Persistencia

Algo3

FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Persistencia

Persistencia significa trascender en el tiempo y/o espacio.

En la informática hace referencia a la característica que puede tener un dato o estado que le permite sobrevivir al proceso que lo creó.

Persistencia en POO

Un ambiente orientado a objetos debe permitir que los objetos persistan, para mantener su vida más allá de la vida de la aplicación.

Este concepto nos permite que un objeto pueda ser usado en diferentes momentos a lo largo del tiempo, por el mismo programa o por otros, así como en diferentes instalaciones de hardware en el mismo momento.

Persistencia en POO

Un **objeto persistente** es aquel que conserva su estado en un medio de almacenamiento permanente, pudiendo ser reconstruido por el mismo proceso que lo generó u otro, de modo tal que al reconstruirlo se encuentre en el mismo estado en que se lo guardó.

Al objeto no persistente lo llamamos **efímero** o **transitorio**.

Persistencia en POO

Tipos de Persistencia:

Nativa

- Provista por la plataforma.

  *Ej: Java, Smalltalk.

No nativa

- A través de una biblioteca externa.
- Programada a mano.

El ideal en cuanto a persistencia sería que cada clase tenga un método para guardar objetos y otro para recuperarlos, consistentes entre sí. Sin embargo, hay una contradicción entre este ideal y el principio de separación de intereses.

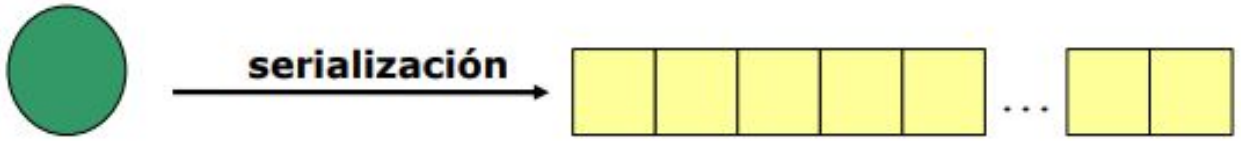
Persistencia en POO

Lo más habitual es que los lenguajes orientados a objetos no soporten la auténtica persistencia, si no, una variante que exige guardar y recuperar objetos en forma activa.

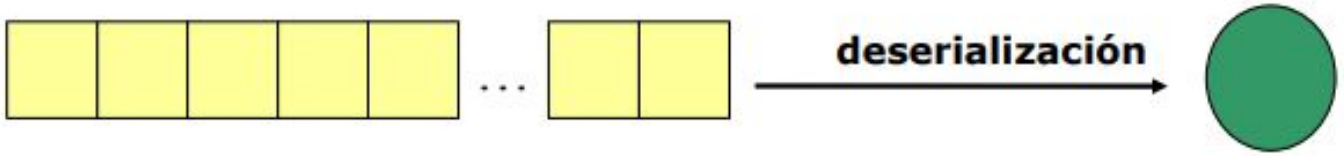
Una excepción es **Smalltalk** y su **persistencia basada en imágenes**.

Serialización

Proceso que consiste en convertir la representación de un objeto en un *stream (flujo o secuencia)* de bytes.



Reconstruir un objeto a partir de un stream de bytes se denomina **deserialización**.



Serialización

Formatos de serialización:

- Formato propietario
  - Más eficiente en términos de uso de almacenamiento y tiempo de traducción.
  - Solo sirve para comunicar aplicaciones basadas en la misma plataforma.

- Uso de lenguaje estándar:
  - Ej: JSON

    -Es el más popular lenguaje de intercambio de información.

    -Al ser formato de texto ocupa más espacio y su estructura no propietaria precisa hacer transformaciones en términos de computo.

Java posee serialización automática en formato propietario, para trabajar con JSON hay bibliotecas.

Persistencia y serialización

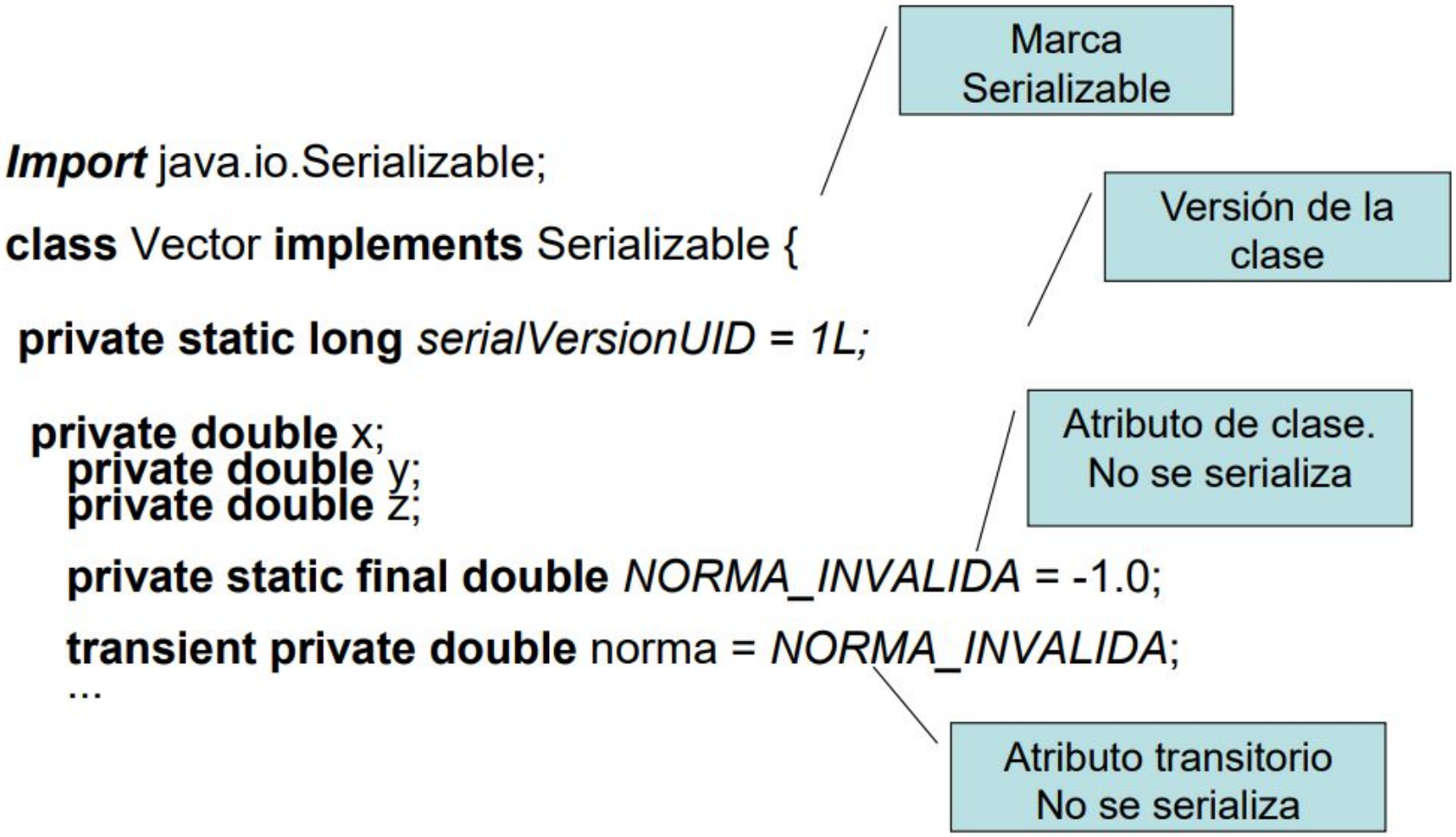
- Para persistir debo primero serializar.
- Serializar no implica necesariamente persistir.
- Otras acciones luego de serializar:
  - Enviar por red.
  - Mantener en memoria.
  - Enviar a una impresora, etc.

Serialización en Java

Para definir una clase serializada, debe implementarse la interfaz **Serializable.**

- No tiene métodos.
- Sirve para avisarle a la máquina virtual que la clase puede serializarse (*marker interface*).
- Todas las subclases de una clase serializable son serializables también.
- Por defecto, se serializan todos los atributos de un objeto que no posean el modificador `transient` (*transitorio)*.
- Si un objeto serializable tiene referencias a otro objeto serializable, también lo serializa.
- Si algún objeto del "árbol de objetos" a serializar no es serializable se lanza la excepción NonSerializableException.
- Los objetos serializables deben tener un constructor sin parámetros para poder ser deserializados correctamente.

Serialización en Java



Persistencia en Java

- Paquete java.io
  - De objeto a archivo

```
UnaClase objeto = new UnaClase();
OutputStream fos = new FileOutputStream("objeto.dat");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(objeto);
```

  - De archivo a objeto

```
InputStream fis = new FileInputStream("objeto.dat");
ObjectInputStream ois = new ObjectInputStream(fis);
Clase objeto = (Clase) ois.writeObject(objeto);
```

Persistencia en Java

Persistencia Nativa

Ventajas

- Es nativa del lenguaje (casi no hay que programar)
- Resuelve referencias circulares

Desventajas

- No es portable a otros lenguajes de manera sencilla .
- No es óptima en cuanto a tamaño (tiene overhead alto).
- La información en el archivo es binario.
- No es extensible ni reparable.

Serialización en Smalltalk

- Objeto a Archivo

```
objeto := MiClase new.
rr := ReferenceStream fileNamed: 'objeto.bin'.
rr nextPut: objeto; close.
```

- Archivo a Objeto

```
rr := ReferenceStream fileNamed: 'objeto.bin'.
objeto := rr next.
rr close.
```

Ej: Serialización no nativa

- Persistencia en Texto (XML)

instancias

```
Disco disco = new Disco(50, “Queen”);
disco.addPista(new Pista(4,”the show must go on”);
disco.addPista(new Pista(3, “inuendo”);
```

Stream de texto

```
<disco duracion=“50” autor=“Queen”>
        <pista duracion=“4” titulo=“the show must go on” />
        <pista duracion=“3” titulo=“inuendo” />
</disco>
```

Persistencia no nativa(XML)

- Responsabilidad: cada clase conoce como serializarse.

- Cada clase serializable tendrá:
  - Un método serializar que devolverá un nodo XML.
  - Un constructor sobrecargado que recibirá un nodo XML

Persistencia no nativa(XML)

```
public class Disco{
    private int duracion;
    private string autor;
    private List pistas = new ArrayList();

    public Disco(XMLNode nodo){
          duracion=Integer.ParseInt)nodo.getAttribute("duracion"));
          autor=nodo.getAttribute("autor");
          List nodosPistas = nodo.getChildrens("pista");
          for(XMLNode nodoPista : nodosPistas){
                  pistas.add(new Pista(nodoPista);
          }
    }

    public XMLNode serializar(){
          XMLNode nodo = CrearNodo();
          nodo.setAttribute("autor",this.autor);
          nodo.setAttribute("duracion",this.duracion);
          for(Pista pista : this.pistas){
                  nodo.addChildren(pista.serializar());
          }
    }

}
```

Deserialización

Serialización

La clase pista es similar

Persistencia no nativa(XML)

- Escritura a archivo

```
XmlDocument doc =CreateDocFromNode(disco.serializar());
XMLHelper.SaveXMLDocToFile(doc,"disco.xml");
```

- Lectura desde archivo

```
XMLDocumento doc =XmlHelper.ReadFromFile("disco.xml");
Disco disco = new Disco(doc.getRootNode);
```

Persistencia no nativa

- Queda en manos del programador
  - Cuestiones de diseño
    - Responsabilidad
      - Cada clase sabe como persistirse
      - Existe un gestor externo que sabe como persistir las clases
    - Formato
      - Binario, texto plano, texto jerárquico (XML)
    - Identidad de los objetos
      - Referencias circulares, duplicación de objetos, etc
  - Mayor versatilidad pero…
  - Mayor complejidad