Взаимодействие с потоками `InputStream` и `OutputStream` обеспечивается с помощью специальных методов `Files.newInputStream(Path, OpenOption...)` и `Files.newOutputStream(Path, OpenOption...)`. Так удается удачно объединить старый ввод-вывод, построенный вокруг пакета `java.io`, и новый файловый ввод-вывод, в основе которого лежит пакет `java.nio`.

СОВЕТ

Не забывайте, что при работе со строками String всегда требуется знать их кодировку. Если вы забудете указать кодировку (это делается с помощью класса StandardCharsets, например, так: new String(byte[], StandardCharsets .UTF_8)), то позже могут возникнуть неожиданные проблемы, связанные именно с ней.

В предыдущих примерах показан тривиальный код для считывания файлов и записи информации в них. Сегодня он часто используется в Java 6 и более старых версиях. Все это — довольно сложный низкоуровневый код, а Java 7 предлагает красивые высокоуровневые абстракции, позволяющие избавиться от множества ненужного шаблонного кода.

Упрощение записи и считывания

Во вспомогательном классе `Files` есть несколько полезных методов, выполняющих обычные задачи считывания всех строк кода или всех байтов из файла. Это означает, что теперь вы можете обойтись без написания шаблонного кода, не считывать в буфер байтовые массивы данных с применением цикла `while`. В следующем фрагменте кода показано, как вызывать вспомогательные методы.

```
Path logFile = Paths.get("/tmp/app.log");
List<String> lines = Files.readAllLines(logFile, StandardCharsets.UTF_8);
byte[] bytes = Files.readAllBytes(logFile);
```

При программировании некоторых ситуаций требуется знать, когда выполнять считывание и когда — запись. Особенно это касается файлов свойств или файлов журнала (логов). Именно здесь может очень пригодиться система уведомлений о внесении новых изменений в файл.

2.4.5. Уведомление об изменении файлов

Java 7 позволяет отслеживать внесение изменений в файл или каталог. Это делается с помощью класса `java.nio.file.WatchService`. Этот класс использует клиентские потоки для отслеживания изменений в зарегистрированных файлах или каталогах. При обнаружении изменения класс возвращает соответствующее событие. Подобное уведомление о событиях может пригодиться при мониторинге безопасности, обновлении информации из файла свойств, а также во многих других случаях. Подобные уведомления идеально подходят для замены сравнительно более медленных механизмов опроса, используемых в некоторых современных приложениях.

В листинге 2.7 служба `WatchService` используется для обнаружения любых изменений в домашнем каталоге пользователя karianna и для вывода информации о таком событии изменения на консоль. Как и в других случаях, в которых

используется непрерывный опрашивающий цикл, всегда стоит добавлять в код легковесный механизм для отключения такого цикла.

Листинг 2.7. Использование WatchService

```
import static java.nio.file.StandardWatchEventKinds.*;
try
{
  WatchService watcher =
    FileSystems.getDefault().newWatchService();

  Path dir =
    FileSystems.getDefault().getPath("/usr/karianna");

  WatchKey key = dir.register(watcher, ENTRY_MODIFY);

  while(!shutdown)
  {
    key = watcher.take();
    for (WatchEvent<?> event: key.pollEvents())
    {
      if (event.kind() == ENTRY_MODIFY)
      {
        System.out.println("Home dir changed!");
      }
    }
    key.reset();
  }
}
catch (IOException | InterruptedException e)
{
  System.out.println(e.getMessage());
}
```

Получив задаваемую по умолчанию службу WatchService, вы регистрируете отслеживание изменений для домашнего каталога пользователя karianna (1). Затем в бесконечном цикле (он продолжается до тех пор, пока не изменится флаг shutdown) (2) к службе WatchService применяется метод take(), дожидающийся, пока будет доступен ключ WatchKey. Как только WatchKey оказывается доступен, код опрашивает этот ключ WatchKey на наличие событий WatchEvents (3). Если обнаруживается событие WatchEvent рода Kind ENTRY_MODIFY (4), то вы сообщаете об этом «во всеуслышание». Наконец, остается еще сбросить ключ (5), чтобы он был готов к приему следующего события.

Вы можете отслеживать и события других видов, например ENTRY_CREATE, ENTRY_DELETE и OVERFLOW (последний случай может означать, что событие было потеряно или отменено).

Далее рассмотрим очень важный новый API абстрагирования, предназначенный для считывания и записи данных. Здесь задействуется асинхронный ввод-вывод с применением интерфейса SeekableByteChannel.