать, что, если объявлены только конструкторы с параметрами, создать объект, не указав параметры, не получится. То есть если в предыдущем примере пропустить описание первого конструктора, строка вида

```
Point p = new Point();
```

вызовет ошибку, так как используется конструктор без параметров.

Если в конструкторе или другом методе возникает конфликт имен, может использоваться ссылка **this**. Она представляет ссылку на объект, вызвавший данный метод. Например, второй конструктор класса Point может быть переписан в виде:

```
Point (int x, int y) {
        this.x = x;
        this.y = y;
}
```

Также **this** может использоваться для вызова из конструктора других конструкторов. Например, можно создать еще один конструктор:

```
Point (int x) {
        this(x, x);
}
```

Так как в языке Java присвоение начальных значений переменным может быть выполнено несколькими способами, следует учитывать порядок назначения начальных значений. Первыми выполняется установка начальных значений при создании полей, а последним выполняется конструктор и заданные в нем действия. То есть в классе **Point** сначала свойства **x** и **y** получают нулевые значения и только затем в конструкторе, их значения меняются.

Статические и константные члены класса. Модификатор static.

Если необходимо, чтобы все объекты этого класса пользовались одним свойством или одним методом, его следует объявить с помощью ключевого слова **static**. Например, необходимо создать переменную, хранящую общее количество объектов:

```
public static int count = 0;
```

Если вы объявили в классе такую переменную, а затем в конструктор добавили строку вида:

count++;

В результате в этой переменной будет количество созданных в вашей программе объектов данного класса.

Статические свойства создаются при запуске программы. К ним можно обращаться не только от имени переменных, но и от имени их класса. Так если мы создали переменную **count** в классе **Point**, мы можем обратиться к ней, записав:

Point.count

Статическими могут быть не только свойства, но и методы. Статические методы, также как и свойства могут вызываться без создания объекта данного класса.

В приведенном в начале примере метод **main** – статический, что позволяет автоматически вызывать его, не создавая объекта класса.

Следует также помнить, что статические методы не могут напрямую получить доступ к не статическим методам и свойствам этого класса. Так, если описать класс следующего вида:

```
public class HelloWorld {
	private int a = 5;
	private void print() {
		System.out.println(a);
	}
	public static void main(String[] args) {
		print();
	}
}
```

При компиляции данного примера будет выдана ошибка компиляции, о невозможности вызова **print** из статического метода, и программа не будет скомпилирована.

Обойти это можно, создав объект класса, и вызывая метод через объект:

```
HelloWorld h1 = new HelloWorld();
h1.print();
```

Статический метод может быть вызван не от имени объекта, а от имени класса. Если класс A имеет статический метод b, его можно вызвать как:

```
A.b();
```

Иногда возникает необходимость в выполнении достаточно сложных действий для присвоения начальных значений статическим свойствам класса. Для этого внутри класса используются специальные блоки, напоминающие методы, но вместо модификаторов и имени метода стоит слово **static**, а также отсутствуют скобки и входные параметры:

```
static {
	операторы
}
```

Если необходимо, чтобы свойство класса не могло изменить своего значения, его следует объявить с помощью ключевого слова **final**.

```
final double PI = 3.14159;
```

При попытке присвоить переменной другое значение, компилятор выдаст ошибку.

Стандартные классы и объекты Java

Ввод с консоли

Для получения ввода с консоли используют класс **Scanner**, который использует **System.in** внутри себя.

Класс **Scanner** находится в пакете java.util, необходимо импортировать:

import java.util.Scanner

Для создания самого объекта **Scanner** в его конструктор передается объект **System.in**. После этого мы можем получать вводимые значения.

Класс **Scanner** имеет следующие методы:

- **next()** – читает введенную строку до первого пробела;
- **nextLine()** – читает всю введенную строку;
- **nextInt()** – читает введенное число int;
- **nextDouble()** – читает введенное число double;
- **nextBoolean()** – читает значение boolean;
- **nextByte()** – читает введенное число byte;
- **nextFloat()** – читает введенное число float;
- **nextShort()** – читает введенное число short.

Пример:

```
public class Program {
	public static void main(String[] args) {
		Scanner in = new Scanner(System.in);
		System.out.print("Name: ");
		String name = in.nextLine();
		System.out.print("Age: ");
		int age = in.nextInt();
		System.out.print("Height: ");
		float height = in.nextFloat();
		System.out.println("Name: " + name);
		System.out.println("Age: " + age);
		System.out.println("Height: " + height);
		in.close();
	}
}
```

Математика

Для работы с математическими функциями используется класс **Math**.

Math содержит несколько математических констант. Наиболее часто используемыми являются **PI** и **E**. Таким образом, чтобы получить значение Пи, следует записать **Math.PI**

Также имеются следующие методы:

- Math.abs(x) – модуль аргумента;
- Math.ceil(x) – округление в большую строну;
- Math.cos(x) – косинус;
- Math.exp(x) – экспонента;
- Math.floor(x) – округление в меньшую сторону;
- Math.log(x) – логарифм;
- Math.max(x,y) – максимальное значение из данных аргументов;
- Math.min(x,y) – минимальное значение;
- Math.pow(x,y) – возведение в степень;
- Math.random() – случайное число от 0 до 1;
- Math.round(x) – округление к ближайшему целому;
- Math.sin(x) – синус;
- Math.sqrt(x) – квадратный корень;
- Math.tan(x) – тангенс.

Все эти методы являются статическими и вызываются через класс Math.

Пример:

x = Math.sqrt(15);

В результате в **x** будет помещен квадратный корень пятнадцати.

Класс Arrays

Класс **Arrays** содержит набор статических методов для работы с массивами. Обычно для данного класса объектов не создается. Он фактически представляет собой контейнер методов для работы с массивами. В классе имеются следующие методы

- static int binarySearch(int[] a, int key) – производит поиск в массиве указанного значения бинарным способом. Если массив не является отсортированным, то результаты поиска неопределены;
- static boolean equals(int[] a, int[] a2) – сравнивает два массива;

- static void fill(int[] a, int val) – назначает указанное значение всем элементам массива. После массива могут быть указаны еще два параметра – номер первого и номер последнего элемента, которым присваиваются значения;
- static void sort(int[] a) – сортирует массив. Также можно отсортировать фрагмент массива, указав, вторым и третьим входными параметрами номера первого и последнего элемента, сортируемой части.

Классы-обертки

Так как примитивные типы не являются объектами, в ряде случаев их использовать не получается. Чтобы обойти данное ограничение, были созданы классы-обертки для примитивных типов.

Данные классы могут использоваться не только для создания объектов, они также хранят некоторые статические методы (например, для преобразования в строки и назад). Чаще всего используются следующие классы-обертки: **Byte, Double, Float, Integer, Long, Short**. Обертки есть для всех примитивных типов данных. Следует отметить одну особенность объектов данных классов – значения в объектах не могут меняться. То есть если создан объект типа Integer со значением 5, изменить это значение не возможно. Если следует менять значение, необходимо создавать новый объект.

```
Integer intOb = new Integ(5);

intOb = new Integer(15);
```

Классы-обертки как правило могут возвращать свое значение, приведенное к требуемому типу. Так для класса Integer имеются следующие методы:

- double doubleValue() – возвращает вещественное удвоенной точности;
- float floatValue() – возвращает вещественное;
- int intValue() – возвращает целое число;
- long longValue() – возвращает длинное целое;
- short shortValue() – возвращает короткое целое.

Также для каждого типа имеется метод для перевода из строкового в числовой тип. Метод является статическим и возвращает значение примитивного типа:

```
static int parseInt(String str)
```