Semana 11

Manejo de Archivos y Serialización

Clase File y FileOutputStream

Definición

Un Archivo es un camino abstracto, no tiene existencia física. Solo cuando se "usa" ese archivo, se afecta el almacenamiento físico subyacente. Cuando el archivo se crea indirectamente, se crea la ruta abstracta. El archivo es una forma en la que los datos se almacenarán según los requisitos.

Pasos para crear un archivo en Java

1. Primario, para crear un nuevo archivo, se utilizan archivos y funciones incorporados que definitivamente arrojarán excepciones aquí para ir a lo seguro. Entonces, para solucionarlo, utilizaremos técnicas de manejo de excepciones. Aquí, usaremos una de ellas conocida como técnicas de bloque de prueba y captura.

2. El trabajo adicional secundario es simplemente que importaremos la clase de archivo para la cual importaremos la clase de archivo.

Sintaxis: Para importar biblioteca de archivos o Clases

```
import java.util.File ;
```

Sintaxis: Para crear un archivo nuevo

```
File object_name = new File(Directory)
```

Sintaxis: Para especificar que un directorio es diferente en diferentes sistemas operativos (suponga que el archivo java está en una carpeta llamada 'Carpeta' que se crea en el escritorio) en Linux y Mac

```
/Users/Usuario/Desktop/Folder/
```

En Windows: se usa ‘\\’ en lugar de ‘/’ y ‘\’ como caracter de escape. Entonces en Windows sería

```
\\Users\\Usuario\\Desktop\\Folder\\
```

Hay dos métodos estándar para crear un nuevo archivo, ya sea directamente con la ayuda de la clase File o indirectamente con la ayuda de FileOutputStream creando un objeto del archivo en ambos enfoques.

- Usando la Clase File
- Usando la Clase FileOutputStream

Clase File	Clase FileOutputStream
Es una clase que es solo un identificador para	Es un flujo de salida que se puede escribir en FileOutputStream JavaDoc
Método: File.createNewFile()	**Método:** FileOutputStream **Ejemplo:** echo > miArchivo.txt
Se utiliza para aquellos objetos que no tienen existencia física	Se utiliza para aquellos objetos que ya existen

Ambas clases proporcionan algunos métodos que se utilizan principalmente para realizar operaciones con archivos. Por ejemplo, para crear, escribir, comparar dos nombres de ruta, verificar si un archivo específico está presente o no, y muchos más. Para entender este tema, primero, considere un ejemplo para ambos enfoques.

1. Comando de terminal utilizado para compilar cualquier código java en la máquina
2. Comando de terminal utilizado para ejecutar cualquier código java en la máquina

javac class_name.java // Para compilación
java class_name // Para ejecución

La terminal del sistema operativo Mac se usará para la implementación y proporcionará una salida para acceder al directorio.

Directorio utilizado: /Users/User/Desktop/Folder/

Ejemplo 1: Creando un nuevo archivo usando la clase File

```
// Programa Java para crear un nuevo archivo usando la clase File

// Importación de nuevos archivos
import java.io.File;
```

```java
import java.io.BufferedReader;

// Importación a medida que convierte bits en cadenas
import java.io.InputStreamReader;

public class Ejemplo {

    // Método Main
    public static void main(String args[])
    {
        // Creación de un nuevo archivo a través del método
        Ejemplo e = new Ejemplo();
        e.newFile();
    }

    // Método para hacer un nuevo archivo
    public void newFile()
    {
        String strPath = "", strName = "";

        // Bloque Try-catch
        try {

            // Crear objeto BufferedReadered
            BufferedReader br = new BufferedReader(
                new InputStreamReader(System.in));
            System.out.println("Enter the file name:");

            // Leyendo el nombre del archivo
            strName = br.readLine();
            System.out.println("Enter the file path:");

            // Leyendo la ruta del archivo
            strPath = br.readLine();

            // Creando el archivo objeto
            File file1
                = new File(strPath + "" + strName + ".txt");

            // Método createNewFile() el método crea un archivo
            // en blanco.
            file1.createNewFile();
        }
```

```
        // Bloque Try-catch
        catch (Exception ex1) {
        }
    }
}
```

Salida:

```
Name of file to be added             : newFile.txt
Directory where file is to be added : /Users/User/Desktop/Folder/

Added file name            : newFile.txt
Added file name directory : /Users/User/Desktop/Folder/
```

Explicación del ejemplo:

1. Para crear un nuevo archivo utilizando el lenguaje Java, aquí se utiliza la clase "File". Ambas clases, "BufferedReader" e "InputStreamReader", se utilizan para crear nombres de archivo y rutas del usuario como entrada. Estas clases se encuentran dentro del paquete "java.io". Entonces, para hacer uso de esas clases, es necesario importar esas clases al comienzo del programa.

2. Aquí se crea la clase, a saber, "Ejemplo". Y dentro de esa clase se define el método “main()” a partir del cual se iniciará la ejecución.

3. Dentro se crea el objeto de método “main()” de la clase. Y este objeto se usa para llamar al método “newFile()”.

4. Fuera del método main(), se declara el método newFile(), que cubre el código para recibir información del usuario y crear el archivo según la entrada.

5. Como el archivo tiene su propio nombre y ruta, es necesario dar un nombre para el archivo (que queremos crear) y una ruta (donde crear el archivo) para el archivo. En esta línea, se declaran dos cadenas en blanco, a saber, strName, strPath. “strName y strPath se utilizan para almacenar el nombre y la ruta del archivo cuando el usuario proporciona esta información.

6. Para tomar el nombre del archivo y la ruta del usuario como entrada, aquí se utilizan la clase BufferedReader y la clase InputStreamReader. El objeto de BufferedReader “br” es útil para tomar valores de entrada dados por el usuario.

7. Esta línea imprime un texto para dar una indicación al usuario como "Ingrese el nombre del archivo:". Para imprimir el texto se utiliza la función “println()”.

8. Aquí se usa el método “readLine()” para tomar datos y almacenarlos, en strName y strPath.

9. Aquí se crea el objeto de la clase File y, como parámetro, se le da al constructor la ruta y el nombre del archivo. En esta línea de código, “.txt” es un formato del archivo. Puedes cambiarlo según tus necesidades. El objeto de la clase File es necesario para llamar a los métodos provistos en su clase.

10. Aquí se llama al método “createNewFile()” con la ayuda del objeto de la clase File. Este método crea un archivo en blanco en una ruta de directorio dada.

11. Por último, encerrado por el bloque "try{ }". Porque, métodos como readLine() y createNewFile() generan una excepción. Entonces, para manejar ese intento de excepción, se usa la captura.

Ejemplo 2: Creando un nuevo archivo usando la clase FileOutputStream

```
// Programa en Java para crear un nuevo archivo
// usando la clase FileOutputStream

// Importando la clase FileOutputStream
import java.io.FileOutputStream;

// Importando la clase BufferedReader para entradas
import java.io.BufferedReader;

// Importación a medida que convierte bits en cadenas
import java.io.InputStreamReader;

// Clase
public class Ejemplo {

    // Método Main
    public static void main(String args[])
    {
        // Creando objeto File
        Ejemplo e = new Ejemplo();
        e.newFile();
    }

    // Función para crear un nuevo archivo
    public void newFile()
```

```
    {
        String strFilePath = "", strFileName = "";

        // Bloque Try-Catch
        try {

            // Creando objeto BufferClass
            BufferedReader br = new BufferedReader(
                new InputStreamReader(System.in));
            System.out.println("Enter the file name:");

            // Solicitar nombre de archivo al usuario
            strFileName = br.readLine();
            System.out.println("Enter the file path:");

            // Solicitar ruta de archivo al usuario
            strFilePath = br.readLine();

            // Creando objeto de la clase FileOutputStream
            FileOutputStream fos = new FileOutputStream(
                strFilePath + "" + strFileName + ".txt");
        }

        // Bloque Try-Catch
        catch (Exception ex1) {
        }
    }
}
```

Salida:

Será igual que el anterior porque solo el enfoque para crear un nuevo archivo ha cambiado el resto del nombre del archivo y el directorio donde se agrega sigue siendo el mismo.

```
Added file name            : newFile.txt
Added file name directory : /Users/User/Desktop/Folder/
```

Explicación del ejemplo:

En el segundo ejemplo, la clase File no se usa para crear un nuevo archivo mediante programación.

1. Para crear un nuevo archivo usando el lenguaje Java, aquí se usa la clase "FileOutputStream" y "BufferedReader" e "InputStreamReader" para tomar el nombre del archivo y la ruta del usuario como entrada. Estas clases se encuentran dentro del paquete "java.io". Entonces, para hacer uso de esas clases, es necesario importarlas al comienzo del programa.

2. La clase se crea a saber, "Ejemplo". Y dentro de esa clase se define el método “main()” a partir del cual se iniciará la ejecución.

3. Dentro del método "main()" se crea el objeto de la clase. Y este objeto se usa para llamar al método “newFile()”.

4. Fuera del método main(), se declara el método newFile(), que cubre el código para recibir información del usuario y crear un archivo según la entrada.

5. En esta línea, se declaran dos cadenas en blanco, a saber, strFileName, strFilePath. “strFileName y strFilePath se utilizan para almacenar el nombre y la ruta del archivo cuando el usuario proporciona esta información.

6. Para tomar el nombre del archivo y la ruta del usuario como entrada, aquí se utilizan la clase BufferedReader y la clase InputStreamReader. El objeto de BufferedReader “br” es útil para tomar valores de entrada dados por el usuario.

7. Esta línea imprime un texto para dar una indicación al usuario como "Ingrese el nombre del archivo:". Para imprimir texto se utiliza la función “println()”.

8. Aquí se usa el método “readLine()” para tomar la entrada y almacenarla, en strFileName y strFilePath.

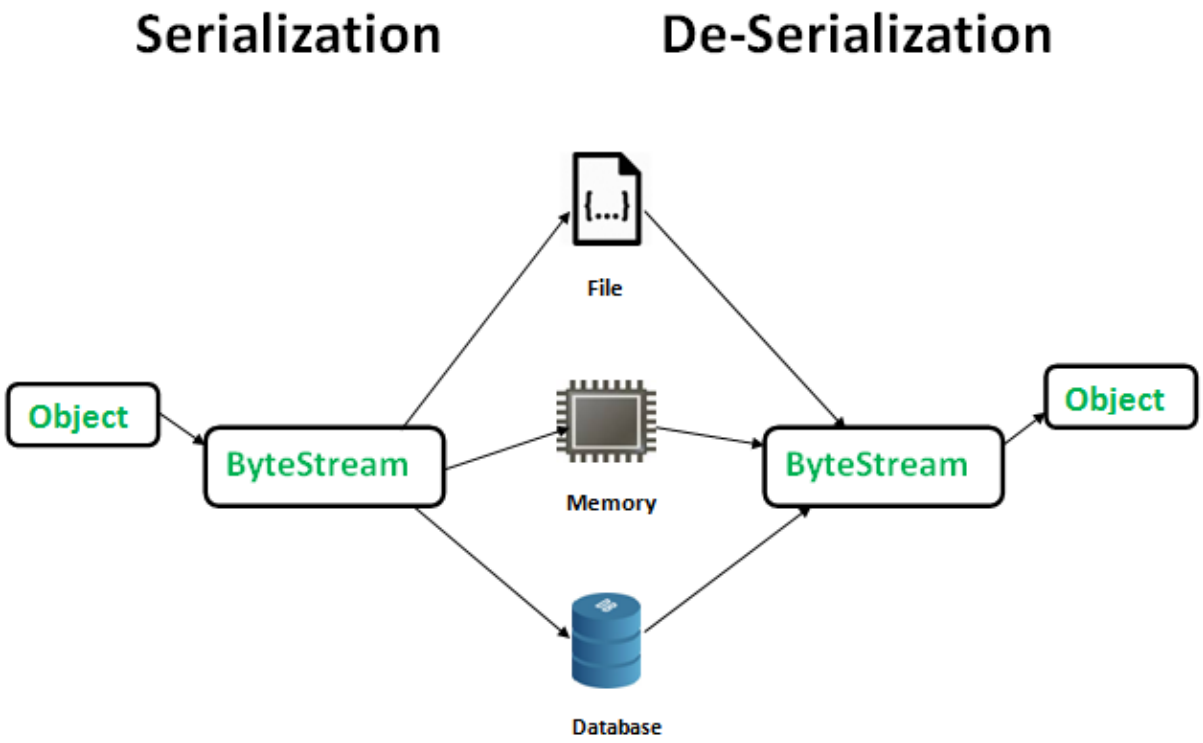
9. Aquí se crea el objeto de la clase FileOutputStream y, como parámetro, la ruta y el nombre del archivo se le dan al constructor. En esta línea de código, “.txt” es un formato del archivo. Puedes cambiarlo según tus necesidades. El objeto de la clase FileOutputStream es necesario para llamar a los métodos proporcionados en su clase. Por ejemplo, si el usuario desea almacenar algún texto en un archivo recién creado mediante programación, el método write() es útil.

10. Por último, encerrado por el bloque "try{ }". Porque el método readLine() genera una excepción. Entonces, para manejar ese intento de excepción, se usa el bloque catch.

Serialización en Java

Definición

La serialización es un mecanismo para convertir el estado de un objeto en un flujo de bytes. La deserialización es el proceso inverso en el que se utiliza el flujo de bytes para recrear el objeto Java real en la memoria. Este mecanismo se utiliza para persistir el objeto.



El flujo de bytes creado es independiente de la plataforma. Por lo tanto, el objeto serializado en una plataforma se puede deserializar en una plataforma diferente. Para hacer que un objeto Java sea serializable, implementamos la interfaz java.io.Serializable. La clase ObjectOutputStream contiene el método writeObject() para serializar un objeto.

```
public final void writeObject(Object obj)
                        throws IOException
```

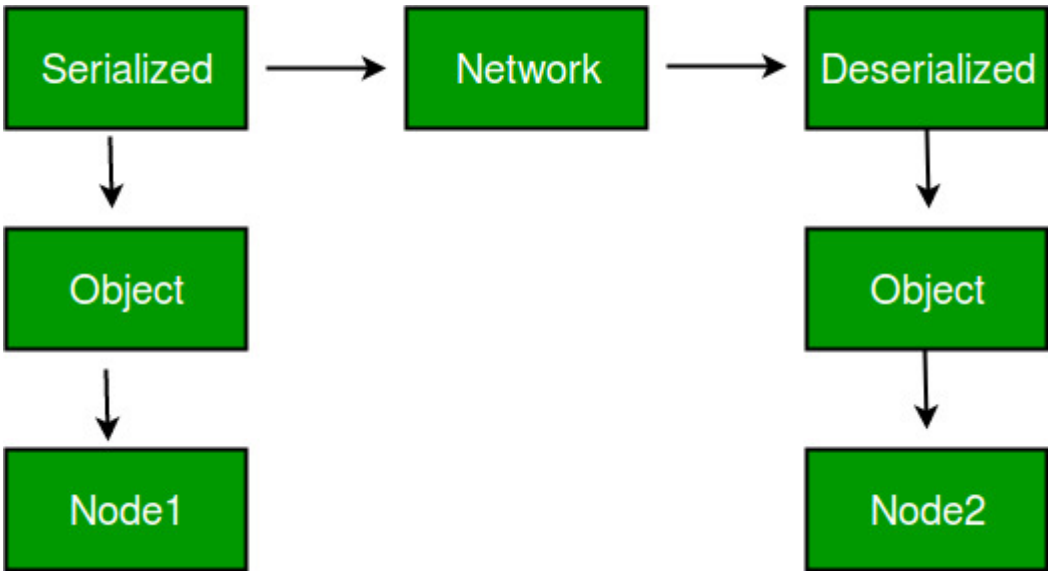
La clase ObjectInputStream contiene el método readObject() para deserializar un objeto.

```
public final Object readObject()
                  throws IOException,
```

```
ClassNotFoundException
```

Ventajas de la Serialización

1. Para guardar/mantener el estado de un objeto. 2. Para viajar un objeto a través de una red.



Solo se pueden serializar los objetos de esas clases que implementan la interfaz java.io.Serializable. Serializable es una interfaz de marcador (no tiene miembro de datos ni método). Se utiliza para "marcar" las clases de Java para que los objetos de estas clases puedan obtener cierta capacidad. Otros ejemplos de interfaces de marcador son:- Clonable y Remote. Puntos para recordar 1. Si una clase principal ha implementado una interfaz Serializable, entonces la clase secundaria no necesita implementarla, pero viceversa no es cierto. 2. Solo los miembros de datos no estáticos se guardan a través del proceso de serialización. 3. Los miembros de datos estáticos y los miembros de datos transitorios no se guardan a través del proceso de serialización. Por lo tanto, si no desea guardar el valor de un miembro de datos no estáticos, hágalo transitorio. 4. Nunca se llama al constructor del objeto cuando se deserializa un objeto. 5. Los objetos asociados deben implementar una interfaz serializable. Ejemplo:

```
class A implements Serializable{

// B also implements Serializable
// interface.
B ob=new B();
}
```

Ejemplo de implementación

```
// Código Java para serialización y deserialización
// de un objeto Java
import java.io.*;

class Demo implements java.io.Serializable
{
    public int a;
    public String b;

    // Constructor por defecto
    public Demo(int a, String b)
    {
        this.a = a;
        this.b = b;
    }

}

class Test
{
    public static void main(String[] args)
    {
        Demo object = new Demo(1, "Sample");
        String filename = "file.ser";

        // Serialización
        try
        {
            // Guardar un objeto en un archivo
            FileOutputStream file = new FileOutputStream(filename);
            ObjectOutputStream out = new ObjectOutputStream(file);

            // Método para la serialización del objeto.
            out.writeObject(object);

            out.close();
            file.close();

            System.out.println("Object has been serialized");

        }
```

```
        catch(IOException ex)
        {
            System.out.println("IOException is caught");
        }


        Demo object1 = null;

        // Deserialización
        try
        {
            // Leer el objeto de un archivo
            FileInputStream file = new FileInputStream(filename);
            ObjectInputStream in = new ObjectInputStream(file);

            // Método para la deserialización del objeto.
            object1 = (Demo)in.readObject();

            in.close();
            file.close();

            System.out.println("Object has been deserialized ");
            System.out.println("a = " + object1.a);
            System.out.println("b = " + object1.b);
        }

        catch(IOException ex)
        {
            System.out.println("IOException is caught");
        }

        catch(ClassNotFoundException ex)
        {
            System.out.println("ClassNotFoundException is caught");
        }

    }
}
```

Salida:

```
Object has been serialized
Object has been deserialized
```

```
a = 1
b = Sample
```