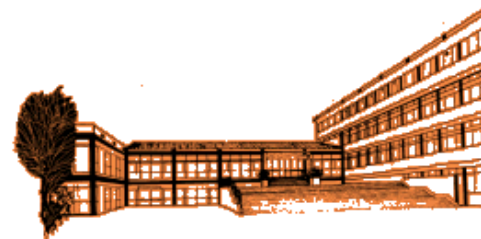




INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

# ISEL



## Instituto Superior de Engenharia de Lisboa

Área Departamental de Engenharia da Electrónica  
das Telecomunicações e dos Computadores

# Infraestruturas Computacionais Distribuídas

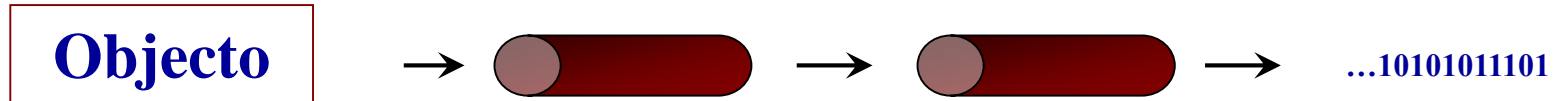
# Seriação em Java



# Seriação

- Classes contidas no **package java.io.\***;
- Transformar qualquer objecto que implemente o interface *Serializable* numa sequência de *bytes*

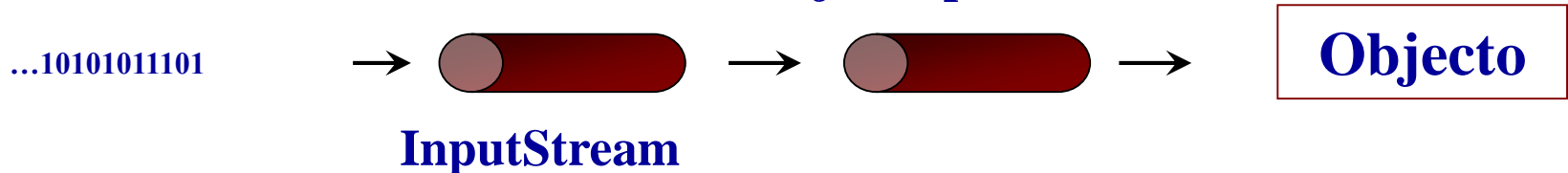
**ObjectOutputStream**



**OutputStream**

- Poder reconstruir a partir da sequência de bytes o objecto original (estado de um objecto)

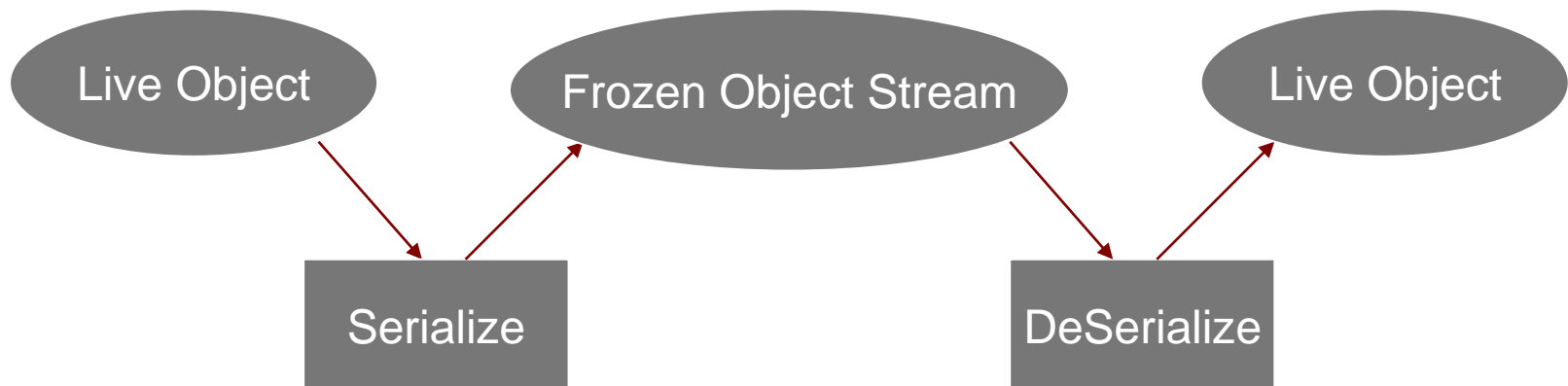
**ObjectInputStream**





# Seriação de Objectos

- ❏ A seriação permite representar um objecto num formato que automatiza o seu armazenamento, disponibilizando igualmente a forma de o reconstruir.
- ❏ Para que os objectos possam ser seriados
  - ❏ A sua classe tem de implementar a interface *Serializable*
  - ❏ Os atributos não persistentes são marcados com a palavra chave *transient*





# Persistência – Escrita num ficheiro

```
import java.io.*;
public class Example {
    // Must enter a filename and a buffer size on the command line, as in:
    // $ java Example outfile.dat 32
    // The above command line will cause this program to write out int values 0 to 99
    // to the file named filename. Output will be buffered in a buffer of size 32.
    public static void main(String[] args) throws IOException {
        if (args.length < 2) {
            System.out.println(
                "Must enter filename and buffsize as arguments.");
            System.exit(0);
        }
        int buffSize = Integer.parseInt(args[1]);
        FileOutputStream fos = new FileOutputStream(args[0]);
        BufferedOutputStream bos = new BufferedOutputStream(fos, buffSize);
        DataOutputStream dos = new DataOutputStream(bos);
        for (int i = 0; i < 100; ++i) {
            dos.writeInt(i);
        }
        dos.close();
    }
}
```



# Seriação de uma *String*


 Criar um *OutputStream*, por exemplo para escrita num ficheiro

```
 OutputStream os = new FileOutputStream("meusDados");
```

 Criar um *ObjectOutputStream*

```
 ObjectOutputStream oos = new ObjectOutputStream(os);
```

 Escrever o objecto (por ex. uma *String*)

```
 String s = new String("Sou um objecto");
```

```
 oos.writeObject(s);
```

 Fecho do ficheiro e do *ObjectOutputStream*

```
 oos.close();
```

```
 os.close();
```



# Reconstrução de uma *String*

📄 Abrir o ficheiro que contém o objecto

```
InputStream is = new FileInputStream("meusDados");
```

📄 Criar um *ObjectInputStream*

```
ObjectInputStream ois = new ObjectInputStream(is);
```

📄 Ler o objecto

```
String s = (String) ois.readObject();
```

📄 Fechar o ficheiro e o *ObjectInputStream*

```
ois.close();
```

```
is.close();
```



# Seriação de uma data

## Escrita de uma data no ficheiro “*afile*”

```
FileOutputStream out = new FileOutputStream("afile") ;  
ObjectOutputStream oos = new ObjectOutputStream(out) ;  
oos.writeObject("Today") ;  
oos.writeObject(new Date()) ;  
oos.flush() ;
```

## Leitura de uma data do ficheiro “*afile*”

```
FileInputStream in = new FileInputStream("afile") ;  
ObjectInputStream ois = new ObjectInputStream(in) ;  
String today = (String) ois.readObject() ;  
Date date = (Date) ois.readObject() ;
```



# Seriação de uma *Hashtable*

```
Hashtable settings = new Hashtable(3);  
settings.put("buffers", new Integer(20));  
settings.put("layers", new Integer(2));  
settings.put("title", "navigator");  
  
...  
FileOutputStream fos = new FileOutputStream("settings.dat");  
ObjectOutputStream oos = new ObjectOutputStream(fos);  
oos.writeObject(settings);  
oos.close();  
fos.close();
```







# Reconstrução de uma *Hashtable*

```
FileInputStream fis = new FileInputStream("settings.dat");  
ObjectInputStream ois = new ObjectInputStream(fis);  
Hashtable settings = (Hashtable)ois.readObject();  
ois.close();  
fis.close();
```



# Criação de uma Classe Seriável

-  Incluir na definição “*implements Serializable*”, serve de marcador a indicar que as instâncias desta classe podem ser seriadas
-  Os atributos estáticos (*static*) e os indicados na declaração com a keyword “*transient*” não são seriados

```
public class Dados implements java.io.Serializable{  
    public String s1;  
    private String s2;  
    private int i;  
  
    public Dados(int i, String s1, String s2) {  
        this.i = i;  
        this.s1 = s1;  
        this.s2 = s2;  
    }  
}
```

**Classe  
Seriável**

**Exemplo de uma  
escrita de um  
objecto num  
ficheiro**

```
Dados dado = new Dados(1234, "string1", "string2");  
// Vamos enviar o objecto Date  
oos = new ObjectOutputStream(new FileOutputStream("meusDados"));  
oos.writeObject(dado);  
oos.flush();  
oos.close();
```



# Atenção ao *caching*

As *streams* para escrita de objectos efectuem *caching* dos objectos enviados

Problema:

```
Xpto x = new Xpto();  
x.xpto = 0;  
oos.writeObject(x);  
oos.flush();  
x.xpto=10;  
oos.writeObject(x);  
oos.flush();
```

```
Xpto w =(Xpto) Ois.readObject();  
System.out.println("Contem: " + w.xpto);  
Xpto w =(Xpto) Ois.readObject();  
System.out.println("Contem: " + w.xpto);
```

Contem: 0

Contem: 0

**ERRADO!**  
**Deveria ser**  
**10**

Solução1:

Ⓢ Chamar o método `objectOutputStream.reset()`, que provoca a limpeza da *cache* de referências para objectos

Solução2:



Ⓢ Após cada escrita fechar o `ObjectOutputStream`



# Justificação

## **Object Serialization Pitfall** (an intellectual error that traps a researcher, perhaps forever or an unforeseen or unexpected or surprising difficulty)

When working with object serialization it is important to keep in mind that the *ObjectOutputStream* maintains a hashtable mapping the objects written into the stream to a handle. When an object is written to the stream for the first time, its contents will be copied to the stream. Subsequent writes, however, result in a handle to the object being written to the stream. This may lead to a couple of problems:

-  If an object is written to the stream then modified and written a second time, the modifications will not be noticed when the stream is deserialized. Again, the reason is that subsequent writes results in the handle being written but the modified object is not copied into the stream. To solve this problem, call the *ObjectOutputStream.reset* method that discards the memory of having sent an object so subsequent writes copy the object into the stream.
-  An *OutOfMemoryError* may be thrown after writing a large number of objects into the *ObjectOutputStream*. The reason for this is that the hashtable maintains references to objects that might otherwise be unreachable by an application. This problem can be solved simply by calling the *ObjectOutputStream.reset* method to reset the object/handle table to its initial state. After this call, all previously written objects will be eligible for garbage collection.

The *reset* method resets the stream state to be the same as if it had just been constructed. This method may not be called while objects are being serialized. Inappropriate invocations of this method result in an *IOException*.



# Considerações

- ❏ **ObjectOutputStream.writeObject(Object)**, escreve recursivamente todo o conteúdo do objecto
- ❏ **ObjectOutputStream**, implementa a interface **DataOutput** para escrever tipos primitivos:
  - writeInt(...), writeFloat(...), etc.
- ❏ **ObjectInputStream**, implementa a interface **DataInput** para ler tipos primitivos:
  - readInt(), readFloat(), etc.
- ❏ **writeObject(Object)** produz a excepção **NotSerializableException** se o objecto não implementar a interface *Serializable*.



# Seriação Tipos primitivos

## **Escrita de uma *String* e de um número no ficheiro “Beans.tmp”**

```
// save a string and double to the stream
String str = "Sample";
double d = 3.14;
FileOutputStream f = new FileOutputStream ("Beans.tmp")
ObjectOutputStream s = new ObjectOutputStream(f);
s.writeObject(str);
s.writeDouble(d);
s.flush();
```

## **Leitura de uma *String* e de um numero do ficheiro “Beans.tmp”**

```
// restore the string and double
FileInputStream f = new FileInputStream("Beans.tmp");
ObjectInputStream s = new ObjectInputStream(f);
String str = (String)s.readObject();
double d = s.readDouble();
```



# Exemplo de Serialização

```
import java.io.* ;
import java.util.* ;
class A implements Serializable {
    public int i = 5 ;
    public String str = "Hi" ;
    public List l = new ArrayList() ;
}
public class ObjSerTest {
    public static void main(String[]args) {
        A a = new A() ;
        a.i = 10 ;           a.str = "Hello" ;
        a.l.add("One") ;     a.l.add("Two") ;
        serialize(a) ;
    }
    private static void serialize(A a) {
        System.out.println("Serializing...");
        try {
            FileOutputStream fos = new FileOutputStream("test.out") ;
            ObjectOutputStream oos = new ObjectOutputStream(fos) ;
            oos.writeObject(a) ;
        } catch (Exception e) {
            System.err.println("Problem: "+e) ;
        }
    }
}
```



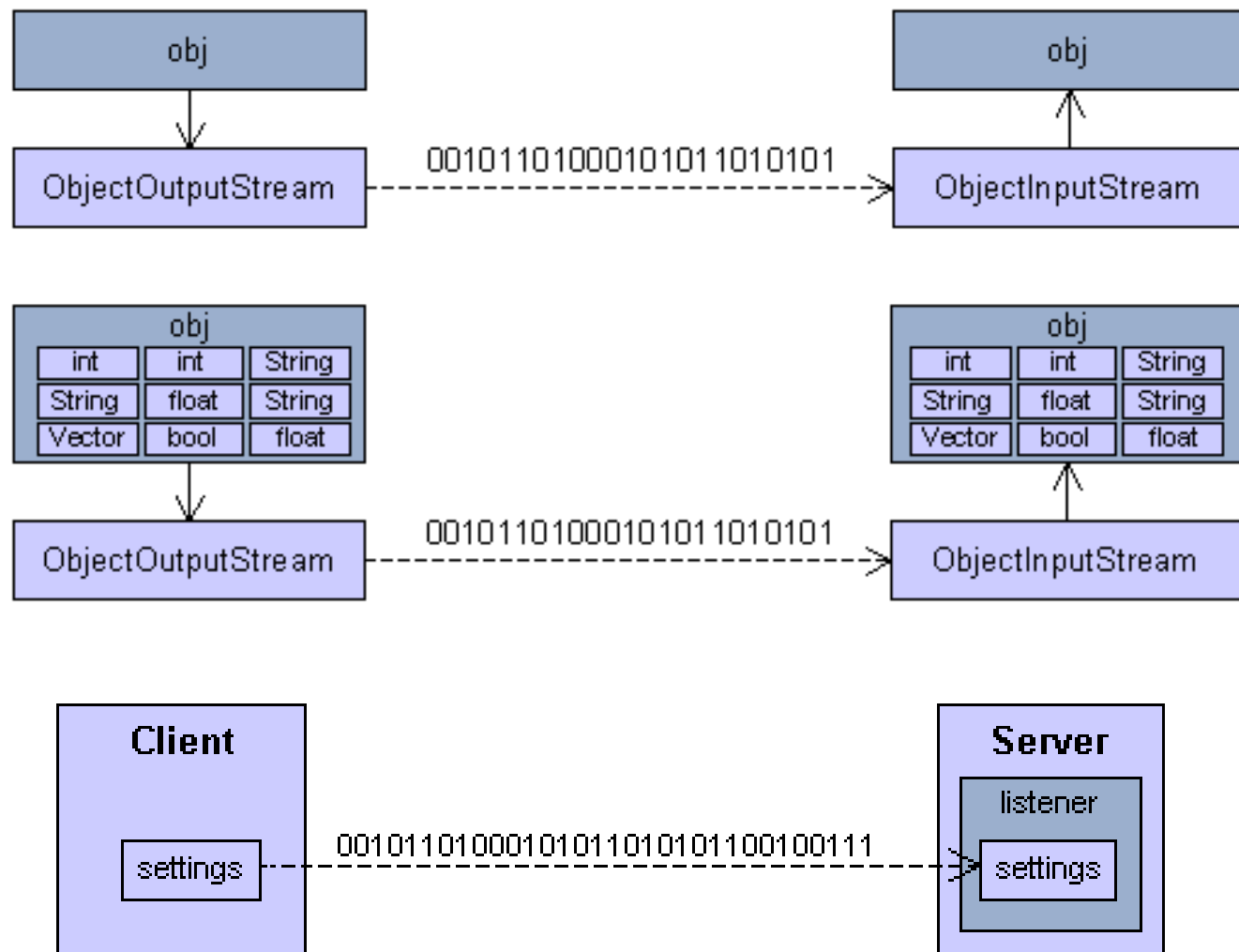
# Exemplo de Reconstrução

```
import java.io.* ;
import java.util.* ;
class A implements Serializable {
    public int i = 5 ;
    public String str = "Hi" ;
    public List l = new ArrayList() ;
}
public class ObjDeSerTest {
    public static void main(String[]args) {
        A a = deserialize() ;
        System.out.println(a.i) ;
        System.out.println(a.str) ;
        System.out.println(a.l) ;
    }
    private static A deserialize() {
        System.out.println("DeSerializing...");
        try {
            FileInputStream fis = new FileInputStream("test.out") ;
            ObjectInputStream ois = new ObjectInputStream(fis) ;
            return (A) ois.readObject() ;
        } catch (Exception e) {
            System.err.println("Problem: "+e) ;
        }
        return null ;
    }
}
```





# Seriação Cliente-Servidor





# Cliente envia Hashtable

```
Hashtable settings = new Hashtable(3);  
settings.put("buffers", new Integer(20));  
settings.put("layers", new Integer(2));  
settings.put("title", "navigator");
```

```
InetAddress ia = InetAddress.getByName("localhost");  
Socket socket = new Socket(ia, 9999);  
ObjectOutputStream oos = new  
    ObjectOutputStream(socket.getOutputStream());
```

```
oos.writeObject(settings);
```

```
oos.flush();  
oos.close();  
socket.close();
```



# Servidor Recebe Hashtable

```
ServerSocket listener = new ServerSocket(9999);  
while(true){  
    Socket socket = listener.accept();  
    DataInputStream dis = new  
        DataInputStream(socket.getInputStream());  
    ObjectInputStream ois = new ObjectInputStream(dis);  
    Hashtable settings = (Hashtable)ois.readObject();  
    System.out.println("Got object: " + settings.toString());  
    ois.close();  
    dis.close();  
    socket.close();  
}  
  
// note-se que este servidor é iterativo, podia também ser  
    concorrente!
```



# Tratamento especial



Classes que requeiram um tratamento especial na Serialização devem implementar os seguintes métodos:



```
private void readObject(java.io.ObjectInputStream stream) throws IOException,  
ClassNotFoundException  
    stream.defaultReadObject();  
    ... //Resto do código específico para ler do Stream os dados escritos no  
        //writeObject do emissor.  
}
```



```
private void writeObject(java.io.ObjectOutputStream stream) throws IOException {  
    stream.defaultWriteObject();  
    ... //Resto do código específico para escrever no Stream os dados que  
        //não são escritos automaticamente (elementos static e transient).  
}
```

**Para efectuar processamento adicional no  
momento da Serialização**



# Exemplo de Utilização de Pipes

 Duas Threads a transferir o objecto `java.util.Date` de 1 em 1 segundo através de Pipes

```
package exemplopipesStream;  
import java.io.PipedInputStream;  
import java.io.PipedOutputStream;  
import java.io.IOException;
```

```
public class PipeEx {
```

```
    public static void main(String[] args) throws IOException{  
        System.out.println("Exemplo de Serialização...");
```

```
        PipedOutputStream out = new PipedOutputStream();  
        PipedInputStream in = new PipedInputStream(out);
```

```
        GeradorDeDados dadosGerador = new GeradorDeDados (out);  
        dadosGerador.start();
```

```
        ConsumidorDeDados dadosConsumidor = new ConsumidorDeDados (in);  
        dadosConsumidor.start();
```

```
    }
```

```
}
```

Note que a classe `Date` implementa `Serializable`

**Criação da Thread geradora do objecto `Date`**

**Criação do Pipe**

**Criação da Thread consumidora do objecto `Date`**



# Gerador de dados

```
package exemplopipesStream;
import java.io.PipedOutputStream;
import java.io.ObjectOutputStream;
import java.util.Date;

public class GeradorDeDados extends Thread{
    PipedOutputStream out;
    ObjectOutputStream oos;
    public GeradorDeDados(PipedOutputStream out) {
        this.out = out;
        this.oos = new ObjectOutputStream(out);
    }

    public void run(){
        System.out.println("Thread Geradora de dados a correr...");
        try{
            for(;;){
                Date d = new Date();
                System.out.println(d.toString());
                // Vamos enviar o objecto Date
                oos.writeObject(d);
                oos.flush();

                Thread.currentThread().sleep(1000); // enviar só de segundo a segundo
            }
        }catch(Exception e){System.err.println(e);}
        try{
            oos.close();
            out.close();
        }catch(Exception ie){ie.printStackTrace();}
    }
}
```

**Objecto a enviar**

**Escrita do objecto  
na stream de dados**



# Consumidor de dados

```
package exemploPipesStream;
import java.io.PipedInputStream;
import java.io.ObjectInputStream;
import java.util.Date;

public class ConsumidorDeDados extends Thread{
    PipedInputStream in;
    ObjectInputStream ois;
    public ConsumidorDeDados(PipedInputStream in) {
        this.in = in;
        ois = new ObjectInputStream(in);
    }

    public void run(){
        System.out.println("Thread Consumidora de dados a correr...");
        try{
            for(;;){
                Object d = ois.readObject();
                System.out.println("Recebi o objecto :" + d.toString());
                if ( d instanceof Date ){
                    System.out.println("Recebi o objecto do tipo Date");
                    Date dd = (Date) d;
                }
            }
        }catch(Exception e){System.err.println(e);}
        try{
            ois.close();
            in.close();
        }catch(Exception ie){ie.printStackTrace();}
    }
}
```

Se o tipo esperado fosse conhecido poderia ser feito logo o cast Ex:

**Date d=(Date) ois.readObject();**

**Leitura do objecto**

**Descobrir se o objecto é uma instancia da classe esperada**



# “Bibliografia” utilizada

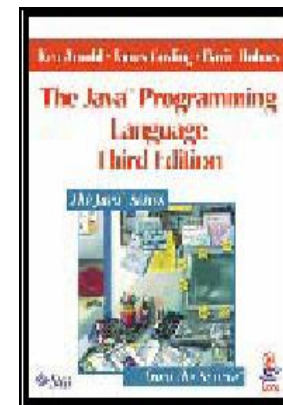
## Hierarquia das classes no *package* java.io

 <http://java.sun.com/j2se/1.4.2/docs/api/java/io/package-tree.html>

## The Java™ Programming Language, 3<sup>rd</sup> Edition

 Arnold, Gosling, Holmes

 Sun® Microsystems



## Segredos da API Java para a serialização de objectos

 <http://developer.java.sun.com/developer/technicalArticles/Programming/serialization>