CS205 Object Oriented Programming in Java

Module 3 - More features of Java (Part 5)

Prepared by

Renetha J.B.

AP

Dept.of CSE,

Lourdes Matha College of Science and Technology

Topics

- **More features of Java :**
  - Working with **Files**

Working with Files

- In Java, all files are **byte-oriented**.
- Java provides methods to
  - read bytes from a file and
  - write bytes to a file.

till now we studied alternatives of scanner and print() andd object se kaise data extract kiya ja saktha hai

Working with Files(contd.)

- Two of the most often-used file stream classes are

  - **FileInputStream**
    - FileInputStream is an input stream to **read data** from a file in the form of sequence of bytes
  - **FileOutputStream**
    - **FileOutputStream** class is an output stream for **writing data** to a file

Working with Files- OPEN a file

- **To open a file,**
  - create an object of one of these classes
  - specify the name of the file as an argument to the constructor.
- If we want to open a file for **reading**
  - Create object of FileInputStream class
- If we want to open a file for **writing**
  - Create object of FileOutputStream class

read from a file
&
write to a file

Working with Files(OPEN a file contd.)

- Main constructors are

FileInputStream(String *fileName*) throws FileNotFoundException

FileOutputStream(String *fileName*) throws FileNotFoundException

  - Here, *fileName* specifies the name of the file (as String i.e. enclose in double quotes) that we want to open.
  - When we create an **input stream**, if the file does not exist, then **FileNotFoundException** is thrown**.**
  - For **output streams**, if the file cannot be created, then **FileNotFoundException is thrown.**
    - When an ***output file*** is opened, any file that is already existing with the same name as output file is destroyed.

Working with Files(OPEN a file) contd.

- To open a file for **reading-**

  We have to create **FileInputStream** class object and pass *filename* as the parameter to the constructor.

E.g. to open the file Sample.txt for reading

```
FileInputStream fileobject;

fileobject = new FileInputStream("Sample.txt");
```

Working with Files(OPEN a file) contd.

- To open a file for **writing**

  We have to create **FileOutputStream** class object and pass *filename* as the parameter to the constructor.

E.g. to open the file Sample.txt for writing

```
FileOutputStream fileobject;
fileobject = new FileOutputStream(“Sample.txt");
```

Working with Files(closing a file)

- After completing file read or write operations, we should **close** the file by calling **close( ).**
- It is defined by both **FileInputStream** and **FileOutputStream :**

```
void close( ) throws IOException
```

Working with Files(closing a file) contd.

E.g. to close file Sample.txt opened for reading

```
FileInputStream fileobject;

fileobject = new FileInputStream(“Sample.txt");

//statements for reading the file

fileobject.close();
```

Working with Files(read a file)

- To **read data** from a file,

  1. First, we have to create **FileInputStream** class object and pass *filename* as the parameter to the constructor.

     E.g.
     ```
     FileInputStream fileobject;
     fileobject = new FileInputStream("Sample.txt");
     ```

  2. Next,we can use a version of **read( )** that is defined within **FileInputStream.**

     ```
     int read( ) throws IOException
     ```

     E.g.
     ```
     int c=fileobj.read();
     ```

- Each time read() called, it reads a single byte from the file and returns the byte as an integer value.
  - **read( ) returns –1 when the end of the file** is encountered.
  - read() can throw an **IOException.**

Read contents from file(E.g)

Write a program to read &display the contents in the file Sample.txt

```
import java.io.*;
class Readfile
{
public static void main(String arg[]) throws IOException
{
FileInputStream f;
try
{
f= new FileInputStream(“Sample.txt");
int c;
    do    {
          c=f.read();
          if(c!=-1)
          {System.out.print((char)c);}
          } while(c!=-1);
}
```

```
catch(FileNotFoundException e)
{
System.out.println("File not found");
return;
}
f.close();


}
}
```

Working of file read program

- In the above program, to read the file, an object pf **FileInputStream** class is created.

FileInputStream f;

f= new FileInputStream(“Sample.txt");

- Here the argument of the constructor in FileInputStream is the name of the file to be read. Here “Sample.txt”
- The following statement **read one byte** from the file and store in integer variable c

  c=f.read();

Working of file read program(contd.)

- The following statement **converts the integer variable c into character** using char(c) and print that character on the output screen (console)

  System.out.print((char)c);

This continues until c is equal to -1(end of file)

Working of file read program(contd.)

- ***Exceptions***(run time error) like FileNtotFoundException (if the given filename is not in the system path) may occur during file operations.
  - So it is better to **enclose file operation statements within try block** for handling exceptions.
- If the file we try to read a file that does not exist, then that exception is caught by the following catch block and the corresponding action in it is done.

```
try{
   //File Operation statements
   }
   catch(FileNotFoundException e)
   {
   System.out.println("File not found");  //print this message if file is not found
   }
```

Working with Files(write to a file)

- To write to a file, we can use the **write( )** method defined by **FileOutputStream.**

```
void write(int byteval) throws IOException
```

  - This method **writes the byte** specified by *byteval to the file.*
  - *Although byteval is declared as* an integer, **only the low-order eight bits are written** to the file.
  - If an error occurs during writing, an **IOException** is thrown

Steps to write data to a file

- To **write data** from a file,
  1. First, we have to create **FileOutputStream** class object and pass *filename* as the parameter to the constructor.
  2. Using write function store the byte value in file

E.g. int c=65;

FileOutputStream fileobject;

fileobject = new **FileOutputStream**("Sample.txt");

fileobject.**write**(*c*);

- Here lower order will be stored.So this will store ASCII value of 65 that is letter A in file Sample.txt

FILE COPY –copy contents from *test.txt* to *cp.txt*


```
import java.io.*;
class Rdfcopy
{ public static void main(String a[]) throws IOException
{
FileInputStream f1=null;
FileOutputStream f2=null;
try
{
f1= new FileInputStream("test.txt");
f2= new FileOutputStream("cp.txt");
int c;
do
{
c=f1.read();
if(c!=-1)
{
f2.write((char)c);
System.out.print((char)c);
}
}while(c!=-1);
}
catch(FileNotFoundException e)
        {
System.out.println("File not found");
return;    }
f1.close();
f2.close();
}
}
```

f2. write (c)

Working of file copy program

- For reading a file, FileInputStream object need to created.
  - Here f1

```
FileInputStream f1=null;
f1= new FileInputStream("test.txt");
```

- For writing to a file, FileOutputStream object need to created
  - Here f2

```
FileOutputStream f2=null;
f2= new FileOutputStream("cp.txt");
```

Working of file copy program(contd)

```
c=f1.read();
if(c!=-1)
{
f2.write((char)c);
System.out.print((char)c);
}
```

char ilottu typecast cheyande need il athu pinneyum tirichu implicityly int lotu cast cheythatte write cheyyu

This means that integer **f1.read()** reads a single byte from file pointed by f1(test.txt) and store in integer variable c.

If c is not -1 (end-of-file) then c is converted int character using(char) casting.

f2.write((char)c);

This statement writes the character equivalent of c into file pointed by f2(cp.txt)

This continues until c==-1

FILE READ - file name given as command line argument

Execution: java Readcommandline test.txt

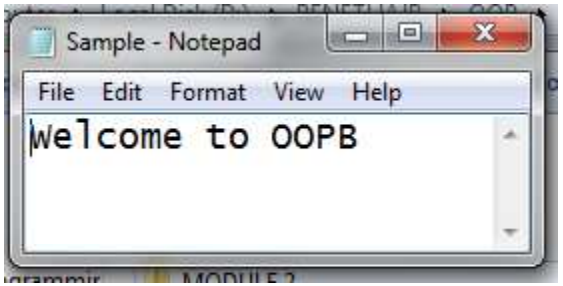
```
import java.io.*;
class Readcommandline
{
public static void main(String arg[]) throws IOException
{
FileInputStream f;
        try
        {
        f= new FileInputStream(arg[0]);
        int c;
                do
                {
                c=f.read();
                if(c!=-1)
                    {System.out.print((char)c);}
                }while(c!=-1);
        }
        catch(FileNotFoundException e)
        {
        System.out.println("File not found");
        return;
}
f.close();}}
```

Example

```
import java.io.*;
 class Writesentencefile
{
public static void main(String arg[]) throws IOException
{
FileOutputStream f;
String s;
try
{
f= new FileOutputStream("Sample.txt");
      s="Welcome to OOP";
byte b[]=s.getBytes();          //converting string into byte array
f.write(b);

 f.write(66);        // write lower bytes. Here we will get ASCII vlue of 66 i.e. letter  B

}
catch(FileNotFoundException e)
{
System.out.println("File not found");
return;
}
f.close();
} }
```



FileReader

- The **FileReader** class creates a **Reader** that we can use to read the contents of a file.
- Its two most commonly used constructors are shown here:

  FileReader(String *filePath)*

  FileReader(File *fileObj)*

  – They can throw a **FileNotFoundException.** Here, ***filePath*** *is the full path name of a file, and* ***fileObj*** *i*s a File object that describes the file.
- The following example shows how to. It reads its own source file, which must be in the current directory.

Read lines from a file and print these to the standard output stream using FileReader


```
import java.io.*;
class FileReaderDemo {
public static void main(String args[]) throws IOException
{
    FileReader fr = new FileReader("Sample.txt");
    BufferedReader br = new BufferedReader(fr);
    String s;
    while((s = br.readLine()) != null)
    {
        System.out.println(s);
    }
fr.close();
}
}
```

FileWriter

- **FileWriter** creates a Writer that you can use to write to a file.
- Its most commonly used constructors are:

  FileWriter(String *filePath)*

  FileWriter(String *filePath, boolean append)*

  FileWriter(File *fileObj)*

  FileWriter(File *fileObj, boolean append)*

  – They can throw an **IOException.** Here, ***filePath*** is the full path name of a file, and ***fileOb***j is a File object that describes the file. If append is true, then output is appended to the end of the file.

FileWriter(contd.)

- **FileWriter** will create the file before opening it for output when you create the object.
  - In the case where we attempt to open a read-only file, an **IOException will be thrown.**
- **getChars( )** method is used to extract the character array equivalent.

public void getChars(int srhStartIndex, int srhEndIndex, char[] destArray, int destStartIndex)

Parameters:

srhStartIndex : Index of the first character in the string to copy.

srhEndIndex : Index after the last character in the string to copy. **destArray** : Destination array where chars wil get copied.

destStartIndex : Index in the array starting from where the chars will be pushed into the array. **Return:** It does not return any value.

Write a string to file using FileWriter

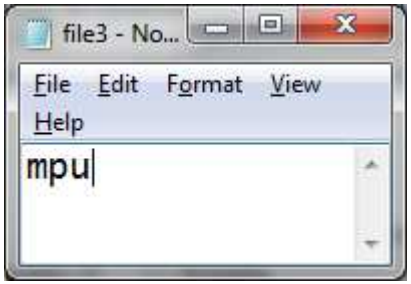

```
import java.io.*;
class FileWriterSimple
{
public static void main(String args[]) throws IOException
{
String source = "Welcome to OOP class\n" + " Study well";
char buffer[] = new char[source.length()]; // allocate space equal to length of string
source.getChars(0, source.length(), buffer, 0);
//copy the characters from position 0 to whole length(end) to buffer at position 0.
FileWriter f1 = new FileWriter("file1.txt");
f1.write(buffer);
f1.close();
}
}
```

Append a string to file using FileWriter


```
import java.io.*;
class FileWriterSimple
{
public static void main(String args[]) throws IOException
{
String source = "Welcome to OOP class\n" + " Study well";
char buffer[] = new char[source.length()]; // allocate space equal to length of string
source.getChars(0, source.length(), buffer, 0);
    //copy the characters from position 0 to whole length(end) from source
    //to buffer at /position 0.
FileWriter f1 = new FileWriter("file1.txt“,true); //append the contents
f1.write(buffer);
f1.close();
}
}
```

Write the alternate letter from string to file1.txt.
Write whole string to file2.txt,
Write the string starting from index 2 and upto 5 letters into file3.txt

```
import java.io.*;
class FileWriterDemo {
public static void main(String args[]) throws IOException {
String source = "Computers";
char buffer[] = new char[source.length()];
source.getChars(0, source.length(), buffer, 0);
FileWriter f0 = new FileWriter("file1.txt");
for (int i=0; i < buffer.length; i += 2) {
f0.write(buffer[i]);      //Write letters at position 0,2,4,6…. ino file1.txt
}
f0.close();
FileWriter f1 = new FileWriter("file2.txt");
f1.write(buffer);         //write all contents in buffer in file2.txt
f1.close();
FileWriter f2 = new FileWriter("file3.txt");
f2.write(buffer,2,3);     //Write letters from 2nd position to three letters
f2.close();
}
}
```



Reference

- Herbert Schildt, Java: **The Complete Reference, 8/e, Tata McGraw Hill, 2011**.