PERSISTENCIA

- Capacidad de los objetos para perdurar más allá de la duración del programa que los creó.

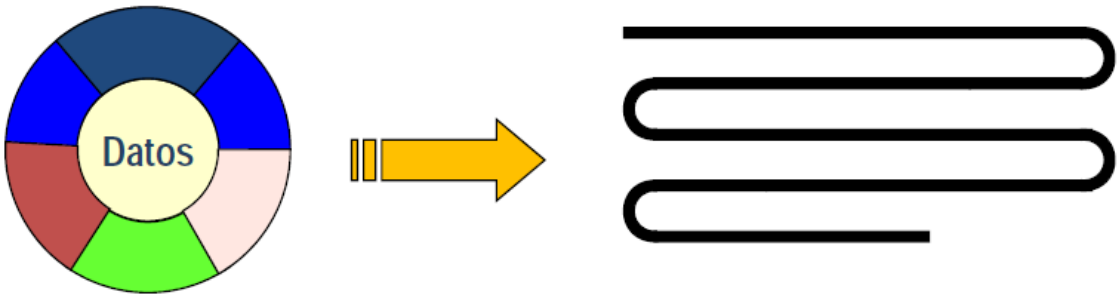
- Se trata de que un objeto pueda mantener su estado entre diferentes ejecuciones del programa.

- Una de las formas que existen para implementar la persistencia es almacenar los objetos en memoria secundaria antes de que el programa termine, y cargarlos de nuevo en memoria primaria en la siguiente ejecución

SERIALIZACIÓN

- Al implementar la persistencia, es frecuente que el estado de los objetos se almacene en un archivo.

- El archivo es una sucesión de bytes. Por tanto, es necesario "traducir" el estado del objeto a una sucesión lineal de bytes. Ese proceso se conoce como serialización.



- La serialización de objetos permite convertir cualquier objeto que implemente a la interfaz Serializable o la interfaz Externalizable en una secuencia de bits que puede ser utilizada posteriormente para reconstruir el objeto original.

- Esta secuencia de bits puede guardarse en un fichero o puede enviarse a otra máquina virtual (que puede estar ejecutándose en otro sistema operativo) para reconstruir el objeto (deserialización) en otro instante o en otra máquina virtual. No tenemos que preocuparnos en absoluto de las diferentes representaciones de datos en distintos ordenadores.

- Los objetos mantienen referencias a otros objetos. Estos otros objetos deben ser también almacenados y recuperados con el fin de mantener las relaciones originales. Por supuesto, todos estos objetos deben ser serializables ya que de lo contrario se lanzará una excepción del tipo NotSerializableException.

- Para reconstruir un objeto (o conjunto de objetos) Java serializado es necesario que la clase (o clases) esté en el classpath con el fin de indentificarla y verificarla antes de restaurar el contenido en una nueva instancia.

```java
import java.util.*;
import java.io.*;
public class Serial {
    public static void main(String arg[]){
        try{
            FileOutputStream archivo= new FileOutputStream("d:\\prueba.dat");
            ObjectOutputStream salida = new ObjectOutputStream(archivo);
            salida.writeObject("Hoy es: ");
            salida.writeObject(new Date());
            salida.close();
        }catch (IOException e) {
            System.out.println("Problemas con el archivo.");
        }
        try {
            FileInputStream archivo=new FileInputStream("d:\\prueba.dat");
            ObjectInputStream entrada=new ObjectInputStream(archivo);
            String hoy = (String) entrada.readObject();
            Date fecha = (Date) entrada.readObject();
            entrada.close();
            System.out.println(hoy + fecha);
        } catch(FileNotFoundException e){
            System.out.println("No se pudo abrir el archivo.");
        } catch (IOException e){
            System.out.println("Problemas con el archivo.");
        } catch (Exception e) {
            System.out.println("Error al leer un objeto.");
        }
    }
}
```

Ilustración 1. Lectura y escritura de objetos

```
Salida del programa: Hoy es: Wed Apr 04 15:35:43 CEST 2012
```

- Un objeto se puede serializar si implementa el interface Serializable. Este interface no declara ningún método, se trata de un interface vacío.

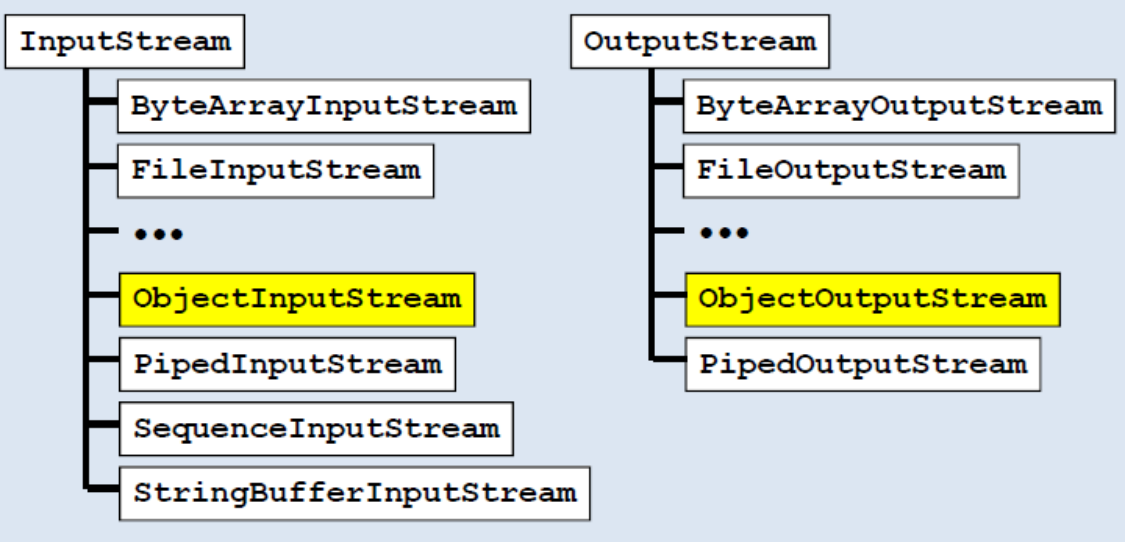
```
import java.io.*;
public interface Serializable{}
```

Para que un objeto sea serializable, todas sus variables de instancia han de ser serializables. Todos los tipos primitivos en Java son serializables por defecto (igual que los arrays y otros muchos tipos estándar).

- La serialización se introdujo en Java para implementar la persistencia y soportar la Invocación Remota de Métodos (RMI) que permite a una aplicación enviar mensajes a un objeto remoto (que

se esté ejecutando en otra máquina virtual). También es necesaria en el caso de los JavaBeans.

FLUJOS PARA ENTRADA Y SALIDA DE OBJETOS



- Los flujos para leer y escribir objetos tienen métodos que reciben o devuelven instancias de la clase Object

  void ObjectOutputStream.writeObject(Object o)

  Serializa un objeto y lo graba en el flujo al que se haya conectado el ObjectOutputStream

  Object ObjectInputStream.readObject()

  Carga el siguiente objeto del flujo al que esté conectado. Para usarlo hay que hacer un cast Ejemplo:

  Perro p = (Perro) miStream.readObject();

Ventajas Serialización:

El modelo de serialización de Java es muy potente

- Permite serializar con gran facilidad el estado de un objeto; requiere poco trabajo de nuestra parte
- No tenemos que elegir formatos de fichero ni escribir código de análisis léxico y sintáctico, etc. etc.
- El flujo al que se conecta el ObjectOutputStream puede hacer que el objeto serializado se grabe en un fichero, o se transmita por la red, o se guarde en un array de bytes, o...
- Los objetos agregados y complejos... ¡se serializan automáticamente!

Notas Extra:

- Al serializar no nos tenemos que preocupar cómo están implementados por dentro.
- Todas las clases de la API estándar de Java son serializables.
- Para serializar se utiliza el flujo ObjectOutputStream y el método writeObject. Si alguno de sus atributos es un objeto se serializa también. No se serializan ni atributos static ni transient.
- Para leer un objeto serializado se utiliza el flujo ObjectInputStream y el método readObject. Este método devuelve un Object por lo que habrá que realizar una conversión.

EL MODIFICADOR TRANSIENT, LA HERENCIA EN OBJETOS SERIALIZABLES, LA INTERFAZ EXTERNALIZABLE PARA EL CURSO QUE VIENE, MÓDULO “ACCESO A DATOS”