
MAHARISHI UNIVERSITY of MANAGEMENT
Engaging the Managing Intelligence of Nature
Computer Science Department
CS390 Foundamental Programming Practices (FPP)
Professor Paul Corazza

Lecture 13:
Working with Files and Databases

Wholeness of the Lesson

Java provides convenient tools for reading and writing files, and for accessing data stored in a database. The relationship between stored data and an executing program parallels the relationship between awareness and its interaction with the world; that interaction is most successful and rewarding if awareness is broad (corresponding to a well-designed program) and is well integrated with the laws of nature, with ways of manifest existence (JDBC).

File I/O in Java

- Java supports reading and writing text streams and binary streams. In this lesson we focus on the API for text streams.

- Internally, Java represents each character with the UTF-16 encoding, which is a 16-bit encoding (= 2 bytes represented by 4 hex digits). For instance, in this encoding, 'A' is represented by `\u0041` (which is the pair of bytes 0, 65 in base 10). So, with this way of encoding characters, the String "ABC" would be represented by the byte sequence [0, 65, 0, 66, 0, 67]. However, the default encoding for the Western world is not UTF-16 but ISO-Latin, which encodes "ABC" as [65, 66, 67]. So, though Java's internal representation of characters uses UTF-16, for purposes of reading and writing text, it uses ISO-Latin encoding as the default (but other encodings can be specified).

Example

```
Charset utf16 = Charset.forName("UTF-16BE");
Charset isolatin = Charset.forName("ISO-8859-1");
String abc = "ABC";
byte[] asUtf16 = abc.getBytes(utf16);
byte[] defaultEnc = abc.getBytes();
byte[] asIsolatin = abc.getBytes(isolatin);

printArray(asUtf16);
printArray(defaultEnc);
printArray(asIsolatin);

String newAbc = new String(asIsolatin);
System.out.println(newAbc);
```

//output

```
[0, 65, 0, 66, 0, 67]
[65, 66, 67]
[65, 66, 67]
ABC
```

Readers

- `Reader` is the superclass of all “readers” in Java, which offer the ability to read in streams of unicode characters in various convenient ways.

- `InputStreamReader` converts raw bytes from some input source to character data, using, by default, the ISO-Latin encoding (other encodings can be specified). `BufferedReader` organizes data stored in a `Reader` object to be read in convenient ways.

Example. Begin with a file `text.txt` containing the line of text “This is test text." (code in the next slide)

```
try {
    InputStream stream1 = System.in;
    InputStream stream2 = new FileInputStream("text.txt");
    BufferedReader reader1 =
      new BufferedReader(new InputStreamReader(stream1));
    BufferedReader reader2 =
      new BufferedReader(new InputStreamReader(stream2));
    System.out.println(reader2.readLine());
    System.out.print("Type something: ");
    System.out.println(reader1.readLine());
    reader1.close();
    reader2.close();
    stream1.close();
    stream2.close();
}

catch(IOException e) {
    System.out.println(e.getMessage());
}
```

//output

```
This is test text.
Type something: hi
hi
```

- If there is no explicit need to convert from raw bytes to characters (this is necessary for example when reading from System.in), the concept of an “input stream” is absorbed into the functionality of readers, so the developer never needs to work with the low level of bytes.

  Use the following Examples as models for how to read text files.

Example:

```
//uses a FileReader
try {
     FileReader reader = new FileReader("text.txt");
     BufferedReader bufreader = new BufferedReader(reader);
     String line = null;
     while((line = bufreader.readLine()) != null){
        System.out.println(line);
     }
     bufreader.close();
     reader.close();
}
catch(IOException e) {
     e.printStackTrace();

}
```

Example:

```
//uses a Scanner
try {
     Scanner sc = new Scanner(new File("text.txt"));
     String line = null;
     while(sc.hasNextLine()) {
        line = sc.nextLine();
        System.out.println(line);
     }
     sc.close();
}
catch(IOException e) {
     e.printStackTrace();
}
```

Writers

- Similarly, there is an OutputStreamWriter that converts raw bytes to a Unicode character stream as output. Convenience methods in a PrintWriter make it possible to format output using print, println, and printf methods, familiar from System.out.

Example:

```
try {
   OutputStream stream1 = System.out;
   OutputStream stream2 =
      new FileOutputStream("text2.txt");
   PrintWriter writer1 =
      new PrintWriter(new
      OutputStreamWriter(stream1));
   PrintWriter writer2 =
      new PrintWriter(new
      OutputStreamWriter(stream2));
   writer1.println("output to console");
   writer2.println("output to file");
   writer1.close();
   writer2.close();
   stream1.close();
   stream2.close();
   }
catch(IOException e) {
   System.out.println(e.getMessage());
}
```

- When there is no explicit need to work with bytes directly, PrintWriter can be used without reference to the conversion class OutputStreamWriter.

The following example is a model for writing text data.

Example

```
try{
    PrintWriter writer =
       new PrintWriter("text3.txt");
    writer.printf("output to %s", "file");
    writer.close();
}
catch(IOException e) {
    System.out.println(e.getMessage());
}
```

- See package lesson13.files

The File Class

1. The `File` class is an abstraction that represents either a file or a directory on the native system's directory system.
2. Methods available in `File` include:
   - `boolean isFile`
   - `boolean isDirectory`
   - `boolean exists`
   - `String getAbsolutePath`
   - `String getParent`
   - `File getParentFile`
   - `boolean mkdir`
   - `boolean mkdirs`
   - `boolean delete`
   - `File[] listFiles`

Main Point

Reading a File in Java is accomplished by using a FileReader (or Scanner). Writing to a File is accomplished by using a FileWriter. More generally, "input" in human life is handled by the senses; "output" is handled by the organs of action. Both have their source in the field of pure creative intelligence.

Interacting with a Database

- JDBC provides an API for interacting with a database using SQL – part of the jdk distribution.

- To interact efficiently with a database, you use the database vendor's *driver* that allows communication between the JVM and the database.

Set-up steps for using JDBC

- Install the database software. In this course we will use JavaDb (originally called Apache Derby). Files for JavaDb come with the jdk distribution; no special installation is required.

- Locate the JDBC driver. This is the platform specific software that the JDBC API uses to communicate with your database. For JavaDB, the jar file is **`derby.jar`**. Typical location:

  C:\Program Files (x86)\Java\jdk1.8.0_45\db\lib

  The jar file will be added to your Eclipse project.

Eight Coding Steps for JDBC

1. Place the driver on the classpath. This is usually done by adding the jar file (for JavaDB, it's derby.jar) as an external jar for the project.

2. Assemble your database url (dburl) that will tell JDBC the name of your data source– typically, store this database url as a constant. The dburl we will use is:

   ```
   jdbc:derby://localhost:1527/FPP_DB;create=true
   ```

   For a MySql database, a typical dburl is:

   ```
   jdbc:mysql:///MyDatabase
   ```

3. Load the driver *if necessary*. This ensures that your code has access to the contents of the driver
   a. Typical line of code to load driver (for a MySql database)

      ```
      Class.forName("org.gjt.mm.mysql.Driver");
      ```

   b. Drivers that meet the JDBC-4 standard are *self-registering* – the jar file containing the driver contains information that allows the driver register itself. JavaDB does it this way, so no special coding is needed to load the driver for JavaDB.

(continued)

4. Create a `Connection` object You pass in the dburl at this stage. Typical line of code for this:

```
DriverManager.getConnection(DB_URL, USERNAME, PASSWORD);
```

   The DB vendor tells you the default username and password for access to the database; for a real project, usernames and passwords will be created and managed carefully.

5. Create a Statement object. The Statement object allows you to execute SQL statements. You create one Statement object for each query; Statement objects are not re-used.

6. Execute an SQL statement. Prepare a String consisting of an SQL query, and pass it to the Statement instance, using either

   `executeQuery(String sql)` (for reads), or

   `updateQuery(String sql)` (for updating or inserting).

7. Process ResultSet. Return value of these query methods is a `ResultSet`. A `ResultSet` encapsulates the rows of data that have been read. Typical code:

```
ResultSet rs = stmt.executeQuery(<your query>);
if(rs.next())
    catalogid = rs.getString("catalogid");
```

8. Close. After a Statement has been used, close it. After you have finished with the database, close the Connection object, with its close() method.

Main Point

JDBC provides an API for interacting with a database using SQL. To interact efficiently with a database, you typically use the database vendor's driver that allows communication between the JVM and the database. This is reminiscent of the Principle of Diving – once the initial conditions have been met, a good dive is automatic. (Here, the initial conditions are correct configuration of the data source and code to load the database driver; once the set up is right, interacting with the database is "effortless".)

Connecting the Parts of Knowledge With the Wholeness of Knowledge

Expansion of consciousness leads to expanded territory of influence

1. Since Java is an OO language, it supports storage and manipulation of data within appropriate objects.
2. To work with real data effectively, Java supports interaction with external data stores (databases) through the use of various JDBC drivers, and the JDBC API.

---

3. **Transcendental Consciousness:** TC is the field of truth, the field of Sat. "Know that by which all else is known." -- Upanishads
4. **Wholeness moving within itself**: In Unity Consciousness, the final truth about life is realized in a single stroke of knowledge.