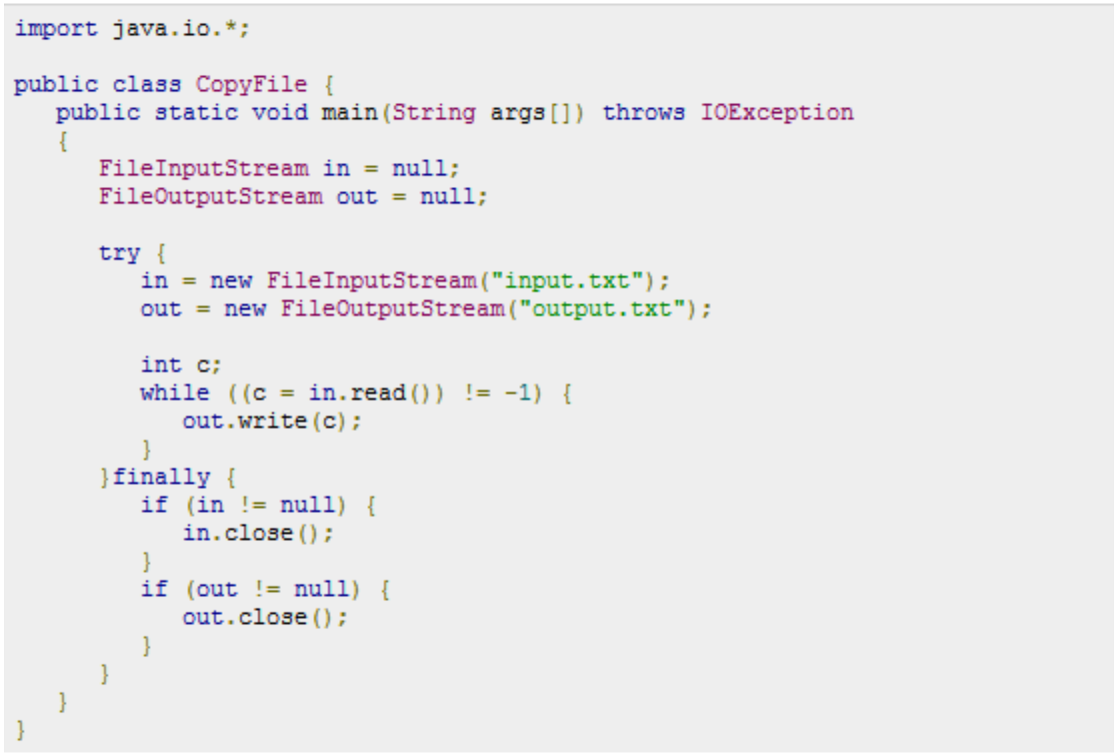
File Handling

Java performs I/O through streams.

A stream can be defined as a sequence of data. The **InputStream** is used to read data from a source and the **OutputStream** is used for writing data to a destination.

Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**. Following is an example which makes use of these two classes to copy an input file into an output file:



```java
import java.io.*;

public class CopyFile {
   public static void main(String args[]) throws IOException
   {
      FileInputStream in = null;
      FileOutputStream out = null;

      try {
         in = new FileInputStream("input.txt");
         out = new FileOutputStream("output.txt");

         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }finally {
         if (in != null) {
            in.close();
         }
         if (out != null) {
            out.close();
         }
      }
   }
}
```

__Note__***:*** *The Java process is started with the project directory as the working directory by default. So you will have to put your input file in the project directory. For other files, you will have to write the absolute path.*

Character Streams

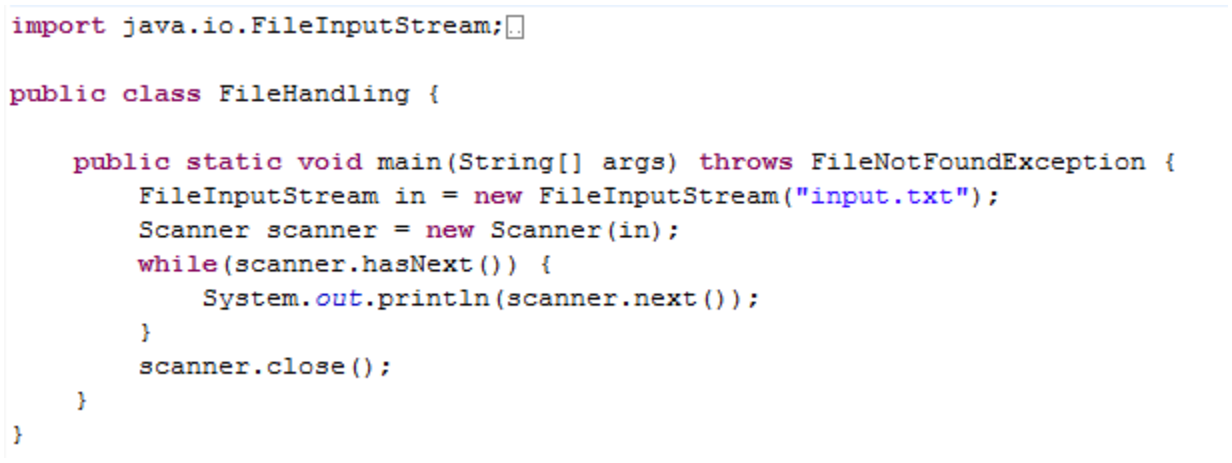
Similarly, we have Character Streams. Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are , **FileReader** and **FileWriter.**. Though internally **FileReader** uses **FileInputStream** and **FileWriter** uses **FileOutputStream** but here major difference is that **FileReader** reads two bytes at a time and **FileWriter** writes two bytes at a time.

```java
import java.io.*;

public class CopyFile {
   public static void main(String args[]) throws IOException
   {
      FileReader in = null;
      FileWriter out = null;

      try {
         in = new FileReader("input.txt");
         out = new FileWriter("output.txt");

         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }finally {
         if (in != null) {
            in.close();
         }
         if (out != null) {
            out.close();
         }
      }
   }
}
```

Example to demonstrate **InputStream** and **OutputStream**

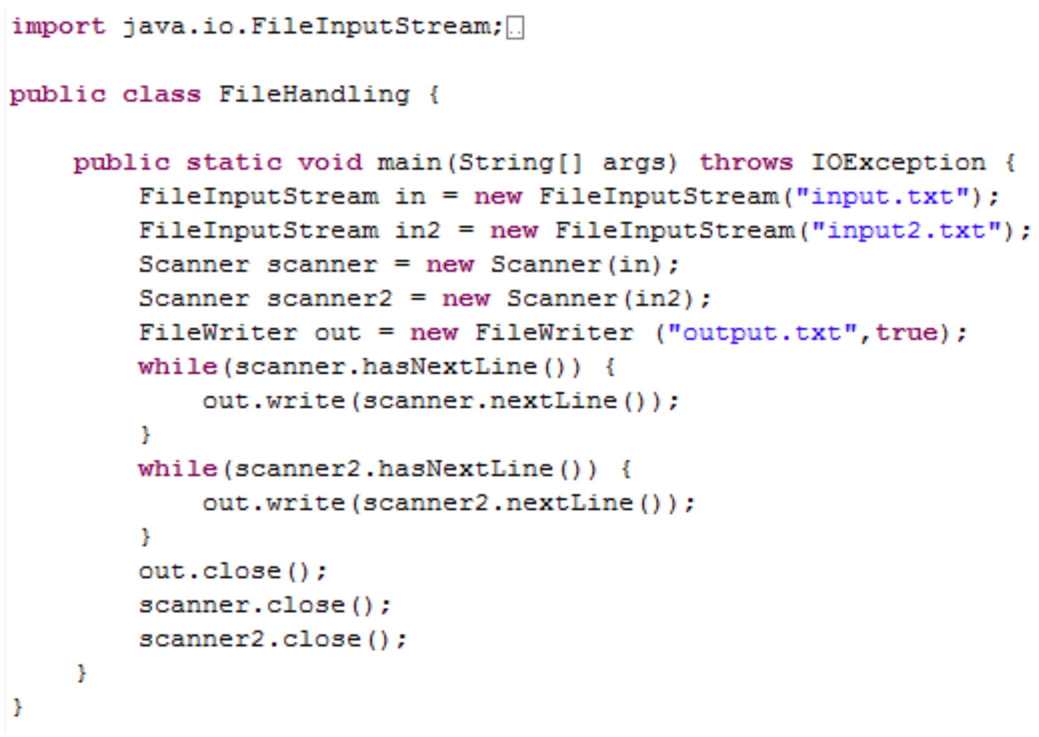
```java
import java.io.*;

public class fileStreamTest{

   public static void main(String args[]){

   try{
      byte bWrite [] = {11,21,3,40,5};
      OutputStream os = new FileOutputStream("test.txt");
      for(int x=0; x < bWrite.length ; x++){
         os.write( bWrite[x] ); // writes the bytes
      }
      os.close();

      InputStream is = new FileInputStream("test.txt");
      int size = is.available();

      for(int i=0; i< size; i++){
         System.out.print((char)is.read() + "  ");
      }
      is.close();
   }catch(IOException e){
      System.out.print("Exception");
   }
   }
}
```

Instead of using ***in.read()*** on **FileInputStream**, we can also provide **FileInputStream** to a **Scanner** and use it.



```java
import java.io.FileInputStream;

public class FileHandling {

    public static void main(String[] args) throws FileNotFoundException {
        FileInputStream in = new FileInputStream("input.txt");
        Scanner scanner = new Scanner(in);
        while(scanner.hasNext()) {
            System.out.println(scanner.next());
        }
        scanner.close();
    }
}
```

To read from two files and writing into another, you can do something like this.



```java
import java.io.FileInputStream;

public class FileHandling {

    public static void main(String[] args) throws IOException {
        FileInputStream in = new FileInputStream("input.txt");
        FileInputStream in2 = new FileInputStream("input2.txt");
        Scanner scanner = new Scanner(in);
        Scanner scanner2 = new Scanner(in2);
        FileWriter out = new FileWriter ("output.txt",true);
        while(scanner.hasNextLine()) {
            out.write(scanner.nextLine());
        }
        while(scanner2.hasNextLine()) {
            out.write(scanner2.nextLine());
        }
        out.close();
        scanner.close();
        scanner2.close();
    }
}
```

Passing the second argument ***true*** in **FileWriter** would append the output in the file, instead of overwriting it.

Directories

A directory is a File which can contains a list of other files and directories. You use **File** object to create directories, to list down files available in a directory.

Creating Directories

There are two useful **File** utility methods, which can be used to create directories:

- The **mkdir( )** method creates a directory, returning true on success and false on failure. Failure indicates that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet.
- The **mkdirs()** method creates both a directory and all the parents of the directory.

Following example creates "/tmp/user/java/bin" directory:

```java
import java.io.File;

public class CreateDir {
   public static void main(String args[]) {
      String dirname = "/tmp/user/java/bin";
      File d = new File(dirname);
      // Create directory now.
      d.mkdirs();
   }
}
```

Listing Directories

```java
import java.io.File;

public class ReadDir {
   public static void main(String[] args) {

      File file = null;
      String[] paths;

      try{
         // create new file object
         file = new File("/tmp");

         // array of files and directory
         paths = file.list();

         // for each name in the path array
         for(String path:paths)
         {
            // prints filename and directory name
            System.out.println(path);
         }
      }catch(Exception e){
         // if any error occurs
         e.printStackTrace();
      }
   }
}
```