File Handling

Contents

- Java provides two ways to handle files
  - Serial access
  - Random access

Serial Access Files

- Serial access is where data records are stored one after the other with no regard to the order.
  - each new item of data being added to the end of the file
- The internal structure of a serial file can be either **binary** or **text**.
- In java, we use the $File$ class for reading and writing files to the hard disk.
  - Class $File$ is contained within package $java.io$
- Examples:

$$File\ results = new\ File("c:\\data\\results.txt");$$

- To read a file, we need to wrap a $File$ object around a $Scanner$ object.
  - We use the $next, nextLine$, $nextInt$, $nextFloat$, … for reading.

$$Scanner\ input\ =\ new\ Scanner(new\ File("inFile.txt"));$$

- To write to a file, we wrap a $File$ object around a $PrintWriter$ object.
  - We use $print$ and $println$ for writing.

$$PrintWriter\ output$$
$$=\ new\ PrintWriter(new\ File("outFile.txt"));$$

- Example:

```
String file_name = "F:\\fci\\Network
programming\\Java Workspace\\src\\CH04\\data.txt";
File myfile = new File(file_name);
PrintWritear writer = new PrintWriter(myfile);
writer.println("Welcome");
writer.println(5);
writer.println(14/7);
writer.println(true);
writer.close();
Scanner reader = new Scanner(myfile);
String x1 = reader.next();
int x2 = reader.nextInt();
float x3 = reader.nextFloat();
boolean x4 = reader.nextBoolean();
System.out.println(x1);
System.out.println(x2);
System.out.println(x3);
System.out.println(x4);
```

Quiz: What if we change the text "$Weclome$" to "$Welcome\ to\ Network\ Programming$"?

- It is very important to close the file after writing to ensure that the file buffer has been emptied and all data written to the file.
- If we open an existing file and write again to it, the data of the file will be overwritten.

```
File file = new File("F:\\fci\\Network
programming\\Java Workspace\\src\\CH04\\data.txt");
PrintWriter writer = new PrintWriter(file);
writer.println("This is a new Text");
writer.close();
```

  - How to solve this problem?

- To append data to an existing file, we can use the $FileWriter$ object.

$$FileWriter(String < fileName >, boolean < append >)$$
$$FileWriter(File < fileName >, boolean < append >)$$

- Example

```
FileWriter file = new FileWriter("F:\\fci\\Network
programming"\\Java Workspace\\src\\CH04\\data.txt",
true);
file.write("This is another text!");
file.close();
```

File Methods

Method	Description
$\boldsymbol{boolean\ canRead}()$	Returns true if file is readable and false otherwise.
$\boldsymbol{boolean\ canWrite}()$	Returns true if file is writeable and false otherwise.
$\boldsymbol{boolean\ delete}()$	Deletes file and returns true/false for success/failure.
$\boldsymbol{boolean\ exists}()$	Returns true if file exists and false otherwise.
$\boldsymbol{String\ getName}()$	Returns name of file.
$\boldsymbol{boolean\ isDirectory}()$	Returns true if object is a directory/folder and false otherwise.
$\boldsymbol{boolean\ isFile}()$	Returns true if object is a file and false otherwise.
$\boldsymbol{long\ length}()$	Returns length of file in bytes.
$\boldsymbol{String}[]\ \boldsymbol{list}()$	Returns array of File objects.
$\boldsymbol{boolean\ mkdir}()$	Creates directory with name of current File object.

- Example: An application for telling whether the input is a file or a directory. If it is a file, the application tells whether it is readable and writable or not and prints its size. If the input is a directory, the application tells whether the directory is readable and writable or not and prints the content of the folder.

```
String filename;
Scanner input = new Scanner(System.in);
System.out.print("Enter name of file/directory ");
System.out.print("or press <Enter> to quit: ");
filename = input.nextLine();
while (!filename.equals("")){ //Not <Enter> key.
    File fileDir = new File(filename);
    if (!fileDir.exists()) {
        System.out.println(filename + " does not exist!");
        break; //Get out of loop.
    }
    if (fileDir.isFile())
        System.out.println("It is a file.");
    else
        System.out.println("It is a directory.");
    if (fileDir.canRead())
        System.out.print("It is readable.");
    else
        System.out.println("It is not readable.");
    if (fileDir.canWrite())
        System.out.println("It is writable");
    else
        System.out.println("It is not writable");

    if (fileDir.isDirectory()) {
        System.out.println("Contents:");
        String[] fileList = fileDir.list();
        for (int i = 0; i < fileList.length; i++)
            System.out.println(" " + fileList[i]);
    } else {
        System.out.print("Size of file: ");
        System.out.println(fileDir.length() + " bytes.");
    }
    System.out.print("\n\nEnter name of next file/directory ");
    System.out.print("or press <Enter> to quit: ");
    filename = input.nextLine();
}
input.close();
```

Command Line Parameters

- To run a java program using cmd
  - Add java bin folder to the path
  - Compile the file using $javac\ filename.java$
  - Execute the file using $java\ filename.class$
- It is possible to supply values (command line parameters) in addition to the name of the program to be executed.
  - The parameters are stored in the array $args$.
- Example: an application that copies the data of one file (the first parameter) to a new file (the next parameter).

```
public static void main(String[] args) throws
FileNotFoundException {
    if (args.length < 2) {
        System.out.println(
                "You must supply TWO file names.");
        System.out.println("Syntax:");
        System.out.println(
                " java Copy <source> <destination>");
        return;
    }
    Scanner source = new Scanner(new File(args[0]));
    PrintWriter destination =
            new PrintWriter(new File(args[1]));
    String input;
    while (source.hasNext()) {
        input = source.nextLine();
        destination.println(input);
    }
    source.close();
    destination.close();
}
```

  - Run the terminal
  - Compile the file: $javac\ Copy.java$
  - Run the file: $java\ CH04.Copy\ CH04/data.txt\ CH04/new.txt$

Random Access Files

- Random access files (direct files) permit nonsequential, or random, access to a file's contents.
- To create a random access file in Java, we create a $RandomAccessFile$ object. The constructor takes two arguments:
  - a string or File object identifying the file.
  - a string specifying the file's access mode.
- The mode value can be:

"r"	Open for reading only. Invoking any of the write methods of the resulting object will cause an $IOException$ to be thrown.
"rw"	Open for reading and writing. If the file does not already exist, then an attempt will be made to create it.
"rws"	Open for reading and writing, as with "rw", and require that every update to the file's content or metadata be written synchronously to the underlying storage device.
"rwd"	Open for reading and writing, as with "rw", and require that every update to the file's content be written synchronously to the underlying storage device.

- Using "rwd" only requires updates to the file's content to be written to storage; using "rws" requires updates to both the file's content and its metadata to be written.
- Example:

$$RandomAccessFile\ ranFile = new\ RandomAccessFile("accounts.dat", "rw");$$

- Before reading or writing a record, it is necessary to position the file pointer.
  - We do this by calling method seek, which requires a single argument specifying the byte position within the file.

$ranFile.seek(500)$; // move the pointer to the 501st byte

- To move the pointer to the correct position for a particular record, we need to know the size of each record.
    - In java, the numeric fields are

int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes

    - For strings, each character is stored in Unicode format, which requires 2 bytes.
- Class $RandomAccessFile$ provides the following methods for manipulating the above types:
    - $readInt, readLong, readFloat, readDouble, readChar$
    - $writeInt, writeLong, writeFloat, writeDouble, writeChar$

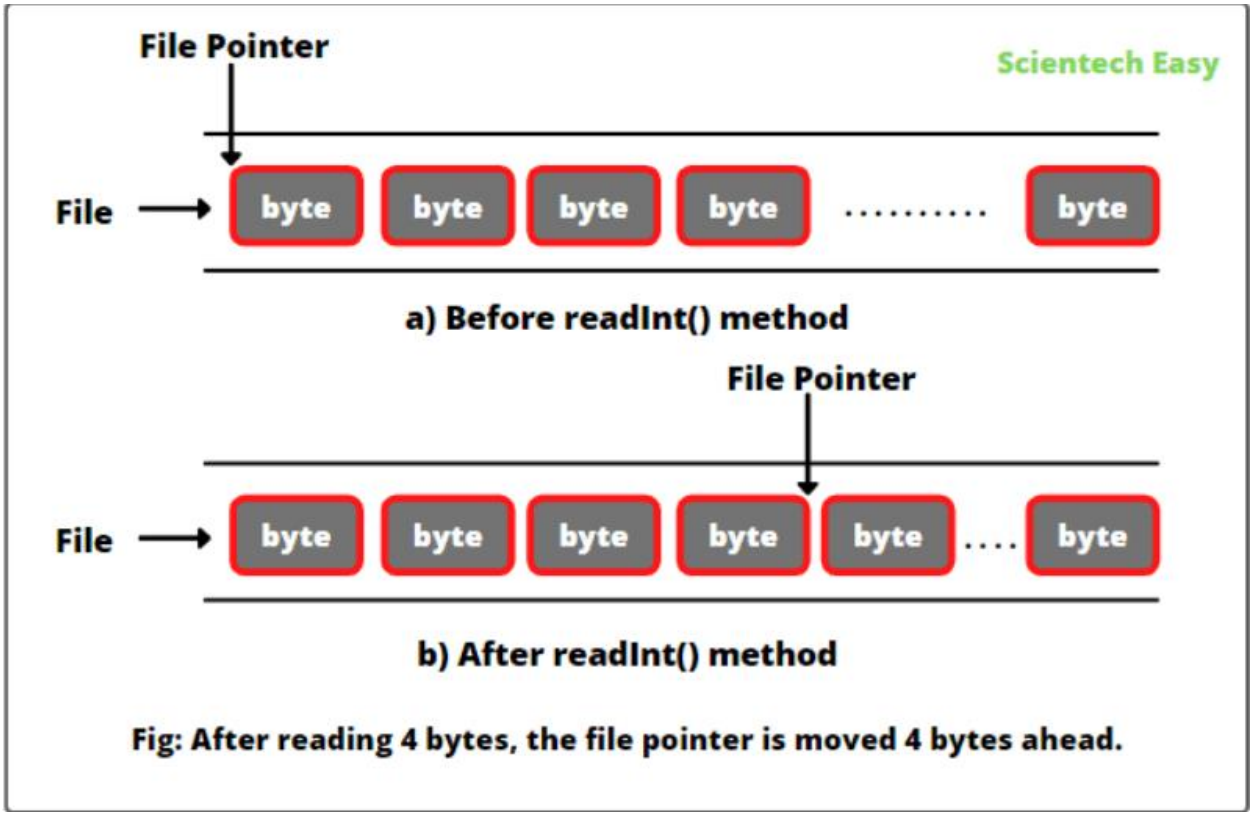

Fig: After reading 4 bytes, the file pointer is moved 4 bytes ahead.

- Example 1:

```
RandomAccessFile file = new
RandomAccessFile("myfile.dat", "rw");


file.writeChar('S');
file.writeInt(2222);
file.writeDouble(222.22);

file.seek(0); // Moving file pointer to the
beginning.

System.out.println(file.readChar());
System.out.println(file.readInt());
System.out.println(file.readDouble());

file.seek(2);
System.out.println(file.readInt());


file.seek(file.length());
file.writeBoolean(true);


file.seek(4);
System.out.println(file.readBoolean());
file.close(); // Closing stream.
```

- Example 2:

  Suppose we wish to set up an accounts file with the following fields:
  - account number (long); $long$ requires 8 bytes
  - surname (String); we will allocate 15 Unicode characters
  - initials (String); we will allocate 3 Unicode characters
  - balance (float); $float$ requires 4 bytes

Now, the one record requires $8 + 15 * 2 + 3 * 2 + 4 = 48$ bytes.
The formula for calculating the position of any record on the file is $(record\ number - 1) * 48$.
To calculate the number of records in a file, get the length of the file then divide it by the length of the individual record

$$ranAccts.length()/48$$

Exercises

- What is I/O redirection in java? Give a code example.
- What is Serialization in java? Give a code example.
- Compare between serial access and random-access files in terms of their definition, usage, advantages, and disadvantages.
- What is $JFileChooser$ in java? Give a code example.