Chapter 17

How to work with file I/O

Objectives

Applied

- Write code that uses the Paths, Path, and Files classes to get information about a file or directory.
- Write code that reads and writes data to a text file using buffered readers and writers.

Knowledge

- Explain the differences between a text file and a binary file.
- Explain how you can layer classes to create filtered streams that can read or write files.
- Explain what a buffer is and describe how it improves the performance of an I/O operation.
- Name and describe three common types of I/O exceptions.

A package for working with directories and files

```
java.nio.file
```

A static method of the Paths class

```
get(String[, String]...)
```

Methods of the Path interface

```
getFileName()
getName(int)
getNameCount()
getParent()
getRoot()
toAbsolutePath()
toFile()
```

Static methods of the Files class

```
exists(Path)
notExists(Path)
isReadable(Path)
isWritable(Path)
isDirectory(Path)
isRegularFile(Path)
size(Path)
newDirectoryStream(Path)
createFile(Path)
createDirectory(Path)
createDirectories(Path)
delete(Path)
```

Code that creates a directory if it doesn't exist

```
String dirString = "c:/murach/java_netbeans/files";
Path dirPath = Paths.get(dirString);
if (Files.notExists(dirPath)) {
    Files.createDirectories(dirPath);
}
```

Code that creates a file if it doesn't exist

```
String fileString = "products.txt";
Path filePath = Paths.get(dirString, fileString);
if (Files.notExists(filePath)) {
    Files.createFile(filePath);
}
```

Code that displays information about a file

```
System.out.println("File name:      " +
    filePath.getFileName());
System.out.println("Absolute path:  " +
    filePath.toAbsolutePath());
System.out.println("Is writable:    " +
    Files.isWritable(filePath));
```

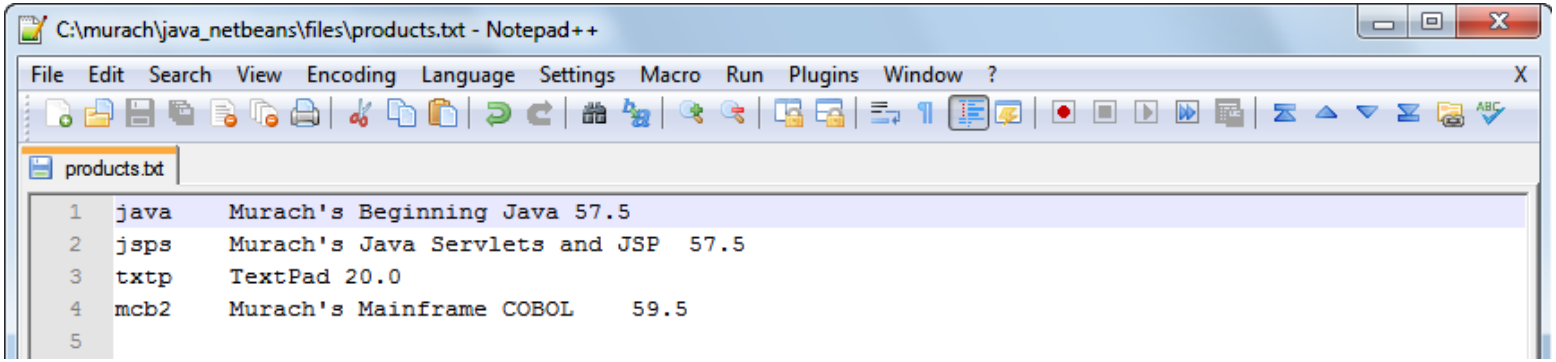
Resulting output

```
File name:      products.txt
Absolute path:  c:\murach\java_netbeans\files\products.txt
Is writable:    true
```

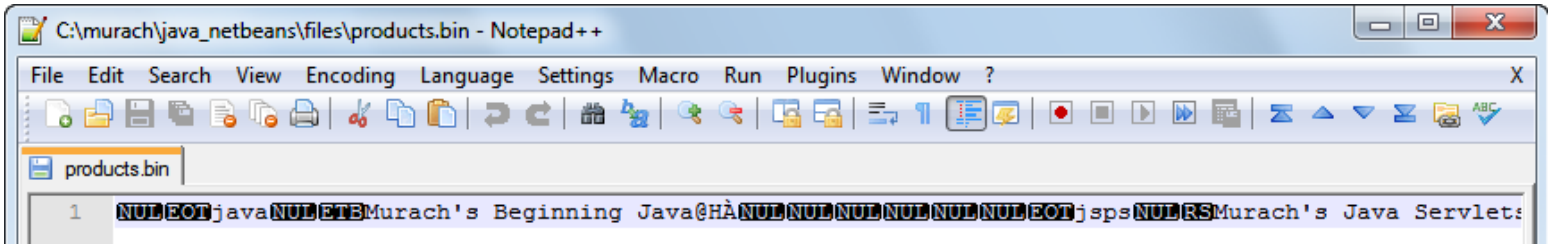
Code that displays the files in a directory

```
if (Files.exists(dirPath) &&
    Files.isDirectory(dirPath)) {

    System.out.println("Directory: " +
         dirPath.toAbsolutePath());

    System.out.println("Files: ");

    DirectoryStream<Path> dirStream =
        Files.newDirectoryStream(dirPath);

    for (Path p: dirStream) {
        if (Files.isRegularFile(p)) {
            System.out.println("     " +
                p.getFileName());
        }
    }
}
```

A text file that's opened by a text editor



A binary file that's opened by a text editor



Two types of files

- **Text**
- **Binary**

Two types of streams

- **Character**
- **Binary**

Import all necessary packages

```
import java.io.*;
import java.nio.file.*;
```

Get a Path object for the file

```
Path productsPath = Paths.get("products.txt");
File productsFile = productsPath.toFile();
```

Write data to the file

```
try (PrintWriter out = new PrintWriter(
                       new BufferedWriter(
                       new FileWriter(productsFile)))) {
    out.println("java\tMurach's Beginning Java\t57.50");
}
catch (IOException e) {
    System.out.println(e);
}
```

Read data from the file

```
try (BufferedReader in = new BufferedReader(
                         new FileReader(productsFile))) {
    String line = in.readLine();
    System.out.println(line);
}
catch (IOException e) {
    System.out.println(e);
}
```

Resulting output

```
java    Murach's Beginning Java          57.50
```

A subset of the IOException hierarchy

```
IOException
    EOFException
    FileNotFoundException
```

Common I/O exceptions

```
IOException
EOFException
FileNotFoundException
```

Code that handles I/O exceptions

```
Path productsPath = Paths.get("products.txt");
if (Files.exists(productsPath)) { // no FileNotFoundException
    File productsFile = productsPath.toFile();
    try (BufferedReader in = new BufferedReader(
                                new FileReader(productsFile))) {
            String line = in.readLine();
            while(line != null) { // no EOFException
                System.out.println(line);
                line = in.readLine();
            }
    }
    catch (IOException e) {       // catch IOException
        System.out.println(e);
    }
}
else {
    System.out.println(
        productsPath.toAbsolutePath() + " doesn't exist");
}
```

A subset of the Writer hierarchy

```
Writer <<abstract>>
    BufferedWriter
    PrintWriter
    OutputStreamWriter
        FileWriter
```

Classes used to connect a character output stream to a file

`PrintWriter`	writes data to a text stream
→`BufferedWriter`	creates a buffer for the stream
→`FileWriter`	connects the stream to a file

Constructors of these classes

Constructor	Throws
`PrintWriter(Writer[, booleanFlush])`	None
`BufferedWriter(Writer)`	None
`FileWriter(File[, booleanAppend])`	IOException
`FileWriter(StringPathName[, booleanAppend])`	IOException

How to connect without a buffer (not recommended)

```
FileWriter fileWriter = new FileWriter("products.txt");
PrintWriter out = new PrintWriter(fileWriter);
```

A more concise way to code the previous example

```
PrintWriter out = new PrintWriter(
    new FileWriter("products.txt"));
```

How to connect to a file with a buffer

```
PrintWriter out = new PrintWriter(
                  new BufferedWriter(
                  new FileWriter("products.txt"));
```

How to connect for an append operation

```
PrintWriter out = new PrintWriter(
                  new BufferedWriter(
                  new FileWriter("products.txt", true)));
```

How to connect with the autoflush feature

```
PrintWriter out = new PrintWriter(
                  new BufferedWriter(
                  new FileWriter("products.txt")), true);
```

Common methods of the PrintWriter class

Method	Throws
`print(argument)`	None
`println(argument)`	None
`flush()`	IOException
`close()`	IOException

Code that appends data to a text file

```
// open an output stream for appending to the text file
PrintWriter out = new PrintWriter(
                  new BufferedWriter(
                  new FileWriter("log.txt", true)));

// write a string and an object to the file
out.print("This application was run on ");
LocalDateTime currentDateTime = LocalDateTime.now();
out.println(currentDateTime);

// flush data to the file and close the output stream
out.close();
```

Code that writes data to a tab-delimited text file

```
// open an output stream for overwriting a text file
PrintWriter out = new PrintWriter(
                  new BufferedWriter(
                  new FileWriter(productsFile)));

// write the Product object to the file
out.print(product.getCode() + "\t");
out.print(product.getDescription() + "\t");
out.println(product.getPrice());

// flush data to the file and close the output stream
out.close();
```

A subset of the Reader hierarchy

```
Reader <<abstract>>
    BufferedReader
    InputStreamReader
        FileReader
```

Classes used to connect to a file with a buffer

`BufferedReader`	reads data from the stream
→`FileReader`	connects the stream to a file

Constructors of these classes

Constructor	Throws
`BufferedReader(Reader)`	`None`
`FileReader(File)`	`FileNotFoundException`
`FileReader(StringPathName)`	`FileNotFoundException`

How to connect a character input stream to a file

```
BufferedReader in = new BufferedReader(
                    new FileReader("products.txt"));
```

Common methods of the BufferedReader class

Method	Throws
`readLine()`	`IOException`
`close()`	`IOException`

Code that reads the records in a text file

```
// read the records of the file
String line;
while((line = in.readLine()) != null) {
    System.out.println(line);
}

// close the input stream
in.close();
```

Sample output

```
This application was run on 2015-05-28T12:06:55.084
This application was run on 2015-05-28T12:07:28.041
```

A method of the String class

```
split(delimiter)
```

Code that reads data from a tab-delimited text file

```
// read the next line of the file
String line = in.readLine();

// parse the line into its columns
String[] columns = line.split("\t");
String code = columns[0];
String description = columns[1];
String price = columns[2];

// create a Product object from the data in the columns
Product p = new Product(code, description,
   Double.parseDouble(price));

// print some Product object data
System.out.println(p.getDescription() + " (" +
   p.getPriceFormatted() + ")");

// close the input stream
in.close();
```

Sample output

```
Murach's Beginning Java ($57.50)
```

The code for the ProductIO class

```
package murach.io;

import java.util.*;
import java.io.*;
import java.nio.file.*;
import murach.business.Product;

public class ProductIO {
    private static final Path productsPath =
       Paths.get("products.txt");
    private static final File productsFile =
        productsPath.toFile();
    private static final String FIELD_SEP = "\t";
    private static List<Product> products = getAll();

    // prevent instantiation of class
    private ProductIO() {}
```

The code for the ProductIO class (cont.)

```
public static List<Product> getAll() {
    // if the products file has already been read
    // don't read it again
    if (products != null) {
        return products;
    }

    products = new ArrayList<>();
    if (Files.exists(productsPath)) {
        try (BufferedReader in = new BufferedReader(
                new FileReader(productsFile))) {

            // read products from file to array list
            String line = in.readLine();
            while(line != null) {
              String[] columns = line.split(FIELD_SEP);
              String code = columns[0];
              String description = columns[1];
              String price = columns[2];
```

The code for the ProductIO class (cont.)

```
                Product p = new Product();
                p.setCode(code);
                p.setDescription(description);
                p.setPrice(Double.parseDouble(price));
                products.add(p);

                line = in.readLine();
            }
        }
        catch(IOException e) {
            System.out.println(e);
            return null;
        }
    }
    return products;
}
```

The code for the ProductIO class (cont.)

```
public static Product get(String code) {
    for (Product p : products) {
        if (p.getCode().equals(code))
            return p;
    }
    return null;
}
```

The code for the ProductIO class (cont.)

```
private static boolean saveAll() {
    try (PrintWriter out = new PrintWriter(
                        new BufferedWriter(
                        new FileWriter(productsFile)))) {

        // write products from array list to file
        for (Product p : products) {
            out.print(p.getCode() + FIELD_SEP);
            out.print(p.getDescription() + FIELD_SEP);
            out.println(p.getPrice());
        }
    }
    catch(IOException e) {
        System.out.println(e);
        return false;
    }
    return true;
}
```

The code for the ProductIO class (cont.)

```
    public static boolean add(Product p) {
        products.add(p);
        return saveAll();
    }

    public static boolean delete(Product p) {
        products.remove(p);
        return saveAll();
    }

    public static boolean update(Product newProduct) {
        // get the old product and remove it
        Product oldProduct = get(newProduct.getCode());
        int i = products.indexOf(oldProduct);
        products.remove(i);

        // add the updated product
        products.add(i, newProduct);
        return saveAll();
    }
}
```

The console

```
Welcome to the Product Manager

COMMAND MENU
list    - List all products
add     - Add a product
del     - Delete a product
help    - Show this menu
exit    - Exit this application

Enter a command: list

PRODUCT LIST
java        Murach's Java Programming         $57.50
jsp         Murach's Java Servlets and JSP    $57.50
mysql       Murach's MySQL                    $54.50
android     Murach's Android Programming      $57.50
html5       Murach's HTML5 and CSS3           $54.50
oracle      Murach's Oracle and PL/SQL        $54.50
javascript  Murach's JavaScript and jQuery    $54.50
```

The console (cont.)

```
Enter a command: add

Enter product code: bjava
Enter product description: Murach's Beginning Java
Enter price: 54.50

Murach's Beginning Java was added to the database.

Enter a command: del

Enter product code to delete: bjava

Murach's Beginning Java was deleted from the database.

Enter a command: exit

Bye.
```

The Main class

```
package murach.ui;

import java.util.List;
import murach.business.Product;
import murach.io.ProductIO;

public class Main {

    public static void main(String args[]) {

        // display a welcome message
        System.out.println(
            "Welcome to the Product Manager\n");

        // display the command menu
        displayMenu();
```

The Main class (cont.)

```
// perform 1 or more actions
String action = "";
while (!action.equalsIgnoreCase("exit")) {

    // get the input from the user
    action = Console.getString(
        "Enter a command: ");
    System.out.println();
```

The Main class (cont.)

```
            if (action.equalsIgnoreCase("list")) {
                displayAllProducts();
            } else if (action.equalsIgnoreCase("add")) {
                addProduct();
            } else if (action.equalsIgnoreCase("del") ||
                       action.equalsIgnoreCase("delete")) {
                deleteProduct();
            } else if (action.equalsIgnoreCase("help") ||
                       action.equalsIgnoreCase("menu")) {
                displayMenu();
            } else if (action.equalsIgnoreCase("exit")) {
                System.out.println("Bye.\n");
            } else {
                System.out.println(
                 "Error! Not a valid command.\n");
            }
        }
    }
```

The Main class (cont.)

```
public static void displayMenu() {
    System.out.println("COMMAND MENU");
    System.out.println("list    - List all products");
    System.out.println("add     - Add a product");
    System.out.println("del     - Delete a product");
    System.out.println("help    - Show this menu");
    System.out.println("exit    - Exit application\n");
}
```

The Main class (cont.)

```
public static void displayAllProducts() {
    System.out.println("PRODUCT LIST");
    List<Product> products = ProductIO.getAll();
    if (products == null) {
        System.out.println(
            "\nError! Unable to get products.\n");
    } else {
        Product p;
        StringBuilder sb = new StringBuilder();
        for (Product product : products) {
            p = product;
            sb.append(StringUtil.padWithSpaces(
                    p.getCode(), 12));
            sb.append(StringUtil.padWithSpaces(
                    p.getDescription(), 34));
            sb.append(p.getPriceFormatted());
            sb.append("\n");
        }
        System.out.println(sb.toString());
    }
}
```

The Main class (cont.)

```
public static void addProduct() {
    String code = Console.getString(
        "Enter product code: ");
    String description = Console.getString(
        "Enter product description: ");
    double price = Console.getDouble(
        "Enter price: ");

    Product product = new Product();
    product.setCode(code);
    product.setDescription(description);
    product.setPrice(price);

    ProductIO.add(product);
    System.out.println("\n" + description
            + " was added to the database.\n");
}
```

The Main class (cont.)

```
    public static void deleteProduct() {
        String code = Console.getString(
            "Enter product code to delete: ");

        Product product = ProductIO.get(code);
        if (product == null) {
            System.out.println(
                "\nError! Unable to get product.");
        } else {
            ProductIO.delete(product);
            System.out.println("\n" +
                product.getDescription() +
                " was deleted from the database.\n");
        }
    }
}
```