Chapter 15
Files, Input/Output Streams, NIO and XML Serialization

Objectives

- In this chapter you'll:
  - Create, read, write and update files.
  - Retrieve information about files and directories using features of the NIO.2 APIs.
  - Learn the differences between text files and binary files.
  - Use class Formatter to output text to a file.
  - Use class Scanner to input text from a file.
  - Use sequential file processing to develop a real-world credit-inquiry program.
  - Write objects to and read objects from a file using XML serialization and the JAXB (Java Architecture for XML Binding) APIs.
  - Use a JFileChooser dialog to allow users to select files or directories on disk.
  - Optionally use java.io interfaces and classes to perform bytebased and character-based input and output.

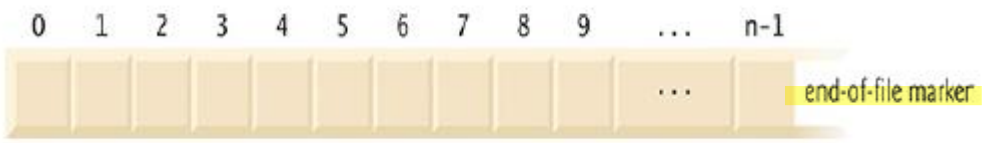
15.1 Introduction

- Data stored in variables and arrays is temporary
  - It's lost when a local variable goes out of scope or when the program terminates
- For long-term retention of data, computers use **files.**
- Computers store files on **secondary storage devices**
  - hard disks, optical disks, flash drives and magnetic tapes.
- Data maintained in files is **persistent data** because it exists beyond the duration of program execution.

15.2 Files and Streams

- Java views each file as a sequential **stream of bytes**

1. Java's NIO APIs also include classes and interfaces that implement so-called channel-based architecture for high-performance I/O. These topics are beyond the scope of this book.



IO：文件处理看作流
有一个文件结束符

fig. 15.1 Java's view of a file of n bytes.

- Every operating system provides a mechanism to determine the end of a file, such as an **end-of-file marker** or a count of the total bytes in the file that is recorded in a system-maintained administrative data structure.

15.2 Files and Streams (cont.)

- File streams can be used to input and output data as bytes or characters.
- **binary files**
  - **byte-based streams** stream结尾基本都是binary files
    - Streams that input and output bytes
- **text files**
  - **character-based streams**
    - Streams that input and output characters
    -

15.2 Files and Streams (cont.)

- A Java program **opens** a file by creating an object and associating a stream of bytes or characters with it.
  - Can also associate streams with different devices.
- Java creates three stream objects when a program begins executing
  - `System.in`
  - `System.out`
  - `System.err`
- Class `System` provides methods **setIn**, **setOut** and **setErr** to **redirect** the standard input, output and error streams, respectively.

对输入输出进行重定向

15.2 Files and Streams (cont.)

- Java programs perform stream-based processing with classes and interfaces from package java.io and the subpackages of **java.nio.**

15.3 Using NIO Classes and Interfaces to Get File and Directory Information

- Interfaces Path and DirectoryStream and classes Paths and Files (all from package java.nio.file):
  - Path interface—Objects of classes that implement Path represent the location of a file or directory.
  - Paths class—Provides static methods used to get a Path object representing a file or directory location.
  - Files class—Provides static methods for common file and directory manipulations
  - DirectoryStream interface—Objects of classes that implement this interface enable a program to iterate through the contents of a directory.

工具类，静态方法，有get方法

15.3 Using NIO Classes and Interfaces to Get File and Directory Information (cont.)

- Creating Path Objects

```
Path path = Paths.get("C:/", "Xmp");
Path path2 = Paths.get("C:/Xmp");

URI u = URI.create("file://C:/Xmp/dd.txt");  //on
Windows
Path p = Paths.get(u);

URI u = URI.create("file:/home/dd.txt");  //on Mac
```

路径可能是拼起来的

15.3 Using NIO Classes and Interfaces to Get File and Directory Information (cont.)

- Example: getting file and directory information

```
import java.io.IOException;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Scanner;

public class FileAndDirectoryInfo
{
   public static void main(String[] args) throws IOException
   {
      Scanner input = new Scanner(System.in);

      System.out.println("Enter file or directory name:");

      // create Path object based on user input
      Path path = Paths.get(input.nextLine());
```

```
      if (Files.exists(path)) // if path exists, output info about it
      {
         // display file (or directory) information
      System.out.printf("%n%s exists%n", path.getFileName());
      System.out.printf("%s a directory%n", Files.isDirectory(path) ? "Is" : "Is not");
      System.out.printf("%s an absolute path%n", path.isAbsolute() ? "Is" : "Is not");
      System.out.printf("Last modified: %s%n", Files.getLastModifiedTime(path));
      System.out.printf("Size: %s%n", Files.size(path));
      System.out.printf("Path: %s%n", path);
      System.out.printf("Absolute path: %s%n", path.toAbsolutePath());

         if (Files.isDirectory(path)) // output directory listing
         {
            System.out.printf("%nDirectory contents:%n");

            // object for iterating through a directory's contents
            DirectoryStream<Path> directoryStream = Files.newDirectoryStream(path);

            for (Path p : directoryStream)
               System.out.println(p);
         }
      }
      else // not file or directory, output error message
      {
         System.out.printf("%s does not exist%n", path);
      }
   }
} // end class FileAndDirectoryInfo
```

是一个目录吗？
isDirectory

转换成绝对路径

对文件夹内容进行遍历

```
Enter file or directory name:
c:\test
c:\test does not exist
```

```
Enter file or directory name:
D:\Intel

Intel exists
Is a directory
Is an absolute path
Last modified: 2015-07-15T09:55:29.405635Z
Size: 0
Path: D:\Intel
Absolute path: D:\Intel

Directory contents:
D:\Intel\ExtremeGraphics
D:\Intel\gp
D:\Intel\Logs
```

```
Enter file or directory name:
E:\test.txt

test.txt exists
Is not a directory
Is an absolute path
Last modified: 2019-02-20T03:13:32.3972Z
Size: 5
Path: E:\test.txt
Absolute path: E:\test.txt
```

15.3 Using NIO Classes and Interfaces to Get File and Directory Information (cont.)

- Separator Characters
  - On a Windows computer, the *separator character* is a backslash (\). On a Linux or macOS system, it's a forward slash (/).
  - Java processes both characters identically in a pathname.
  - For example, if we were to use the path c:\Program Files\Java\jdk1.6.0_11\demo/jfc Java would still process the path properly.

Good Programming Practice 15.1

When building `Strings` *that represent path information, use* `File.separator` *to obtain the local computer's proper separator character rather than explicitly using* `/` *or* `\`*. This constant is a* `String` *consisting of one character—the proper separator for the system.*

Common Programming Error 17.1

Using \ as a directory separator rather than \\ in a string literal is a logic error. A single \ indicates that the \ followed by the next character represents an escape sequence. Use \\ to insert a \ in a string literal.

15.4 Sequential Text Files

- Sequential-access files store records in order by the record-key field.
- Text files are human-readable files.

15.4.1 Creating a Sequential-Access Text File

- Java imposes no structure on a file
  - Notions such as records do not exist as part of the Java language.
  - You must structure files to meet the requirements of your applications.

```
12 public class CreateTextFile {
13    public static void main(String[] args) {
14       Scanner input = new Scanner(System.in);
15       System.out.printf("%s%n%s%n? ",
16          "Enter account number, first name, last name and balance.",
17          "Enter end-of-file indicator to end input.");
18
19       // open clients.txt, output data to the file then close clients.txt
20       try (Formatter output = new Formatter("clients.txt")) {
21          while (input.hasNext()) { // loop until end-of-file indicator
22             try {
23                // output new record to file; assumes valid input
24                output.format("%d %s %s %.2f%n", input.nextInt(),
25                   input.next(), input.next(), input.nextDouble());
26             }
27             catch (NoSuchElementException elementException) {
28                System.err.println("Invalid input. Please try again.");
29                input.nextLine(); // discard input so user can try again
30             }
31
32             System.out.print("? ");
33          }
34       }
35       catch (SecurityException | FileNotFoundException |
36          FormatterClosedException e) {
37          e.printStackTrace();
38          System.exit(1); // terminate the program
39       }
40    }
41 }
```

try后面的括号里面写：autoclosable（可以自动掉close方法的）

当输入不是double抛出错误

15.4.1 Creating a Sequential-Access Text File (cont.)

- `Formatter` outputs formatted `String`s to the specified stream.
- The constructor with one `String` argument receives the name of the file, including its path.
  - If a path is not specified, the JVM assumes that the file is in the directory from which the program was executed.
- If the file does not exist, it will be created.
- If an existing file is opened, its contents are **truncated**.

15.4.1 Creating a Sequential-Access Text File (cont.)

- A **SecurityException** occurs if the user does not have permission to write data to the file.
- A **FileNotFoundException** occurs if the file does not exist and a new file cannot be created.
- the **FormatterClosedException** that occurs if the Formatter object is closed when you attempt to use it to write into a file.

15.4.1 Creating a Sequential-Access Text File (cont.)

- `Scanner` method `hasNext` determines whether the end-of-file key combination has been entered.
- A **NoSuchElementException** occurs if the data being read by a `Scanner` method is in the wrong format or if there is no more data to input.
- `Formatter` method **format** works like `System.out.printf`
- A **FormatterClosedException** occurs if the `Formatter` is closed when you attempt to output.
- `Formatter` method **close** closes the file.
  - If method `close` is not called explicitly, the operating system normally will close the file when program execution terminates.

15.4.1 Creating a Sequential-Access Text File (cont.)

- `static` method `System.exit` terminates an application.
  - An argument of `0` indicates successful program termination.
  - A nonzero value, normally indicates that an error has occurred.
  - The argument is useful if the program is executed from a **batch file** on Windows or a **shell script** on UNIX/Linux/Mac OS X.

Operating system	Key combination
UNIX/Linux/Mac OS X	*<Enter> <Ctrl> d*
Windows	*<Ctrl> z*

Fig. 17.6 | End-of-file key combinations.

Sample data			
100	Bob	Jones	24.98
200	Steve	Doe	-345.67
300	Pam	White	0.00
400	Sam	Stone	-42.16
500	Sue	Rich	224.62

Fig. 17.8 | Sample data for the program in Figs. 17.5–17.7.

15.4.2 Reading Data from a Sequential-Access Text File

- If a `Scanner` is closed before data is input, an **IllegalStateException occurs.**

```
5  import java.io.IOException;
6  import java.lang.IllegalStateException;
7  import java.nio.file.Files;
8  import java.nio.file.Path;
9  import java.nio.file.Paths;
10 import java.util.NoSuchElementException;
11 import java.util.Scanner;
12
13 public class ReadTextFile {
14    public static void main(String[] args) {
15       // open clients.txt, read its contents and close the file
16       try(Scanner input = new Scanner(Paths.get("clients.txt"))) {
17          System.out.printf("%-10s%-12s%-12s%10s%n", "Account",
18             "First Name", "Last Name", "Balance");
19
20          // read record from file
21          while (input.hasNext()) { // while there is more to read
22             // display record contents
23             System.out.printf("%-10d%-12s%-12s%10.2f%n", input.nextInt(),
24                input.next(), input.next(), input.nextDouble());
25          }
26       }
27       catch (IOException | NoSuchElementException |
28          IllegalStateException e) {
29          e.printStackTrace();
30       }
31    }
32 }
33
```

catch后会自动关闭

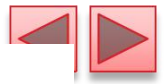
15.4.3 Case Study: A Credit-Inquiry Program

- To retrieve data sequentially from a file, programs start from the beginning of the file and read all the data consecutively until the desired information is found.
- It might be necessary to process the file sequentially several times (from the beginning of the file) during the execution of a program.
- Class `Scanner` does not allow repositioning to the beginning of the file.
  - The program must close the file and reopen it.

MenuOption enum

```
4  // enum type for the credit-inquiry program's options.
5  public enum MenuOption {
6     // declare contents of enum type
7     ZERO_BALANCE(1),
8     CREDIT_BALANCE(2),
9     DEBIT_BALANCE(3),
10    END(4);
11
12    private final int value; // current menu option
13
14    // constructor
15    private MenuOption(int value) {this.value = value;}
16 }
17
18
```

```
// Fig. 15.8: CreditInquiry.java
import java.io.IOException;
import java.lang.IllegalStateException;
import java.nio.file.Paths;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class CreditInquiry {
   private final static MenuOption[] choices = MenuOption.values();

   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);

      // get user's request (e.g., zero, credit or debit balance)
      MenuOption accountType = getRequest(input);

      while (accountType != MenuOption.END) {
         switch (accountType) {
            case ZERO_BALANCE:
               System.out.printf("%nAccounts with zero balances:%n");
               break;
            case CREDIT_BALANCE:
               System.out.printf("%nAccounts with credit balances:%n");
               break;
            case DEBIT_BALANCE:
               System.out.printf("%nAccounts with debit balances:%n");
               break;
         }

         readRecords(accountType);
         accountType = getRequest(input); // get user's request
      }
   }
```



```
40     // obtain request from user
41     private static MenuOption getRequest(Scanner input) {
42        int request = 4;
43
44        // display request options
45        System.out.printf("%nEnter request%n%s%n%s%n%s%n%s%n",
46           " 1 - List accounts with zero balances",
47           " 2 - List accounts with credit balances",
48           " 3 - List accounts with debit balances",
49           " 4 - Terminate program");
50
51        try {
52           do { // input user request
53              System.out.printf("%n? ");
54              request = input.nextInt();
55           } while ((request < 1) || (request > 4));
56        }
57        catch (NoSuchElementException noSuchElementException) {
58           System.err.println("Invalid input. Terminating.");
59        }
60
61        return choices[request - 1]; // return enum value for option
62     }
63
```

```
64    // read records from file and display only records of appropriate type
65    private static void readRecords(MenuOption accountType) {
66       // open file and process contents
67       try (Scanner input = new Scanner(Paths.get("clients.txt"))) {
68          while (input.hasNext()) { // more data to read
69             int accountNumber = input.nextInt();
70             String firstName = input.next();
71             String lastName = input.next();
72             double balance = input.nextDouble();
73
74             // if proper account type, display record
75             if (shouldDisplay(accountType, balance)) {
76                System.out.printf("%-10d%-12s%-12s%10.2f%n", accountNumber,
77                   firstName, lastName, balance);
78             }
79             else {
80                input.nextLine(); // discard the rest of the current record
81             }
82          }
83       }
84       catch (NoSuchElementException | IllegalStateException |
85          IOException e) {
86          System.err.println("Error processing file. Terminating.");
87          System.exit(1);
88       }
89    }
90
```

```
90
91      // use record type to determine if record should be displayed
92      private static boolean shouldDisplay(
93         MenuOption option, double balance) {
94         if ((option == MenuOption.CREDIT_BALANCE) && (balance < 0)) {
95            return true;
96         }
97         else if ((option == MenuOption.DEBIT_BALANCE) && (balance > 0)) {
98            return true;
99         }
100        else if ((option == MenuOption.ZERO_BALANCE) && (balance == 0)) {
101           return true;
102        }
103
104        return false;
105     }
106  }
```

```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - Terminate program

? 1
```

15.4.4 Updating Sequential-Access Files

- The data in many sequential files cannot be modified without the risk of destroying other data in the file.
- Rewriting the entire file is uneconomical to update just one record, but reasonable if a substantial number of records need to be updated.

15.5 XML Serialization

- Sometimes we want to write an entire object to or read an entire object from a file or as you'll see in the online REST Web Services chapter over a network connection.
- XML (eXtensible Markup Language) is a widely used language for describing data. The APIs for manipulating objects as XML are built into Java SE

15.5 XML Serialization

- In this section, we'll manipulate objects using JAXB (Java Architecture for XML Binding).

15.5.1 Creating a Sequential File Using XML Serialization

- JAXB works with POJOs (plain old Java objects)— no special superclasses or interfaces are required for XML-serialization support.
- By default, JAXB serializes only an object's public instance variables and public read–write properties.
  - a read–write property is defined by creating get and set methods with specific naming conventions.

```
3  // Fig. 15.9: Account.java
4  // Account class for storing records as objects.
5  public class Account {
6     private int accountNumber;
7     private String firstName;
8     private String lastName;
9     private double balance;
10
11    // initializes an Account with default values
12    public Account() {this(0, "", "", 0.0);}
13
14    // initializes an Account with provided values
15    public Account(int accountNumber, String firstName,
16       String lastName, double balance) {
17       this.accountNumber = accountNumber;
18       this.firstName = firstName;
19       this.lastName = lastName;
20       this.balance = balance;
21    }
22
```

```
23      // get account number
24      public int getAccountNumber() {return accountNumber;}
25
26      // set account number
27      public void setAccountNumber(int accountNumber)
28         {this.accountNumber = accountNumber;}
29
30      // get first name
31      public String getFirstName() {return firstName;}
32
33      // set first name
34      public void setFirstName(String firstName)
35         {this.firstName = firstName;}
36
37      // get last name
38      public String getLastName() {return lastName;}
39
40      // set last name
41      public void setLastName(String lastName) {this.lastName = lastName;
42
43      // get balance
44      public double getBalance() {return balance;}
45
46      // set balance
47      public void setBalance(double balance) {this.balance = balance;}
48  }
```

```java
// Fig. 15.10: Accounts.java
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlElement;

public class Accounts {
   // @XmlElement specifies XML element name for each object in the List
   @XmlElement(name="account")
   private List<Account> accounts = new ArrayList<>(); // stores Accounts

   // returns the List<Accounts>
   public List<Account> getAccounts() {return accounts;}
}
```

15.5.1 Creating a Sequential File Using XML Serialization(cont.)

- @XmlElement(name="account")

实现接口，可序列化

- The annotation @XMLElement indicates that the private instance variable should be serialized.
- The annotation is required because the instance variable is not public and there's no corresponding public read– write property.
- name="account" used to represent each of the List's Account objects in the serialized output.

```
// Fig. 15.11: CreateSequentialFile.java
// Writing objects to a file with JAXB and BufferedWriter.
import java.io.BufferedWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.NoSuchElementException;
import java.util.Scanner;
import javax.xml.bind.JAXB;

public class CreateSequentialFile {
   public static void main(String[] args) {
      // open clients.xml, write objects to it then close file
      try(BufferedWriter output =
         Files.newBufferedWriter(Paths.get("clients.xml"))) {

         Scanner input = new Scanner(System.in);

         // stores the Accounts before XML serialization
         Accounts accounts = new Accounts();

         System.out.printf("%s%n%s%n? ",
            "Enter account number, first name, last name and balance.",
            "Enter end-of-file indicator to end input.");

```

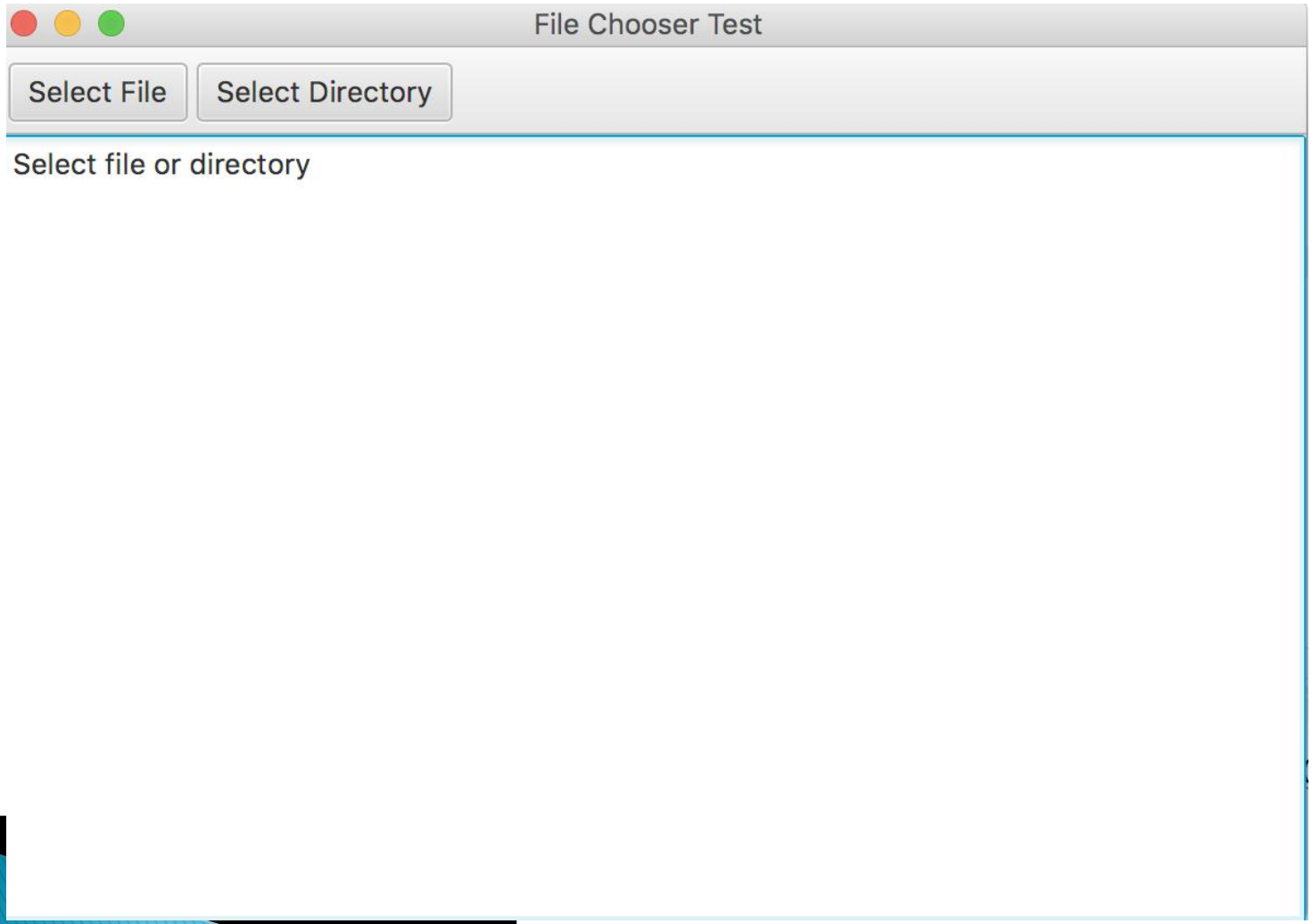
```
28            while (input.hasNext()) { // loop until end-of-file indicator
29               try {
30                  // create new record
31                  Account record = new Account(input.nextInt(),
32                     input.next(), input.next(), input.nextDouble());
33
34                  // add to AccountList
35                  accounts.getAccounts().add(record);
36               }
37               catch (NoSuchElementException elementException) {
38                  System.err.println("Invalid input. Please try again.");
39                  input.nextLine(); // discard input so user can try again
40               }
41
42               System.out.print("? ");
43            }
44
45            // write AccountList's XML to output
46            JAXB.marshal(accounts, output);
47         }
48         catch (IOException ioException) {
49            System.err.println("Error opening file. Terminating.");
50         }
51      }
52   }
```

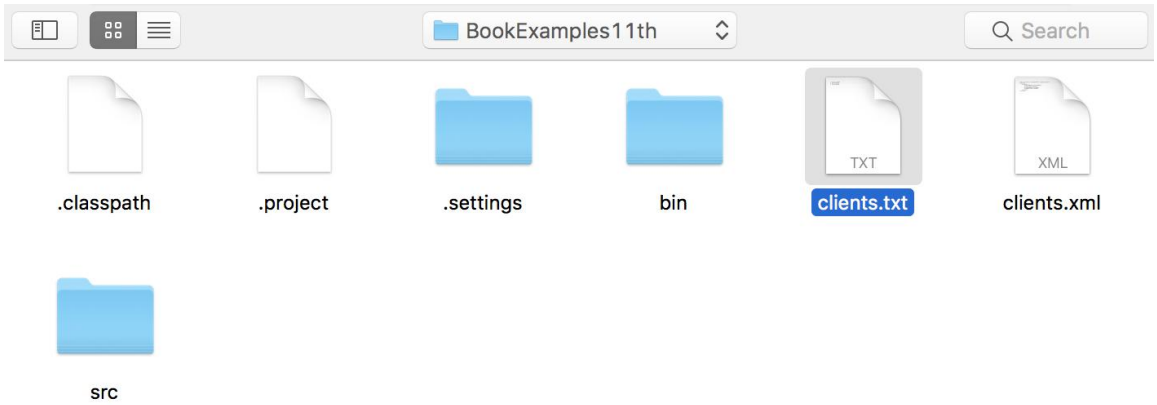
```
// Fig. 15.13: ReadSequentialFile.java
import java.io.BufferedReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import javax.xml.bind.JAXB;

public class ReadSequentialFile {
   public static void main(String[] args) {
      // try to open file for deserialization
      try(BufferedReader input =
         Files.newBufferedReader(Paths.get("clients.xml"))) {
         // unmarshal the file's contents
         Accounts accounts = JAXB.unmarshal(input, Accounts.class);

         // display contents
         System.out.printf("%-10s%-12s%-12s%10s%n", "Account",
            "First Name", "Last Name", "Balance");

         for (Account account : accounts.getAccounts()) {
            System.out.printf("%-10d%-12s%-12s%10.2f%n",
               account.getAccountNumber(), account.getFirstName(),
               account.getLastName(), account.getBalance());
         }
      }
      catch (IOException ioException) {
         System.err.println("Error opening file.");
      }
   }
}
```

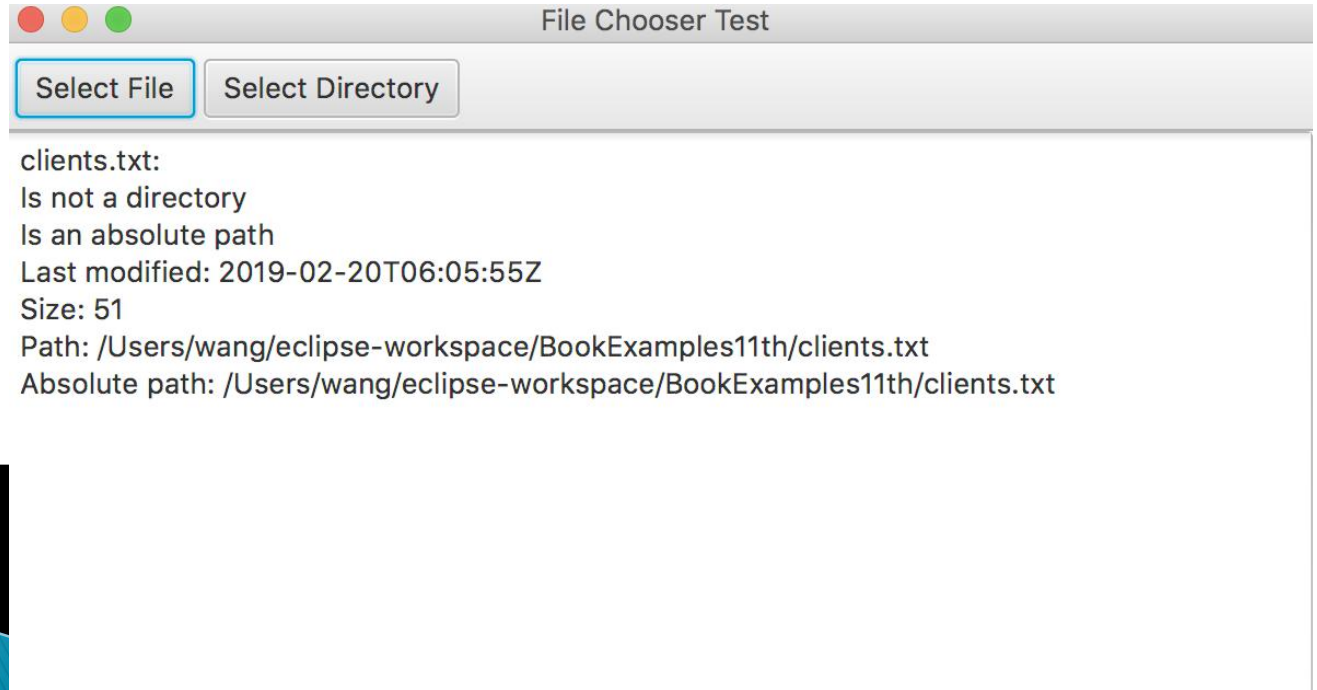
15.6 FileChooser and DirectoryChooser Dialogs

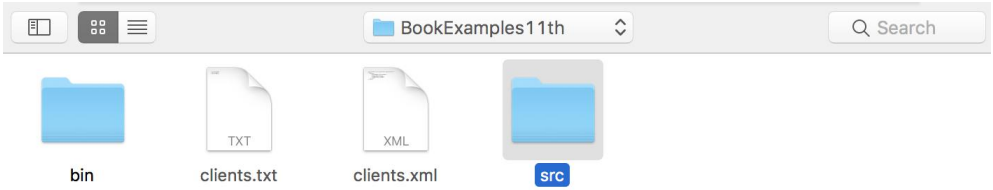
- JavaFX classes FileChooser and DirectoryChooser (package javafx.stage) display dialogs that enable the user to select a file or directory, respectively.

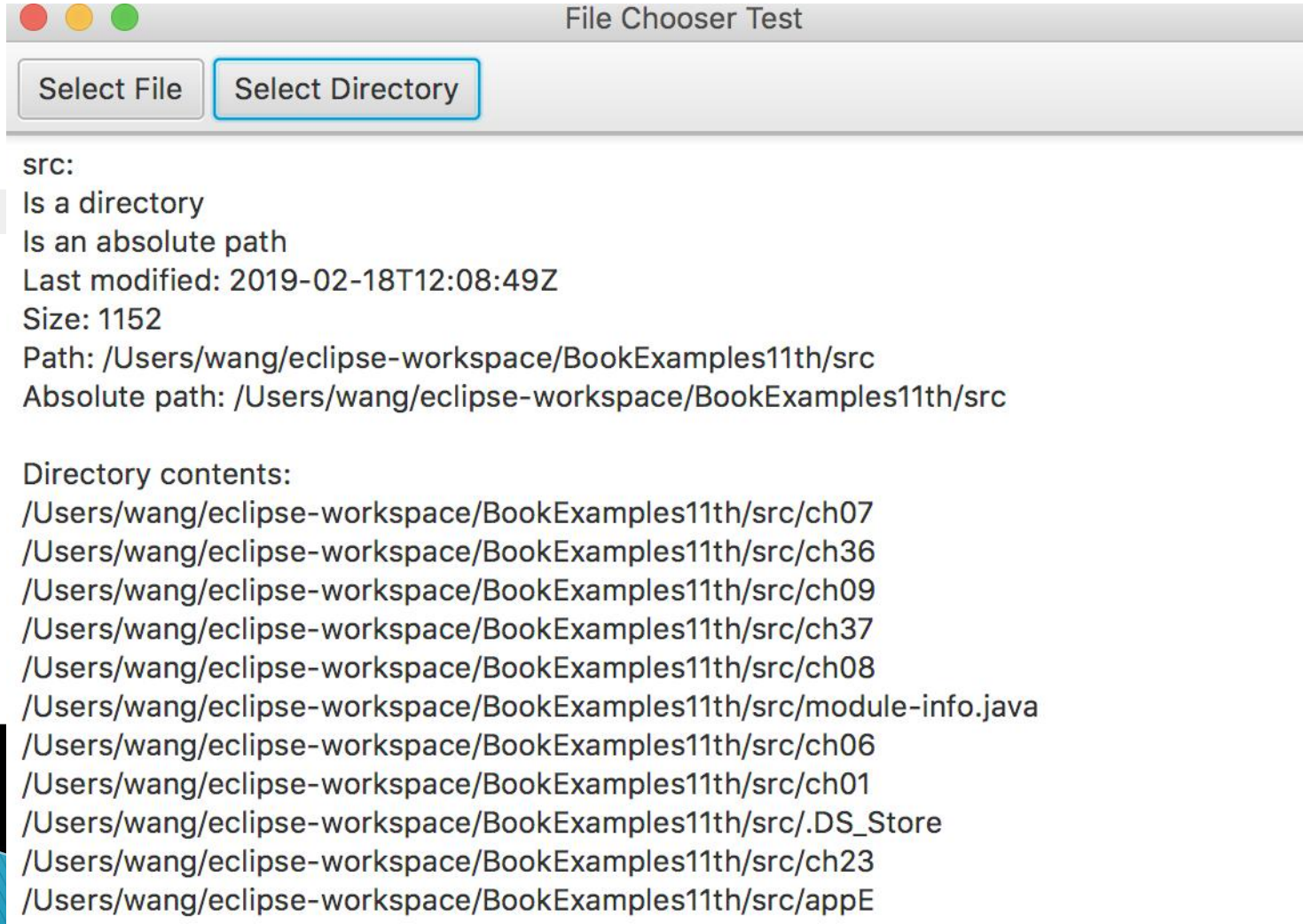
15.6 FileChooser and DirectorChooser Dialogs (cont.)

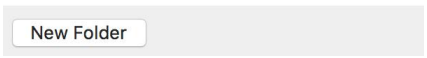


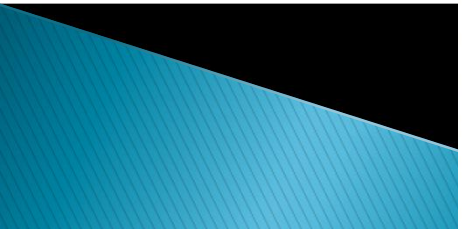

BookExamples11th
Search
.classpath
.project
.settings
bin
TXT
clients.txt
XML
clients.xml
src
Hide extension
Cancel
Open


File Chooser Test
Select File
Select Directory
clients.txt:
Is not a directory
Is an absolute path
Last modified: 2019-02-20T06:05:55Z
Size: 51
Path: /Users/wang/eclipse-workspace/BookExamples11th/clients.txt
Absolute path: /Users/wang/eclipse-workspace/BookExamples11th/clients.txt


BookExamples11th
Search
bin
clients.txt
clients.xml
src
TXT
XML


File Chooser Test
Select File
Select Directory
src:
Is a directory
Is an absolute path
Last modified: 2019-02-18T12:08:49Z
Size: 1152
Path: /Users/wang/eclipse-workspace/BookExamples11th/src
Absolute path: /Users/wang/eclipse-workspace/BookExamples11th/src
Directory contents:
/Users/wang/eclipse-workspace/BookExamples11th/src/ch07
/Users/wang/eclipse-workspace/BookExamples11th/src/ch36
/Users/wang/eclipse-workspace/BookExamples11th/src/ch09
/Users/wang/eclipse-workspace/BookExamples11th/src/ch37
/Users/wang/eclipse-workspace/BookExamples11th/src/ch08
/Users/wang/eclipse-workspace/BookExamples11th/src/module-info.java
/Users/wang/eclipse-workspace/BookExamples11th/src/ch06
/Users/wang/eclipse-workspace/BookExamples11th/src/ch01
/Users/wang/eclipse-workspace/BookExamples11th/src/.DS_Store
/Users/wang/eclipse-workspace/BookExamples11th/src/ch23
/Users/wang/eclipse-workspace/BookExamples11th/src/appE


New Folder



15.6 FileChooser and DirectorChooser Dialogs (cont.)

- Creating the JavaFX GUI
- BorderPane
  - 600* 400
  - fx:id borderPane:
  - In the BorderPane's top, we placed a ToolBar
  - In the BorderPane's center, we placed a TextArea control with the fx:id textArea.

15.6 FileChooser and DirectorChooser Dialogs (cont.)

- Drag other controls onto the ToolBar.
- For the first Button, we set:
  - the Text property to "Select File",
  - the fx:id property to selectFileButton and
  - the On Action event handler to selectFileButtonPressed.
- For the second Button, we set:
  - the Text property to "Select Directory",
  - the fx:id property to selectDirectoryButton and
  - the On Action event handler to selectDirectoryButtonPressed.

```
import javafx.application.Application;

public class FileChooserTest extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        Parent root =
            FXMLLoader.load(getClass().getResource("FileChooserTest.fxml"));

        Scene scene = new Scene(root);
        stage.setTitle("File Chooser Test"); // displayed in title bar
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

```
3  // Fig. 13.6: FileChooserTestController.java
4  // Displays information about a selected file or folder.
5  import java.io.File;
6  import java.io.IOException;
7  import java.nio.file.DirectoryStream;
8  import java.nio.file.Files;
9  import java.nio.file.Path;
10 import java.nio.file.Paths;
11 import javafx.event.ActionEvent;
12 import javafx.fxml.FXML;
13 import javafx.scene.control.Button;
14 import javafx.scene.control.TextArea;
15 import javafx.scene.layout.BorderPane;
16 import javafx.stage.DirectoryChooser;
17 import javafx.stage.FileChooser;
18
19 public class FileChooserTestController {
20    @FXML private BorderPane borderPane;
21    @FXML private Button selectFileButton;
22    @FXML private Button selectDirectoryButton;
23    @FXML private TextArea textArea;
24
```

```
25      // handles selectFileButton's events
26      @FXML
27      private void selectFileButtonPressed(ActionEvent e) {
28         // configure dialog allowing selection of a file
29         FileChooser fileChooser = new FileChooser();
30         fileChooser.setTitle("Select File");
31
32         // display files in folder from which the app was launched
33         fileChooser.setInitialDirectory(new File("."));
34
35         // display the FileChooser
36         File file = fileChooser.showOpenDialog(
37            borderPane.getScene().getWindow());
38
39         // process selected Path or display a message
40         if (file != null) {
41            analyzePath(file.toPath());
42         }
43         else {
44            textArea.setText("Select file or directory");
45         }
46      }
```

设置初始位置

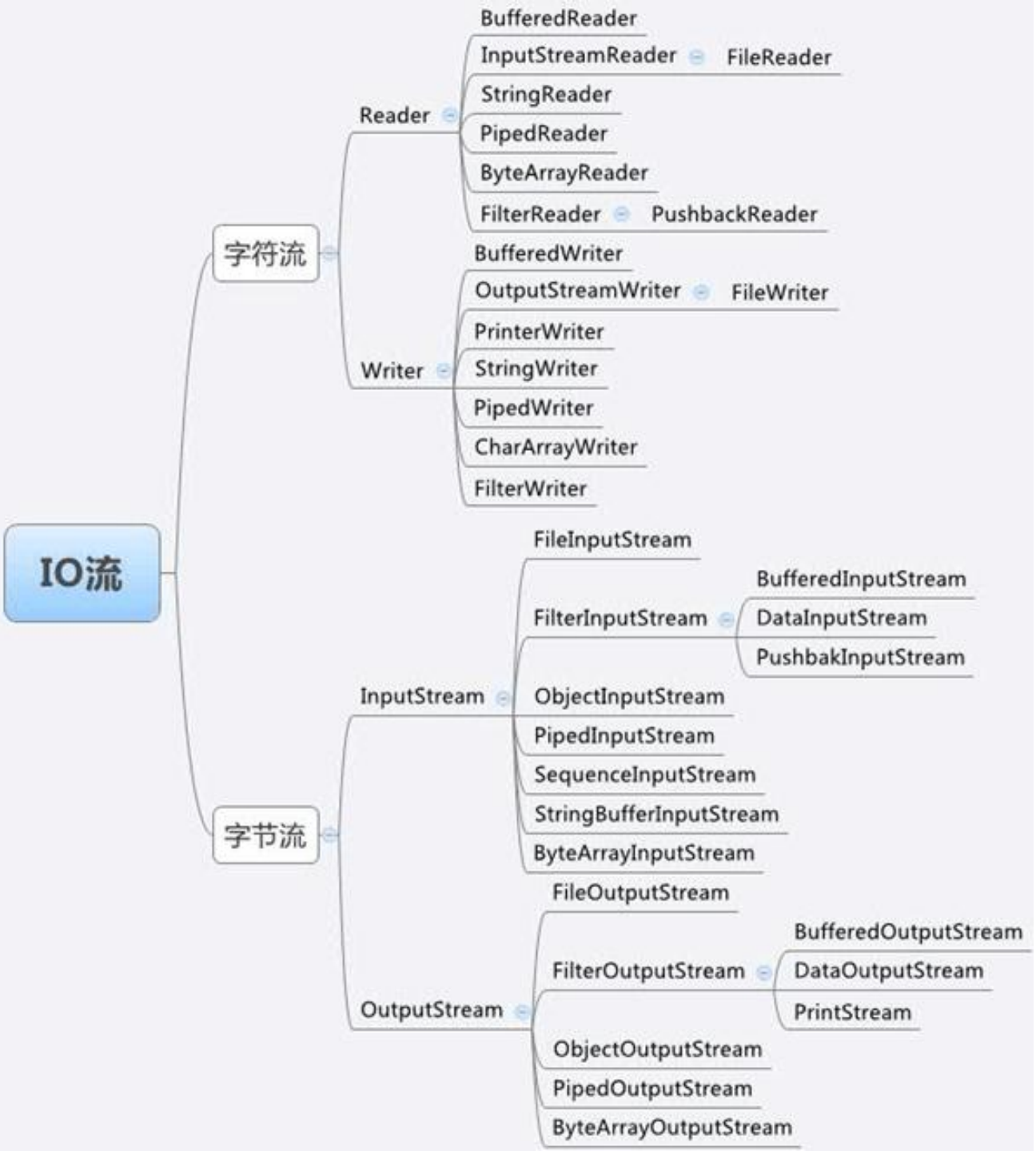
```
48      // handles selectDirectoryButton's events
49      @FXML
50      private void selectDirectoryButtonPressed(ActionEvent e) {
51         // configure dialog allowing selection of a directory
52         DirectoryChooser directoryChooser = new DirectoryChooser();
53         directoryChooser.setTitle("Select Directory");
54
55         // display folder from which the app was launched
56         directoryChooser.setInitialDirectory(new File("."));
57
58         // display the FileChooser
59         File file = directoryChooser.showDialog(
60            borderPane.getScene().getWindow());
61
62         // process selected Path or display a message
63         if (file != null) {
64            analyzePath(file.toPath());
65         }
66         else {
67            textArea.setText("Select file or directory");
68         }
69      }
70
```

```
71    // display information about file or directory user specifies
72    public void analyzePath(Path path) {
73       try {
74          // if the file or directory exists, display its info
75          if (path != null && Files.exists(path)) {
76             // gather file (or directory) information
77             StringBuilder builder = new StringBuilder();
78             builder.append(String.format("%s:%n", path.getFileName()));
79             builder.append(String.format("%s a directory%n",
80                Files.isDirectory(path) ? "Is" : "Is not"));
81             builder.append(String.format("%s an absolute path%n",
82                path.isAbsolute() ? "Is" : "Is not"));
83             builder.append(String.format("Last modified: %s%n",
84                Files.getLastModifiedTime(path)));
85             builder.append(String.format("Size: %s%n", Files.size(path)));
86             builder.append(String.format("Path: %s%n", path));
87             builder.append(String.format("Absolute path: %s%n",
88                path.toAbsolutePath()));
89
90             if (Files.isDirectory(path)) { // output directory listing
91                builder.append(String.format("%nDirectory contents:%n"));
92
93                // object for iterating through a directory's contents
94                DirectoryStream<Path> directoryStream =
95                   Files.newDirectoryStream(path);
96
97                for (Path p : directoryStream) {
98                   builder.append(String.format("%s%n", p));
99                }
.00             }
.01
.02             // display file or directory info
.03             textArea.setText(builder.toString());
.04          }
.05          else { // Path does not exist
.06             textArea.setText("Path does not exist");
.07          }
.08       }
.09       catch (IOException ioException) {
.10          textArea.setText(ioException.toString());
.11       }
.12    }
.13 }
.14
```

https://docs.oracle.com/javase/8/javafx/api/javafx/stage/FileChooser.html

15.7 Additional `java.io` Classes

- This section overviews additional interfaces and classes (from package `java.io`) for byte-based input and output streams and character-based input and output streams.


IO流
字符流
Reader
BufferedReader
InputStreamReader
FileReader
StringReader
PipedReader
ByteArrayReader
FilterReader
PushbackReader
Writer
BufferedWriter
OutputStreamWriter
FileWriter
PrinterWriter
StringWriter
PipedWriter
CharArrayWriter
FilterWriter
字节流
InputStream
FileInputStream
FilterInputStream
BufferedInputStream
DataInputStream
PushbakInputStream
ObjectInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
ByteArrayInputStream
OutputStream
FileOutputStream
FilterOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
ObjectOutputStream
PipedOutputStream
ByteArrayOutputStream

15.7.1 Interfaces and Classes for Byte-Based Input and Output

- InputStream and OutputStream classes
  - abstract classes that declare methods for performing byte-based input and output
  - InputStream Method Summary
    - int available()

      Returns the number of bytes that can be read
    - void close()
    - void mark(int readlimit)

      Marks the current position in this input stream.
    - boolean markSupported()

15.7.1 Interfaces and Classes for Byte-Based Input and Output (cont.)

- abstract int read()

  Reads the next byte of
- int read(byte[] b)
- int read(byte[] b, int off, int len)
- void reset()

  Repositions this stream to the position at the time the mark method was last called on this input stream.
- long skip(long n)

  Skips over and discards n bytes of data from this input stream.

15.7.1 Interfaces and Classes for Byte-Based Input and Output (cont.)

- OutputStream Method Summary
  - void close()
    - Closes this output stream and releases any system resources associated with this stream.
  - void flush()
    - Flushes this output stream and forces any buffered output bytes to be written out.
  - void write(byte[] b)
    Writes b.length bytes from the specified byte array to this output stream.
  - void write(byte[] b, int off, int len)
    - Writes len bytes from the specified byte array starting at offset off to this output stream.
  - abstract void write(int b)
    - Writes the specified byte to this output stream.

15.7.1 Interfaces and Classes for Byte-Based Input and Output (cont.)

- PipedInputStream and PipedOutputStream classes
  - Establish pipes between two threads in a program
  - Pipes are synchronized communication channels between threads
- FilterInputStream and FilterOutputStream classes
  - Provides additional functionality to stream, such as aggregating data byte into meaningful primitive-type units

15.7.1 Interfaces and Classes for Byte-Based Input and Output (cont.)

- PrintStream class
  - Performs text output to a specified stream
  - Support the print() and println() methods for all types, including Object.
- DataInput and DataOutput interfaces
  - For reading and writing primitive types to a file
  - DataInput implemented by classes RandomAccessFile and DataInputStream, DataOutput implemented by RandomAccessFile and DataOuputStream
- SequenceInputStream class enables concatenation of several InputStreams – program sees group as one continuous InputStream
  - Examples: SequenceInputStreamDemo.java RandomAccessFileDeom

15.7.1 Interfaces and Classes for Byte-Based Input and Output (cont.)

- Buffering is an I/O-performance-enhancement technique
  - Greatly increases efficiency of an application
  - Output (uses BufferedOutputStream class)
    - Each output statement does not necessarily result in an actual physical transfer of data to the output device – data is directed to a region of memory called a buffer (faster than writing to file)
    - When buffer is full, actual transfer to output device is performed in one large physical output operation (also called logical output
    - Partially [illegible]er can be forced out with method flush

15.7.1 Interfaces and Classes for Byte-Based Input and Output (cont.)

- Input (uses BufferedInputStream class)
  - Many logical chunks of data from a file are read as one physical input operation (also called logical input operation)
  - When buffer is empty, next physical input operation is performed

- ByteArrayInputStream and ByteArrayOutputStream classes used for inputting from byte arrays in memory and outputting to byte arrays in memory

Performance Tip 17.1

Buffered I/O can yield significant performance improvements over unbuffered I/O.

15.7.2 Interfaces and Classes for Character-Based Input and Output

- Reader and Writer abstract classes
  - Unicode two-byte, character-based streams
- FileReader and FileWriter classes
  - Can use to read/write the contens of a file.
- BufferedReader and BufferedWriter classes
  - Enable buffering for character-based streams
- CharArrayReader and CharArrayWriter classes
  - Read and write streams of characters to character arrays
- LineNumberReader class 通过行号读和写
  - Buffered character stream that keeps track of number of lines read
- PipedReader and PipedWriter classes
  - Implement piped-character streams that can be used to transfer information between threads
- StringReader and StringWriter classes
  - Read characters from and write characters to Strings

ZIP Archives

- ZIP archives store one or more files in (usually) compressed format.
- Each ZIP archive has a header with information such as the name of the file and the compression method that was used.

a typical code sequence to read through a ZIP file

```
ZipInputStream zin = new ZipInputStream(new
FileInputStream(zipname));
ZipEntry entry;
while ((entry = zin.getNextEntry()) != null
{
  analyze entry;
  read the contents of zin;
  zin.closeEntry();
}
zin.close();
```

在基础流之上又做了一层封装

to read a text file inside a ZIP file, you can use the following loop:

```
Scanner in = new Scanner(zin);
while (in.hasNextLine())
   do something with in.nextLine();
```

a typical code sequence to place entry into the ZIP file

```
FileOutputStream fout = new
FileOutputStream("test.zip");
ZipOutputStream zout = new
ZipOutputStream(fout);
for all files
{
   ZipEntry ze = new ZipEntry(filename);
   zout.putNextEntry(ze);
   send data to zout;
   zout.closeEntry();
```

Two examples:

- ZipCompress.java

  - Create a zip file and then input an entry in it.
- ZipTest.java

ZipTest

File

```
// Fig. 14.18: CreateSequentialFile.java
// Writing objects sequentially to a file with class ObjectOutputStream.
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.NoSuchElementException;
import java.util.Scanner;

import com.deitel.jhtp6.ch14.AccountRecordSerializable;

public class CreateSequentialFile
{
  private ObjectOutputStream output; // outputs data to file

  // allow user to specify file name
  public void openFile()
  {
```

CreateSequentialFile.java

See also:http m/java/tar/

Rar file : http://orz.javaeye /blog/110479

Save image in the file(.jpg)

- JPEGImageEncoder
  - Encodes buffers of image data into JPEG data streams.
  - Required to provide image data in a Raster or a BufferedImage
- JPEGImageDecoder
  - This interface describes a JPEG data stream decoder.
  - Takes an InputStream that contains JPEG encoded image data.
  - Returned in either a Raster or a BufferedImage.

```java
public class Test {
    public static void main(String[] args) {
        File out = new File("/Users/wangjue/DownLoads/1.jpg");
        //将图片写入ImageIO流
        try {
            BufferedImage img = ImageIO.read(out);
        //将图片写出到指定位置（复制图片）
            OutputStream ops = new FileOutputStream(new File("/Users/wangjue
            ImageIO.write(img, "jpg", ops);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

```java
/**
 * 将{@link BufferedImage}生成formatName指定格式的图像数据
 * @param source
 * @param formatName 图像格式名，图像格式名错误则抛出异常
 * @return
 */
public static byte[] wirteBytes(BufferedImage source,String formatName){
  Assert.notNull(source, "source");
  Assert.notEmpty(formatName, "formatName");
  ByteArrayOutputStream output = new ByteArrayOutputStream();
  Graphics2D g = null;
  try {
    for(BufferedImage s=source;!ImageIO.write(s, formatName, output);){
      if(null!=g)
        throw new IllegalArgumentException(String.format("not found writer for '%s'",formatN
ame));
      s = new BufferedImage(source.getWidth(),
          source.getHeight(), BufferedImage.TYPE_INT_RGB);
      g = s.createGraphics();
      g.drawImage(source, 0, 0,null);
    }
  } catch (IOException e) {
    throw new RuntimeException(e);
  } finally {
    if (null != g)
      g.dispose();
  }
  return output.toByteArray();
}
```

Write and receive data (.xls)

- JXL:
- Download Xls.jar: http://www.andykhan.com/jexcelapi/
- Examples: ExcelTest.java

- POI:
- EXAMPLEs: https://www.cnblogs.com/SimonHu1993/p/8202391.html

An other Class: Runtime

- Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running.
- Process exec(String command)
  - Executes the specified string command in a separate process.

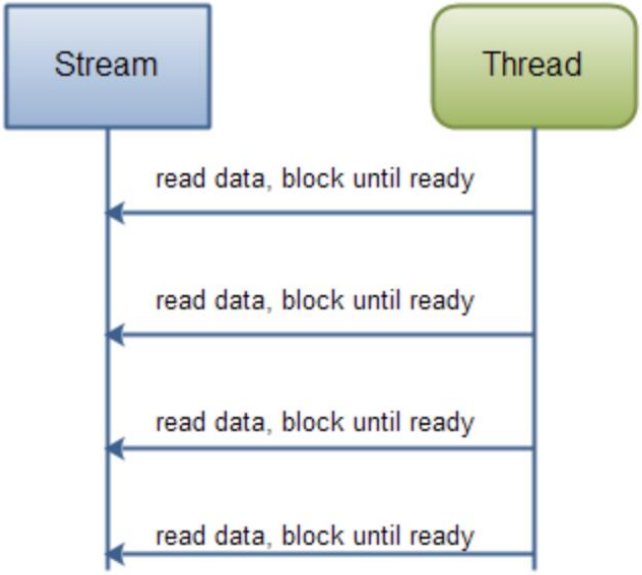
  - Exec1.java
  - Exec2.java

Main Differences Between Java NIO and IO

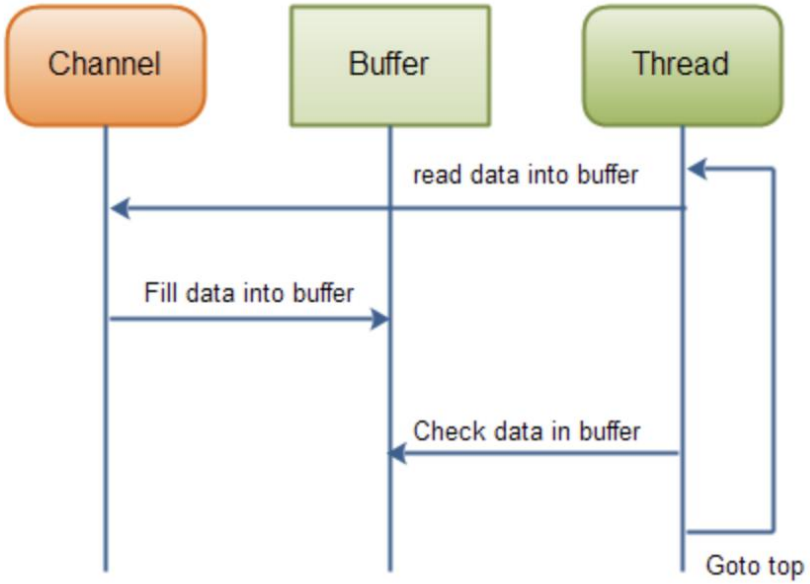
IO	NIO
Stream oriented	Buffer oriented
Blocking IO	Non blocking IO
	Selectors

线程阻塞
等待东西传过来
基于流

基于buffer



Java IO: Reading data from a blocking stream.



Java NIO: Reading data from a channel until all needed data is in buffer.

- http://ifeve.com/overview/