Streams y Persistencia en Java

Agustín J. González

ELO-326: Seminario II

2do. Sem. 2001

Clases bases para Entrada y Salida

- Un Stream es simplemente una fuente o destino de bytes.
- Los streams más comunes son los archivos. También pueden ser caracteres de un string o bloque de memoria o bytes de un “socket”. La idea es que el programador los trate de la misma forma que archivos.
- En Java se usan diferentes mecanismos de formateo los datos, buffer, y destino o fuente de bytes.

Clases bases para Salida (formato)

- En Java hay una clase clase abstracta OutputStream. Ésta especifica un número de operaciones para escribir un byte, para escribir un arreglo de bytes, y para cerrar el stream.
- Clases derivadas de ésta son FileOutputStream y ByteArrayOutputStream, las cuales son clases concretas (no abstractas). Sus operaciones de write envían bytes a archivo o memoria.
- Para dar formato a los datos (imprimir números y strings) se dispone de la case PrintStream.
- PrintStream es una clase adaptadora (Adapter), la cual recibe en su constructor el objeto con el cual podrá escribir (write) los bytes.
- Ej.
  FileOutputStream fout = new FileOutputStream(“output.dat”);
  PrintStream pout = new PrintStream(fout);

Clases bases para Salida (Buffer)

- Además de formato es preciso disponer de un buffer para mejorar el desempeño de algunos dispositivos de entrada y salida.
- Para incorporar un buffer y con ello crear la cadena que se conoce como filtrado, se usa:
  PrintStream pout = new PrintStream(new BufferedOutputStream(new FileOutpurStream(“Output.dat”)));

- Los filtros se pueden construir usando cualquier combinación de cases encadenando una después de la otra.
- La primera clase es aquella que permite leer o escribir objetos, y la última clase de la cadena envía o recibe los bytes. Las clases intermedias pueden hacer el trabajo que necesitemos (buffer, encriptación etc.)

Clases bases para Salida

- Para escribir texto, usamos PrintStream y sus operaciones print o println.
- PrintStream out;
  Employee harry;
  ....
  out.println(3.14);
  out.print(“Harry : “);
  out.print(harry);
- Cuando se imprimen objetos, se invoca el método toString del objeto.
- La clase Object implementa este método, el cual imprime la clase del objeto y su dirección. Lo recomendable es sobremontar este método para producir un string que tenga más sentido para ese objeto.

Clases bases para Entrada y Salida Binaria

- Para salida binaria, usamos las clases DataInputStream y DataOutputStream. Ellas proveen las siguientes operaciones:
-

DataInputStream	DataOutputStream
readInt	writeInt
readShort	writeShort
readLong	writeLong
readFloat	writeFloat
readDouble	writeDouble
readBoolean	writeBoolean
readChar	writeChar
	writeChars

- Ejemplo:

```
DataOutputStream out = new DataOutputStream(new
FileOutputStream(“output.dat”));
out.writeDouble(3.14);
out.writeChars(“Harry”);
```

Clases bases para Entrada y Salida Binaria

- Una propiedad importante se las operaciones previas es que son independiente del procesador (tamaño de datos). Se usa el ordenamiento de la red big-endian.
- La comunicación es segura pues la clase se encarga de hacer las conversiones si el procesador es little-endian.
- Ej. Para guardar un string, primero se puede guardar el tamaño y luego los caracteres.
  String s;
  ....
  out.writeInt(s.length());
  out.writeChars(s);
- Para leer el string de vuelta:
  int len = in.readInt();
  StringBuffer b = new StringBuffer(len);
  for (int i=0; i <len; i++) b.append(in.readChar());
  String s = b.toString();

Ejemplo: Lectura de una página WEB

- URL url = new URL(“www.elo.utfsm.cl”);
  InputStream is = url.openStream();
  DataInputStream in = new DataInputStream(is);
  String s;
  while((s=in.readLine()) != null)
  {
  ..... // porcesar s ...
  }

Ejemplo: Clase para salida formateada

- Este ejemplo ofrece una opción similar al printf de C para salida formateada. Éste tipo de operación no existe en Java.
- Ejemplo de uso del servicio que buscamos

```
public class PrintfStreamTest
{  public static void main(String[] args)
   {  String[] n = { "zero", "one", "two", "three", "four", "five", "six",
         "seven", "eight", "nine", "ten" };
      PrintfStream ps = new PrintfStream(System.out);
      for (int i = 0; i < n.length; i++)
      {  ps.printf("%-8s", n[i]);
         ps.printf("|%6.2f\n", i * 1.0);
      }
   }
}
```

El archivo que implementa la clase PrintfStream está aquí.

Archivos

- Se dispone de las clase FileInputStream y FileOutputStream.
- Hay que recordar cerrar el archivo.
  FileInputStream fin = new FileInputStream (“input.dat”);
  ...
  fin.close();
- No hay operaciones muy útiles para trabajar con los archivos en forma directa. Debemos utilizar un adaptador.
  DataInputStream in = new DataInputStream(fin);
- Si deseamos hacer ambos lectura y salida de un archivo, se debe usar la clase RandomAccessFile. Esta clase no hereda de InputStream ni OutputStream.
- Como no hay múltiple herencia en Java, esta clase implementa las interfaces DataInput y DataOutput.
- Para abrir un archivo random:
  RandomAccessFile in = new RandomAccessFile(“input.dat”, “r”);
  RandomAccessFile inOut = new RandomAccessFile(“input.dat”, “rw”);
- Los archivos random disponen de funciones para hacer un seek como en C.

String como un Stream

- Se emplean las clases ByteArrayOutputStream y ByteArrayInputStream.
- Para escribir sobre un string usando print().

```
Date bday = new Data(1975, 6,16);
ByteArrayOutputStream bout = new ByteArrayOutputStream();
PrintStream out = new PrintStream(bout);
out.print(“Birthday: “);
out.println(bday);
String b = out.toString();
```

- Si deseamos hacer una lectura desde memoria podemos usar algo similar.

```
byte [] imageBytes;
...
ByteArrayInputStream in = new ByteArrayInputStream(imageBytes);
Image img = imageLoader.getImage(in);
```

Persistencia en Java

- Un objeto se dice persistente cuando es almacenado en un archivo u otro medio permanente. Un programa puede grabar objetos persistentes y luego recuperarlos en un tiempo posterior.
- A diferencia de C++ que sólo soporta persistencia a través de bibliotecas propietarias por lo cual su portabilidad y generalidad es limitada, Java se provee un mecanismo de serialización para almacenar objetos en disco.
- La serialización se obtiene llamando al método writeObject de la clase ObjectOutputStream para grabar el objeto, para recuperarlo llamamos al método readObject de la clase ObjectInputStream.
- La serialización además de persistencia, se puede usar para transferir objetos desde una máquina a otra a través de un socket (Seminario I).
- Sólo objetos que implementen la interfaz Serializable pueden ser escritos a stream. La clase de cada objeto es codificada incluyendo el nombre de la clase y la firma de la clase (su prototipo) los valores de los sus campos y arreglos, y la clausura de cualquier otro objeto referenciado desde el objeto inicial.

Persistencia en Java

- Múltiples referencias a un único objeto son codificadas usando un mecanismo de referencias compartidas de modo que el “grafo” de objetos puede ser restaurado con la misma forma original.
- Ejemplo: para escribir un objeto,

```
FileOutputStream ostream = new FileOutputStream("t.tmp");
    ObjectOutputStream p = new ObjectOutputStream(ostream);
    p.writeInt(12345);
    p.writeObject("Today");
    p.writeObject(new Date());
    p.flush();
    ostream.close();
```

- Clases que requieren manejos especiales durante el proceso de serialización o deserialización deben implementar los métodos:

```
private void readObject(java.io.ObjectInputStream stream)
    throws IOException, ClassNotFoundException;
 private void writeObject(java.io.ObjectOutputStream stream)
    throws IOException
```

Persistencia en Java (cont)

- Para recuperar el objeto previo usamos:

```
FileInputStream istream = new FileInputStream("t.tmp");
  ObjectInputStream p = new ObjectInputStream(istream);

  int i = p.readInt();
  String today = (String)p.readObject();
  Date date = (Date)p.readObject();
  istream.close();
```

- En este caso no hay gran dificultad por cuanto todos los objetos String y Date implementan la interfaz serializable.
- Tipos de datos primitivos pueden ser escritos usando los métodos apropiados de DataOutput. Strings también pueden ser escritos usando el métodowriteUTF.
- Análogamente la lectura de tipos primitivos se puede efectuar usando métodos de DataInput.