# Serialização

# Serialização

*Serialization* – Escrever, para um *stream*, um objecto e todos aqueles que ele directa ou indirectamente referencia.

*Deserialization* – Restaurar, a partir de um *stream*, um objecto serializado e todos os que ele directa ou indirectamente referencia.

Para poder ser serializado um objecto tem de ser instância de uma classe que implemente a interface **`Serializable`**.

- Ex: generalidade das classes de java.lang, arrays, contentores.

# *Streams* para serialização

Filtros de *streams* para serialização de dados (especificados pelas interfaces **`DataOutput`** e **`DataInput`**):

- **`DataOutpuStream`** `writeInt(int), writeFloat(float), ...`
- **`DataInputStream`** `int readInt(), float readFloat(), ...`

Os objectos podem ser serializados com:

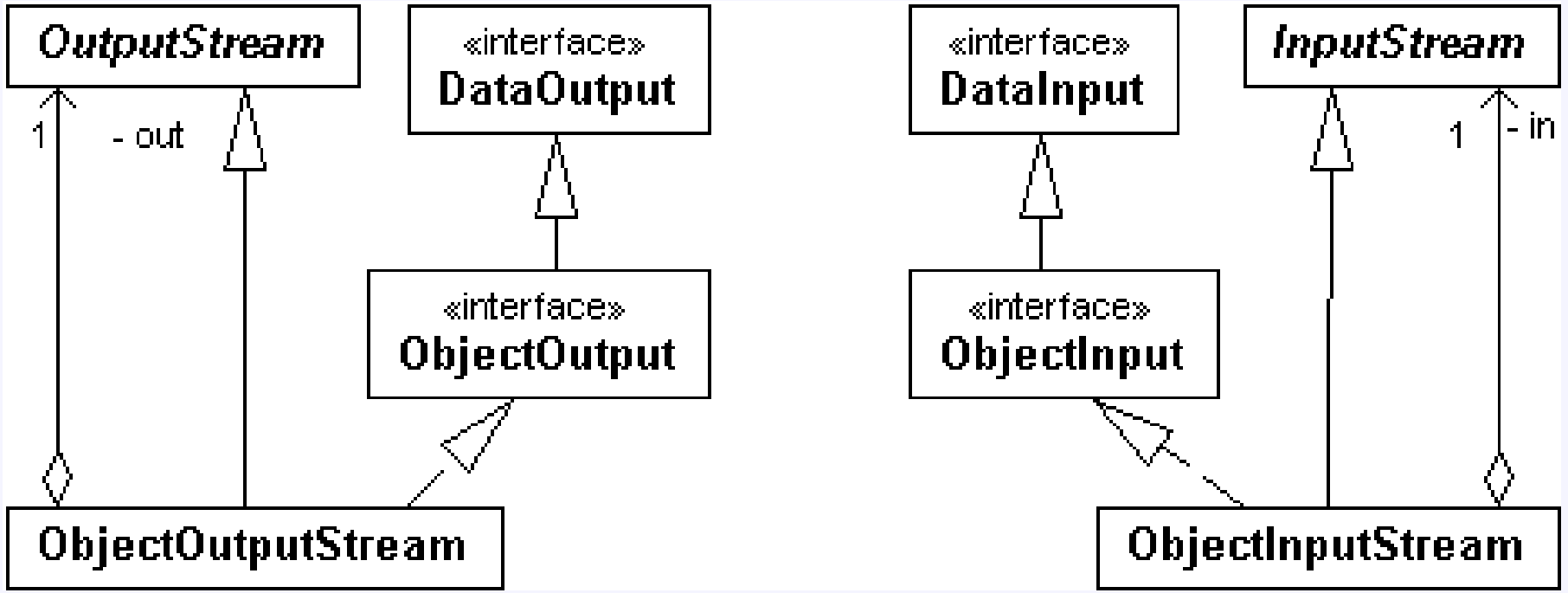
- **`ObjectOutputStream`** `writeObject(Object)`
- **`ObjectInputStream`** `Object readObject()`

(especificados pelas interfaces **`ObjectOutput`** e **`ObjectInput`**)

```
Object obj =...; // Qualquer objecto
OutputStream out =...;
...
ObjectOutputStream os;
os = new OjectOutputStream( out );
os.writeObject( obj );
```

```
Object obj;  // Apenas referência
InputStream in =...;
...
ObjectInputStream is;
is = new OjectInputStream( in );
obj = is.readObject();
```

# Serialização de Objectos



# Serializar um objecto

O método `writeObject(Object obj)` de `ObjectOutputStream` serializa o valor de cada campo para a *stream* `ObjectOutputStream`:

- Para cada campo `p` de tipo primitivo é invocado o método `writeXXX(p)` especificado por `DataOutput`.
- Para cada campo `r` de tipo complexo é invocado recursivamente o método `writeObject(r)`.
- Os atributos `static` e `transient` não são serializados.

O método `readObject()` de `ObjectInputStream` retorna um novo objecto em memória igual ao que foi serializado com `writeObject(obj)`.

# interface `Serializable`

As classes de objectos serializáveis devem implementar a interface `Serializable`. Para marcar a classe e o compilador gerar automaticamente os métodos: `writeObject` e `readObject`.

Os métodos `writeObject` e `readObject` podem ser redefinidos para realizar uma serialização proprietária, ou para lançar a excepção `NotSerializableException`.

```
private void writeObject(ObjectOutputStream out)
  throws IOException {
    out.defaultWriteObject();
}
private Object readObject(ObjectInputStream in)
  throws IOException, ClassNotFoundException {
    in.defaultReadObject();
}
```

Os métodos `defaultReadObject()` e `defaultWriteObject()` é que fazem o trabalho "pesado".

# writeObject & readObject

O método `writeObject(Object)` de `ObjectOutputStream`:

➔ invoca `writeObject(ObjectOutputStream)` do objecto a escrever.

O método `readObject()` de `ObjectInputStream`

➔ invoca o método `readObject(ObjectInputStream)` do objecto a ler.

```
class ObjectOutputStream {
  void writeObject( Object obj )
  throws IOException {
    Escreve_ID_do_objecto;
    if ( ! objecto_escrito ) {
      Escreve_UID_da_classe;
      if ( ! classe_escrita )
        Escreve_classe;
      obj.writeObject( this );
    }
  }
  ...
}
```

```
class ObjectIntputStream {
  Object readObject()
  throws IOException, ClassNotFoundException {
    Object obj;
    Lê_ID_do_objecto;
    if ( ! objecto_lido ) {
      Lê_UID_da_classe;
      if ( ! classe_lida ) Lê_classe;
      Class c = Class.forName( nome_classe );
      obj = c.newInstance(); // Quase isto.
      obj.readObject( this );
    }
    else obj = objecto_lido;
    return obj;
  }
  ...
}
```

# Serializar um objecto… Exemplo

Uma única chamada a `writeObject()` escreve um grafo de objectos ligados por referências entre si.

Exemplo:

- Object #1, type = Employee[]
  - [0] Object #2, type = Employee
    - Field: name, Value: Flinstone
    - Field: salary, Value: …
    - Field: buddy, Value: Object #3, type = Employee:
      - Field: name, Value: Burne
      - Field: salary, Value: …
      - Field: buddy, Value: Object #2 (already described)
  - [0] Object #3 (already described)

# Versão da classe serializável

Qualquer classe que implementa a interface `Serializable`, tem um número de versão.

```
static final long serialVersionUID = 3045685432790453O9L;
```

- Só é possível reconstruir um objecto serializado se a versão da classe lida do *stream* for igual à da classe carregada.
- Quando o atributo `serialVersionUID` não for definido, é automaticamente gerado um pelo compilador, que muda por cada alteração da classe.
- O utilitário **`serialver`** gera um número de versão a partir da declaração de uma classe serializável.

# Exemplo de uma classe serializável

```
enum DiaDaSemana{DOMINGO, SEGUNDA, TERCA,QUARTA, QUINTA, SEXTA, SABADO}
class Data implements Serializable{
   private static final long serialVersionUID = 1L;
   public byte dia, mes;
   public short ano;
   public transient DiaDaSemana diaSemana;
   public Data( int d, int m, int a ) {
      dia=(byte)d; mes=(byte)m; ano=(short)a;
      diaSemana = getDiaSemana();
   }
   public String toString() {
      return ""+dia+'/'+mes+'/'+ano+','+diaSemana;
   }
   private void readObject( ObjectInputStream in )
     throws ClassNotFoundException, IOException{
       in.defaultReadObject();
       diaSemana = getDiaSemana();
   }
   private DiaDaSemana getDiaSemana(){
      Calendar cal = Calendar.getInstance();
      cal.set(Calendar.DAY_OF_MONTH, dia);
      cal.set(Calendar.MONTH, mes - 1); // MONTH between [0,11]
      cal.set(Calendar.YEAR, ano);
      int w = cal.get(Calendar.DAY_OF_WEEK);
      return DiaDaSemana.values()[w - 1];// DAY_OF_WEEK [1, 7]
   }
}
```