Università degli Studi di Bologna
Facoltà di Ingegneria

Fondamenti di Informatica T2
Modulo 2

Corso di Laurea in Ingegneria Informatica

Anno accademico 2012/2013

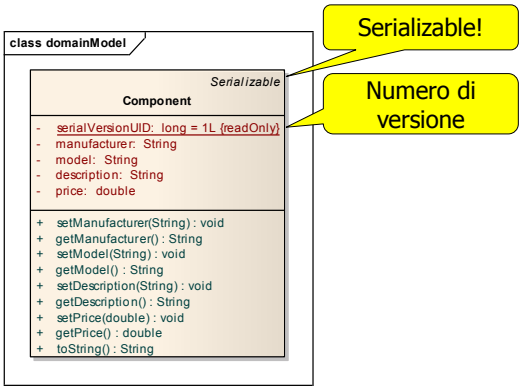
File, ancora file…

File in Java…

- I concetti di base sono analoghi a quelli del C
  - File/Stream binari/di testo
  - Dispositivi visti come Stream:
    - in C visti solo stdin/stdout
    - diversi tipi di Stream in Java (System.in, System.out, stream di rete, …)
- In Java, per operare con i file c'è qualche sovrastruttura in più… che risulta molto d'aiuto!
- Ma lavorare con i file rimane sempre **l'incubo dello studente**!

Serializzazione di entità "semplice"

Si supponga di avere un'entità **`Component`** che rappresenti un componente di un PC in termini di marca, modello, descrizione, costo



Riassunto... delle puntate precedenti

- L'interfaccia **`Serializable`** consente di marcare l'oggetto come serializzabile – può essere letto/scritto tramite **`ObjectInputStream`**/**`ObjectOutputStream`**
  - Senza quell'interfaccia gli stream di serializzazione si rifiutano di lavorare: non tutto deve poter essere serializzato!!
- La costante **`serialVersionUID`** consente al sistema di serializzazione di riconoscere se gli oggetti letti da file hanno la stessa versione degli oggetti istanziabili
  - Consente di evitare che un cambiamento di "schema" dell'oggetto provochi una catastrofe all'atto della deserializzazione
  - Cosa accadrebbe se venisse aggiunto/tolto un field e si tentasse di caricare un file contenente oggetti con schema diverso?

Serial Version UID

Una prova

- si serializzi un insieme di componenti con
  `serialVersionUID = 1L;`
- si modifichi
  `serialVersionUID = 2L;`
- Si deserializzino i componenti precedentemente scritti

Risultato

- Un'eccezione di tipo **`InvalidClassException`**…
- …con messaggio: "**`domainModel.Component; local class incompatible: stream classdesc serialVersionUID = 1, local class serialVersionUID = 2`**"

Serial Version UID

IMPORTANTE

Ogni volta che si cambia lo schema/struttura di una classe serializzabile occorre ricordare di modificare anche la costante **`serialVersionUID`**

Funzionalità da Ottenere

- Lettura/Scrittura su stream di testo
- Lettura/Scrittura su stream binario

- In C:

```
Boolean readComponentFromTxt(FILE *f, Component *c);
void writeComponentToTxt(FILE *f, Component c);
Boolean readComponentFromBin(FILE *f, Component *c);
void writeComponentToBin(FILE *f, Component c);
```

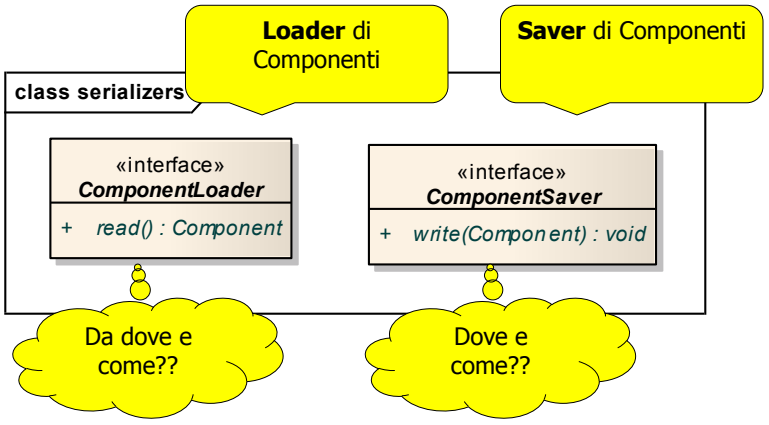
- In Java?

Struttura Esterna

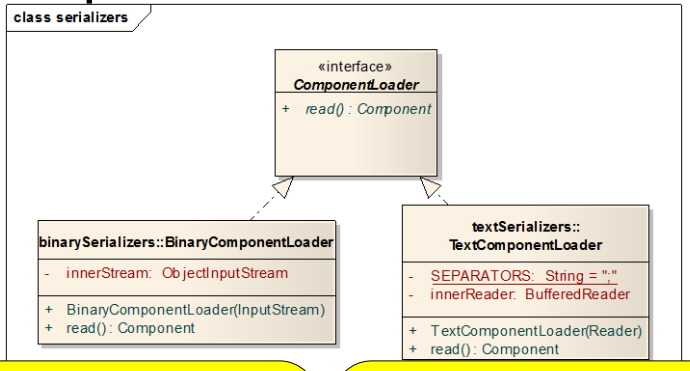
- Soliti desideri: architettura flessibile e riusabile

- Quindi:
  - **Astrazioni** per modellare gli oggetti che consentono di **caricare** e **salvare**
  - **Diverse implementazioni** per lavorare su *stream* di testo e *stream* binari
  - Chi usa le astrazioni sa cosa salva/carica (`Component`) ma **non sa né dove/da dove, né il modo o il formato in cui i dati siano salvati**

Astrazioni



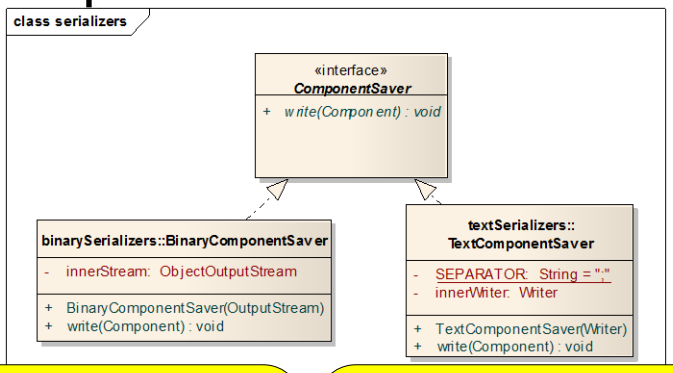
Implementazioni – Loader



- **Incapsula** un `InputStream` (soluzione generale)
- Internamente lavora con un `ObjectInputStream` (completamente incapsulato)
- **Il formato dei dati letti è completamente nascosto**

- **Incapsula** un `Reader` (soluzione generale)
- Internamente lavora con un `BufferedReader` (completamente incapsulato)
- **Il formato dei dati letti è completamente nascosto**

Implementazioni – Saver



- **Incapsula** un `OutputStream` (soluzione generale)
- Internamente lavora con un `ObjectOutputStream` (completamente incapsulato)
- **Il formato dei dati scritti è completamente nascosto**

- **Incapsula** un `Writer` (soluzione generale)
- **Il formato dei dati scritti è completamente nascosto**

Struttura Interna

- La logica è del tutto simile a quella già studiata in Fondamenti di Informatica T-1 con le stesse varianti…
  - …sia per gli stream binari…
  - …sia per gli stream di testo.

Lettura/scrittura - Eccezioni

- Rilanciare tutte le eccezioni non è la scelta giusta → **quali rilanciare?**

- **Rilanciare** (`throws`) le eccezioni **valide in tutti i casi**
  - `IOException` → se si lavora con *stream* si ha sempre a che fare con `IOException`
- **Catturare** (`try-catch`) le eccezioni **valide solo in casi specifici**

Lettura/scrittura - Eccezioni

- Nella lettura dei dati, valutare come comportarsi (*catch* o non *catch*…)

File Binari

  - Lettura tramite `ObjectInputStream`
    - `ClassNotFoundException`: lo *stream* si riferisce a classi che non esistono nel contesto corrente: cosa si sta caricando?
      → incapsulare in eccezione più generica
    - `InvalidClassException`: problema di versionamento delle classi (il serialVersionUID non corrisponde)
      → incapsulare in eccezione più generica
    - `EOFException`: raggiunta la fine dello stream – **da trattare e non rilanciare!**

Lettura/scrittura - Eccezioni

File di Testo

– Lettura tramite **`Reader`**/**`BufferedReader`**

- **`NumberFormatException`**: tipici errore di conversione da stringa a numero

  → incapsulare in eccezione più generica

- I problemi da gestire quando si legge da file di testo sono **gli stessi** che ci sono quando si legge da file binari… **ma vanno gestiti A MANO!**

Lettura/scrittura - Eccezioni

- Poiché dal tipo di eccezioni rilanciate, traspare la reale implementazione: si perde parte dell'incapsulamento!
- Quindi si incapsulano le eccezioni da mascherare in una di uso generale
  - Es: **`ClassNotFoundException`** è specifica della lettura da file binari
- **`InvalidDataException`** deriva da **`Exception`** e non aggiunge nessuna informazioni aggiuntiva… se non il proprio tipo (scusate se è poco!)

Lettura/scrittura – Binario

- Facilissimo!!
- Si legge/scrive l'oggetto INTERO tramite un **`ObjectInputStream`** o un **`ObjectOutputStream`**
- Ricordarsi trattare correttamente le eccezioni come prima specificato.

Codice!

```
package serializers.binarySerializers;

public class BinaryComponentLoader implements ComponentLoader
{
   private ObjectInputStream innerStream;

   public BinaryComponentLoader(InputStream inputStream)
                                    throws IOException
   {
        innerStream = new ObjectInputStream(inputStream);
   }
   …
}
```

Ha il compito di caricare oggetti di tipo **`Component`** da stream binario

Incapsula un **`InputStream`**

Internamente lavora con un **`ObjectInputStream`**

Codice!

```
@Override
public Component read()
                    throws IOException, InvalidDataException
{
     try {
        return (Component)innerStream.readObject();
     }
     catch (ClassNotFoundException e) {
        throw new InvalidDataException("Dati non validi", e);
     }
     catch (InvalidClassException e) {
        throw new InvalidDataException("Dati non validi", e);
     }
     catch (EOFException e) {
        return null;
     }
}
```

Viene letto l'oggetto INTERO

Problemi di deserializzazione → Eccezioni da rilanciare

File terminato → Viene restituito `null`

Codice!

```
package serializers.binarySerializers;

public class BinaryComponentSaver implements ComponentSaver
{
   private ObjectOutputStream innerStream;

   public BinaryComponentSaver(OutputStream outputStream)
                                    throws IOException
   {
        innerStream = new ObjectOutputStream(outputStream);
   }

   @Override
   public void write(Component c) throws IOException
   {
        innerStream.writeObject(c);
   }
}
```

Ha il compito di salvare oggetti di tipo `Component` su stream binario

Incapsula un `OutputStream`

Internamente lavora con un `ObjectOutputStream`

Viene scritto l'oggetto INTERO

Lettura/scrittura – Testo

- Si legge/scrive l'oggetto campo per campo tramite un **`BufferedReader`**/**`Writer`**
- **Scrittura**: facile (si usa direttamente il **`Writer`** o un **`PrintWriter`**)
- **Lettura**: leggere una riga del file alla volta e "smontarla" tramite uno **`StringTokenizer`**
- Ricordarsi di rilanciare le opportune eccezioni

Il carattere newline?

- La JVM è portabile ed il compilato Java è eseguibile ovunque esista una JVM
- Diversi sistemi operativi possono rappresentare il newline in modo diverso (**`\n`**, **`\r`**, **`\n\r`**, oppure??)
- È una proprietà di sistema operativo e, come tale, è possibile recuperarla:

```
System.getProperty("line.separator");
```

Quali proprietà?

System property	Description
java.version	Java Runtime Environment version
java.vendor	Java Runtime Environment vendor
java.vendor.url	Java vendor URL
java.home	Java installation directory
java.vm.specification.version	Java Virtual Machine specification version
java.vm.specification.vendor	Java Virtual Machine specification vendor
java.vm.specification.name	Java Virtual Machine specification name
java.vm.version	Java Virtual Machine implementation version
java.vm.vendor	Java Virtual Machine implementation vendor
java.vm.name	Java Virtual Machine implementation name
java.specification.version	Java Runtime Environment specification version
java.specification.vendor	Java Runtime Environment specification vendor
java.specification.name	Java Runtime Environment specification name

Quali proprietà?

System property	Description
java.class.version	Java class format version number
java.class.path	Java class path
java.library.path	List of paths to search when loading libraries
java.io.tmpdir	Default temp file path
java.compiler	Name of JIT compiler to use
java.ext.dirs	Path of extension directory or directories
os.name	Operating system name
os.arch	Operating system architecture
os.version	Operating system version
file.separator	File separator ('/' on UNIX)
path.separator	Path separator (':' on UNIX)
line.separator	Line separator ('\n' on UNIX)
user.name	User's account name
user.home	User's home directory
user.dir	User's current working directory

Lettura/Scrittura – Testo

- Un esempio di file da leggere…

Samsung;Monitor 24" L3424P;Response Time: 2ms;258.0
Samsung;HD 1TB 7200rpm 8MB Cache SATA2;Seek Time 5ms;80.0
Asus;eee box 206;Atom N270 + ATI3434HD, 1GB RAM, 320GB HD;420.0
Trust;Keyboard 300;Ita;10.0

Codice!

```
package serializers.textSerializers;

public class TextComponentLoader implements ComponentLoader
{
   private static String SEPARATORS = ";";
   private BufferedReader innerReader;

   public TextComponentLoader(Reader reader)
   {
       this.innerReader = new BufferedReader(reader);
   }
   …
}
```

Ha il compito di caricare oggetti di tipo **`Component`** da stream di testo

I campi dell'oggetto sono separati da un ";"

Incapsula un **`Reader`**

Internamente lavora con un **`BufferedReader`**

Codice!

```
…
@Override
public Component read()
                throws IOException, InvalidDataException
{
     String line = innerReader.readLine();
     if (line != null)
     {
         StringTokenizer tokenizer = new StringTo
         Component c = new Component();

         if (!tokenizer.hasToken())
                 throw new InvalidDataException("Manufacturer"));
         c.setManufacturer(tokenizer.nextToken(SEPARATORS));

         if (!tokenizer.hasToken())
                 throw new InvalidDataException("Model");
         c.setModel(tokenizer.nextToken(SEPARATORS));

         if (!tokenizer.hasToken())
                 throw new InvalidDataException("Description");
         c.setDescription(tokenizer.nextToken(SEPARATORS));
         …
}
```

L'oggetto di tipo **`Component`** viene letto campo per campo

Codice!

```
     …
     if (!tokenizer.hasToken())
                 throw new InvalidDataException("Price");

      try
      {
         c.setPrice(
             Double.parseDouble(tokenizer.nextToken()));
      }
      catch (NumberFormatException e)
      {
         throw new InvalidDataException("Price Format", e);
      }
      return c;
 }
 else
 {
     return null;
 }
```

Check di validità!

Se il file è terminato viene restituito **`null`**

Oppure…

```
@Override
public Component read() throws IOException, InvalidDataException {
    String line = innerReader.readLine();
    if (line != null) {
        StringTokenizer tokenizer = new StringTokenizer(line);
        Component c = new Component();
        try {
            c.setManufacturer(tokenizer.nextToken(SEPARATORS));
            c.setModel(tokenizer.nextToken(SEPARATORS));
            c.setDescription(tokenizer.nextToken(SEPARATORS));
            c.setPrice(Double.parseDouble(tokenizer.nextToken()));
        } catch(NoSuchElementException e) {
            throw new InvalidDataException ("Dati mancanti", e);
        } catch(NumberFormatException e) {
            throw new InvalidDataException("Numero non valido", e);
        }
   }
```

Codice più compatto ma meno informazioni date sugli errori avvenuti…

Codice!

```
package serializers.textSerializers;

public class TextComponentSaver implements ComponentSaver
{
   private static String SEPARATOR = ";";

   private Writer innerWriter;

   public TextComponentSaver(Writer writer)
   {
        this.innerWriter = writer;
   }
   …
}
```

Ha il compito di salvare oggetti di tipo **Component** su stream di testo

I campi degli oggetti DEVONO essere separati da un ";"!

Incapsula e lavora con un **Writer**

Codice!

```
…
@Override
public void write(Component c)  throws IOException
{
     innerWriter.write(c.getManufacturer());
     innerWriter.write(SEPARATOR);

     innerWriter.write(c.getModel());
     innerWriter.write(SEPARATOR);

     innerWriter.write(c.getDescription());
     innerWriter.write(SEPARATOR);

     innerWriter.write(Double.toString(c.getPrice()));
     innerWriter.write("\n");
}
```

L'oggetto di tipo `Component` viene scritto campo per campo

Gli oggetti sono separati da ***NEWLINE***

Contenitore: quanto grande?

- Non c'è problema, c'è il JCF!

All together, now!

- Come si comporta la libreria sul campo?
- Verifiche:
  - Caricare da file (diversi formati) e stampare su console
  - Copiare da un file ad un altro (diversi formati)

Codice!

```
public class FileTestMain
{
   public static void copy(ComponentLoader from,
                           ComponentSaver to)
                           throws IOException, InvalidDataException
   {
        Component c;
        while ((c = from.read()) != null)
        {
            to.write(c);
        }
    }
    …
}
```

Copia da un file ad un altro

Codice!

Legge un file e di stampa su console gli oggetti letti

```
public static void readAndPrint(ComponentLoader reader)
                  throws IOException, InvalidDataException
{
     Component c;
     while ((c = reader.read()) != null)
        System.out.println(c);
}
```


Codice!

```
…
public static void main(String[] args)
               throws FileNotFoundException, IOException,
                                 InvalidDataException
{
     Reader reader = new FileReader("components.txt");
     readAndPrint(new TextComponentLoader(reader));
     reader.close();

     reader = new FileReader("components.txt");
     OutputStream outputStream =
               new FileOutputStream("components.dat");
     copy(new TextComponentLoader(reader),
               new BinaryComponentSaver(outputStream));
     reader.close();
     outputStream.close();
     …
}
```

IMPORTANTE: ricordarsi sempre di chiudere gli stream!

Codice!

```
…
InputStream inputStream =
              new FileInputStream("components.dat");
Writer writer = new FileWriter("newComponents.txt");
copy(new BinaryComponentLoader(inputStream),
          new TextComponentSaver(writer));
inputStream.close();
writer.close();
```

IMPORTANTE: ricordare **SEMPRE** di **chiudere gli stream**!

E se…

- Gli oggetti sono composti?
  - Se su stream binario, no problem!
  - Se su stream di testo, dipende dal formato
    - Per avere un buon design sarebbe bene realizzare **UNA COPPIA LOADER/SAVER** per ogni classe (componente o composto)
      - L'oggetto composto usa i reader/writer degli oggetti componenti
    - In casi semplici, ci si può **accontentare** di una unica coppia LOADER/SAVER

E se...

- Gli oggetti sono in gerarchia?
  - Se su stream binario, no problem!
  - Se su stream di testo, dipende dal formato
    - In generale, in ogni riga (all'inizio?) c'è l'indicazione della classe da istanziare
    - Successivamente, in base al tipo, si leggono i campi specifici
    - Come nel caso precedente, nel caso più generale (e complesso) si può prevedere una coppia loader/saver per ogni oggetto in gerarchia…
    - …in casi specifici (**semplici**) ciò potrebbe essere superfluo – **dipende dal contesto!**

Esercizio (per casa)

- Il componente per il calcolo del codice fiscale può essere finalmente completato!
- Sul sito del corso c'è il file (di testo) contenente tutti i comuni d'Italia ed i relativi codici

1. Come sono fatti i dati?
2. Realizzare la lettura e memorizzare in un opportuno contenitore
   - Qual è il contenitore che minimizza i tempi di ricerca?
3. Integrare il tutto nel componente per il calcolo del codice fiscale