Начинаем программировать на языке Java. Часть 1

Дмитрий Рамодин

Задумывая этот практикум, я первым делом задал себе вопрос: какими должны быть статьи, обучающие программированию на новом языке? Проанализировав опыт различных изданий, я пришел к выводу, что не стоит начинать с нуля, — для того написаны толстые тома, приобрести которые можно в любом книжном магазине. Я пишу для тех, кто уже имел дело с языком C++. Руководствуясь материалами этого практикума, программист, имеющий опыт написания приложений на C++, сможет освоить написание программ на языке Java. Хочу предупредить читателей: возможно, некоторые места вы поймете не сразу. Не стоит огорчаться, впоследствии, когда мы с вами начнем писать конкретные программы, многое прояснится.

1. Коротко о Java

Язык Java — это объектно-ориентированный язык программирования, ведущий свою историю от известного языка C++. Но в отличие от последнего Java является языком интерпретируемым, программы, написанные на нем, способны работать в разных местах сети и не зависят от платформы, на которой выполняются написанные на нем приложения. Java сознательно избегает арифметики с указателями и прочих ненадежных элементов, которыми изобилует C++, поэтому, разрабатывая на нем приложения, вы предотвратите многие проблемы, обычные при создании программного обеспечения.

В терминах языка Java маленькое приложение, которое встраивается в страницу Web, называется аплет. Собственно говоря, создание аплетов — основное применение для Java. Аплеты снискали себе звание подлинных украшений для Web. Аплет может быть и окном анимации, и электронной таблицей, и всем, что только можно себе

представить. Но это не значит, что на Java нельзя писать нормальные приложения с окнами. Этот язык программирования изначально был создан для обычных приложений, выполняющихся в Internet и интрасетях, и уж потом стал использоваться для изготовления аплетов.

Элементарные строительные блоки в Java называются классами (как и в C++). Класс состоит из данных и кода для работы с ними. В средствах для разработки на языке Java все стандартные классы, доступные программисту, объединены для удобства в упаковки — еще одни элементарные блоки Java-программ.

Вот простейшая программа, приводимая во многих учебниках по Java:

```
class JavaTest
{
 public static void main(String args[])
 {
  System.out.println("Hello, World!");
 }
}
```

Запустим компилятор Java под названием javac и получим готовый класс Java - JavaTest.class. Если вы хотите посмотреть, как этот класс работает, выполните его при помощи команды java JavaTest. При этом необходимо набрать имя запускаемого класса точно так, как оно написано в исходном тексте программы, т.е. с соблюдением регистра, иначе вы получите сообщение об ошибке.

Рассмотрим поэлементно исходный текст нашего примера. Вся программа состоит из одного класса с именем JavaTest. У этого класса имеется единственный метод main, аналогичный функции main в языках программирования C и C++ и определяющий место, с которого программа начинает выполняться (так называемая точка входа). Модификатор доступа public перед именем метода main указывает на то, что этот метод доступен всем классам, желающим его вызвать, независимо от прав доступа и от места их расположения. Модификатор static говорит о том, что для всех экземпляров класса CafeTest и в наследуемых от него классах существует лишь один метод main, разделяемый между всеми классами, которые, возможно, будут унаследованы от JavaTest. Это помогает избежать появления множества точек входа в программу, что вызовет ошибку.

Через переменную-массив args типа String (строка) передаются параметры командной строки класса. В Java первый элемент списка параметров соответствует первому параметру, а не имени запускаемой программы, как это принято в языках C и C++. Доступ к нему можно осуществить через выражение args[0]. Строка System.out.println("Hello, World!") посылает строку текста в стандартный поток вывода, т. е. на экран. Мы отправляем сообщение стандартному классу System, который отвечает за основные системно-независимые операции, например вывод текста на консоль. А уже из этого класса мы вызываем класс стандартного потока вывода. Следом идет вызов метода println, который, собственно, и отображает строку текста на экране монитора, по завершению чего переводит курсор на следующую строку. В Java все методы класса описываются только внутри него. Таким образом, отпадает необходимость в передвижении по тексту в поиске методов классов.

Обработанные классы компилятор записывает в отдельные файлы. Так, если вы опишете в одном исходном файле сразу несколько классов, то в результате компиляции получите несколько файлов с расширением class, по одному для каждого класса, и каждый из них будет иметь то же имя, что и соответствующий класс.

Замечание:

Ранее я упоминал о Java как об интерпретируемом языке. Это касается только времени исполнения. А для получения готовых к исполнению классов исходные тексты на Java должны быть откомпилированы в промежуточный код, называемый байт-кодом.

Если вы освоите содержимое упаковки Java с именем java.awt, которая расшифровывается как Abstract Windowing Toolkit (Набор абстрактной работы с оконной системой), то вам откроются неисчислимые возможности по созданию интерфейсов и оконной графики. Эта упаковка обеспечивает машинно-независимый интерфейс управления оконной системой любой оконной операционной системы. В состав java.awt входят более 40 классов, отвечающих за элементы графической среды пользователя (GUI). В основном awt применяется при написании аплетов для страниц Web. При просмотре страницы на Web-сервере аплет передается на машину пользователя, где и запускается на выполнение.

Теперь мы можем рассмотреть аплет, который делает то же самое, что и уже рассмотренный ранее пример, т. е. выводит строку на экран:

```
import java.awt.*;
public class JavaTest
    extends java.applet.Applet
{
 public void init() {}
 public void paint(Graphics graph)
 {
  graph.drawString("Hello, World!",20,30);
 }
}
```

Первой строкой в аплет включаются все необходимые классы из упаковки java.awt, о которой мы только что говорили. Ключевое слово import имеет приблизительно то же значение, что и оператор #include языков C и C++. Далее следует описание класса нашего аплета, предваряемое модификатором доступа public. Его задача — дать возможность использовать наш класс извне, т. е. запускать его из внешних программ. Если этого слова не будет, компилятор выдаст сообщение об ошибке, указав, что аплету требуется описание интерфейса доступа. Далее следует ключевое слово extends и название класса. Так в Java обозначается процесс наследования. Этим словом мы указываем компилятору унаследовать (расширить) стандартный класс java.applet.Applet, отвечающий за создание и работу аплета. Метод init вызывается в процессе инициализации аплета. Сейчас этот метод пуст, но впоследствии, возможно, вы будете пользоваться им для своих нужд. За отображение строки отвечает другой метод — paint. Он вызывается в тот момент, когда требуется перерисовать данные на экране. Здесь с помощью метода drawString стандартного класса Graphics рисуется строка "Hello, World!" с экранными координатами 20,30.

2. Типы

Идентификаторы языка Java должны начинаться с буквы любого регистра или символов "_" и "$". Далее могут следовать и цифры. Например, _Java - правильный идентификатор, а 1_$ — нет. Еще одно ограничение Java проистекает из его свойства использовать для хранения символы кодировки Unicode, т. е. можно применять только символы, которые имеют порядковый номер более 0xC0 в раскладке символов Unicode.

В стандарте языка Java имеются три типа комментариев:

```
/*Comment*/;
//Comment;
/** Comment*/.
```

Первые два представляют собой обычные комментарии, применяемые как в Java, так и в C++. Последний — особенность Java и введен в этот язык для автоматического документирования. После написания исходного текста утилита автоматической генерации документации собирает тексты таких комментариев в один файл.

Цифровые литералы схожи с аналогичными в языке C++. Правила для целых чисел предельно просты. Если у цифры нет суффикса и префикса, то это десятичное число. У восьмеричных чисел перед цифрой стоит ноль, а для шестнадцатеричных префикс состоит из ноля и буквы X (0x или 0X). При добавлении к цифре буквы L числу присваивается тип long. Примеры: 23 (десятичное), 0675 (восьмеричное), 0x9FA (шестнадцатеричное), 456L (длинное целое).

Теперь обратимся к числам с плавающей точкой. Для них предусмотрены два вида описаний: обычное и экспоненциальное. Обычные числа с плавающей точкой записываются в такой же форме, как и те числа, которые мы пишем на бумаге от руки: 3.14, 2.73 и т.д. Это же относится и к экспоненциальному формату: 2.67E4, 5.0E-10. При добавлении суффиксов D и F получаются числа типов double и float соответственно. Например, 2.71D и 0.981F.

Коснемся подробнее числовых типов. В языке Java появился новый 8-битный тип — byte. Тип int, в отличие от аналогичного в C++, имеет длину 32 бит. А для 16-битных чисел предусмотрен тип short. В соответствии со всеми этими изменениями тип long увеличился, став 64-битным.

В стандарт Java был введен тип boolean, которого так долго ждали программисты, использующие C++. Он может принимать лишь два значения: true и false.

По сравнению с C++ массивы Java претерпели значительные изменения. Во-первых, изменились правила их описания. Массив теперь может быть описан двумя следующими способами:

```
type name[];
type[] name;
```

При этом массив не создается, лишь описывается. Следовательно, для резервирования места под его элементы надо воспользоваться динамическим выделением с помощью ключевого слова new, например:

```
char[] arrayName;
arrayName[] = new char[100];
```

или совместить описание массива с выделением под него памяти:

```
char array[] = new char[100];
```

Многомерных массивов в Java нет, поэтому приходится прибегать к ухищрениям. Например, создать многомерный массив можно как массив массивов:

```
float matrix[][] = new float[5][5];
```

В Java возможно приведение разнообразных типов к типу "массив":

```
varName = (array_type[]) other_varName;
```

3. Классы

Рассмотрим теперь, как описываются основные базовые строительные блоки языка Java - классы. Схема синтаксиса описания класса такова:

```
[Модификаторы]
     class ИмяКласса
     [extends ИмяСуперкласса]
     [implements ИменаИнтерфейсов]
{
   Данные класса;
   Методы;
}
```

где:

Модификаторы — ключевые слова типа static, public и т.п., модифицирующие

поведение класса по умолчанию;

ИмяКласса — имя, которое вы присваиваете классу;

ИмяСуперкласса — имя класса, от которого наследуется ваш класс;

ИменаИнтерфейсов — имена интерфейсов, которые реализуются данным классом (об этом в следующем занятии).

Типичный пример класса мы уже приводили ранее. Это класс аплета, выводящего строку на экран.

Схема описания методов класса сродни описанию простых функций в языках C и C++:

```
[Модификаторы]
      ВозвращаемыйТип ИмяМетода
      (Список Параметров)

{
   [Тело Метода]
}
```

В показанном ранее примере описан всего один общедоступный метод Paint, возвращающий тип void и принимающий один параметр graph типа Graphics.

Как и в C++, в классах Java имеются конструкторы. Их назначение полностью совпадает с назначением аналогичных методов C++. Конструкторы могут быть перегружены (overload), т. е. в одном классе может быть несколько конструкторов, отличающихся передаваемыми параметрами.

В отличие от C++ в языке Java предусмотрен единственный способ распределения памяти - оператором new. В отношении выделения блоков памяти во многом действуют те же правила, что и в C++. Но есть и исключение: в Java имеется возможность динамического задания имени создаваемого класса, как говорится, "на лету":

```
ClassVar = new ("Class" + "Name");
```

Здесь операцией конкатенации (объединения) строк создается новая строка "ClassName", которая передается оператору new в качестве параметра. В свою очередь,

new создает класс с именем ClassName типа Object (каждый раз, когда тип создаваемого объекта не указан, предполагается тип Object). По мнению автора, разработчики Java позаимствовали операцию конкатенации строк "+" из языка программирования Pascal. Кстати, о типе Object. Это базовый тип для любого класса в Java (заимствовано из идеологии языка SmallTalk). Даже когда вы, создавая новый класс, явно не указываете его предка, предполагается, что это класс Object.

Классы и их отдельные члены могут быть статическими. В этом случае они помечаются ключевым словом static. Преимущество статических членов состоит в том, что они становятся разделяемыми между всеми классами-потомками и экземплярами класса. Это значит, что, ссылаясь на несколько унаследованных классов или несколько экземпляров, на самом деле вы ссылаетесь на один и тот же член класса, расположенный в одном и том же участке памяти. В дополнение к стандартным статическим определениям в Java есть инициализаторы — блоки кода, помеченные ключевым словом static. Их задача — инициализация статических переменных. При загрузке класса сначала выполняются блоки инициализации, а уже потом начинается присвоение значений простым переменным, которые инициализируются в порядке их описания. То же справедливо и для блоков инициализации. В примере, показанном ниже, переменные инициализируются в следующем порядке: xxx, yyy.

```
class StaticClass
{
   short zzz = 10;
   static int xxx;
   static float yyy;
   static
   {
     xxx = 12345;
     yyy = 3.1415;
   }
}
```

Далее выполняется блок инициализации, и уже потом производится инициализация переменной zzz.

4. Модификаторы доступа

В языке C++ определены три модификатора доступа: private, protected и public. Язык Java обладает тем же набором модификаторов, но расширенным модификатором friendly. Однако все эти модификаторы ведут себя несколько по-другому, в основном из-за того, что в семантику Java были введены новые модули — упаковки (packages), о которых мы уже говорили. Каждая упаковка содержит в себе набор классов и интерфейсов для выполнения какой-либо определенной задачи. Так, например, упаковка java.applet отвечает за работу аплетов, что явствует из ее названия. Соответственно упаковка java.io хранит в себе все необходимое для выполнения операций ввода-вывода и т. д. Модификаторы доступа стали контекстно-чувствительными, т. е. зависят от того, размещается ли класс, к которому производится доступ, в одной упаковке с вызывающим его классом или нет.

В таблице отражена возможность доступа к данным из того или иного класса. Расшифруем теперь то, что здесь изображено. В первом столбце приводятся модификаторы доступа данных и методов, к которым производится обращение. Столбец "Класс" говорит нам, что сам класс имеет право обращаться к своим данным и методам независимо от того, какой модификатор доступа им присвоен. Следующий столбец "Наследник" объясняет, что класс-наследник может обращаться к данным и методам своего предка, исключительно если они имеют спецификатор доступа protected или public, причем в случае protected оговаривается, что доступ к методам и данным protected класса предка возможен, лишь если класс-наследник располагается с ним в той же самой упаковке, в противном случае компилятор не позволит вам доступ. Столбец "Упаковка" говорит о том, что все классы, располагающиеся в одной и той же упаковке, могут обращаться к данным и методам друг друга, если только они не объявлены как private. При этом совершенно не имеет значения иерархия наследования. И последний столбец показывает, что классы, расположенные на одной машине сети, могут обращаться лишь к общедоступным данным и методам, размещенным на другой сетевой машине. Ну а теперь несколько подробнее о каждом из модификаторов.

4.1. Public

Любой класс может обратиться к данным и методам другого класса из любого компьютера сети, если он имеет модификатор доступа. Постарайтесь описывать методы как public лишь в крайнем случае, когда это действительно необходимо. А

объявлять переменные внутри класса как public не стоит — для этого надо бы предусмотреть отдельные методы, которые и должны делать это за вас.

4.2. Protected

Модификатор доступа protected позволяет обращаться к данным и методам класса лишь самому классу, классам, хранящимся в этой же упаковке, и унаследованным классам, но лишь в том случае, если они находятся в одной упаковке с классом-предком. Обычно такой модификатор применяют для того, чтобы закрыть доступ к данным и методам для тех классов, которые не состоят в "родственных отношениях" с защищаемым классом. Обратите внимание на то, что в Java классы считаются родственными, не только если они унаследованы друг от друга, но и просто хранятся в одной и той же упаковке.

Предположим, что в упаковке Nums имеется некий класс First и что он содержит переменную и метод, объявленные protected:

```
package Nums;
class First

{
   protected int protVar;
   protected void protMethod()
   {
     System.out.println("protMeth called!");
   }
}
```

Если теперь в той же упаковке описать другой класс с именем Second, то он сможет свободно обращаться к методам и данным класса First, не обращая внимания на то, что Second не был унаследован от First:

```
package Nums;
class Second
{

   void protAccessMethod()
```

```
    {
      First ap = new First();
      ap.protVar = 345;
      ap.protMethod();
    }
}
```

Напомню, что унаследованные классы могут беспрепятственно обращаться к данным и методам, отмеченным модификатором protected, только в том случае, если класс-предок располагается в той же упаковке, что и сами классы-наследники.

4.3. Private

Из названия private (частный, собственный) следует то, что к данным и методам, отмеченным этим ключевым словом, доступ отсутствует. Это позволено лишь самому классу - владельцу данных и методов. Если же кто-то попытается обратиться к private-данным или методам, то компилятор Java немедленно выдаст сообщение об ошибке компиляции. Если ваш класс не будет в дальнейшем наследоваться, то лучше использовать модификатор private, а не protected.

4.4. Friendly

В Java существует еще один модификатор доступа — friendly. Этот модификатор не пишется явно, но подразумевается, если не указан никакой другой модификатор доступа. При использовании friendly к данным может обратиться любой класс и метод в той же самой упаковке.

Мир ПК #11-12/96