POO

Streams

Almacenamiento del estado de un programa

SERIALIZACION

- Convertir el Estado de los objetos de un programa en bytes que se pueden almacenar en un archivo o transmitir a través de la red.
- Los datos que corresponden al Estado de un programa se utilizaran solo con el mismo programa.

TEXTO PLANO

- Escribir los datos a un archivo con delimitadores que otros programas puedan entender y procesar.

- El almacenamiento se puede realizar con caracteres o bytes.

Almacenamiento del estado de un programa

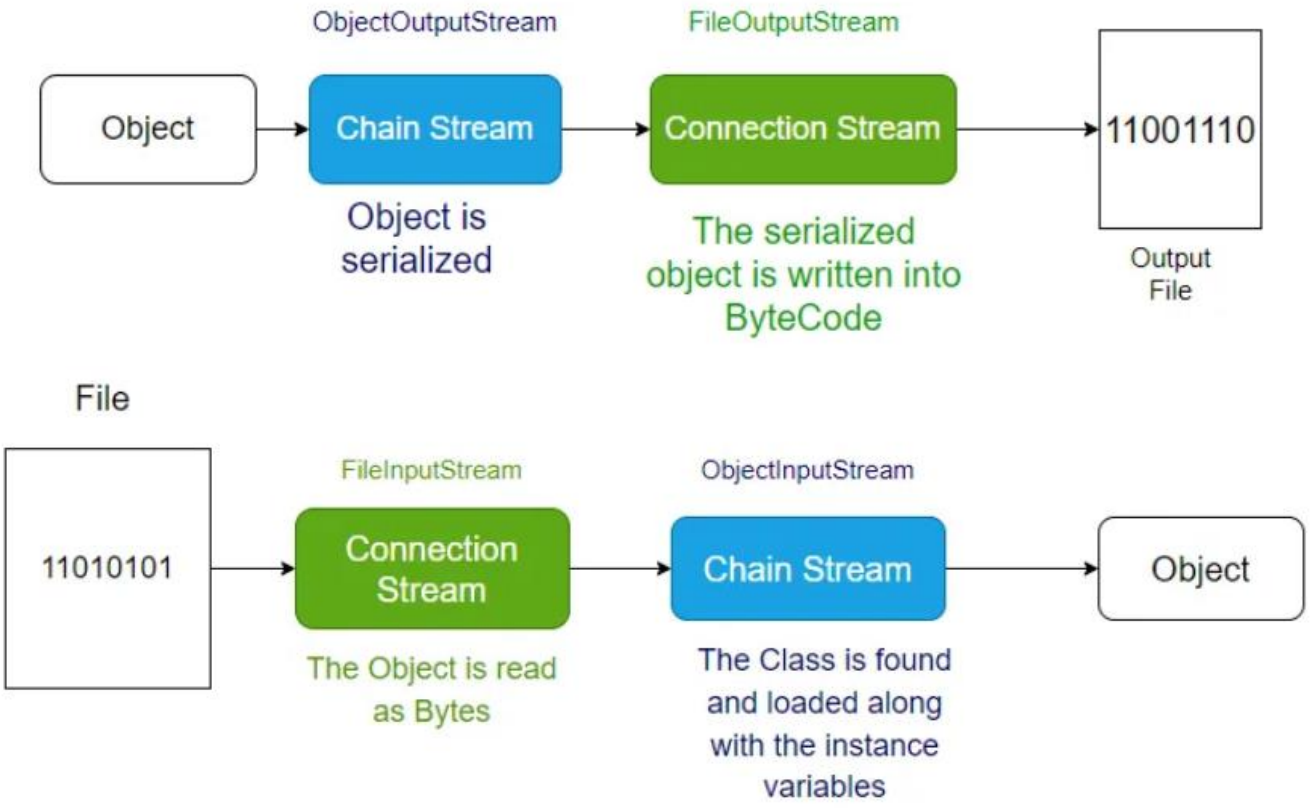
- Los archivos serializados no se pueden leer fácilmente, pero para nuestro programa es más fácil y seguro restaurar los datos de esta forma.
- Los archivos de texto son fáciles de leer, cada línea puede corresponder a un Objeto separando los elementos de su estado con comas.

```
¨Ìsrjava.util.ArrayListxÅ“ô«aùIsizexpwsr9crunchify.com.tutorial.CrunchifySerializeDeserial
ize$ItemI"¶)Ω˙ΩfÍDcostIquantityLdesctLjava/lang/String;LitemIDq~Lthis$0t6Lcrunchify/com/
tutorial/
CrunchifySerializeDeserialize;xp@0tiPadtITEM101sr4crunchify.com.tutorial.CrunchifySerializ
eDeserializeA„«Ç
√XJxpsq~@Ç∏tiPhonetITEM102q~    x
```

```
Make,Model,Description,Price
Dell,P3421W,"Dell 34, Curved, USB-C Monitor",2499.00
Dell,"","Alienware 38 Curved ""Gaming Monitor""",6699.00
Samsung,,"49"" Dual QHD, QLED, HDR1000",6199.00
Samsung,,"Promotion! Special Price
49"" Dual QHD, QLED, HDR1000",4999.00
```

¿Que se necesita para serializar un objeto a un archivo?

- Un archivo
- Un flujo de datos (FileOutputStream)
- Un objeto que permita escribir los objetos (ObjectOutputStream)



Streams de Entrada/Salida (I/O)

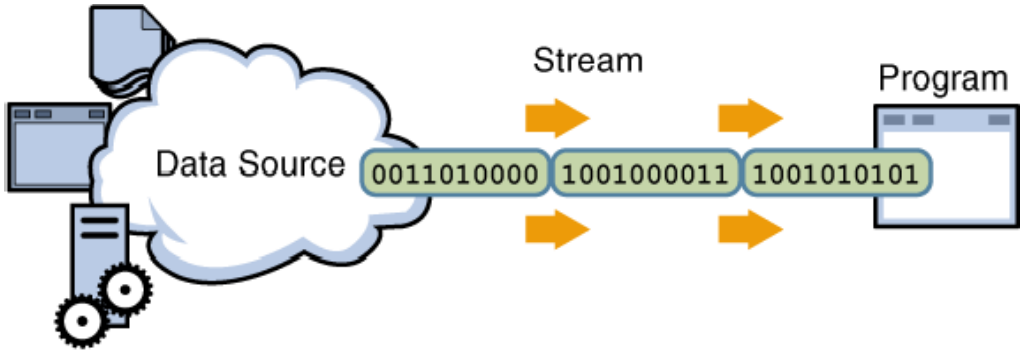
- Un Stream I/O representa una fuente de entrada o un destino de salida
- Un stream puede representar varios tipos diferentes de fuentes y destinos:
  - ficheros en disco, dispositivos, otros programas, un socket de red y arrays de memoria
- Los streams soportan varios tipos de datos
  - bytes simples, tipos de datos primitivos, caracteres localizados, y objetos
- Algunos streams son de paso y otros de conversión

Streams de Entrada/Salida (I/O)

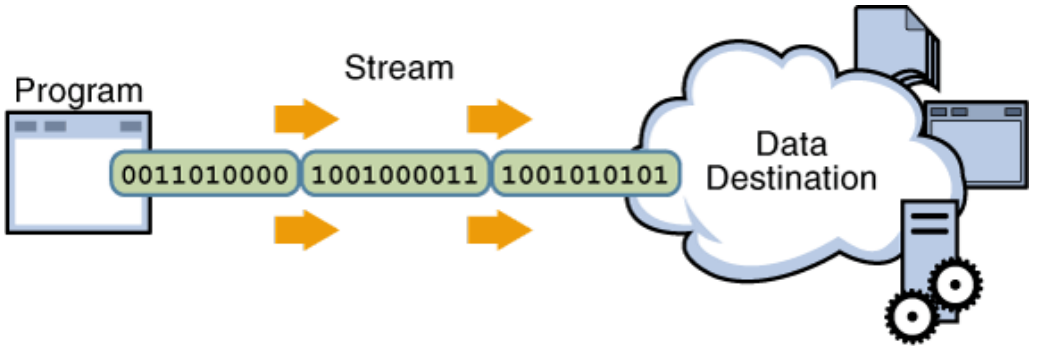
- Sin importar cómo trabajan internamente, todos los streams presentan el mismo modelo simple a los programas que los usan
  - Un stream es una secuencia de bytes
- La entrada/salida basada en streams soprta la lectura o escritura de datos secuencialmente
- Un stream se puede abrir para leer o escribir, pero no la leer y escribir

Stream de Entrada (Input Stream)

– Un programa usa un input stream para leer datos de una fuente, uno a la vez (secuencialmente)



– Un programa usa un output stream para escribir datos a un destino, uno a la vez (secuencialmente)



Tipos de Streams generales

- Byte y Character Streams
  - Character vs. Byte
- Input y Output Streams
  - Basados en fuente o destino
- Node y Filter Streams
  - Si los datos en un stream son o no manipulados o transformados

Streams Byte y Character

- Byte streams
  - Para datos binarios
  - Clases raíz de byte streams (ambas son abstractas):
    - Clase InputStream
    - Clase OutputStream
- Character streams
  - Para caracteres Unicode
  - Clases raíz de character streams (ambas abstractas):
    - Clase Reader
    - Clase Writer

Streams Input y Output

- Input o source streams
  - Pueden leer de estos streams
  - Clases raíz de todos los input streams:
    - Clase InputStream
    - Clase Reader
- Output o sink (destino) streams
  - Pueden escribir en estos streams
  - Clases raíz de todos los output streams:
    - Clase OutputStream
    - Clase Writer

Streams Nodo y Filtro

- Node streams (sumidero de datos)
  - Contienen la funcionalidad básica de lectura o escritura de una ubicación específica
  - Los tipos de nodo streams incluyen ficheros, memoria y pipes
- Filter streams (stream de procesado)
  - Capa sobre los streams nodo entre hilos de ejecución o procesos
  - Para una funcionalidad adicional – alterando o gestionando datos en el stream

Clases Abstractas

- InputStream y OutputStream
- Reader y Writer

Control del flujo de una operación I/O

Crear un objeto stream y asociarlo con la fuente de datos

Dar al objeto stream la funcionalidad deseada a través del encadenamiento del stream

while (hay más información)

    leer (escribir) siguiente dato desde (a)

el stream cerrar el stream

Byte Stream

- Los programas usan byte streams para realizar input y output de bytes (8-bit)
- Todas las clases byte stream descienden de InputStream y OutputStream
- Hay varias clases byte stream
  - FileInputStream y FileOutputStream
- Se usan de forma similar; la diferencia es la forma en que se construyen
- Se deben usar en I/O primitivo o de bajo nivel

Ejemplo: FileInputStream y FileOutputStream

```
public class CopyBytes {
  public static void main(String[] args) throws IOException {
    FileInputStream in = null;
    FileOutputStream out = null;
    try {
      in = new FileInputStream("prueba.txt");
      out = new FileOutputStream("byteprueba.txt");
      int c;
      while ((c = in.read()) != -1) {
        out.write(c);
      }
    } finally {
      if (in != null) { in.close(); }
      if (out != null) {
        out.close();
      }
    }
  }
}
```

Character Stream

- Java utiliza el código Unicode para los caracteres
- La I/O character stream convierte automáticamente este formato interno a y del conjunto de caracteres locales
- Todas las clases character stream descienden de Reader y Writer
- Como en los byte streams, hay clases character stream que se especializan en I/O de ficheros: FileReader y FileWriter

Ejemplo: FileReader y FileWriter

```
public class CopyCharacters {
  public static void main(String[] args) throws IOException {
    FileReader inputStream = null;
    FileWriter outputStream = null;
    try {
      inputStream = new FileReader("prueba.txt");
      outputStream = new FileWriter("characteroutput.txt");
      int c;
      while ((c = inputStream.read()) != -1) {
        outputStream.write(c);
      }
    } finally {
      if ((inputStream != null) {inputStream.close(); }
      if (outputStream != null) {
        outputStream.close();
      }
    }
  }
}
```

Buffered Streams

- Un unbuffered I/O significa que cada solicitud de lectura o escritura es gestionado directamente por el sistema operativo subyacente (ineficiente)
- Para reducir esta sobrecarga, Java implementa los buffered I/O streams
  - Con los buffered input streams se leen datos desde un area de memoria conocida como buffer; la API nativa se invoca sólo cuando el buffer está vacío
  - Para los buffered output streams la API se invoca cuando el buffre está lleno

Creación de Buffered Streams

- Un programa puede convertir un unbuffered stream en un buffered stream usando envolventes. Ejemplo:

```
inputStream =
new BufferedReader(new FileReader("prueba.txt"));
outputStream =
new BufferedWriter(new FileWriter("charoutput.txt"));
```

- Las clases buffered stream son:
  - *BufferedInputStream* y *BufferedOutputStream* crean buffered byte streams
  - *BufferedReader* and *BufferedWriter* crean buffered character streams

Ejemplo: escribe matriz con BufferedOutputStream

```
import java.io.*;
public class EscribeMatrizBufOutSt {
  static double[][] data = {
    { Math.exp(2.0), Math.exp(3.0), Math.exp(4.0) },
    { Math.exp(-2.0), Math.exp(-3.0), Math.exp(-4.0) },
  };
  public static void main(String[] args) {
    int row = data.length;
    int col = data[0].length;
    int i, j;
    for (i = 0; i < row; i++) {
      for (j = 0; j < col; j++) {
     System.out.println("dato[" + i + "][" + j + "] = " +
                        data[i][j]);
      }
    }
```

Ejemplo: escribe matriz con BufferedOutputStream

```
    if (args.length > 0) {
       try {
         DataOutputStream out =
          new DataOutputStream(new BufferedOutputStream(
          new FileOutputStream(args[0])));
         out.writeInt(row);  out.writeInt(col);
         for (i = 0; i < row; i++) {
           for (j = 0; j < col; j++) {
             out.writeDouble(data[i][j]);
           }
         }
          out.close();
       } catch (IOException e) {}
    }
  }
}
```

Ejemplo: lee matriz con BufferedInputStream

```
import java.io.*;
public class LeeMatrizBufInp {
  static double[ ][ ] data;
  public static void main(String[] args) {
    if (args.length > 0) {
      try {
          DataInputStream in =
            new DataInputStream(new BufferedInputStream(
            new FileInputStream(args[0])));
          int row = in.readInt();
          System.out.println("fila = " + row);
          int col = in.readInt();
          System.out.println("columna = " + col);
          data = new double[row][col];
          for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
              data[i][j] = in.readDouble();
              System.out.println("dato[" + i + "][" + j
                                 + "] = " + data[i][j]);
            }
          }
      } catch (IOException e) {}
    }  }
}
```

Uso de Reader y Writer

```
BufferedReader inp =
   new BufferedReader(new FileReader("matriz.dat"));

BufferedReader inp =
 new BufferedReader(new InputStreamReader(System.in));

PrintWriter out =
   new PrintWriter(new BufferedWriter(
         new FileWriter("matriz.dat")));

Writer out =
   new BufferedWriter(new
         OutputStreamWriter(System.out));
```

Estándar Streams en Java

- Tres estándar streams
  - Estándar Input, accedido a través de *System.in*
  - Estándar Output, accedido a través de *System.out*
  - Estándar Error, accedido a través de *System.err*
- Estos objetos son definidos automáticamente y no requieren ser abiertos
- System.out y System.err son definidos como objetos PrintStream

Clase File

- La clase File no es un stream
- Es importante porque las clases stream manipulan objetos File
- Son una representación abstracta de los ficheros y pathname de directorios

Ejemplo File Class

```
import java.io.*;

public class FileInfoClass {
   public static void main(String[] args) {
      String fileName = args[0];
      File fn = new File(fileName);
      System.out.println("Nombre: " + fn.getName());
      if (!fn.exists()) {   // Comprueba si el fichero existe
         System.out.println(fileName + " no existe");
         System.out.println("Crea directorio temporal...");
         fileName = "temp";
         fn = new File(fileName); fn.mkdir();
         System.out.println(fileName +
                            (fn.exists()? " existe":" no existe"));
         System.out.println("Elimina directorio temporal...");
         fn.delete();
       }
       System.out.println(fileName + " es un " +
                          (fn.isFile()? "fichero":" directorio"));
```

Ejemplo File Class

```
if (fn.isDirectory()) {
  String content[] = fn.list();
  System.out.println("Contenido de este directorio:");
  for (int i = 0; i < content.length; i++) {
    System.out.println(content[i]);
  }
}
if (!fn.canRead()) {
  System.out.println(fileName + " no se puede leer");
  return;
}
System.out.println(fileName + " is " + fn.length()
                   + " bytes long");
System.out.println(fileName + " es " + fn.lastModified());

if (!fn.canWrite()) {
   System.out.println(fileName + " no se puede escribir");
}
}
```

Internacionalización: codificación de caracteres

- Por defecto, la codificación de caracteres está especificada por una propiedad del sistema
  - file.encoding=ISO8859_1 (ISO-8859-1) ASCII
- Se puede usar otras codificaciones mediante:

```
BufferedReader in =
new BufferedReader(new InputStreamReader(
        new FileInputStream("foo.in"), "GB18030"));
PrintWriter out =
    new PrintWriter(new BufferedWriter(
     new OutputStreamWriter(
          new FileOutputStream("foo.out", "GB18030"))));
```