



Контрольная работа – 2 семестр

Цель работы ознакомиться с языком Java для написания простых программ.

Для решения поставленной цели, необходимо решить следующие задачи:

- ознакомиться с материалом;
- ответить на вопросы;
- выполнить задание.

Реализация принципов объектно-ориентированного подхода в Java

Описатели ограничения доступа

Такие элементы языка Java как *класс*, *поле* и *метод* имеют *ограничения доступа*. Т.е. для них можно указать, где они будут доступны, а где нет.

Для описания ограничений доступа используются ключевые слова **public**, **private**, **protected**. Они являются опциональными описателями и дают нам три варианта ограничений доступа плюс четвертый вариант, если не указан не один из этих описателей.

public - означает, что данный элемент доступен без каких-либо ограничений;

private - доступ разрешен только из данного класса; **protected** - доступ разрешен из данного класса и из всех классов-потомков(это связано с наследованием классов, которое мы будем рассматривать позже), а также из всех классов данного пакета(что такое пакеты мы также будем рассматривать позже).

без описателя - доступ разрешен из всех классов данного пакета.

Для классов применим только один описатель - **public**. Кроме того, классы могут не иметь никакого описателя ограничения доступа.

Как мы уже рассматривали на 1-м занятии, каждый java-файл должен содержать **public**-класс с именем этого файла и наоборот, каждый **public**-класс должен быть в java-файле с именем этого класса. Классы без описателя **public** могут быть в любом java-файле.

Для полей и методов применимы все 4 варианта ограничения доступа.

Мы, наконец, дошли до некоторой "критической массы" и можем написать несколько примеров программ так, что не все в этих программах будет непонятно. Отдельные элементы этих примеров, которые выходят за рамки рассмотренного материала, мы рассмотрим с содержательной стороны, т.е. разберемся, что они делают, не вдаваясь в подробности, что за понятия в них использованы.

Один из таких элементов - это вывод результатов. Он выглядит так

```
System.out.println( < то, что нужно напечатать > );
```

В скобках можно задать текст или переменную или выражение. Подробнее мы рассмотрим это на конкретных примерах.

Пример (файл XUser.java)

```
// Это простой демонстрационный пример
```

```
// Два класса X и XUser взаимодействуют друг с другом
```

```
class X {  
  
public int x = 0; // В классе X одно поле    void f(int y) {  
  
    x = y;  
  
}  
  
}  
  
public class XUser {    void f() {    int n = 6; // Переменная n
```

```

X x1 = new X();   X x2 = new X();   x1.x = 5;   x1.f👉 ;

x2.f(2);

System.out.println("Поле x из x1: " + x1.x);

System.out.println("Поле x из x2: " + x2.x);

}

public static void main(String args[]) {   XUser ref = new XUser();   ref.f(); }

```

Трансляция

```
j.bat XUser.java
```

Запуск

```
jr.bat XUser
```

Результат работы

```
Поле x из x1: 6
```

```
Поле x из x2: 2
```

Для того, чтобы разобраться в примере достаточно проследить всю цепочку его выполнения шаг за шагом, начиная от метода main. В заключение обратим

внимание на строки

```
System.out.println("Поле x из x1: " + x1.x);
```

```
System.out.println("Поле x из x2: " + x2.x);
```

Параметром в данной конструкции является строка. Но для строк, как мы рассмотрим позднее, определена операция сложения строк друг с другом, с целыми и вещественными числами и даже с объектами.

Возврат значения в методе класса

Для возврата значения из метода класса используется оператор return вида return < выражение>;

Например,

```

int f(int a, int b) {

    return a+b; // возвращает сумму двух аргументов данного метода

}

```

Как мы уже рассматривали, тип возвращаемого значения указывается в заголовке метода перед его именем. Тип выражения в операторе return должен соответствовать типу возвращаемого значения.

Ключевое слово this

Разберемся подробнее с вызовом метода. Мы рассматривали, что вызов метода всегда производится для некоторого объекта. При этом внутри метода можно обращаться к полям класса, не указывая объект.

Это реализуется путем передачи скрытого параметра - ссылки на тот объект, для которого вызван данный метод. Всегда, когда мы пишем обращение к полю или методу того же класса, что и сам данный метод, неявно подставляется этот скрытый параметр. В некоторых случаях требуется обратиться к этому параметру явно. Такая возможность предусмотрена. Для этого в языке определено ключевое слово **this**.

Например,

```

class A {   private int size = 0;

    public void setSize(int size) {   this.size = size;   }   ...

}

```

Этот пример демонстрирует, как можно реализовать установку значения private-поля класса, т.е. поля недоступного извне класса. Для этого в классе реализован открытый (public) метод setSize(...).

В этом методе имя параметра совпадает с именем поля класса

(size). Такое в Java разрешено, более того, это является общепринятым стилем. Но совпадение имен заставляет нас писать слово `this` с точкой перед именем поля (внутри метода имя `size` обозначает параметр, а не поле).

Мы рассмотрели базовые возможности языка, реализующие концепцию объектно-ориентированного программирования. Естественно, что это только "вершина айсберга".

Статические поля и методы класса

Мы уже встречались в программах с описателем **static** в конструкции `"public static void main(String args[])"`. Пора разобраться с тем, что он означает. В Java есть статические поля и статические методы. Для указания того, что поле или метод являются статическими, используется описатель **static** перед именем типа поля или метода. Например,

```
class SomeClass {    static int t = 0; // статическое поле    ...

    public static void f() { // статический метод        ...

    }

}
```

Для поля описатель `static` означает, что такое поле создается в единственном экземпляре вне зависимости от количества объектов данного класса. Статическое поле существует даже в том случае, если не создано ни одного экземпляра класса. Статические поля класса размещаются VM Java отдельно от объектов класса в некоторой области памяти в момент первого обращения к данному классу. Рассмотрим это на примере (файл `Proba2.java`)

// Демонстрация статических полей класса

```
public class Proba2 {

    int a = 10;        // обычное поле    static int cnt = 0; // статическое поле

    public void print() {

        System.out.println("cnt=" + cnt);

        System.out.println("a=" + a);

    }

    public static void main(String args[]) {

        Proba2 obj1 = new Proba2();

        cnt++;        // увеличим cnt на 1

        obj1.print();

        Proba2 obj2 = new Proba2();

        cnt++;        // увеличим cnt на 1

        obj2.a = 0;    obj1.print();    obj2.print();

    }

}
```

В данном примере поле **cnt** является статическим. При печати обоих объектов в конце метода **main(...)** будет отпечатано одно и то же значение поля **cnt**. Это объясняется тем, что оно присутствует в одном экземпляре.

По аналогии со статическими полями, статические методы не привязаны к конкретному объекту класса. При вызове статического метода перед ним можно указать не ссылку, а имя класса. Например,

```
class SomeClass {    static int t = 0; // статическое поле    ...

    public static void f() { // статический метод        ...

    }

}

SomeClass.f(); ...
```

Поскольку статический метод не связан с каким либо объектом, внутри такого метода нельзя обращаться к нестатическим полям класса без указания объекта перед именем поля. К статическим полям класса такой метод может обращаться свободно.

Это легко продемонстрировать на примере Proba2.java. Попробуем описать метод print(...) как статический. При трансляции измененной программы мы получим сообщение об ошибке в строке

```
System.out.println("a=" + a);
```

Действительно, здесь идет обращение к нестатическому полю **a** и, поскольку метод print(...) является статическим, то неизвестно к какому объекту относится это поле.

Если мы хотим сделать метод print(...) статическим (что, в общемто, не соответствует общим принципам построения объектноориентированных программ), то мы должны определить в нем параметр типа Proba2 и использовать его для доступа к полю **a**.

Перепишем этот пример (файл Proba3.java)

// Демонстрация статических полей и методов класса

```
public class Proba3 {  
  
    int a = 10;      // обычное поле  
  
    static int cnt = 0; // статическое поле  
  
    public static void print(Proba3 obj) {  
  
        System.out.println("cnt=" + cnt);  
  
        System.out.println("a=" + obj.a);  
  
    }  
  
    public static void main(String args[]) {  
  
        Proba3 obj1 = new Proba3();  
  
        cnt++;      // увеличим cnt на 1  
  
        Proba3.print(obj1);  Proba3 obj2 = new Proba3();  
  
        cnt++;      // увеличим cnt на 1  
  
        obj2.a = 0;  print(obj1);  print(obj2);  
  
    }  
  
}
```

Обратим внимание на то, что при первом вызове print(...) перед ним указано имя класса, а в двух других случаях вызова print(...) вызывается без каких-либо уточнителей. В действительности и при первом вызове мы могли не ставить префикс "Proba3.", поскольку мы вызываем статический метод класса из другого метода класса, а при этом указывать имя объекта или класса не обязательно.

Статические методы используются достаточно часто. В составе стандартной библиотеки Java есть целые классы, в которых все методы статические (Math, Arrays, Collections).

Ответить на вопросы

[← Лекции - 2 семестр](#)

Перейти на...

[Идентификация личности ►](#)