Курс: «Разработка программного обеспечения на Java»

Тема: Методы (на примере статических методов). Область видимости. Рекурсия

План

1. Методы
    1. Что такое метод?
    2. Необходимость использования методов
    3. Синтаксис объявления методов
    4. Использование ключевого слова void при работе с методами
    5. Вызов метода
    6. Аргументы
    7. Возврат значения из метода (return)
2. Область видимости
3. Рекурсия

1. Методы

1.1 Что такое метод?

Java является объектно-ориентированным языком, поэтому любой программный код должен определятся внутри класса. Класс в свою очередь, состоит из полей (или свойств) и методов. В качестве поля может использоваться переменные и константы примитивного типа, а также, объекты другого класса.

Методами называется последовательность инструкций, которые направлены на решение конкретной задачи. Аналогичное понятие (не в ООП языках) – функция или процедура, но согласно терминологии объектно-ориентированного программирования, называются методами. Сам по себе метод является фрагментом программы, которому присвоено уникальное имя (или идентификатор). Имя метода можно использовать для вызова этого метода из других частей программы. При вызове метода происходят действия, которые перечислены в теле метода.

Метод, который вызывается без использования конкретного объекта класса называется статическим.

1.2 Необходимость использования методов

В процессе развития компьютерных систем, усложнялись и задачи, поставленные перед разработчиками. ООП языки стали прекрасной заменой процедурным языкам, потому что позволяют программисту создавать классы, которые представляют собой объекты из реального мира. Свойства этих объектов описываются полями, а логика и поведение объектов задается методами класса. Как правило, вызов метода ориентирован на изменения свойств объекта, но статические методы позволяют выполнять общую логику, работать с данными, не привязанными к конкретному объекту. Таким образом, программа представляет собой набор объектов, которые взаимодействуют между собой при помощи методов. С помощью статических методов можно определять логику, которая не привязана к конкретному объекту, а является общей для них.

1.3 Синтаксис объявления методов

```
<модификатор доступа> <static> <тип возвращаемого значения> <имя
метода>(<аргументы>) {
    <тело метода>
}
```

- **модификатор доступа** определяет доступность данного метода для вызова в разных частях программы

- ключевое слово **static** означает то, что метод принадлежит всему классу, в котором он объявлен, а не конкретному экземпляру этого класса.
- **тип возвращаемого значения** определяет тип данных (как примитивные, так и объекты классов), которые будут возвращены в результате выполнения метода.
- **имя метода** должно быть уникальное в рамках класса, в котором объявлен метод. Имя метода не может начинаться с цифр, и не может называться другим ключевым словом, зарезервированным в языке Java. По правилам именований, имя метода должно начинаться с маленькой буквы, а если имя класса состоит из нескольких слов, каждое последующее слово должно писаться слитно и с большой буквы.
- **аргументы** метода задают его входные данные, которые влияют на результат работы метода или на его свойства. Аргументы могут отсутствовать.
- **тело метода** обязательно заключается в пустые скобки, может быть пустое.

1.4 Использование ключевого слова void при работе с методами

Ключевое слово **void** используется как тип возвращаемого значения в тех методах, которые не предполагают возвращать какой-либо результат выполнения.

1.5 Вызов метода

Рассмотрим вызов метода на примере:

```
public class Main {
   static void hello() {
     System.out.println("Hello world!")
   }

   public static void main(String[] args){
     hello(); // вызов метода
   }
}
```

Метод main(String[] args) является точкой входа в программу. Внутри этого метода можно написать логику программы, но лучше разбить ее на несколько методов, которые будут решать определенные задачи приложения, а в методе main будет происходить только вызовы этих методов.

В нашем случае произошел вызов метода hello(), который выведет в консоль строку «Hello world!». Т.к. метод main и метод hello находятся в одном классе,

можно вызвать метод без указания класса, в котором он находится. Рассмотрим пример, когда вызываемый метод объявлен в другом классе:

```
public class Main {
   public static void main(String[] args){
     A.hello(); // вызов метода
   }
}

class A {
   static void hello() {
     System.out.println("Hello world!");
   }
}
```

В таком случае, обязательно следует указать класс, в котором необходимо вызвать указанный метод, иначе произойдет ошибка. Давайте рассмотрим почему это необходимо. Преимущество объектно-ориентированного подхода в том, что пространство имен определяется классом, т.е. множество классов могут состоять из методов, которые имеют аналогичное название, но выполняют разную функцию.

Рассмотрим пример:

```
public class Main {
   public static void main(String[] args){
     A.hello(); // вызов метода в классе A
     B.hello(); // вызов метода в классе B
   }
}

class A {
   static void hello() {
     System.out.println("Привет мир!");
   }
}

class B {
   static void hello() {
     System.out.println("Привет, ШАГ!");
   }
}
```

1.6 Аргументы

Существуют формальные и фактические аргументы (параметры). Формальные аргументы описывают входные данные метода внутри скобок при объявлении метода. Каждый аргумент должен быть описан в виде: тип и имя. Аргументами может быть переменная примитивного типа, массив или объект класса. Фактические аргументы передаются в круглых скобках при вызове метода. При вызове не указывается тип, только конкретное значение параметра.

Рассмотрим работу с аргументами на примере:

```
public class Main {
   static void hello(String name) {
      System.out.println("Привет, " + name);
   }

   static void main(String[] args){
      hello("Андрей");
   }
}
```

Метод hello(String name) заранее не знает с какими данными он будет работать, но он определяет тип и имя параметра, который следует передать при вызове метода.

1.7 Возврат значения из метода (return)

Если метод предполагает возвращать результат выполнения, необходимо указать соответствующий тип возвращаемого значения при объявлении метода, а затем, при помощи оператора return возвратить результирующее выражение.

Рассмотрим пример:

```
public class Main {

  static int [] createRandomArray(int count, int max) {
      int [] array = new int [count];
      Random rand = new Random();
      for(int i = 0; i < count; i++)
         array[i] = rand.nextInt(max);
      return array;
  }
```

```
    static int arraySum(int [] array) {
      int sum = 0;
      for(int i = 0; i < array.length; i++)
         sum += array[i];
      return sum;
    }

    static void printArray(int [] array) {
      for(int i = 0; i < array.length; i++)
         System.out.print(array[i] + " ");
      System.out.println();
    }

    public static void main(String[] args){
        int [] array = createRandomArray(10, 100);

        printArray(array);
        System.out.println("Сумма элементов массива: " + arraySum(array));
    }
}
```

2. Область видимости

Область видимости переменной определяется либо на уровне класса (аналог глобальных переменных в других языках) или блоком кода (тело метода или оператора).

Для полей и методов, которые вынесены на уровень класса действуют модификаторы доступа. Их всего четыре:

- default (без ключевого слова)
- public
- private
- protected

Доступ по умолчанию ограничен пакетом, в котором объявлен класс. К примеру, проект состоит из пакетов a и b. В пакете a есть три класса: A1, A2, A3. В классе A1 определен метод с доступом по умолчанию. Данный метод будет доступен для вызова из всех классов пакета a. В классе A1 его можно вызывать без указания имени класса, в классах A2 и A3 перед вызовом необходимо указать класс A1. В пакете b ни один из классов не сможет обратиться к данному методу.

Модификатор доступа public расширяет область видимости метода на все пакеты, которые включены в проект.

Если вы хотите скрыть метод, используйте private доступ. С таким ключевым словом, доступ к методу будет иметь только тот класс, в котором он объявлен.

Protected отличается от private лишь тем, что данный метод может быть доступен еще и в классах наследниках.

Для локальных переменных область видимости задается фигурными скобками. Пример: тело метода, цикла, оператора ветвлений. Также, блок кода можно определить в любой точке кода. Переменная, объявленная в таком блоке кода, будет доступна только от места объявления до конца фигурной скобки.

Если имя локальной переменной совпадает с именем поля класса, то при обращении в блоке кода, будет использоваться локальная переменная. Для того, чтобы из любого места обратиться к полю или методу класса используйте ключевое слово this.

3. Рекурсия

Рекурсией называется метод, который вызывает сам себя (вызов происходит внутри тела метода). Это актуально при решении задач, для которых необходимо повторять одни и те действия, но с разными данными. Количество рекурсивных вызовов должно быть конечно и минимально, потому что рекурсия может существенно замедлять работу программы, т.к. все методы ожидают завершения вызываемых внутри себя методов, при этом хранят данные в памяти. Большинство задач на рекурсию можно заменить работой с циклами.

Пример рекурсии - вычисление факториала числа n:

```
static int fact (int n) {
  if (n==1) {
    return 1;
  } else if (n==2) {
    return 2;
  } else {
    return fact(n-1) * n;
  }
}
```