UNIVERSITÀ DI PISA

Programmazione di reti
Corso B

11 Ottobre 2016
Lezione 4

Recupero lezione

- **Confermato**
  - Mercoledì **19 ottobre**, **14:00 -16:00**, aula **L1**

Contenuti

- Concetti utili Java
    - Java IO
        - Stream, Reader, Writer
- Serializzazione Java
- JSON

Java IO

- Comunicazione (inviare e ricevere dati) tra il programma e il resto del mondo (un altro programma - anche a distanza, un utente, il disco, lo schermo)

- *Stream* - flussi di dati - concetto di base per Java IO
  - unidirezionali : *input / output*, ordinati (FIFO)
  - bloccanti - il *thread* si ferma fin che l'operazione finisce
  - il fonte dei dati è astratizzato dall'interfaccia - implementato da classi diverse
    - *file*
    - *bytearray*
    - rete

OutputStream

```
public abstract void write(int b) throws IOException
```
scrive un solo *byte* (meno significativo del int)

```
public void write(byte[] data) throws IOException
public void write(byte[] data, int offset, int
length)
throws IOException
```
scrivono più *byte*

```
public void flush() throws IOException
```
svuota i *buffer* - per evitare *deadlock* e prima di chiudere

```
public void close() throws IOException
```
chiude lo *stream* (mette *end-of-stream* nella coda)

InputStream

```
public abstract int read() throws IOException
```
Legge 1 *byte*, restituisce -1 se *stream* è finito

```
public int read(byte[] input) throws IOException
```
Prova di leggere abbastanza *byte* per riempire l'array. Restituisce il numero di *byte* letti. Può restituire 0, che vuol dire che non ha trovato nessun *byte* ready. Non si blocca.

```
public int read(byte[] input, int offset, int length) throws
IOException
```
Riempie l'array a partire da offset. Se length=0 *end-of-stream* restituisce 0.

```
int read = 0;
int toRead = 1024;
byte[] input = new byte[toRead];
while (read < toRead) {
   int readNow = in.read(input, read, toRead - read);
   if (readNow == -1) break; // end of stream
      read += readNow;
}
```

InputStream

```
public long skip(long n) throws IOException
```
Salta un numero di byte.

```
public int available() throws IOException
```
Il minimo numero di byte available

```
public void close() throws IOException
```
Chiude lo stream

Dati

- Scrivere nei *stream* : ogni tipo di dati deve essere trasformato in un `byte[]`

- Leggere dai *stream*: i `byte[]` devono essere interpretati (decodificati per ottenere il dato originale)

```
public class Streams {
   public static void main(String[] args) {
      int integer=2435;
      double real=2456754.6;
      char character='A';
      String name="Alina";
      int nameByteLength=name.getBytes().length;
      FileOutputStream out=null;
      FileInputStream in = null;
      try {
         out= new FileOutputStream("data.dat");
         out.write(int2ByteArray(integer));
         out.write(double2ByteArray(real));
         out.write(int2ByteArray(character));
         out.write(name.getBytes());
         System.out.println("I wrote "+integer+" "+real+" "+character+" "+name);
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }finally{
         try {if (out != null) out.close();
         } catch (IOException e) {}
      }
```

```
try {
    in= new FileInputStream("data.dat");

    byte[] bytes=new byte[Integer.BYTES];
    in.read(bytes);
    int readInteger=byteArray2Int(bytes);

    bytes=new byte[Long.BYTES];
    in.read(bytes);
    double readReal=byteArray2Double(bytes);

    bytes=new byte[Integer.BYTES];
    in.read(bytes);
        char readCharacter=(char)(byteArray2Int(bytes));

        bytes=new byte[nameByteLength];
        in.read(bytes);
        String readName=new String(bytes);

        System.out.println("I read "+readInteger+" "+readReal+" "+readCharacter+"
"+readName);
     } catch (FileNotFoundException e) {e.printStackTrace();
     } catch (IOException e) {e.printStackTrace();
     }finally{try {
           if (in != null)
              in.close();} catch (IOException e) {}
     }}}
```

```java
public static byte[] int2ByteArray(int x)
{
   byte[] result= new byte[Integer.BYTES];
   for (int i=0;i<Integer.BYTES;i++)
   {
      result[Integer.BYTES-i-1]=(byte) (x>>(i*8));
   }
   return result;
}
```

```java
public static byte[] double2ByteArray(double x)
{
   byte[] result= new byte[Long.BYTES];
   long bits=Double.doubleToLongBits(x);
   for (int i=0;i<Long.BYTES;i++)
   {
      result[Long.BYTES-i-1]=(byte) (bits>>(i*8));
   }
   return result;
}
```

```
public static int byteArray2Int(byte[] bytes){
   int result=0;
   for (byte b : bytes){
      result=(result<<8)+(b>=0?b:(b+256));
   }
   return result;
}

public static double byteArray2Double(byte[] bytes){
   long result=0;
   for (byte b : bytes){
      result=(result<<8)+(b>=0?b:(b+256));
   }
   return Double.longBitsToDouble(result);
}
```

data.dat

```
ÉABæYLÃÃÕAAlina
```

```
Output:
I wrote 2435 2456754.6 A Alina
I read 2435 2456754.6 A Alina
```

Java IO

- I *filter* - usati con stream per trasformare dati prima di scrivere/ dopo aver letto
  - criptare/decriptare
  - comprimere
  - ottimizzare usando *buffer*
- I *reader/writer* - specializzati sul lavoro su testo

Filter classes

```
DataOutputStream out = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(“data.dat”)));
```

Metodi per scrivere vari tipi di dati

```
public final void writeBoolean(boolean b) throws IOException
public final void writeByte(int b) throws IOException
public final void writeShort(int s) throws IOException
public final void writeChar(int c) throws IOException
public final void writeInt(int i) throws IOException
public final void writeLong(long l) throws IOException
public final void writeFloat(float f) throws IOException
public final void writeDouble(double d) throws IOException
public final void writeChars(String s) throws IOException
public final void writeBytes(String s) throws IOException
public final void writeUTF(String s) throws IOException
```

`DataInputStream` legge gli stessi tipi di dati.

Filter

```
BufferedOutputStream out = new
BufferedOutputStream(new
FileOutputStream(“data.txt”), int 1024);
```

Usa un *buffer* di 1kb. Migliore performance. La scrittura può essere forzata facendo `flush()`. Simile per `BufferedInputStream`.

```
CipherOutputStream out = new
CipherOutputStream(new BufferedOutputStream(new
FileOutputStream(“data.txt”)),Cipher.getInstance
(“AES/CBC/NoPadding"));
```

Codifica i dati usando AES, poi usa un buffer sul stream grezzo.

Reader/Writer

- Simili a *Input/OutputStream* però lavorano con caratteri **Unicode** (lunghezza dipende dalla codifica) invece di *byte*

- opzione migliore per lavorare con testo - indipendenza dalla piattaforma e supporto a vari tipi di *encoding*

- *encoding* - mappa un carattere a un numero (un codice)

- il codice di un carattere può essere diverso in *encoding* diversi

Encoding

Carattere	ASCII	UTF-8	UTF-16	ISO-8859-1
A	41	41	41	41
B	42	42	42	42
Y	59	59	59	59
Z	5A	5A	5A	5A
é	82	C3A9	E9	E9
è	8A	C3A8	E8	E8
à	85	C3A0	E0	E0
ò	95	C3B2	F2	F2

Writer

```
void write(char[] text, int offset, int length) throws
IOException
void write(int c) throws IOException
void write(char[] text) throws IOException
void write(char[] text, int offset, int length) throws
IOException
void write(String s) throws IOException
void write(String s, int offset, int length) throws
IOException
void flush() throws IOException
void close() throws IOException
```

Interfaccia quasi uguale al `OutputStream`, però avendo il `char` alla base.

Reader

```
abstract int read(char[] text, int offset, int
length)
throws IOException
int read() throws IOException
int read(char[] text) throws IOException
long skip(long n) throws IOException
void close() throws IOException
```

metodi simili agli `InputStream`

```
boolean ready()
```

controlla se c’è qualcosa da leggere, non quanti caratteri ci sono

Reader/Writer

Da usare senza un *stream* grezzo:

`FileReader/Writer`
scrivere/leggere testo da un *file*

`StringReader/Writer`
scrivere/leggere testo da una stringa

`CharArrayReader/Writer`
scrivere/leggere testo da un *array* di `char`

`BufferedReader/Writer`
funzionalità di buffer sopra *reader/writer* grezzi

OutputStreamWriter / InputStreamReader

Da usare con un stream grezzo sotto. Traduce byte in caratteri in base alla encoding usata.

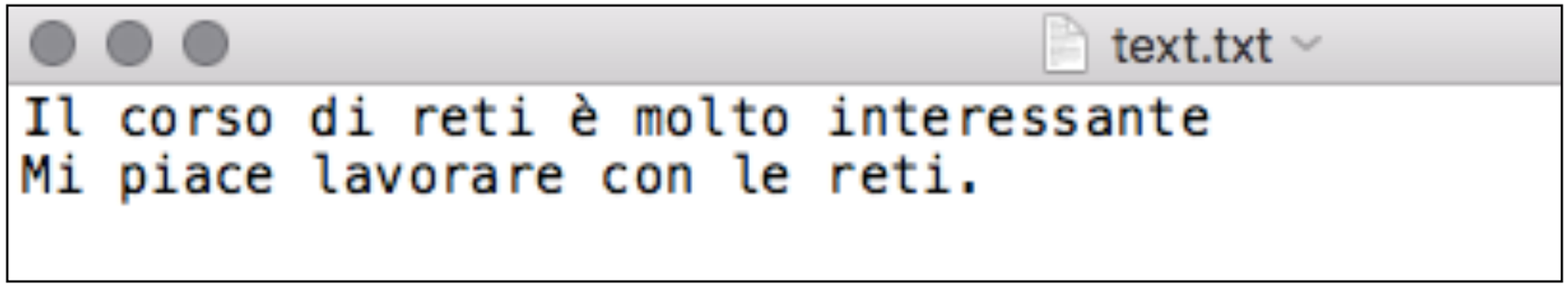
```
public OutputStreamWriter(OutputStream out)
public OutputStreamWriter(OutputStream out, String
encoding)
throws UnsupportedEncodingException

public InputStreamReader(InputStream in)
public InputStreamReader(InputStream in, String encoding)
throws UnsupportedEncodingException

public String getEncoding()
```

```java
public class WriterTest {

   public static void main(String[] args) {
      BufferedWriter w=null;
      try {
         w= new BufferedWriter(new OutputStreamWriter(
            new FileOutputStream("text.txt")));
         w.write("Il corso di reti è molto interessante\r\n”);
         w.write("Mi piace lavorare con le reti.");
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }finally{
         try {w.close();} catch (IOException e) {}
      }
   }
}
```

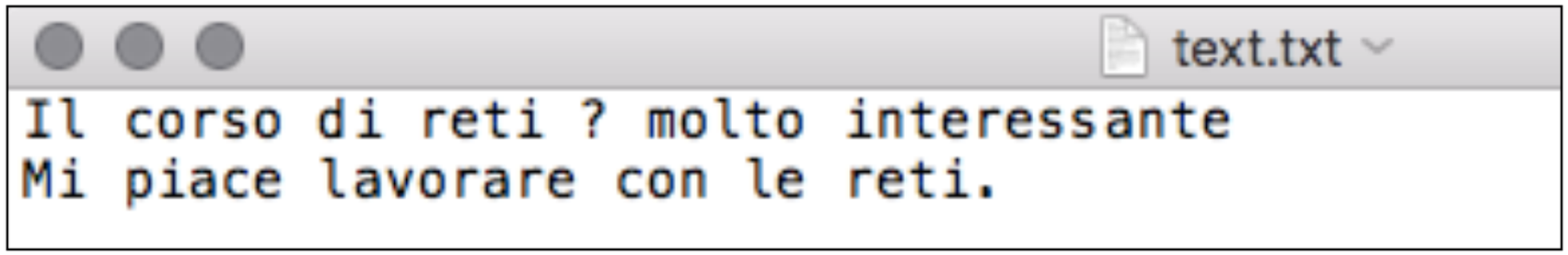


```java
public class ReaderTest {

   public static void main(String[] args) {
      BufferedReader r=null;
      try {
         r= new BufferedReader(new InputStreamReader
            (new FileInputStream("text.txt")));
         String line= r.readLine();
         while (line!=null){
            System.out.println("Read: "+line);
            line=r.readLine();
         }
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }finally{
         try {r.close();} catch (IOException e) {}
      }
   }
}
```

```
Read: Il corso di reti è molto interessante
Read: Mi piace lavorare con le reti.
```

```
w= new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream("text.txt"),"US-ASCII"));
```



```java
w= new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream("text.txt"),"UTF-8"));
```

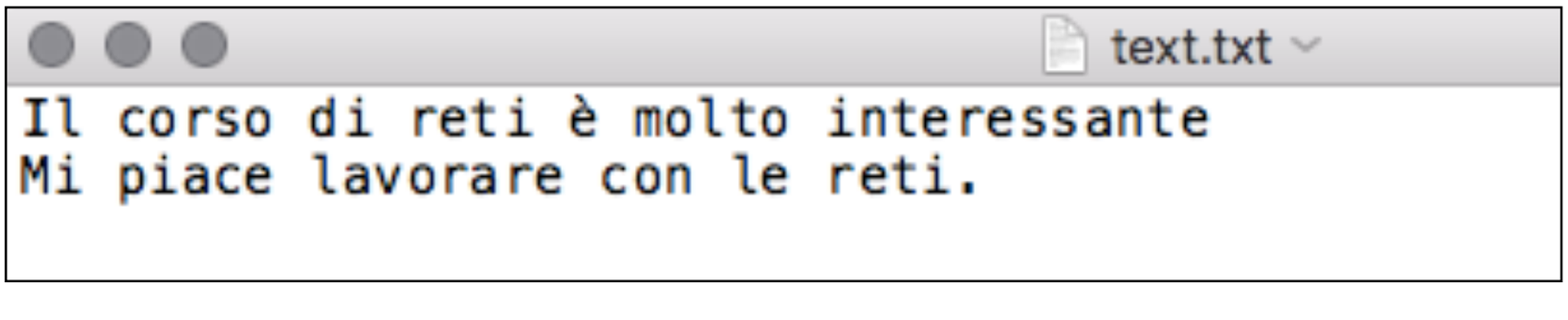
text.txt

```
Il corso di reti è molto interessante
Mi piace lavorare con le reti.
```

```java
r= new BufferedReader(new InputStreamReader
    (new FileInputStream("text.txt"),"UTF-16"));
```

```
Read: 鏄湇獯槧杤瑩?ㄎ浯汴漠楮瑥杤獳慮瑥ഉന䵩ø楬挠慤湵浡摬渠扶?整椮
```

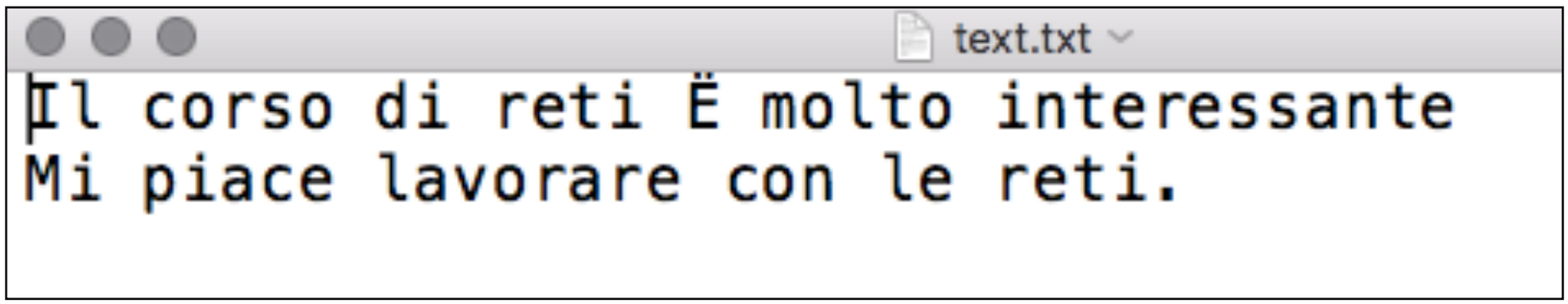
```
w= new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream("text.txt"),"UTF-16"));
```




```
r= new BufferedReader(new InputStreamReader
    (new FileInputStream("text.txt"),"UTF-8"));
```

```
Read: ??Il corso di reti ? molto interessante
Read:
Read: Mi piace lavorare con le reti.
```

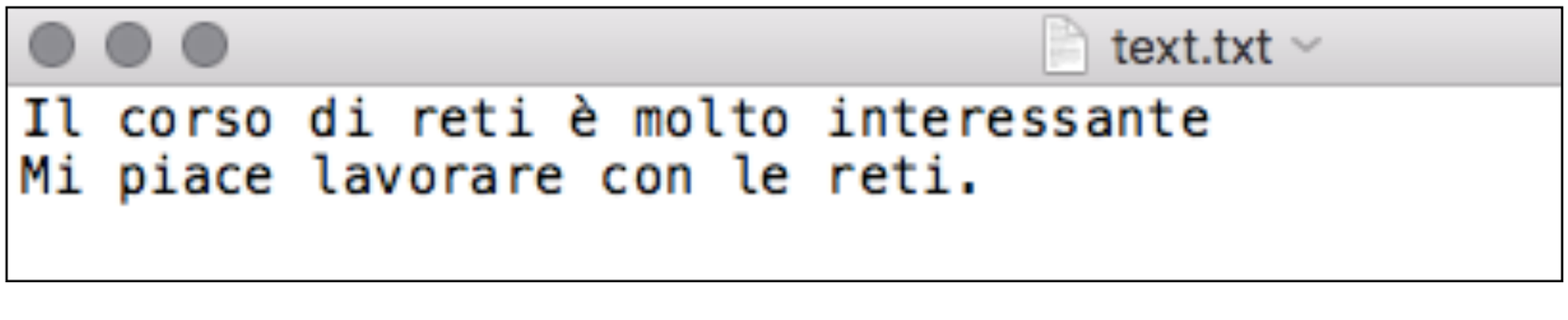
```
w= new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream(“text.txt"),"windows-1252"));
```




```
r= new BufferedReader(new InputStreamReader
   (new FileInputStream("text.txt"),"UTF-8"));
```

```
Read: Il corso di reti ? molto interessante
Read: Mi piace lavorare con le reti.
```

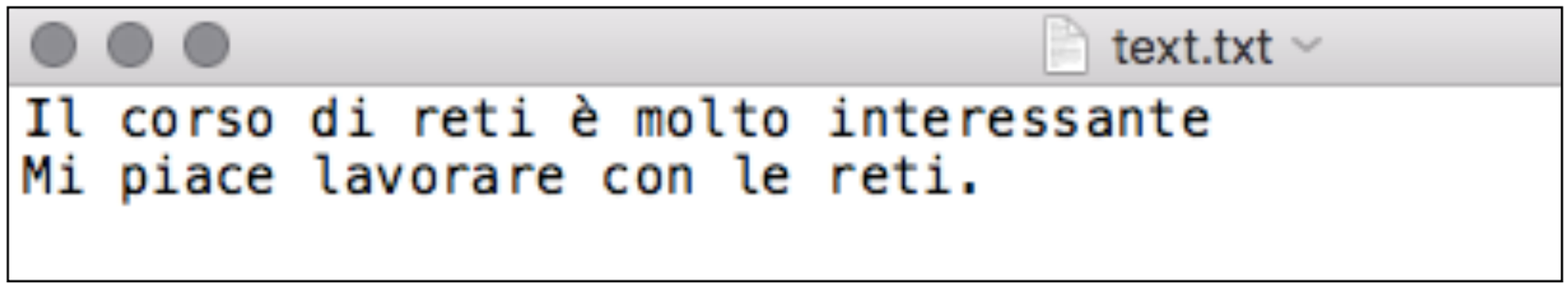
```
w= new BufferedWriter(new OutputStreamWriter(
        new FileOutputStream("text.txt"),"UTF-8"));
```




```
r= new BufferedReader(new InputStreamReader
   (new FileInputStream(“text.txt"),"windows-1252"));
```

```
Read: Il corso di reti Ã¨ molto interessante
Read: Mi piace lavorare con le reti.
```

```
w= new BufferedWriter(new OutputStreamWriter(
       new FileOutputStream("text.txt"),"UTF-8"));
```




```
r= new BufferedReader(new InputStreamReader
   (new FileInputStream(“text.txt"),"Big5"));
```

```
Read: Il corso di reti 癡 molto interessante
Read: Mi piace lavorare con le reti.
```

try con risorse

```
try {
        r= new BufferedReader(new InputStreamReader(new
  FileInputStream("text.txt")));
        String line= r.readLine();
        while (line!=null){
            System.out.println("Read: "+line);
            line=r.readLine();
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }finally{
        try {r.close();} catch (IOException e) {}
    }
```

Funziona per tutte le risorse che implementano interfaccia AutoCloseable

```
try (BufferedReader r= new BufferedReader(new
InputStreamReader(new FileInputStream("text.txt")))) {
    ……….
    } catch (…) {
            ……….
    } //no finally required, file closed at the end of the block
```

Serializzazione

- Trasformare un oggetto in una rappresentazione binaria (sequenza di *byte*) che può essere scritta in un stream (e.g. per memorizzarla in un *file*).
- L'oggetto può essere ricreato dalla sequenza di byte: deserializzazione.
- Il processo è indipendente dalla piattaforma però dipendente dal linguaggio di programmazione: deve essere Java per serializzazione e deserializzazione
- 2 Passi:
  - Creare una classe serializzabile
  - Usare `ObjectInputStream/ObjectOutputStream` per lavorare con gli oggetti.

`Serializable`

- Un oggetto è serializzabile se la classe implementa `Serializable`
- Tutti i tipi di dati primitivi sono serializzabili.
- Per gli oggetti non-primitivi, se implementano `Serializable`, sono serializzabili (controllare documentazione)
- Per essere serializzabile, un tipo di dati custom deve contenere solo attributi serializzabili o `transient/static`
- Gli attributi `transient/static` non vengono serializzati
- Serializzazione implicita: gli attributi vengono serializzati in automatico
- Serializzazione *custom*: la classe sovrascrive la serializzazione implicita

Serializable

- La sequenza di byte ottenuta contiene informazioni che identificano il tipo di oggetto (la gerarchia di classi) e il valore degli attributi

- Se l'oggetto contiene un riferimento ad un secondo oggetto, anche esso viene serializzato (un albero di oggetti)

- Al momento della deserializzazione, il JVM deve trovare la definizione della classe (il file .class), altrimenti viene lanciata una `ClassNotFoundException`

- Questo diventa più problematico quando si lavora in rete.

- I metodi dell'oggetto non vengono serializzati, sono estratti dalla definizione della classe (*file* `.class`)

Serial Version

- Per garantire che i processi di serializzazione e deserializzazione usano versioni compatibili di una classe, Java definisce un UID (*unique ID*) per la definizione di una classe

- è indicato includere un `serialVersionUID` nella definizione della classe. Altrimenti la JVM genera uno per noi (in base alla definizione della classe), però questo può causare problemi se si lavora su piattaforme diverse.

- pe dichiarare la versione:

```
private static final long serialVersionUID = 15L;
```

Serial Version

- Se la classe evolve però è sempre *backwards-compatible*, allora si deve usare lo stesso `serialVersionUID` anche nelle versioni nuove
  - es: aggiungere attributi, aggiungere/rimuovere classi interne, trasformare attributi *transient* in *non-transient*
- Se la nuova versione non e compatibile con quella precedente, il `serialVersionUID` deve essere cambiato
  - es: rimuovere attributi, trasformare attributi *non-transient* in *transient*

```java
public class Student implements Serializable{

    private static final long serialVersionUID = 1L;

    private String fname, lname;
    private String streetAddress;
    private int addressNo;

    public Student(String fname, String lname,String street, int no){
        this.fname=fname;
        this.lname=lname;
        this.streetAddress= street;
        this.addressNo=no;
    }
    public String toString(){
        StringBuilder sb= new StringBuilder();
        sb.append(this.fname);
        sb.append(" ");
        sb.append(this.lname);
        sb.append(" living at ");
        sb.append(this.addressNo);
        sb.append(" ");
        sb.append(this.streetAddress);
        sb.append(" street.");
        return sb.toString();
    }
}
```

Tutti gli attributi sono serializzabili quindi la classe è serializzabile

ObjectOutputStream

- Un *filter*, da usare con *stream raw*, dedicato a serializzare tipi di dati primitivi e oggetti serializzabili

```
void writeBoolean(boolean val)

void writeByte(int val)

void writeBytes(String str)

void writeChar(int val)

void writeChars(String str)

void writeDouble(double val)
```

ObjectOutputStream

```
void writeFloat(float val).

void writeInt(int val)

void writeLong(long val)

void writeShort(int val)

void writeUTF(String str)
```

ObjectOutputStream

```
void  writeObject(Object o)
```

Scrive l'informazione sulla classe e i valori degli attributi da serializzare (non *transient* e non statici). Se un oggetto viene scritto due volte, la prima volta un riferimento è generato e la seconda volta solo il riferimento viene scritto.

Attenzione: se l'oggetto viene modificato dopo essere stato scritto la prima volta, la modifica non è salvata la seconda volta.

```
Student s=  new Student(“Robert”,”Brown","Dawson",12);
out.writeObject(s);
s.setFname(“Robbie”);
out.writeObject(s);
```

ObjectOutputStream

```
void writeUnshared(Object o)
```

Uguale a **`writeObject`** però oggetti ripetuti sono scritti come nuovi senza fare riferimento agli precedenti.

ObjectInputStream

- *Filter* specializzato in deserializzazione
- Metodi lanciano EOFException alla fine del *stream*

```
boolean readBoolean()

byte readByte()

char readChar()

double  readDouble()

float readFloat()
```

ObjectInputStream

```
int readInt()

long readLong()

short readShort()

int readUnsignedByte()

int readUnsignedShort()

String readUTF()
```

ObjectInputStream

`Object readObject()`

Legge l'informazione sulla classe e i valori degli attributi da deserializzare (non *transient* e non statici). Se si incontra un riferimento ad un oggetto già letto, si restituisce quel oggetto direttamente.

ObjectInputStream

`Object readUnshared()`

Da usare per leggere oggetti scritti con `writeUnshared`

```
public class WriteStudentMain {

   public static void main(String[] args) {
      ArrayList<Student> students= new ArrayList<Student>();
      students.add(new Student("Robert","Brown","Dawson",12));
      students.add(new Student("Michael","Reds","Pearse",40));
      students.add(new Student("Joanna","Moore","Collins",62));
      students.add(new Student("Ann","Brown","Buffallo",132));

      try (ObjectOutputStream out= new ObjectOutputStream(
            new FileOutputStream("students.ser"));) {
         out.writeObject(students);
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

students.ser — Edited

```
¨Ìsrjava.util.ArrayListxÅ“ô«aùIsizexpwsrStudentI
addressNoLfnametLjava/lang/String;Llnameq~L
streetAddressq~xp
tRoberttBrowntDawsonsq~(tMichaeltRedstPearsesq~>tJoa
nnatMooretCollinssq~ÑtAnnq~tBuffallox
```

291 byte

```
public class ReadStudentMain {

   public static void main(String[] args) {
      try(ObjectInputStream in= new ObjectInputStream(
            new FileInputStream("students.ser"));){
         ArrayList<Student> stds= (ArrayList<Student>) in.readObject();
         for (Student s : stds){
            System.out.println(s);
         }
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      }
   }
}
```

```
Robert Brown living at 12 Dawson street.
Michael Reds living at 40 Pearse street.
Joanna Moore living at 62 Collins street.
Ann Brown living at 132 Buffallo street.
```

Se cambio `serialVersionUID` della classe `Student` (dopo aver generato il file), il reader mi lancia `IOException`.

```
java.io.InvalidClassException: Student; local class incompatible: stream classdesc
serialVersionUID = 1, local class serialVersionUID = 2
    at java.io.ObjectStreamClass.initNonProxy(ObjectStreamClass.java:616)
    at java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:1623)
    at java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1518)
    at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1774)
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1351)
    at java.io.ObjectInputStream.readUnshared(ObjectInputStream.java:461)
    at ReadStudentMain.main(ReadStudentMain.java:19)
```

```
public class WriteStudentMain {

   public static void main(String[] args) {
      ArrayList<Student> students= new ArrayList<Student>();
      students.add(new Student("Robert","Brown","Dawson",12));
      students.add(new Student("Michael","Reds","Pearse",40));
      students.add(new Student("Joanna","Moore","Collins",62));
      students.add(new Student("Ann","Brown","Buffallo",132));

      try (ObjectOutputStream out= new ObjectOutputStream(
            new FileOutputStream("students.ser"));) {
         out.writeObject(students);
         for (Student s : students){
            out.writeObject(s);
         }
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Riscriviamo gli oggetti Student uno per uno

students.ser — Edited

```
¨Ìsrjava.util.ArrayListxÅ“ô«aùIsizexpwsrStudentI
addressNoLfnametLjava/lang/String;Llnameq~L
streetAddressq~xp
tRoberttBrowntDawsonsq~(tMichaeltRedstPearsesq~>tJoa
nnatMooretCollinssq~ÑtAnnq~tBuffalloxq~q~q~
q~
```

312 byte

```
public class ReadStudentMain {

   public static void main(String[] args) {
      try(ObjectInputStream in= new ObjectInputStream(
            new FileInputStream("students.ser"));){
         ArrayList<Student> stds= (ArrayList<Student>) in.readObject();
         for (Student s : stds){
            System.out.println(s);
         }
         Student s;
         while (true){
            try{
            s = (Student) in.readObject();
            System.out.println(s);
            } catch (EOFException e){
               System.out.println("Stream ended");
               break;
            }        }
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      }
   }
}
```

```
Robert Brown living at 12 Dawson street.
Michael Reds living at 40 Pearse street.
Joanna Moore living at 62 Collins street.
Ann Brown living at 132 Buffallo street.
Robert Brown living at 12 Dawson street.
Michael Reds living at 40 Pearse street.
Joanna Moore living at 62 Collins street.
Ann Brown living at 132 Buffallo street.
Stream ended
```

```
for (Student s : students){
                out.writeUnshared(s);
            }
```

students.ser — Edited

```
¨Ìsrjava.util.ArrayListxÅ“ô«aùIsizexpwsrStudentI
addressNoLfnametLjava/lang/String;Llnameq~L
streetAddressq~xp
tRoberttBrowntDawsonsq~(tMichaeltRedstPearsesq~>tJoa
nnatMooretCollinssq~ÑtAnnq~tBuffalloxsq~
q~q~q~sq~(q~  q~
q~
sq~>q~
q~q~sq~Ñq~q~q~
```

392 byte

```
s = (Student) in.readUnshared();
```

Mescolando *read/write object/unshared* otteniamo errori se ci sono riferimenti agli oggetti. Se tutti gli oggetti sono comunque unici, non otteniamo errori. Meglio usare **`readUnshared`** con **`writeUnshared`** e **`readObject`** con **`writeObject`**

Serializable

- Serializzazione *custom* - implementare metodi per scrivere/leggere dal `ObjectOutput/InputStream`

```
private void writeObject(ObjectOutputStream os)

private void readObject(ObjectInputStream is)
```

```java
public class Address {
   int no;
   String street;

   public Address(int n, String street){
      this.no=n;
      this.street=street;
   }

   public String toString()
   {
      StringBuilder sb = new StringBuilder();
      sb.append(this.no);
      sb.append(" ");
      sb.append(this.street);
      sb.append(" street.");
      return sb.toString();
   }
}
```

Classe non serializzabile (e.g. non può essere cambiata perché ereditata da versione precedente del programma)

```java
public class Student implements Serializable{

   private static final long serialVersionUID = 1L;

   private String fname, lname;
   private transient Address address;

   public Student(String fname, String lname,String street, int no){
      this.fname=fname;
      this.lname=lname;
      this.address= new Address(no, street);
   }

   public String toString(){
      StringBuilder sb= new StringBuilder();
      sb.append(this.fname);
      sb.append(" ");
      sb.append(this.lname);
      sb.append(" living at ");
      sb.append(this.address);
      return sb.toString();
   }
```

Deve usare **Address** qui, marcandolo con **transient** perché non serializzabile

```
private void writeObject(ObjectOutputStream out){
   try {
      out.defaultWriteObject();
      out.write(this.address.getNo());
      out.writeUTF(this.address.getStreet());
   } catch (IOException e) {
      System.out.println("Could not write object");
   }
}

private void readObject(ObjectInputStream in){
      try {
         in.defaultReadObject();
         int no =in.readInt();
         String street = in.readUTF();
         this.address= new Address(no, street);
      } catch (IOException e) {
         System.out.println("Could not write object");
      } catch (ClassNotFoundException e) {
         System.out.println("Class not found or wrong version");
      }
}
```

Scrivo gli attributi serializzabili in automatico

Scrivo l'attributo non serializzabile a mano

JSON

- Formato di dati *lightweight* e indipendente dalla piattaforma
- Basato su testo
- Usato per interscambiare dati - alternativa a serializzazione - non dipende dal linguaggio di programmazione
- Leggibile e parsabile facilmente
- 2 strutture di base:
  - coppie **chiave:valore**
  - liste ordinate di valori
- Struttura ad albero

Esempio

{“students":[

{"fname":"Robert", “lname”:"Brown", “street”:”Dawson”, “number”:12},

{"fname":"Michael", “lname”:"Reds", “street”:”Pearse”, “number”:40},

{"fname":"Joanna", “lname”:"Moore", “street”:”Collins”, “number”:62},

{"fname":"Ann", “lname”:"Brown", “street”:”Buffalo”, “number”:132},

]}

Sintassi

- Oggetto - attributi rappresentati da coppie **chiave:valore** separate da “,”, delimitate da “{}”
- Array - lista di valori delimitata da “[]”, separati da “ ”
  ,”
- I valori possono essere: stringhe, numeri, boolean, oggetti, array o null
- Le chiavi sono stringhe

JSON in Java

- *package* “JSON simple “
- scaricare *jar* è collegare al progetto come *referenced library (external)*
- offre classi specializzate in creare oggetti JSON dalla loro rappresentazione testuale (deserializzazione) e trasformare oggetti nel formatto JSON di testo (serializzazione)

Tipi di dati

- Stringa JSON -> `String`
- Numero JSON -> `Double, Long`
- Boolean JSON -> `Boolean`
- Array JSON - > `ArrayList`
- Oggetto JSON - > `HashMap`

JSONObject

- Rappresenta le coppie **chiave:valore** come elementi di una map

- Implementa interfaccia `Map`: metodi `put(chiave, valore)`, `get(chiave)`, etc.

- Metodo per trasformare la *map* in una stringa in formato JSON

```
String toJSONString()
```

- Metodo per scrivere il testo JSON in un *writer*

```
void writeJSONString(java.io.Writer out)
```

JSONArray

- Estende `ArrayList` - metodi `set, get, remove, add` etc
- Metodi per trasformare in testo JSON

```
java.lang.String toJSONString()

void writeJSONString(java.io.Writer out)
```

JSONParser

- Trasforma testo JSON in oggetto o array JSON

```
java.lang.Object parse(java.io.Reader in)

java.lang.Object parse(java.lang.String s)
```

```
import org.json.simple.JSONObject;

public class JsonStudent{

   private String fname, lname;
   private String streetAddress;
   private long addressNo;

   public JsonStudent(String fname, String lname,String street, long no){
      this.fname=fname;
      this.lname=lname;
      this.streetAddress= street;
      this.addressNo=no;
   }

   public String toString(){
      StringBuilder sb= new StringBuilder();
      sb.append(this.fname);
      sb.append(" ");
      sb.append(this.lname);
      sb.append(" living at ");
      sb.append(this.addressNo);
      sb.append(" ");
      sb.append(this.streetAddress);
      sb.append(" street.");
      return sb.toString();
   }
```

```
@SuppressWarnings("unchecked")
public JSONObject toJson(){
    JSONObject result= new JSONObject();
    result.put("fname", this.fname);
    result.put("lname", this.lname);
    JSONObject address= new JSONObject();
    address.put("no",this.addressNo);
    address.put("street", this.streetAddress);
    result.put("address", address);
    return result;
}
```

```
    public String getLname() {
        return this.lname;
    }

    public static JsonStudent fromJSON(JSONObject object) {
        return new JsonStudent(
                (String)object.get("fname"),
                (String)object.get("lname"),
                (String)((JSONObject)object.get("address")).get("street"),
                (long)((JSONObject)object.get("address")).get("no"));
    }

}
```

```java
public class JSONWriter {

   @SuppressWarnings("unchecked")
   public static void main(String[] args) {
      ArrayList<JsonStudent> students= new ArrayList<JsonStudent>();
      students.add(new JsonStudent("Robert","Brown","Dawson",12));
      students.add(new JsonStudent("Michael","Reds","Pearse",40));
      students.add(new JsonStudent("Joanna","Moore","Collins",62));
      students.add(new JsonStudent("Ann","Brown","Buffallo",132));

      JSONArray jsonStudents= new JSONArray();
      for (JsonStudent s : students){
         jsonStudents.add(s.toJson());
      }

      try(FileWriter registro= new FileWriter("registro.json");){
         jsonStudents.writeJSONString(registro);
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

registro.json


```
[{"fname":"Robert","lname":"Brown","address":{"no":12,"street":"Dawson"}},
{"fname":"Michael","lname":"Reds","address":{"no":40,"street":"Pearse"}},
{"fname":"Joanna","lname":"Moore","address":{"no":62,"street":"Collins"}},
{"fname":"Ann","lname":"Brown","address":{"no":132,"street":"Buffallo"}}]
```

```java
public class JSONReader {

   public static void main(String[] args) {
      ArrayList<JsonStudent> students= new ArrayList<JsonStudent>();
      JSONParser parser= new JSONParser();
      try(FileReader registro= new FileReader("registro.json");){
         JSONArray array= (JSONArray) parser.parse(registro);
         for (Object jo: array){
            students.add(JsonStudent.fromJSON((JSONObject)jo));
            System.out.println(students.get(students.size()-1));
         }
      } catch (FileNotFoundException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      } catch (ParseException e) {
         e.printStackTrace();
      }
   }
}
```

```
Robert Brown living at 12 Dawson street.
Michael Reds living at 40 Pearse street.
Joanna Moore living at 62 Collins street.
Ann Brown living at 132 Buffallo street.
```

Conclusioni serializzazione

- usando meccanismo Java `Serializable`
    - interoperabilità tra piattaforme - si
    - interoperabilità tra linguaggi - no
    - *heavy weight* - si memorizzano informazioni extra sui tipi di dati
- usando meccanismi universali
    - e.g. JSON
    - interoperabilità tra piattaforme - si
    - interoperabilità tra linguaggi - si
    - *light weight* - si memorizzano solo i dati

Esercizi

1. File search

- File molto lungo
- Ricerca di una parola
- N thread diversi
- primo thread che trova la parola ferma tutti gli altri

Esercizi

2. Producer/consumer - la mensa

- Estensione dell'assignment 4
- Lavoratori scrivono in un file alla fine della giornata il numero di piati processati (ogni avvio del programma è una nuova giornata)
- Un altro programma legge il numero di piati e calcola lo stipendio per tutti i giorni inclusi nel *file*