Программирование
Тема 4.1Java.Введение в классы и объекты. Передача параметров в методы

Классы и объекты

- **Описание класса:**

```
[ Модификатор доступа ] class ИмяКласса [extends Object]
    {   // члены класса – поля и методы
        Модификатор доступа Тип имяПоля  [  = значение ];
        Модификатор доступа Тип имяМетода (Параметры)  {
            // тело метода    }
    }
```

- **Модификаторы доступа:**
  - **public - общедоступный;**
  - **protected - защищенный;**
  - **private - закрытый;**
- **Если модификаторы доступа не указаны, то по умолчанию используется пакетный уровень доступа**

Классы и объекты

- Класс – это конструкция языка программирования Java, которая объединяет в себе переменные и функции (методы), которая может определять новый тип данных, который можно использовать для создания объектов этого типа.
- В отличие от C++, в Java объявление класса совмещено с его определением, то есть методы класса нужно реализовывать (писать код тела) сразу же при их описании. Как правило, в Java класс описывается в отдельном файле.
- Модификаторы доступа - это ключевые слова, которые определяют доступность класса или его членов.
- В языке Java предусмотрены следующие модификаторы доступа: public, protected, private.
- Для членов класса (полей и методов) разрешено использование всех указанных выше модификаторов.
- Поля и методы класса, помеченные как public, доступны через объект по имени из классов текущего пакета и из классов других пакетов (если класс, содержащий public поля и методы, имеет уровень доступа public).
- Поля и методы класса, помеченные как protected, доступны по имени только из методов своего класса и классов потомков, даже если классы потомки находятся в другом пакете. Также protected поля и protected методы доступны через объект по имени для любого класса текущего пакета.
- Private-поля и private методы доступны по имени только из методов своего класса.
- Если не указан ни один из модификаторов доступа, то считается что элементы класса (поля и методы) имеют пакетный уровень доступа, то есть доступны по имени только из методов своего класса и классов потомков текущего пакета, а также доступны через объект по имени для любого класса текущего пакета
- Для классов верхнего уровня, т.е. не вложенных (внутренних), применимы только общедоступный (public) и пакетный уровни доступа. Если класс объявлен как public, то к нему можно получить доступ отовсюду. Если же модификатор доступа не указан, то класс имеет пакетную область видимости и доступ к нему имеют только классы из того же пакета.

Классы и объекты

- **Создание объектов:**

  ИмяКласса имяОбъекта; // объявление ссылки на объект

  имяОбъекта = new ИмяКласса([параметры конструктора]);

- **Доступ к полям и методам объекта:**

  имяОбъекта.имяПоля = значение;
  ИмяОбъекта.имяМетода(параметры);

- **Варианты конструкторов:**
  - **Конструкторы с параметрами;**
  - **Конструкторы без параметров (конструкторы по умолчанию).**

- **Специальная ссылка this**

Классы и объекты

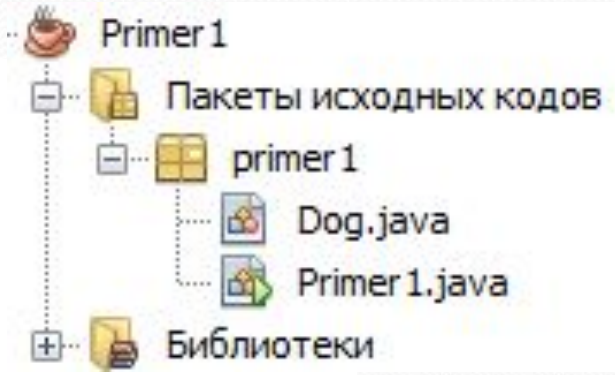
- Объект в Java и других языках программирования – это переменная типа класс.
- Механизм создания объектов:
  1. Создается ссылочная переменная в стеке для хранения адреса будущего объекта;
  2. В динамической памяти (куче) выделяется пространство для размещения объекта со всеми его полями;
  3. Поля объекта инициализируются значениями по умолчанию;
  4. Выполняется явная инициализация полей объекта, если она была задана программистом;
  5. Выполняется конструктор;
  6. Ссылка на созданный объект (его адрес) записывается в соответствующую ссылочную переменную.
- Использование объекта осуществляется посредством доступа к его элементам (полям и методам). Для доступа к элементам объекта вне класса, которому принадлежит объект используется операция «точка» после переменной-ссылки на объект, если конечно этот доступ разрешен модификаторами доступа. Если элементы объекта имеют модификатор доступа private, то доступ к ним осуществляется через public-методы.
- Поля объекта – это переменные внутри класса. Совокупность значений полей объекта описывает состояние объекта. Поля объекта, в отличие от локальных переменных, инициализируются значениями по умолчанию: числовые элементы – нулями; символьные – значениями ‘\0’ (нулевой символ); логические – значениями false; ссылки на объекты – значениями null. Поля объекта могут быть инициализированы явным образом при объявлении.
- Метод – это функция, описанная внутри класса. *Замечание*. Функция – это именованный фрагмент кода, к которому можно обращаться через имя из других частей программы.
- Описание метода включает заголовок и тело. Тело – совокупность операторов. Метод может принимать параметры и возвращать значение

Классы и объекты

- Конструктор класса – это специальный метод, название которого совпадает с именем класса. Конструктор вызывается автоматически при создании объекта. Конструктор не может возвращать значение. Каждый класс обязан иметь конструктор. Если в классе никакого конструктора явно не написано, то автоматически создается конструктор без параметров, который называется конструктором по умолчанию. Если в классе явно описан какой-либо конструктор (конструктор с параметрами или конструктор без параметров), то конструктор по умолчанию не создается. Конструкторы также, как и другие методы, могут иметь модификаторы доступа. Один класс может иметь несколько конструкторов с разными параметрами (перегрузка (overload) конструктора).
- С помощью конструкторов можно управлять процессом создания объекта и производить какие-то действия, обычно связанные с инициализацией объекта: присвоение полям объекта значений по умолчанию; соединение с БД; соединение с сетью; создание других объектов и др.
- В Java нет деструкторов класса. Уничтожение неиспользуемых объектов осуществляется автоматически «сборщиком мусора», специальным механизмом JVM. Объект удаляется, когда в последующей программе на него нет ни одного обращения.
- Ссылка this – это скрытый параметр метода, который хранит ссылку на текущий объект, который вызывает метод.

Классы и объекты

- **Пример 1:**

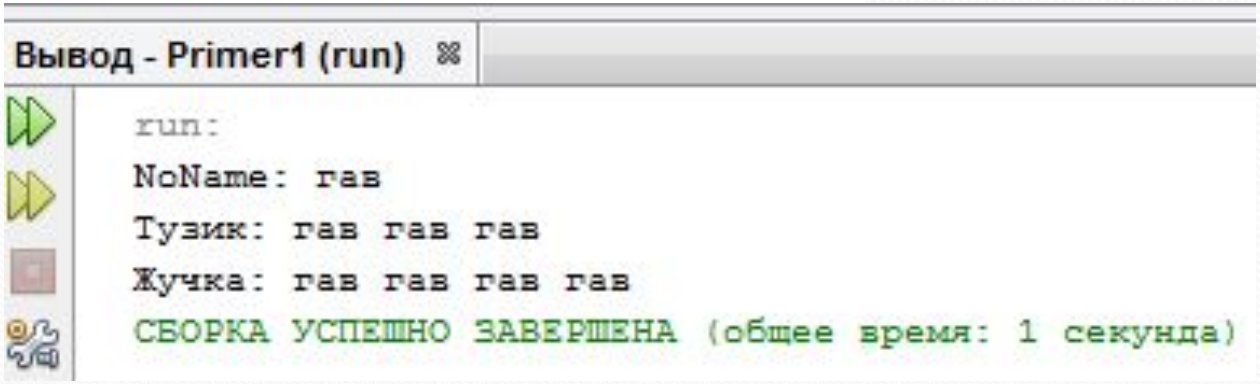



```
package primer1;
class Dog
  {  private String name; // кличка
     private int age;       // возраст
     String getName() {return  name;}
     int getAge() { return  age; }
     void setName (String newName){name = newName;}
     void  setAge(int  newAge) { age = newAge;}
     Dog(){ age = 1; name = "NoName";};
     Dog(String name, int age)
      { this.name = name; this.age = age; }
     void voice()
     { System.out.print(name + ": ");
       for (int i = 1; i <= age; i++)
            System.out.print("гав ");
        System.out.println();
     }
}
```

Классы и объекты

- **Пример 1:**

```
package primer1;
public class Primer1 {
  public static void main (String args [])
    { Dog dog1 = new Dog();
      dog1.voice();
      Dog dog2 = new Dog("Тузик", 3);
      dog2.voice();
      dog1.setName("Жучка");
      dog1.setAge(4);
      dog1.voice();
    }
}
```



Статические поля и методы

- **Статические поля**

```
public class Student
{
      private String name;
      private int course;
      private static int id = 1;
}

   …
 Student.id = 5;

public class Math
{
  public static final double PI = 3.14159265358979323846;
}
```

Статические поля и методы

- Поле, имеющее модификатор static, существует в одном экземпляре. Если же поле не является статическим, то каждый объект имеет его копию.
- Например, добавим в класс Student статическое поле id. Теперь каждый объект класса Student имеет копии полей name и course. А статическое поле id существует в единственном экземпляре и доступно всем экземплярам класса Student (только им, поскольку поле объявлено как private). Даже если объектов класса Student нет вообще, статическое поле id существует. Оно принадлежит классу, а не конкретному объекту. Чтобы обратиться к этому полю нужно указать имя класса (а не объекта).
- Если какой-либо объект класса Student модифицирует значение статического поля, то новое значение получат и другие объекты класса Student.
- Статические переменные используются довольно редко. В то же время *статические константы* используются гораздо чаще. Например, класс Math имеет статическую константу PI. Обратиться к этой константе в программе можно с помощью выражения Math.PI. Если бы ключевое слово static было пропущено, константа PI была бы обычным константным полем экземпляра класса Math. Это значит, что для доступа к такой константе нужно было бы создать объект класса Math, причем каждый подобной объект имел бы свою копию константы PI.
- Как уже упоминалось выше, применять открытые поля не следует никогда, поскольку любой сможет изменить их значения без использования методов объекта. Однако открытые константы (т. е. поля, объявленные с ключевым словом final) можно и нужно использовать.

Статические поля и методы

- **Статические методы**

```
    double x;
    double y;
    x = Math.pow(x, y);
…
private static int id = 1;

      public static int getId()
      {
            return id; // Возвращает статическое поле id
      }
…
int n = Student.getId(); // Вызов статического метода
```

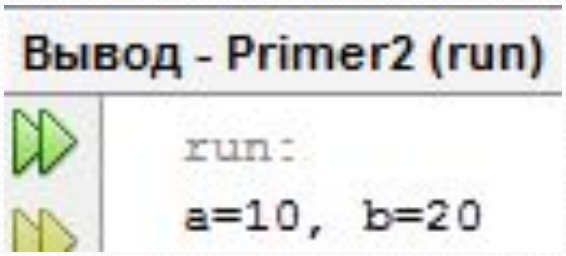
Статические поля и методы

- Для работы со статическими методами не нужно создавать объектов. Например, метод pow из класса Math – статический. Этот метод не использует ни одного объекта класса Math. Другими словами, он не имеет неявного параметра this. Следовательно, статические методы – это методы, не имеющие параметра this.
- В принципе, можно создать объект и вызвать его статический метод, однако это не дает никаких преимуществ, поскольку статический метод все равно не может получить доступ к нестатическим полям объекта. *Статические методы имеют доступ только к статическим полям*.
- Чтобы вызвать статический метод, нужно указать имя класса.
- Хотя для вызова статического метода можно использовать и объекты, тем не менее, в этом случае рекомендуется использовать только имя класса, а не объекта.
- Статические методы следует применять в двух ситуациях:
  - Когда методу не нужен доступ к состоянию объекта, поскольку все необходимые параметры задаются явно (например, в методе Math.pow);
  - Когда методу нужен доступ лишь к статическим полям класса (например, метод Student.getId).
- Термин "статический" – исторический курьез. Сначала ключевое слово static было введено в языке Си для обозначения локальных переменных, которые не уничтожались при выходе из блока. В этом контексте слово "статический" имеет смысл: переменная продолжает существовать после выхода из блока, а также при повторном входе в него.
- Затем слово "статический" в языке Си приобрело второе значение – глобальные переменные и функции, к которым нельзя получить доступ из других файлов. Ключевое слово static было просто использовано повторно, чтобы не вводить новое.
- В языке Си++ это ключевое слово было использовано в третий раз, получив совершенно новую интерпретацию, для обозначения переменных и функций, принадлежащих классу, но не принадлежащих ни одному объекту этого класса. Именно это значение ключевое слово static имеет и в языке Java.

Передача параметров в методы

- **Пример 2.**
  - **Вариант 1**

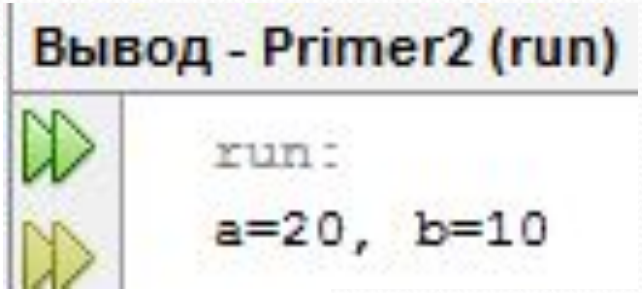
```
package primer2;
class Primer2
{ static void swap(int i1, int i2)
        {int c = i1; i1= i2; i2= c;}
         public static void main(String[] args)
         { int a = 10;int b = 20;
             swap(a, b);
             System.out.println("a="+a+", b="+b);
         }
}
```



Передача параметров в методы

- **Пример 2.**
  - **Вариант 2**

```
package primer2;
class MyInteger
{ int i;
  MyInteger(int i) { this.i= i; }
 }
class Primer2
{ static void swap(MyInteger i1, MyInteger i2)
  {int c = i1.i;  i1.i = i2.i; i2.i = c; }
  public static void main(String[] args)
  { MyInteger a = new MyInteger(10);
    MyInteger b = new MyInteger(20);
     swap(a, b);
     System.out.println("a="+a.i+", b="+b.i);
    }
}
```



Передача параметров в методы

- Метод – это функция, описанная внутри класса. Описание метода включает заголовок и тело. Тело – совокупность операторов. Метод может принимать параметры и возвращать значение с помощью оператора return. Если метод возвращает значение простого типа или ссылку на объект, то тип возвращаемого значения должен быть указан в заголовке метода. Если метод не возвращает значение, то тип возвращаемого значения в заголовке метода должен быть void.
- Если в заголовке метода описаны формальные параметры, то при его вызове нужно указать фактические параметры. В языке Java при вызове метода фактические параметры передаются в формальные по значению, то есть значения фактических параметров копируются в соответствующие формальные параметры. Изменение формального параметра не влияет на значение фактического параметра. При передаче ссылок на объекты копируется ссылка (адрес объекта). После такого копирования, и фактический параметр, и формальный параметр, ссылаются на один и тот же объект.
- В примере 2 (вариант 1) не удалось обменять значения переменных a и b, так как в метод swap эти переменные передаются по значению, то есть метод swap работает с копиями значений переменных a и b.
- Чтобы обменять значения переменных a и b необходимо создать дополнительный класс MyInteger, полем которого является переменная типа int (создать класс оболочку для переменной типа int) – пример 2 (вариант 2).

Передача параметров в методы

- **Пример 3.** Массив в качестве параметра и возвращаемого значения

```
package primer2;
import java.util.Scanner;
class Primer2 {
 static  int[] input()
  {int n; // количество спортсменов
    Scanner in = new Scanner(System.in);
    System.out.print("Укажите колличество стортсменов: ");
    n = in.nextInt();
    int[] a = new int [n];
    System.out.print("Введите время заплыва каждого из ");
    System.out.println(n + " спортсменов (в секундах):");
    for (int i = 0; i < n; i++)
      a[i] = in.nextInt();
    return a;
  }
```

Передача параметров в методы

- **Пример 3.** Массив в качестве параметра и возвращаемого значения

```
static  void minim(int[] a)
 {int min = a[0];
   for (int i = 0; i < a.length; i++)
     if (a[i] < min)
         min = a[i];
    System.out.print("Лучшее время заплыва (в секундах): "+ min+ "\n");
 }

 public static void main(String[] args) {
   int[] a = input();
   minim(a);
   }
 }
```

Вывод - Primer2 (run)

```
run:
Укажите колличество стортсменов: 5
Введите время заплыва каждого из 5 спортсменов (в секундах):
5 8 2 3 4
Лучшее время заплыва (в секундах): 2
```

Передача параметров в методы

- **Пример 4.** Возврат двух значений из метода

```
package primer3;
import java.util.*;

class Out
{ int num;
}
public class Primer3 {
static int maxValue(int []m, Out numValues) {
  int result = Integer.MIN_VALUE;
  for (int i=0; i<m.length; i++) {
    if (m[i] == result) {
      numValues.num ++;
    }
    if (m[i]>result) {
      result = m[i];
      numValues.num = 1;
    }
  }
  return result;
}
```

Передача параметров в методы

- **Пример 4.** Возврат двух значений из метода

```
public static void main(String[] args) {
    //int [] numValues = new int[1];
      Out numValues = new Out();
    int []m = new int[] {1,5,3,5,2};
    int result = maxValue(m, numValues);
    System.out.println("Maximum value is "+result+
            ". Found "+numValues.num+" times");
    }
}
```

Вывод - Primer3 (run)

```
run:
Maximum value is 5. Found 2 times
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 0 секунд)
```

Передача параметров в методы

- **Пример 5.** Возврат двух значений из метода

```
package primer3;
import java.util.*;

public class Primer3 {
public static int maxValue(int []m, int []numValues) {
  int result = Integer.MIN_VALUE;
  for (int i=0; i<m.length; i++) {
    if (m[i] == result) {
      numValues[0] ++;
    }
    if (m[i]>result) {
      result = m[i];
      numValues[0] = 1;
    }
  }
  return result;
}
```

Передача параметров в методы

- **Пример 5.** Возврат двух значений из метода

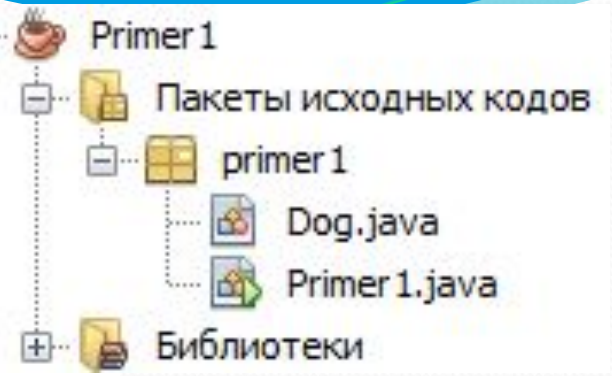
```
public static void main(String[] args) {
    int [] numValues = new int[1];
    int []m = new int[] {1,5,3,5,2};
    int result = maxValue(m, numValues);
    System.out.println("Maximum value is "+result+
            ". Found "+numValues[0]+" times");
    }
}
```

Вывод - Primer3 (run)

```
run:
Maximum value is 5. Found 2 times
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 0 секунд)
```

Массивы объектов. Интерфейс Comparable

- **Пример 6.** Сортировка массива объектов




```
package primer1;
class Dog implements Comparable
  {  String name; // кличка
     int age;       // возраст
     Dog(String name, int age)
      { this.name = name; this.age = age; }
     void voice()
     { //System.out.print(name + ": ");
       for (int i = 1; i <= age; i++)
           System.out.print("гав ");
        System.out.println();
     }
  @Override
     public int compareTo(Object obj)
     {  Dog tmp = (Dog)obj;
        // return name.compareTo(tmp.name);
        if (tmp.age < age) return 1;
          else if (tmp.age > age) return -1;
                  else return 0;
     }
}
```

Массивы объектов. Интерфейс Comparable

- **Пример 6.** Сортировка массива объектов

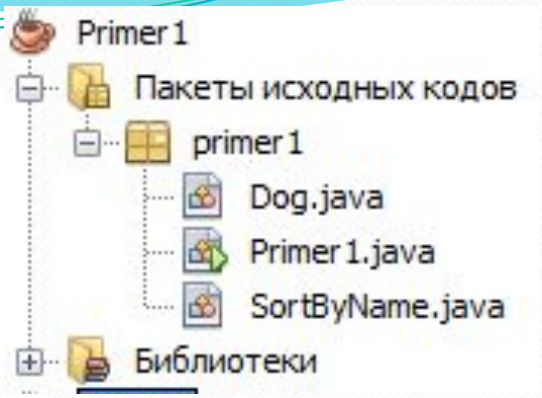
```
package primer1;
import java.util.Arrays;
public class Primer1 {
  public static void main (String args [])
    { Dog[] dogs = new Dog[3];
      dogs[0] = new Dog("Шарик",3);
      dogs[1] = new Dog("Тузик",2);
      dogs[2] = new Dog("Жучка",5);
      Arrays.sort(dogs);
      for(int i = 0; i < dogs.length; i++)
      { System.out.print(dogs[i].name +
                  " - возраст: ");
        dogs[i].voice();
      }
    }
}
```

```
run:
Тузик - возраст: гав гав
Шарик - возраст: гав гав гав
Жучка - возраст: гав гав гав гав гав
```

```
run:
Жучка - возраст: гав гав гав гав гав
Тузик - возраст: гав гав
Шарик - возраст: гав гав гав
```

Массивы объектов. Интерфейс Comparator

- **Пример 7.** Сортировка массива объектов




```
package primer1;
class Dog implements Comparable <Dog>
   {   String name; // кличка
       int age;       // возраст
       Dog(String name, int age)
        { this.name = name; this.age = age; }
       void voice()
       {   for (int i = 1; i <= age; i++)
                System.out.print("гав ");
            System.out.println();
       }
   @Override
       public int compareTo(Dog obj)
       { if (obj.age < age) return 1;
            else if (obj.age > age) return -1;
                      else return 0;
       }
}
```

Массивы объектов. Интерфейс Comparator

- **Пример 7.** Сортировка массива объектов

```
package primer1;
import java.util.Arrays;
public class Primer1 {
  public static void main (String args [])
    { Dog[] dogs = new Dog[3];
      dogs[0] = new Dog("Шарик",3);
      dogs[1] = new Dog("Тузик",2);
      dogs[2] = new Dog("Жучка",5);
      Arrays.sort(dogs);
      for(int i = 0; i < dogs.length; i++)
      { System.out.print(dogs[i].name +
                  " - возраст: ");
        dogs[i].voice();
      }
       SortByName sortName = new SortByName();
       Arrays.sort(dogs,sortName);
       System.out.println();
      for(int i = 0; i < dogs.length; i++)
      { System.out.print(dogs[i].name +
                  " - возраст: ");
        dogs[i].voice();
      }
    }
}
```

Массивы объектов. Интерфейс Comparator

- **Пример 7.** Сортировка массива объектов

```
package primer1;
import java.util.Comparator;
class SortByName implements Comparator <Dog>
{@Override
  public int compare(Dog obj1, Dog obj2)
      {   return obj1.name.compareTo(obj2.name);
      }
}
```

```
run:
Тузик - возраст: гав гав
Шарик - возраст: гав гав гав
Жучка - возраст: гав гав гав гав гав

Жучка - возраст: гав гав гав гав гав
Тузик - возраст: гав гав
Шарик - возраст: гав гав гав
```

Контрольные вопросы

1. Понятие класса. Синтаксис описания класса в Java. Модификаторы доступа: характеристика и примеры использования.
2. Понятие объекта. Создание и уничтожение объектов в Java. Доступ к полям и методам объекта в Java. Конструкторы: назначение и типы. Примеры.
3. Передача параметров простых и ссылочных типов в методы. Примеры.
4. Статические поля и методы: назначение и примеры использования.
5. Массивы объектов. Сортировка массива объектов. Примеры.