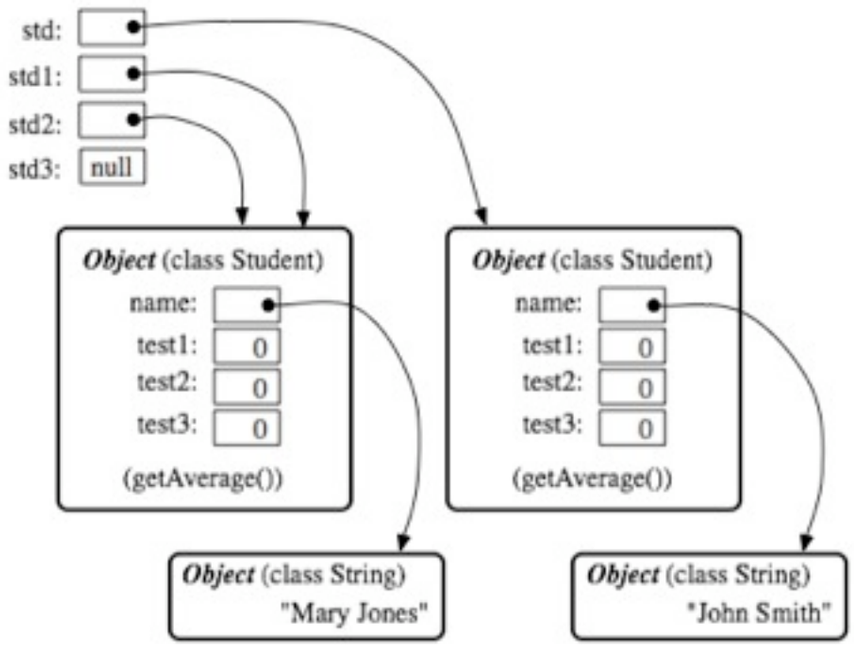
Serializzazione

A cosa serve?

Ad ottenere una rappresentazione di una struttura dati che possiamo memorizzare, trasmettere via rete



Cosa possiamo serializzare?

OK	NO
Tipi primitivi, stringhe	Riferimenti
Istanze di serializzabili	Istanze di classi che usano risorse di sistema (File, Socket, Thread, ...)

Riferimenti

La serializzazione genera una rappresentazione dei dati copiando il contenuto degli oggetti, non i loro riferimenti!

Con la deserializzazione otteniamo dei nuovi oggetti, diversi da quelli originali ma con lo stesso valore

Classi serializzabili

Implementano `Serializable`

```
public class Person implements Serializable {
    private static final long serialVersionUID = 911897921968418815 5L;
    private final String name;
    private final int height;
    private final Date dateOfBirth;
}
```

Contengono tipi di dati primitivi o stringhe

Contengono oggetti serializzabili

`serialVersionUID`

È un identificativo della versione della classe serializzata.

Serve per individuare problemi tra diverse versioni della stessa classe che potrebbero avere attributi diversi.

Serializzazione

Dobbiamo usare il metodo `writeObject` della classe `ObjectOutputStream`

Questo metodo genera una rappresentazione dell'oggetto passato come parametro e lo scrive su uno stream (Ad esempio un file o un socket).

Esempio

```
public static void main(String args[]) throws IOException {
   Person pippo = new Person("Pippo", new Date(1, 2, 1990), 180);

   ByteArrayOutputStream out = new ByteArrayOutputStream();

   ObjectOutputStream s = new ObjectOutputStream(out);
   s.writeObject(pippo);
}
```

ByteArrayOutputStream memorizza i dati in un array di byte, ma avremmo potuto usare uno stream qualsiasi. Dopo la `writeObject` quell'array conterrà una rappresentazione di `pippo`.

Deserializzazione

Dobbiamo usare il metodo duale `readObject` della classe `ObjectInputStream`

Questo metodo legge i dati dallo stream e restituisce il primo oggetto disponibile.

Esempio

```
public static void main(String args[]) throws IOException, ClassNotFoundException {
    ByteArrayInputStream in = new FileInputStream(“pippo.dat”);
    ObjectInputStream inputStream = new ObjectInputStream(in);
    Person pippo = (Person) inputStream.readObject();
}
```

Attenzione alle eccezioni!
Oltre agli errori dovuti alla lettura dallo stream potremmo avere una `ClassNotFoundException` se la classe dell’oggetto che vogliamo deserializzare non è nel classpath.

Attributi non serializzabili

A volte può capitare che una classe abbia degli attributi che non possiamo (o che non vogliamo) serializzare.

Marcandoli come `transient` saranno ignorati dalla serializzazione.

La deserializzazione (di default) non inizializza gli attributi `transient`. Dobbiamo quindi lasciare che lo faccia chi userà l'oggetto oppure fornire un metodo di deserializzazione personalizzato

Esempio

```
public class Person implements Serializable {
   private static final long serialVersionUID = 9118979219684188155L;
   private final String name;
   private final Date dateOfBirth;
   private final int height;

   private final Image image;
   private transient Image thumbnailImage;
```

Aggiungiamo a persona una immagine personale e una miniatura di questa immagine.

Per evitare di sprecare spazio vogliamo serializzare solamente l'immagine originale.

Deserializzazione

La deserializzazione di default non inizializza i campi transient che perciò assumono il valore `null`.

Possiamo però personalizzare il metodo di deserializzazione facendo override del metodo `readObject`.

Esempio

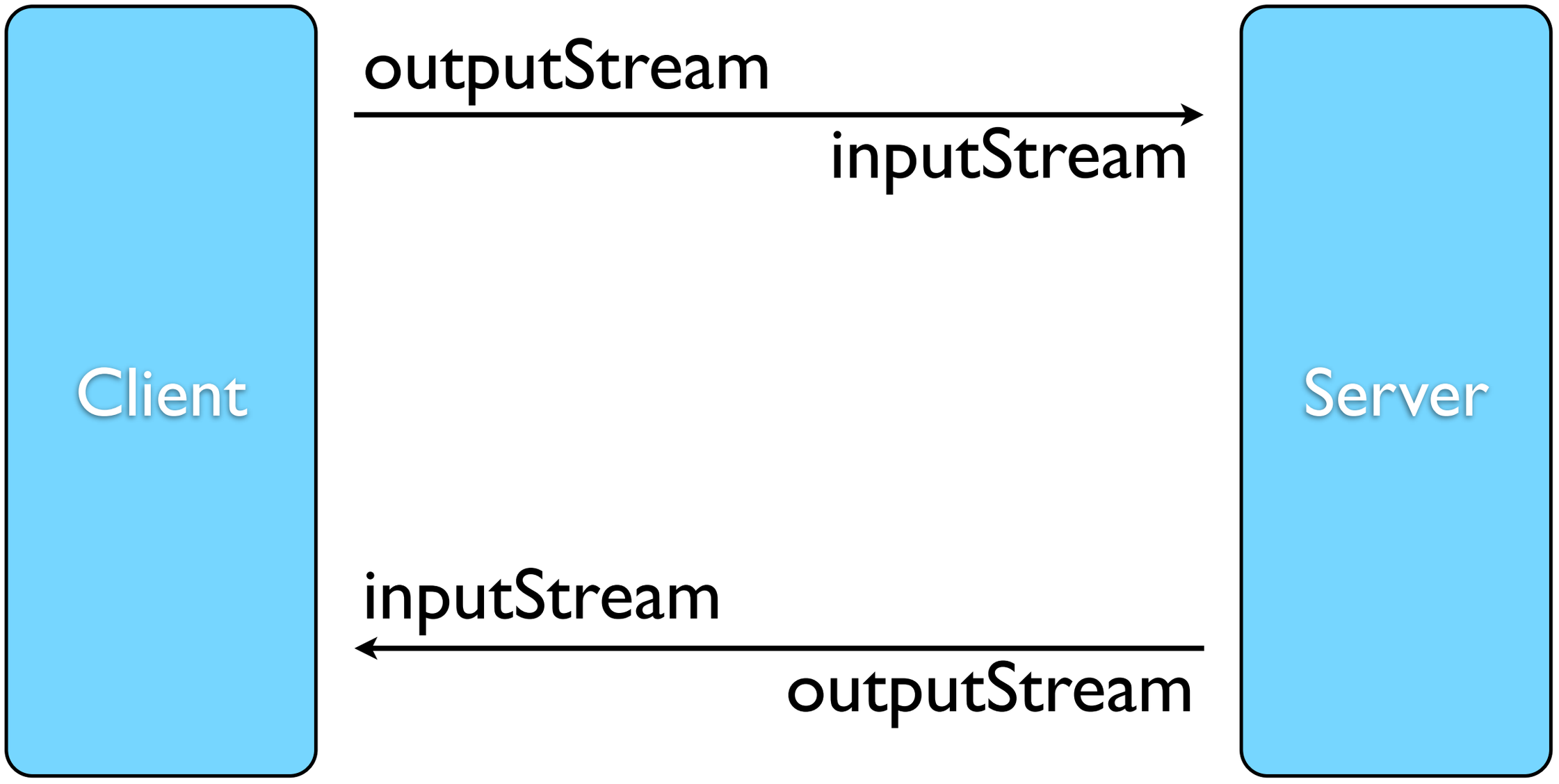
```
public class Person implements Serializable {
   private static final long serialVersionUID = 911897921968418815 5L;
   private final String name;
   private final Date dateOfBirth;
   private final int height;
   private final Image image;
   private transient Image thumbnailImage;
```

```
   private void readObject(ObjectInputStream inputStream) throws IOException,
ClassNotFoundException {
      inputStream.defaultReadObject();
      generateThumbnail();
   }
```

Prima lasciamo che il metodo di serializzazione di default inizializzi l'oggetto, poi possiamo invocare i metodi appropriati per inizializzare gli altri attributi.

Serializzazione e Socket

Comunicazione via socket



Input e output stream

Di default possono trasmettere solamente sequenze di byte, però possiamo wrapparli in `ObjectInputStream` e `ObjectOutputStream` per trasmettere oggetti serializzabili.

In questo modo possiamo gestire il protocollo di comunicazione più semplicemente e ad un livello di astrazione più elevato

Protocollo di comunicazione

Anche usando degli oggetti la gestione del protocollo di comunicazione è completamente compito dell'utente.

Gli stream si comportano come delle code, quindi gli oggetti possono essere letti solo nell'ordine in cui sono stati scritti.

È meglio controllare sempre che l'oggetto letto sia conforme con quanto previsto dal protocollo per evitare eccezioni quando effettuiamo il cast.