

Serialização

Serialização

- E se quisermos ler ou escrever Objectos em Ficheiros?
 - **Serialização:** permite tornar persistentes os objectos
- O processo de Serialização é complicado em muitas linguagens
 - Podemos ter objectos contendo referências para outros objectos...
- Java permite implementar Serialização de forma simples
- **Definição:** Serialização é o processo de transformar um objecto numa sequência (stream) de bytes

Serialização

- Para que uma classe seja serializável basta que implemente a interface *Serializable* (que é uma interface vazia!)

```
package java.io;  
public interface Serializable {  
    // there's nothing in here!  
};
```

- **Serializable** - Permite simplesmente indicar quais as classes serializáveis

Condições de Serialização

- A classe deve ser declarada como public
- A classe deve implementar `Serializable`
- Todos os atributos (dados) devem ser serializáveis:
 - Tipos primitivos (int, double, ...)
 - Objectos serializáveis

Serialização - Algumas Considerações

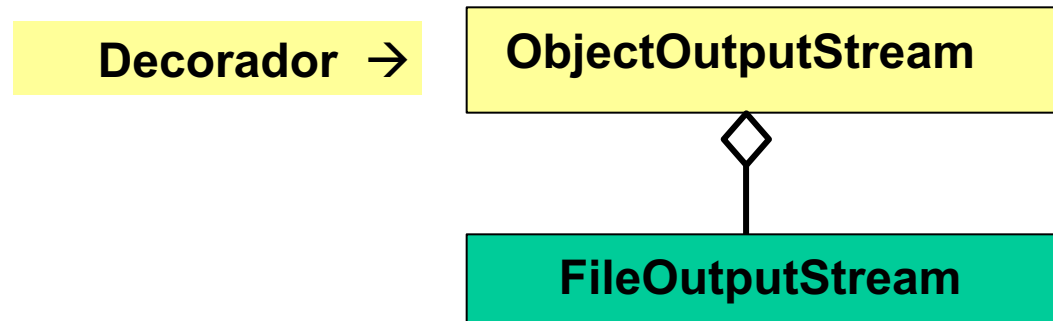
- Um atributo definido como `transient` não será “empacotado” no processo de serialização.
 - No processo de desserialização os atributos assumirão valores de defeito.
- Atributos do tipo `static` não são serializados.
- Se uma classe B serializable tem uma super-classe A que não é serializable, então objectos do tipo B podem ser serializados ... desde que a classe A tenha um construtor sem argumentos acessível.

Serialização - serialVersionUID

- Atributo **Muito Importante**
- Deve ser sempre incluído na Classe:
 - `private static final long serialVersionUID = 75264722956L;`
- Não deve ser alterado em versões futuras das classes, excepto...
- ... ambas as versões gerarem objectos incompatíveis
 - A compatibilidade de novas versões com objectos antigos depende da natureza das alterações.

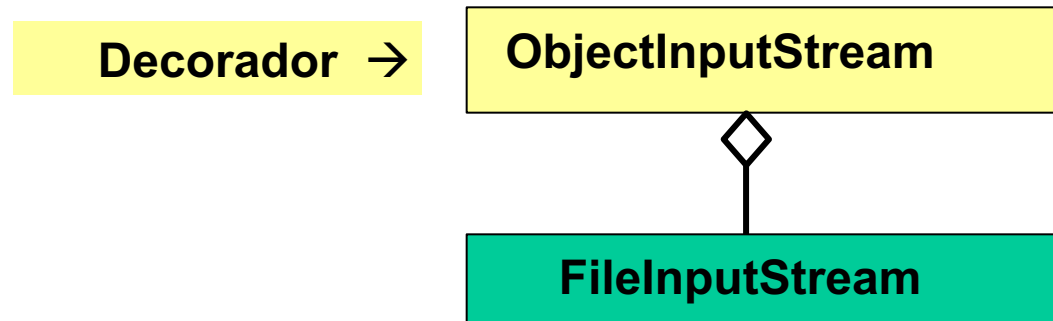
Escrita de Objectos em Ficheiro

```
ObjectOutputStream objectOut =  
new ObjectOutputStream(  
    new FileOutputStream(fileName));  
  
objectOut.writeObject(serializableObject);  
  
objectOut.close( );
```



Leitura de Objectos de Ficheiro

```
ObjectInputStream objectIn =  
new ObjectInputStream(  
    new FileInputStream(fileName));  
  
myObject = (ObjectType)objectIn.readObject();  
  
objectIn.close( );
```



Exemplo - Serialização

- **ObjectOutputStream**

```
FileOutputStream out = new FileOutputStream("Time");  
ObjectOutputStream s = new ObjectOutputStream(out);  
s.writeObject("Today");  
s.writeObject(new Date());  
s.flush();
```

- **ObjectInputStream**

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

- A leitura faz-se pela mesma ordem da escrita

Exemplo - Escrita Objectos

```
public static void main(String[] args) throws FileNotFoundException, IOException {  
    Data d = new Data(11,2,2001);  
    Pessoa p = new Pessoa("Carlos Costa", 234342124, new Data(22,11,1972));  
  
    ObjectOutputStream objectOut = new ObjectOutputStream(  
        new FileOutputStream("c:/out.bin"));  
  
    objectOut.writeObject(d);  
    objectOut.writeObject(p);  
    objectOut.close();  
}
```



C:\out.bin

ACED	0005	7372	0007	494F	2E44	6174	612D	C51E	40EE	1B7D	8E02	0003	4900	0361sr..IO.Data-..@..}....I..a
6E6F	4900	0364	6961	4900	036D	6573	7870	0000	07D1	0000	000B	0000	0002	7372	noI..dial..mesxp.....sr
0009	494F	2E50	6573	736F	6171	314B	6257	84CE	2102	0003	4900	0262	694C	0008	..IO.Pessoaq1KbW...!...I..biL..
6461	7461	4E61	7363	7400	094C	494F	2F44	6174	613B	4C00	046E	6F6D	6574	0012	dataNasct..LIO/Data;L..nomet..
4C6A	6176	612F	6C61	6E67	2F53	7472	696E	673B	7870	0DF7	C6EC	7371	007E	0000	Ljava/lang/String;xp....sq.~..
0000	07B4	0000	0016	0000	000B	7400	0C43	6172	6C6F	7320	436F	7374	61	t..Carlos Costa

Serialização - Utilização

- Persistência
 - Com FileOutputStream
 - Armazena as estruturas de dados em ficheiro para mais tarde recuperar
- Cópia
 - Com ByteArrayOutputStream
 - Armazena as estruturas de dados em memória (array) para poder criar duplicados
- Comunicações
 - Utilizando um stream associado a um Socket
 - Envia as estruturas de dados para outro computador

Serialização - Deep Copy

```
// serialize object
ByteArrayOutputStream mOut = new ByteArrayOutputStream();
ObjectOutputStream serializer = new ObjectOutputStream(mOut);
serializer.writeObject(serializableObject);
serializer.flush();

// deserialize object
ByteArrayInputStream mIn = new
    ByteArrayInputStream(mOut.toByteArray());
ObjectInputStream deserializer = new ObjectInputStream(mIn);
Object deepCopyOfOriginalObject = deserializer.readObject();
```

Serialização - Controlar o Processo

- Como podemos controlar o processo de serialização e desserialização?

Reescrevendo os métodos **writeObject** e **readObject**:

...

String a;

transient **String** b;

```
private void writeObject(ObjectOutputStream stream)
    throws IOException {
    stream.defaultWriteObject();
    stream.writeObject(b);    // forçar a ser. de b
}
```

```
private void readObject(ObjectInputStream stream)                                throws
IOException, ClassNotFoundException {
    stream.defaultReadObject();
    b = (String) stream.readObject();
}
...
```

Jar Files

O que são Jar files?

- O Java Archive (JAR) permite a inclusão de múltiplos ficheiros num único ficheiro arquivo.
- Tipicamente, o ficheiro JAR contém “.class files” e recursos auxiliares associados com applets ou aplicações.
- Os ficheiros JAR são compactados em formato ZIP
 - Podemos utilizar o “Winzip” para manipular JARs

Vantagens

- **Compressão:** O arquivo JAR comprime os seus conteúdos.
 - Aumento da eficiência no transporte (- tempo download) e arquivo (- espaço disco)
- **Segurança:** Os ficheiros JAR podem ser assinados digitalmente.
 - autenticação da proveniência.
 - privilégios do software baseados na certificação da origem.

Vantagens

- **Packaging for extensions**: é possível adicionar novas funcionalidades ao Java *core platform*, utilizando arquivos Jar.
- **Package Sealing**: forçar a consistência de versões.
 - Todas as classes definidas no package devem ser encontradas no mesmo arquivo Jar.
- **Package Versioning**: suporta informação relativa ao software: vendedor, versão, etc.
- **Portabilidade**: suporte de JARs é uma componente standard do Java *platform's core API*.

Java Archive Tool - comando jar

Operações

- create a JAR file
- view the contents of a JAR file
- extract the contents of a JAR file
- extract specific files from a JAR file
- run an application packaged as a JAR file (version 1.2 -- requires Main-Class manifest header)

Comando

`jar cf jar-file input-file(s)`

`jar tf jar-file`

`jar xf jar-file`

`jar xf jar-file archived-file(s)`

`java -jar app.jar`

Jar - Manifest File

- Ficheiro especial que contém diversos tipos de ‘Meta’ informação relativas ao arquivo JAR:
 - electronic signing, version control, package sealing, entry-point, ...
- Na criação de um JAR é criada uma “default manifest file”

META-INF/MANIFEST.MF



Manifest-Version: 1.0

Created-By: 1.6.0 (Sun Microsystems Inc.)

Executable JAR archive

- Como tornar uma aplicação em Java num arquivo JAR executável?

1. Colocar todas as classes num directório (estrutura árvore)
2. Criar um arquivo JAR com esse directório
3. Adicionar na Manifest File um *entry-point*
Main-Class: classname
4. A main-class deve ter o método
public static void main(String[] args)

```
Manifest-Version: 1.0
```

```
Class-Path: .
```

```
Main-Class: aula1.Palindrome
```

5. Para executar
\$ java -jar app.jar