1. Дополнительные пакеты

Когда была выпущена версия Java 1.0, она включила набор из восьми пакетов, названных *ядром API* (core API). Каждый последующий выпуск добавлялся к API-ядру. Сейчас API языка Java содержит большое количество пакетов. Многие из новых пакетов поддерживают специальные области. Здесь рассмотрим два пакета: `java.lang.reflect` и `java.text`. Они поддерживают отражение и форматирование текста, соответственно.

Отражение (reflection) — способность программного обеспечения анализировать себя. Возможности форматирования текста пакета `java.text` имеют много применений. Здесь рассматривается форматирование строк даты и времени.

1.1. Пакеты ядра Java API

Все пакеты ядра API Java 2 перечислены в табл. 17.1. Там же кратко описаны их функции.

Таблица 17.1. Пакеты API ядра Java

Пакет	Первичная функция
`java.applet`	Поддерживает конструкцию апплета
`java.awt`	Обеспечивает возможности графических интерфейсов пользователя
`java.awt.color`	Поддерживает цветовые пространства и профили
`java.awt.datatransfer`	Передает данные к системному буферу обмена или от него
`java.awt.dnd`	Поддерживает операции перетаскивания мыши
`java.awt.event`	Обрабатывает события
`java.awt.font`	Представляет различные типы шрифтов
`java.awt.geom.`	Позволяет работать с геометрическими формами
`java.awt.im`	Разрешает ввод японских, китайских и корейских символов в компоненты редактирования текста
`java.awt.image`	Обрабатывает изображения
`java.awt.image.renderable`	Поддерживает независимые от визуализации изображения
`java.awt.print`	Поддерживает общие возможности печати
`java.beans`	Позволяет формировать программные beans-

	компоненты
java.beans.beancontext	Обеспечивает среду выполнения для beans-компонентов
java.io	Вводит и выводит данные
java.lang	Обеспечивает основные функциональные возможности языка
java.lang.ref	Активизирует некоторые взаимодействия со сборщиком мусора
java.lang.reflect	Анализирует код времени выполнения
java.math	Обрабатывает большие целые и десятичные числа
java.net	Поддерживает работу в сети
java.rmi	Обеспечивает удаленный вызов методов
java.rmi.activation	Активизирует постоянные объекты
java.rmi.dgc	Управляет распределенной сборкой мусора
java.rmi.registry	Отображает имена на ссылки удаленных объектов
java.rmi.server	Поддерживает вызов удаленных методов
java.security	Обрабатывает сертификаты, ключи, классификаторы, сигнатуры и другие функции защиты
java.security.acl	Управляет списками управления доступом
java.security.cert	Анализирует и управляет сертификатами
java.security.interfaces	Определяет интерфейсы для DSA-ключей (Digital Signature Algorithm, алгоритм цифровой сигнатуры)
java.security.spec	Определяет параметры ключей и алгоритма
java.sql	Общается с SQL базой данных (Structured Query Language, язык структурированных запросов)
java.text	Поддерживает форматирование, поиск и манипуляции с текстом
java.util	Содержит общие утилиты
java.util.jar	Создает и читает JAR-файлы (архивные файлы Java)
java.util.zip	Читает и записывает сжатые и несжатые ZIP-файлы
javax.swing	Содержит "облегченные" (Swing) компоненты
javax.swing.border	Рисует специализированные границы вокруг Swing-компонентов
javax.swing.colorchooser	Позволяет пользователю выбирать цвет Swing-компонента
javax.swing.event	Определяет события, генерируемые Swing-компонентами
javax.swing.filechooser	Позволяет пользователю библиотеки Swing

	выбирать файл (классы поддержки компонента JFileChooser)
javax.swing.plaf	Поддерживает plaf-свойства (pluggable look-and-feel) библиотеки Swing. Эти классы предназначены для разработчиков, создающих собственные модули-приложения стилей
javax.swing.plaf .basic	Реализует базовые (Basic) plaf-стили интерфейса пользователя (для создания графической среды в стиле Windows)
javax.swing.plaf.metal	Реализует платформно-независимый (Metal) plaf-стиль интерфейса пользователя
javax.swing.plaf.mulfi	Сочетает вспомогательный и заданный по умолчанию plaf-стили интерфейсов пользователя (технология мультиплексирования стилей)
javax.swing.table	Обеспечивает таблицы (классы поддержки компонента JTable)
javax.swing.text	Обеспечивает текстовые компоненты (классы поддержки Swing-технологии создания документов)
j avax.swing.text.html	Позволяет создавать собственные редакторы HTML-файлов (классы библиотеки HTMLEditorKit)
javax.swing.text.html.rtf	Позволяет создавать собственные редакторы RTF-файлов
javax.swing.tree	Классы поддержки для работы со Swing-компонентом JTree
javax.swing.undo	Классы, обеспечивающие реализацию функций отмены/повторения выполненных действий (в технологии Swing)
CORBA	Пакет поддержки CORBA-технологии. Содержит простой модуль ORB, написанный на языке Java
org.OMG.CORBA.DynAnyPackage	Подпакет org.OMG.CORBA
org.OMG.CORBA.ORBPackage	Подпакет org.OMG.CORBA
org.OMG.CORBA.portable	Подпакет org.OMG.CORBA
org.OMG.CORBA.TypeCodePackage	Подпакет org. OMG. CORBA
Org.OMG.CORBA.CosNaming	Обеспечивает именование в языке 1DL (Interface Definition Language, язык описания интерфейса)
org.OMG.CORBA.CosNaming.NamingContextPackage	Определяет исключения для именований в языке 1DL (подпакет предыдущего пакета)

1.2. Отражение

Отражение — это способность программного обеспечения к самоанализу. Его обеспечивает пакет java.lang.reflect и элементы класса Class. Отражение — важная возможность, необходимая при использовании компонентов, называемых Java Beans. Она позволяет анализировать программный компонент и описывать его возможности динамически, во время выполнения, а не во время компиляции. Например, используя отражение, можно определять, какие методы, конструкторы и поля поддерживает данный класс.

Пакет java.lang.reflect содержит один интерфейс, называемый Member, который определяет методы, позволяющие получать информацию о поле, конструкторе или методе класса. В этом пакете имеются также семь классов, перечисленные в табл. 17.2.

Таблица 17.2. Классы, определенные в пакете java.lang.reflect

Класс	Первичная функция
AccessibleObject	Позволяет обходить заданные по умолчанию проверки управления доступом. (Добавлен в Java 2)
Array	Позволяет динамически создавать и манипулировать массивами
Constructor	Обеспечивает информацию о конструкторе
Field	Обеспечивает информацию о поле
Method	Обеспечивает информацию о методе
Modifier	Обеспечивает информацию о модификаторах доступа класса и члена
ReflectPermission	Разрешает отражение private или protected членов класса. (Добавлен Java 2)

Следующее приложение иллюстрирует простое использование возможностей отражения Java. Оно печатает конструкторы, поля и методы класса java.awt.Dimension. Программа начинается использованием метода forName() класса Class, чтобы получить объект типа Class для java.awt.Dimension. Как только он получен, используются методы getConstructors(), getFields() и getMethods() для анализа этого объекта. Они возвращают массивы объектов Constructor, Field и Method, которые обеспечивают информацию об объекте. Классы Constructor, Field и Method определяют несколько методов, которые можно использовать для получения информации об объекте. Однако каждый поддерживает метод toString(). Он позволяет использовать объекты типа Constructor, Field и Method в качестве аргументов метода println(), как показано в следующей программе.

Программа 140. Состав класса

```
// Файл ReflectionDemo1.java
// Демонстрирует отражение.
import java.lang.reflect.*;
public class ReflectionDemo1 {
   public static void main(String args[]) {
      try {
         Class c = Class.forName("java.awt.Dimension");
         System.out.println("\nConstructors:\n");
         Constructor constructors[] = c.getConstructors();
         for(int i = 0; i < constructors.length; i++) {
            System.out.println(" " + constructors[i]);
         }
         System.out.println("\nFields:\n");
         Field fields[] = c.getFields();
         for(int i = 0; i < fields.length; i++) {
            System.out.println(" " + fields[i]);
         }
         System.out.println("\nMethods:\n");
         Method methods[] = c.getMethods();
         for(int i = 0; i < methods.length; i++) {
            System.out.println(" " + methods[i]);
         }
      }
      catch(Exception e) {
         System.out.println("Exception:  " + e) ;
      }
   }
}
```

Вывод этой программы:

```
Constructors:

 public java.awt.Dimension(java.awt.Dimension)
 public java.awt.Dimension()
 public java.awt.Dimension(int,int)

Fields:

 public int java.awt.Dimension.width
 public int java.awt.Dimension.height

Methods:

 public int java.awt.Dimension.hashCode()
 public boolean java.awt.Dimension.equals(java.lang.Object)
 public java.lang.String java.awt.Dimension.toString()
 public java.awt.Dimension java.awt.Dimension.getSize()
 public void java.awt.Dimension.setSize(double,double)
 public void java.awt.Dimension.setSize(int,int)
 public void java.awt.Dimension.setSize(java.awt.Dimension)
 public double java.awt.Dimension.getHeight()
 public double java.awt.Dimension.getWidth()
```

```
 public java.lang.Object java.awt.geom.Dimension2D.clone()
 public void java.awt.geom.Dimension2D.setSize(java.awt.geom.Dimension2D)
 public final native java.lang.Class java.lang.Object.getClass()
 public final native void java.lang.Object.notify()
 public final native void java.lang.Object.notifyAll()
 public final native void java.lang.Object.wait(long) throws
java.lang.InterruptedException
 public final void java.lang.Object.wait(long,int) throws
java.lang.InterruptedException
 public final void java.lang.Object.wait() throws
java.lang.InterruptedException
```

Следующий пример применяет способности отражения Java получать public-методы класса. Программа начинается с построения экземпляра класса A. Используя его объектную ссылку, метод getClass() возвращает объект Class в качестве класса A. Метод getDeclaredMethods() возвращает массив объектов Method, который описывает только методы, объявленные этим классом. Методы, унаследованные от суперклассов, таких как Object, не включаются.

Затем обрабатывается каждый элемент массива methods. Метод getModifiers() возвращает значение типа int, содержащее флажки, которые описывают, какой модификатор доступа применяется для этого элемента. Класс Modifier обеспечивает набор методов, показанных в табл. 17.3, которые могут использоваться для просмотра этих значений. Например, статический метод isPublic() возвращает true, если его аргумент включает модификатор доступа public, иначе – false. В следующей программе, если метод поддерживает общий доступ, его имя получается методом getName() и затем распечатывается.

Программа 141. Получение сведений о методах класса

```
// Файл ReflectionDemo2.java
// Показывает public-методы.
import java.lang.reflect.*;
public class ReflectionDemo2 {
   public static void main(String args[]) {
      try {
         A a = new A();
         Class c = a.getClass();
         System.out.println("Public Methods:");
         Method methods[] = c.getDeclaredMethods();
         for(int i = 0; i < methods.length; i++) {
            int modifiers = methods[i].getModifiers();
            if(Modifier.isPublic(modifiers)) {
               System.out.println (" " + methods[i].getName ());
            }
         }
      }
      catch(Exception e) {
         System.out.println("Exception:  " + e) ;
```

```
      }
   }
}

class A {
   public void a1() { }
   public void a2()  { }
   protected void a3() { }
   private void a4() { }
}
```

Вывод этой программы:

```
Public Methods:
 a1
 a2
```

Таблица 17.3. Методы класса `Modifier`, проверяющие характеристики модификаторов доступа

Метод	Описание
`static boolean isAbstract(int val)`	Возвращает `true`, если в `val` установлен флажок `abstract`, иначе — `false`
`static boolean isFinal(int val)`	Возвращает `true`, если в `val` установлен `final`, иначе — `false`
`static boolean isInterface (int val)`	Возвращает `true`, если в `val` установлен флажок `interface`, иначе — `false`
`static boolean isNative(int val)`	Возвращает `true`, если в `val` установлен флажок `native`, иначе — `false`
`static boolean isPrivate(int val)`	Возвращает `true`, если в `val` установлен флажок `private`, иначе — `false`
`static boolean isProtected(int val)`	Возвращает `true`, если в `val` установлен флажок `protected`, иначе — `false`
`static boolean isPublic(int val)`	Возвращает `true`, если в `val` установлен флажок `public`, иначе — `false`
`static boolean isStatic(int val)`	Возвращает `true`, если в `val` установлен флажок `static`, иначе — `false`
`static boolean isStrict(int val)`	Возвращает `true`, если в `val` установлен флажок `strict`, иначе — `false`
`static boolean isSynchronized(int val)`	Возвращает `true`, если в `val` установлен флажок `synchronized`, иначе — `false`
`static boolean isTransient(int val)`	Возвращает `true`, если в `val` установлен флажок `transient`, иначе — `false`
`static boolean isVolatile(int val)`	Возвращает `true`, если в `val` установлен флажок `volatile`, иначе — `false`

1.3. Текстовое форматирование

Пакет java.text позволяет форматировать, отыскивать и обрабатывать текст. В этом разделе кратко рассматриваются наиболее часто используемые классы, которые форматируют информацию даты и времени.

Класс DateFormat

DateFormat — абстрактный класс, который обеспечивает способность форматировать и синтаксически анализировать дату и время. Метод getDateInstance() возвращает экземпляр класса DateFormat, который может форматировать информацию даты. Он доступен в следующих формах:

```
static final DateFormat getDateInstance()
static final DateFormat getDateInstance(int style)
static final DateFormat getDateInstance(int style, Locale locale)
```

Параметр style принимает одно из следующих значений: DEFAULT, SHORT, MEDIUM, LONG или FULL. Это int-константы, определенные в DateFormat. Они предоставляют различные подробности относительно формата даты. Параметр locale — одна из статических ссылок, определенных в классе Locale. Если style и/или locale не определены, используются значения по умолчанию.

Один из наиболее часто используемых методов в этом классе — format(). Он имеет несколько перегруженных форм, одна из которых:

```
String format(Date d)
```

имеет в качестве параметра объект d класса Date, который должен быть отображен. Метод возвращает строку, содержащую отформатированную информацию.

Следующая программа иллюстрирует, как форматировать информацию даты. Она начинается с создания Date-объекта, который собирает текущую информацию даты и времени, а затем выводит эту информацию, используя различные стили и регионы.

Программа 142. Форматы даты

```
// Файл DateFormatDemo.java
// Демонстрирует форматы даты.
import java.text.*;
import java.util.*;
public class DateFormatDemo {
   public static void main(String args[]) {
      Date date = new Date();
      DateFormat df;
      df = DateFormat.getDateInstance(DateFormat.SHORT, Locale.JAPAN);
      System.out.println("Japan:  " + df.format(date));
```

```
        df = DateFormat.getDateInstance(DateFormat.MEDIUM, Locale.KOREA);
        System.out.println("Korea:  " + df.format(date));
        df = DateFormat.getDateInstance(DateFormat.LONG,  Locale.UK);
        System.out.println("United Kingdom:  " + df.format(date));
        df = DateFormat.getDateInstance(DateFormat.FULL,  Locale.US);
        System.out.println("United States:  " + df.format(date));
        df = DateFormat.getDateInstance(DateFormat.FULL,  Locale.GERMAN);
        System.out.println("German(FULL) :  " + df.format(date));
        df = DateFormat.getDateInstance(DateFormat.MEDIUM,  Locale.GERMAN);
        System.out.println("German(MEDIUM) :  " + df.format(date));
        df = DateFormat.getDateInstance(DateFormat.SHORT,  Locale.GERMAN);
        System.out.println("German(SHORT) :  " + df.format(date));
        df = DateFormat.getDateInstance(DateFormat.FULL, Locale.getDefault());
        System.out.println("Default(FULL) :  " + df.format(date));
        df = DateFormat.getDateInstance(DateFormat.MEDIUM,
                                        Locale.getDefault());
        System.out.println("Default(MEDIUM) :  " + df.format(date));
        df = DateFormat.getDateInstance(DateFormat.SHORT,
                                       Locale.getDefault());
        System.out.println("Default(SHORT) :  " + df.format(date));
    }
}
```

Пример вывода этой программы:

```
Japan:  13/05/05
Korea:  2013. 5. 5
United Kingdom:  05 May 2013
United States:  Sunday, May 5, 2013
German(FULL) :  Sonntag, 5. Mai 2013
German(MEDIUM) :  05.05.2013
German(SHORT) :  05.05.13
Default(FULL) :  5 Май 2013 г.
Default(MEDIUM) :  05.05.2013
Default(SHORT) :  05.05.13
```

Метод `getTimeInstance()` возвращает объект типа `DateFormat`, который может форматировать информацию времени. Он доступен в следующих версиях:

```
static final DateFormat getTimelnstance()
static final DateFormat getTimelnstance(int style)
static final DateFormat getTimelnstance(int style, Locale locale)
```

Параметр `style` принимает одно из следующих значений: `DEFAULT`, `SHORT`, `MEDIUM`, `LONG` или `FULL`. Это `int`-константы, определенные в `DateFormat`. Они предоставляют различные подробности относительно формата времени. Параметр `locale` — одна из статических ссылок, определенных в классе `Locale`. Если `style` и/или `locale` не определены, используются значения по умолчанию.

Следующий пример программы иллюстрирует, как форматировать информацию времени. Он начинается с создания объекта типа `Date`,

который собирает текущие сведения о дате и времени, а затем выводит информацию времени, используя различные стили и регионы.

Программа 143. Форматирование времени

```
// Файл TimeFormatDemo.java
// Демонстрирует форматы времени.
import java.text.*;
import java.util.*;
public class TimeFormatDemo {
   public static void main(String args[]) {
      Date date = new Date();
      DateFormat df;
      df = DateFormat.getTimeInstance(DateFormat.SHORT, Locale.JAPAN);
      System.out.println("Japan:  " + df.format(date));
      df = DateFormat.getTimeInstance(DateFormat.LONG,  Locale.UK);
      System.out.println("United Kingdom:  " + df.format(date));
      df = DateFormat.getTimeInstance(DateFormat.FULL, Locale.CANADA);
      System.out.println("Canada:  " + df.format(date));
      df = DateFormat.getTimeInstance(DateFormat.FULL,  Locale.GERMAN);
      System.out.println("German(FULL) :  " + df.format(date));
      df = DateFormat.getTimeInstance(DateFormat.MEDIUM,  Locale.GERMAN);
      System.out.println("German(MEDIUM) :  " + df.format(date));
      df = DateFormat.getTimeInstance(DateFormat.SHORT,  Locale.GERMAN);
      System.out.println("German(SHORT) :  " + df.format(date));
      df = DateFormat.getTimeInstance(DateFormat.FULL, Locale.getDefault());
      System.out.println("Default(FULL) :  " + df.format(date));
      df = DateFormat.getTimeInstance(DateFormat.MEDIUM,
                                      Locale.getDefault());
      System.out.println("Default(MEDIUM) :  " + df.format(date));
      df = DateFormat.getTimeInstance(DateFormat.SHORT,
                                      Locale.getDefault());
      System.out.println("Default(SHORT) :  " + df.format(date));
   }
}
```

Пример вывода этой программы:

```
Japan:  18:12
United Kingdom:  18:12:13 MSK
Canada:  6:12:13 o'clock PM MSK
German(FULL) :  18:12 Uhr MSK
German(MEDIUM) :  18:12:13
German(SHORT) :  18:12
Default(FULL) :  18:12:13 MSK
Default(MEDIUM) :  18:12:13
Default(SHORT) :  18:12
```

Класс `DateFormat` включает также метод `getDateTimeInstance()`, который может форматировать информацию как дат, так и времени.

Класс SimpleDateFormat

`SimpleDateFormat` — конкретный подкласс `DateFormat`. Он позволяет определять ваши собственные образцы форматирования, которые используются для отображения даты и времени.

Один из его конструкторов:

```
SimpleDateFormat (String formatString)
```

Параметр `formatString` описывает, как отображается информация даты и времени. Пример его применения:

```
SimpleDateFormat sdf = SimpleDateFormat("dd MMM yyyy hh:iran:ss zzz");
```

Символы, используемые в строке форматирования, определяют информацию, которая отображается. Табл. 24.4 перечисляет эти символы и дает описание каждого.

Таблица 17.4. Символы строк форматирования для SimpleDateFormat

Символ	Описание
a	AM или PM (Ante Meridiem/Post Meridiem)
d	День месяца (1-31)
h	Час в AM/PM(1-12)
K	Час в дне (1-24)
m	Минута в часе (0-59)
s	Секунда в минуте (0-59)
w	Неделя года (1-52)
y	Год
z	Временная зона
D	День года (1—366)
E	День недели (например, Четверг)
F	День недели в месяце
G	Эра (AD (Anno Domini, наша эра) или BC (Before Crist, до нашей эры))
H	Час в дне (0-23)
K	Час в AM/PM (0-11)
M	Месяц

S	Миллисекунда
W	Неделя месяца (1-5)
'	Escape-символ

В большинстве случаев количество повторений символа определяет, как этb данные представляются. Текстовая информация отображается в сокращенной форме, если символ образца воспроизведен меньше чем четыре раза. Иначе используется несокращенная форма. Например, образец zzzz может отображать Pacific Daylight Time, а образец zzz — PDT.

Для чисел количество повторений символа образца определяет, сколько цифр представляется. Например, hh:mm:ss может представлять 01:51:15, но h:m:s отображает то же значение времени как 1:51:15.

Наконец, M. или MM заставляют отображать месяц как одну или две цифры. Однако три или большее количество повторений M. отображает месяц, как текстовую строку.

Следующая программа показывает, как этот класс используется:

Программа 144. Управление форматированием

```
// Файл SimpleDateFormatDemo.java
// Демонстрирует SimpleDateFormat.
import java.text.*;
import java.util.*;

public class SimpleDateFormatDemo {
   public static void main(String args[]) {
      Date date = new Date();
      SimpleDateFormat sdf;
      sdf = new SimpleDateFormat("hh:mm:ss");
      System.out.println(sdf.format(date));
      sdf = new SimpleDateFormat("dd MMM yyyy hh:mm:ss zzz");
      System.out.println(sdf.format(date));
      sdf = new SimpleDateFormat("E MMM dd yyyy");
      System.out.println(sdf.format(date));
   }
}
```

Пример вывода этой программы:

```
06:14:54
05 май 2013 06:14:54 MSK
Вс май 05 2013
```