Лексика, семантика и основные управляющие конструкции языка Java

(продолжение)

Типы данных, операции

- К простым типам данных относятся логический тип данных *boolean*, и числовые типы данных.

- Числовые типы данных представляют собой:

-целочисленные типы данных -*byte, short, int, long, char.*

-типы данных с плавающей точкой - *float, double.*

- К ссылочным типам данных относятся классы, интерфейсы и массивы.

Операции над целыми типами

- Арифметические операции и операции сравнения производятся аналогично соответствующим операциям в C/C++.
- Конструкторы, методы и константы для работы с целочисленными значениями находятся в стандартных Java классах-оболочках: *Integer*, *Long* и Character.
- Встроенные в Java целочисленные операторы не сигнализируют о переполнении значений. Исключение составляют операторы деления (/) и нахождения остатка (%), вызывающее исключение если знаменатель равен нулю.

Операции над вещественными типами

- Арифметические и операции сравнения производятся аналогично соответствующим операциям в C/C++
- Целые и вещественные значения можно смешивать в операциях. К обычным вещественным числам добавлены три значения: *POSITIVE_INFINITY* – положительная бесконечность, *NEGATIVE_INFINITY* – отрицательная бесконечность, *NaN* - "не число”
- Конструкторы, методы и константы для работы со значениями с плавающей точкой находятся в стандартных Java классах-оболочках *Float*, *Double* и *Math*.

Логический тип данных

- Логический (булевский) тип данных служит для представления логической величины, имеющей одно из двух возможных значений: *true* – истина, *false* – ложь
- 0 (ноль) не является эквивалентом для значения "ложь“
- Над логическими данными можно выполнять операции присваивания, сравнение на равенство и неравенство, а также логические операции выполняемые аналогично соответствующим операциям в C/C++.

Операции присваивания

=,+=, - =,*=,/=, %=, &=, |=, ^=,<<=, >>=, >>>=

- выполняются аналогично соответствующим операциям в C/C++.

```
int n = 0;
…
n -= 2;  // n = n – 2
```

Условная операция ?:

- выполняются аналогично соответствующей операции в C/C++

```
int y = 123;
…
int x = (y >= 0 ? 1 : -1);
boolean b = y < 0;
```

Выражения

Из констант и переменных, операций над ними, вызовов методов и скобок составляются выражения. При вычислении выражения выполняются четыре правила:

1. Операции одного приоритета вычисляются слева направо. Исключение: операции присваивания вычисляются справа налево
2. Левый операнд вычисляется раньше правого
3. Операнды полностью вычисляются перед выполнением операции
4. Перед выполнением составной операции присваивания значение левой части сохраняется для использования в правой части

Пример.

```
int a = 3, b = 5;
b += a += b += 7;
```

В результате вычислений b получит значение 27.

Приоритет операции

Операции перечислены в порядке убывания приоритета:

- Постфиксные операции ++ и --.
- Префиксные операции ++ и --, дополнение ~ и отрицание !
- Приведение типа (тип)
- Умножение *, деление / и взятие остатка %.
- Сложение + и вычитание - .
- Сдвиги <<, >>, >>>
- Сравнения >, <, >=, <=
- Сравнения ==, !=
- Побитовая конъюнкция &
- Побитовое исключающее ИЛИ ^
- Побитовая дизъюнкция |
- Конъюнкция &&
- Дизъюнкция ||
- Условная операция ?:
- Присваивания =,+=, -=, *=, /=, %=, \&=, ^=, |=, <<=, >>=, >>>=

Приведение типов

- При проведении компиляции Java программ происходит проверка соответствия типов данных и предотвращение неверных операций присваивания.
- Все сомнительные программные конструкции приводят к выдаче компилятором сообщений об ошибке в приведении типов.
- Java поддерживает неявное приведение значений целого типа данных в типы с плавающей точкой. Обратное преобразование некорректно и компилятор выдаст ошибку.
- Явное приведение типов осуществляется при помощи оператора:
- *(тип) значение_другого_типа*
- Результат арифметической операции имеет тип *int*, кроме случая, когда один из операндов типа long. В этом случае результат будет типа *long*. Перед выполнением арифметической операции всегда происходит повышение типов byte, short, char в тип *int*, а может быть, и в тип *long*, если другой операнд типа *long*
- Нельзя преобразовать тип данных с плавающей точкой в логический тип данных и наоборот.
- При преобразовании типа данных с большей точностью представления значений (например, *double*) к типу данных, поддерживающих меньшую точность представления (например, *float*) возможна потеря данных.

Управляющие конструкции

Управляющие конструкции предназначены для управления последовательностью исполнения операторов.

В Java существуют следующие управляющие конструкции:

- условие if-else
- переключатель switch
- цикл с предусловием while
- цикл с послесловием do-while
- цикл для определенного диапазона for
- обработчик исключений try-catch-finally

Операторы и блоки

Символом завершения оператора в Java является знак "точка с запятой".

Для группировки операторов в блок используются фигурные скобки:

```
{
    оператор1;
    оператор2;
}
```

Блок является составным оператором и используется так же, как и отдельный оператор.

Условие if-else

Условие if-else служит базовой управляющей конструкцией любого алгоритмического языка и в Java имеет вид:

if (логическоеВыражение)
оператор1
[else
оператор2];

Вначале работы этой управляющей конструкции вычисляется *логическоеВыражение* и если оно верно, то управление передается *оператору1*, в противном случае управление передается *оператору2*.

Переключатель switch

Переключатель *switch* сначала вычисляет *целочисленноеВыражение*, после чего происходит поиск метки *case* с соответствующим вычисленным значением. Если такая метка найдена, то управление передается ей, иначе управление передается метке *default*. Если метка *default* отсутствует, то весь блок операторов управляющей конструкции *switch* игнорируется. Операторы, следующие за найденной меткой, исполняются последовательно, пока не встретится ключевое слово *break*, после чего происходит выход из управляющей конструкции:

```
switch (целочисленноеВыражение)
{
    case значение1: оператор1;
    case значение2: оператор2;
    ...
    default:
        оператор0;
}
```

Цикл с предусловием while

Цикл с предусловием имеет вид:

while (логическоеВыражение) оператор;

Вначале работы цикла проверяется значение логического выражение, и если оно истинно, то выполняется оператор, а затем управление передается в начало цикла, и снова происходит проверка логического выражения.

Цикл исполняется до тех пор, пока логическое выражение имеет значение true.

Цикл с послеусловием do-while

Цикл с послеусловием имеет вид:

```
do
    оператор
while (логическоеВыражение);
```

Цикл с послеусловием аналогичен циклу с предусловием, за исключением того, что сначала выполняется оператор, а потом проверяется истинность логического выражения, и если оно истинно, то управление снова передается в начало цикла. В противном случае происходит выход из цикла.

Цикл for

Цикл for используется для исполнения цикла тогда, когда задан диапазон изменения значений переменных и имеет вид:

for (инициализацияПеременных; логическоеЗначение; приращение) оператор;

где *инициализацияПеременных* имеет вид:

[тип] имяПеременной1 = начальноеЗначение1
[, имяПеременной2 = начальноеЗначение2 [, ...]]

Все переменные, разделенные запятой, должны принадлежать одному и тому же типу.

Приращение имеет вид:

значение1 [, значение2 [, ...]]

Существует вариант бесконечного цикла for:

for (;;) оператор;

из которого можно выйти только при помощи выражения прерывания цикла *break*, выражения выхода из метода *return* или посредством возбуждения исключения.

Выход из цикла break

Для принудительного выхода из управляющих конструкций циклов *while, do - while, for* используется ключевое слово *break*.

Переход к началу цикла continue

Для того, чтобы в процессе исполнения цикла *while*, *do-while* или *for* вернуться к его началу, проигнорировав дальнейшую последовательность операторов, применяется ключевое слово *continue*.

Метки

Определение метки имеет вид:

имяМетки: цикл

где *цикл* представляет собой цикл *while, do - while* или for того уровня, из которого следует выйти при помощи выражения *break имяМетки* или к началу которого следует перейти при помощи выражения *continue имяМетки*.

Использование ключевого слова *break* имеет вид:

break имяМетки;

Использование ключевого слова *continue* имеет вид:

continue имяМетки;

Возврат из метода return

Выражение для выхода из метода имеет следующий вид:

return значение;

где тип значение совпадает с возвращаемым типом, определенным для соответствующего метода.

Если метод объявлен, как не возвращающий значения, то есть *void*, или если ключевое слово *return* используется в теле конструктора, то выражение для выхода из метода имеет вид:

return;

Единица компиляции (compilation unit)

Единица компиляции Java хранится в текстовом .java-файле и является единичной порцией входных данных для компилятора. Каждый .java-файл состоит из трех частей:

1. объявление пакета;
2. import-выражения;
3. объявления верхнего уровня

Объявление пакета одновременно указывает, какому пакету будут принадлежать все объявляемые ниже типы. Это выражение может отсутствовать, что означает, что эти классы располагаются в безымянном пакете (другое название - пакет по умолчанию).

Пример:

package java.lang;

Единица компиляции (compilation unit)

import-выражения позволяют обращаться к типам из других пакетов по их простым именам, "импортировать" их. Эти выражения также необязательны.

Пример:

import java.io;

Наконец, объявления верхнего уровня содержат объявления одного или нескольких типов. Название "верхнего уровня" противопоставляет эти классы и интерфейсы, располагающиеся в пакетах, внутренним типам, которые являются элементами и располагаются внутри других типов.

Эта часть также является необязательной, в том смысле, что компилятор не выдаст ошибки в случае ее отсутствия. Однако, никаких .class-файлов сгенерировано тоже не будет.

Единица компиляции (compilation unit)

Пакеты представляют собой структуры, в которых хранятся скомпилированные байт-коды классов. Физически пакет представляет собой каталог на диске, в котором записаны скомпилированные коды классов.

Классы и интерфейсы

Классы и объекты

Данные и соответствующие им методы объединены в одну структуру, которая в ООП называется объектом.

Объект - это программная модель объектов реального мира или абстрактных понятий, представляющая собой совокупность переменных, задающих состояние объекта, и связанных с ними методов, определяющих поведение объекта.

Классы и объекты

Для описания объектов в Java используется понятие класс. Все коды в языке Java находятся внутри классов.

Класс – это шаблон по которому создаётся объект, он задаёт прототип, описывающий переменные и методы. Конструирование объекта на основе некоторого класса называется созданием экземпляра этого класса (instance).

Создание класса

Объявление класса в языке Java имеет вид:

```
[модификаторы] class имя_класса
            [extends суперКласс]
            [implements СписокИнтерфейсов]
{
 переменные и методы класса
}
```

Объекты создаются с помощью команды `new`.

Создание класса

В качестве модификаторов класса могут использоваться следующие ключевые слова:

- `public` – элементы класса доступны вне текущего пакета, которому принадлежит данный класс
- (`по умолчанию`) - элементы класса доступны из пакета, в который входит данный класс
- `abstract` - класс не имеет экземпляров класса
- `final` - класс не имеет подклассов

Доступ к элементам класса

Все поля и методы доступны для методов данного класса. Для того, чтобы ограничить к ним доступ из других классов, используются спецификаторы доступа к полям и методам класса:

`public` – открытый - к элементам класса можно обращаться из любого другого класса

`protected` – защищенный - к элементам класса можно обращаться только из класса–потомка

(`по умолчанию`) – пакетный - к элементам класса можно обращаться только из классов, принадлежащих тому же самому пакету, что и данный класс

`private` – закрытый - к элементам класса можно обращаться только из самого данного класса

Поля класса

Объявление поля класса имеет вид:

```
[модификаторы] типДанных поле [= начальноеЗначение]
```

Поля делятся на два типа:

- статические (поля класса) - поля, совместно используемые всеми экземплярами одного класса;
- динамические (поля экземпляра класса) - поля, уникальные для каждого экземпляра класса.

Статические поля помечаются модификатором `static`. Они используются для синхронизации данных в различных экземплярах одного и того же класса.

Поля класса

Полям класса или экземпляра класса можно присваивать инициализирующие значения, соответствующие их типу данных.

Значение динамических полей инициализируются при создании экземпляра класса.

Если поле объявляется с модификатором `final`, то значение такого поля в программе не может быть изменено - в противном случае компилятор выдаст сообщение об ошибке. Такие финальные переменные исполняют в Java роль констант.

Обращение к переменной класса (статическому полю) имеет вид:

```
класс.статическоеПоле
```

Обращение к переменной экземпляра класса (динамическому полю) имеет вид:

```
экземплярКласса.поле
```

Объявление и создание экземпляра класса

Объявление экземпляра класса (то есть объекта, принадлежащего соответствующему типу класса) имеет вид:

```
класс экземплярКласса
```

Объявление объекта создает не сам объект, а только ссылку на этот объект. Доступ к объекту осуществляется только по его ссылке. Объекты создаются при помощи оператора new, за которым следует конструктор класса:

```
экземплярКласса = new класс(параметры)
```

Класс в языке Java связно хранит внутри себя кроме полей и методов также блоки инициализации, конструкторы и другие классы, встроенные в текущий класс.

пример

```
 // объявление класса
class Point {
     int x, y;
     boolean isVisible;
  // блок инициализатор:
  {
      x = 0; y = 0;
      isVisible = false;
  }
}

// объявление объекта
Point p;

// создание объекта
p = new Point();
p.x = 100;
p.y = 200;
p.isVisible = true;
```

Вложенные, локальные и анонимные классы

Вложенный класс – это класс, объявленный внутри объявления другого класса.

```
public class EnclosingClass {
   ...
   public class NestedClass { // вложенный класс
   ...
   }
}
```

Вложенные, локальные и анонимные классы

Локальный класс – это класс, объявленный внутри блока кода. Область видимости локального класса ограничена тем блоком, внутри которого он объявлен.

```
public class EnclosingClass {
   ...
   public void enclosingMethod(){
    // Этот класс доступен только внутри enclosingMethod()
    public class LocalClass { // локальный класс
        ...
    }
   }
}
```

Вложенные, локальные и анонимные классы

Анонимный класс – это локальный класс без имени.

```
// Параметр конструктора – экземпляр анонимного класса,
// реализующего интерфейс Runnable
new Thread(
    new Runnable() {
        public void run() {...}
    }
  ).start();
```

Конструкторы

Для инициализации объекта при его создании используются конструкторы.

Конструктор имеет список параметров и имя, совпадающее с именем класса. В классе может быть несколько конструкторов, имеющих разные списки параметров.

Конструктор не возвращает значения и вызывается при создании нового экземпляра. Вызов конструктора происходит после инициализации по умолчанию полей объекта и вызова блоков инициализации полей экземпляра класса.

Конструкторы

Создание конструктора класса имеет вид:

```
[модификаторДоступа] класс (списокПараметров)[throws исключение]
{
    телоКонструктора
}
```

В качестве спецификатораДоступа можно использовать одно из следующих ключевых слов:

- public – доступ из любого другого класса
- protected - доступ из классов-потомков
- (по умолчанию) – доступ из классов пакета
- private – доступ из данного класса, обычно используется для запрета создания экземпляра текущего класса

Конструкторы

Конструкторы имеют место только для экземпляров класса.

Для создания кода инициализации статических переменных, принадлежащих всему классу, используются блоки статической инициализации.

Инициализацию объекта можно производить с помощью *конструктора по умолчанию*. Этот конструктор не содержит параметров при его вызове для создания экземпляра данного класса.

Конструкторы

Если в классе определен хотя бы один конструктор с непустым списком параметров, то для инициализации экземпляра класса без передачи параметров необходимо указать дополнительный конструктор с пустым списком параметров.

Последовательность инициализации полей класса при вызове конструктора с параметрами определяется следующим образом:

· сначала инициализируются поля класса (по умолчанию или явно);

· исполняются блоки инициализации;

· происходит инициализация полей класса при помощи конструктора.

Методы

Метод объявляется следующим образом:

```
[модификаторы]Возвращаемый_тип имя_метода() программный_блок
```

или

```
[модификаторы]Возвращаемый_тип имя_метода(тип имя_параметра, тип
   имя_параметра, ...) программный_блок
```

Модификаторы определяют способы последующего использования метода.

Методы

Перед определением метода можно задать следующие ключевые слова:

abstract - этот метод абстрактный, т. е. в программе нельзя создавать его экземпляры;

static - метод класса;

native - метод реализован на языке, зависящем от платформы;

final - метод не может переопределяться в классах потомках;

synchronized - разрешена работа только одного потока команд данного метода;

public, *private* или *protected* - спецификаторы доступа.

Методы

Внутри метода или в теле конструктора могут использоваться следующие выражения:

this - ссылка на текущий объект;

super - ссылка на базовый объект;

super.имя_метода() - метод базового класса;

this(...) - вызов конструктора данного класса;

super(...) - вызов конструктора базового класса;

return или *return значение* - возвращает управление вызвавшей программе и передает ей указанное значение.

Методы

Метод может быть *перегруженным*, т.е. существует несколько его версий с одним и тем же именем, но с различным списком параметров.

Статические методы могут перегружаться нестатическими и наоборот.

Простые типы данных всегда передаются через параметры методов по значению.

Ссылочные типы данных и массивы всегда передаются через параметры методов по ссылке.

Методы с переменным числом параметров

Объявление:

```
[модификаторы]Возвращаемый_тип имя_метода(тип... имя_параметра) программный_блок
```

При передаче параметров в метод из них автоматически создается массив.

Пакеты. Импортирование

Классы группируются в *пакеты* классов.

Пакеты представляют собой структуры, в которых хранятся скомпилированные байт-коды классов. Физически пакет представляет собой каталог на диске, в котором записаны скомпилированные коды классов.

Все стандартные объекты Java находятся в пакетах.

Для использования классов пакета в программе необходимо выполнить операцию подключения данного пакета. Такая процедура носит название импортирование пакета и имеет следующий синтаксис:

Пакеты. Импортирование

```
import имя_пакета.{имя_класса | * };
```

В случае * подключаются все классы пакета.

Статический импорт: Константы и статические методы класса можно использовать без указания принадлежности к классу, если применить статический импорт, как это показано в примерах:

```
import static java.lang.Math.*;

import static java.lang.Math.E;
```

ООП средствами Java. Инкапсуляция

Инкапсуляция - сокрытие данных внутри объекта, и обеспечение доступа к ним с помощью общедоступных методов.

```
public class Point {
   private int x, y;
   public void move(x1,y1)
   {
    x = x1;
    y = y1;
   }
   public int getX() {
    return x;
   }
   public int getY() {
    return y;
   }
}
```