- поддержка Java-XML Web Service (JAX-WS) для создания приложений поколения Web 2.0;
- улучшены возможности интернационализации ПО, в том числе использования различных региональных форматов и методов преобразования данных;
- новый набор java.awt.Desktop API;
- поддержка области состояния: два новых класса, SystemTray и TrayIcon;
- модернизация в Java Foundation Classes (JFC) и Swing;
- Java-XML Binding (JAXB 2.0);
- JDBC 4.0.

Простое приложение

Изучение любого языка программирования удобно начинать с программы вывода обычного сообщения.

```
// пример # 1 : простое линейное  приложение: First.java
package chapt01;

public class First {
      public static void main(String[] args) {
       // вывод строк
           System.out.print("Мустанг ");
           System.out.println("уже здесь!");
      }
   }
```

В следующем примере то же самое будет сделано с использованием метода класса, реализованного на основе простейшего применения объектно-ориентированного программирования:

```
/* пример # 2 : простое объектно-ориентированное приложение :
FirstProgram.java */
package chapt01;

public class FirstProgram {
      public static void main(String[] args) {
      //объявление и создание объекта firstObject
            MustangLogic firstObject = new MustangLogic();
      //вызов метода, содержащего вывод строки
            firstObject.jumpMustang();
      }
}
// пример # 3 : простой класс: MustangLogic
class MustangLogic {
      public void jumpMustang() {// определение метода
      // вывод строки
            System.out.println("Мустанг уже здесь!");
      }
}
```

Здесь класс **FirstProgram** используется для того, чтобы определить метод **main()**, который запускается автоматически интерпретатором Java и может называться контроллером этого простейшего приложения. Метод **main()** содержит аргументы-параметры командной строки **String[] args**, представляющие массив строк, и является открытым (**public**) членом класса. Это означает, что метод **main()** виден и доступен любому классу. Ключевое слово **static** объявляет методы и переменные класса, используемые при работе с классом в целом, а не только с объектом класса. Символы верхнего и нижнего регистров здесь различаются, как и в C++. Тело метода **main()** содержит объявление объекта

```
MustangLogic firstObject = new MustangLogic();
```

и вызов его метода

```
firstObject.jumpMustang();
```

Вывод строки "Мустанг уже здесь!" в примере осуществляет метод **println()** (**ln** – переход к новой строке после вывода) свойства **out** класса **System**, который подключается к приложению автоматически вместе с пакетом **java.lang**. Приведенную программу необходимо поместить в файл **FirstProgram.java** (расширение **.java** обязательно), имя которого совпадает с именем класса.

Объявление классов предваряет строка

```
package chapt01;
```

указывающая на принадлежность классов пакету с именем **chapt01**, который является на самом деле каталогом на диске. Для приложения, состоящего из двух классов, наличие пакетов не является необходимостью. При отсутствии слова **package** классы будут отнесены к пакету по умолчанию, размещенному в корне проекта. Если же приложение состоит из нескольких сотен классов (вполне обычная ситуация), то размещение классов по пакетам является жизненной необходимостью.

Классы из примеров 2 и 3 сохраняются в файлах **FirstProgram.java**. На практике рекомендуется хранить классы в отдельных файлах.

Простейший способ компиляции – вызов строчного компилятора из корневого каталога (в нем находится каталог **chapt01**):

```
javac chapt01/FirstProgram.java
```

При успешной компиляции создаются файлы **FirstProgram.class** и **Mustang.class**. Запустить этот виртуальный код можно с помощью интерпретатора Java:

```
java chapt01.FirstProgram
```

Здесь к имени приложения **FirstProgram.class** добавляется имя пакета **chapt01**, в котором он расположен.

Чтобы выполнить приложение, необходимо загрузить и установить последнюю версию пакета, например с сайта **java.sun.com**. При инсталляции рекомендуется указывать для размещения корневой каталог. Если JDK установлена в директории (для Windows) **c:\jdk1.6.0**, то каталог, который компилятор Java будет рассматривать как корневой для иерархии пакетов, можно вручную задавать с помощью переменной среды окружения **CLASSPATH** в виде:

```
CLASSPATH=.;c:\jdk1.6.0\
```

Переменной задано еще одно значение '**.**' для использования текущей директории, например **c:\temp**, в качестве рабочей для хранения своих собственных приложений.

Чтобы можно было вызывать сам компилятор и другие исполняемые программы, переменную **PATH** нужно проинициализировать в виде

```
PATH=c:\jdk1.6.0\bin
```

Этот путь указывает на месторасположение файлов **javac.exe** и **java.exe**. В различных версиях операционных систем путь к JDK может указываться различными способами.

Однако при одновременном использовании нескольких различных версий компилятора и различных библиотек применение переменных среды окружения начинает мешать эффективной работе, так как при выполнении приложения поиск класса осуществляется независимо от версии. Когда виртуальная машина обнаруживает класс с подходящим именем, она его и подгружает. Такая ситуация предрасполагает к ошибкам, порой трудноопределимым. Поэтому переменные окружения лучше не определять вовсе.

Следующая программа отображает в окне консоли аргументы командной строки метода **main()**. Аргументы представляют последовательность строк, разделенных пробелами, значения которых присваиваются объектам массива **String[] args**. Объекту **args[0]** присваивается значение первой строки и т.д. Количество аргументов определяется значением **args.length**.

```
/* пример # 4 : вывод аргументов командной строки: OutArgs.java */
package chapt01;

public class OutArgs {
      public static void main(String[] args) {
            for (String str : args)
                  System.out.printf("Apr-> %s%n", str);
      }
}
```

В данном примере используется новый вид цикла версии Java 5.0 **for** языка Java и метод форматированного вывода **printf()**. Тот же результат был бы получен при использовании традиционного цикла

```
for (int i = 0; i < args.length; i++)
   System.out.println("Apr-> " + args[i]);
```

Запуск этого приложения осуществляется с помощью следующей командной строки вида:

```
java chapt01.OutArgs 2007 Mustang "Java SE 6"
```

что приведет к выводу на консоль следующей информации:

```
Apr-> 2007
Apr-> Mustang
Apr-> Java SE 6
```

Приложение, запускаемое с аргументами командной строки, может быть использовано как один из способов ввода строковых данных.