Alma Mater Studiorum-Università di Bologna
Scuola di Ingegneria

Gestione dell'I/O in Java
I/O con JAR

Corso di Laurea in Ingegneria Informatica

Anno accademico 2021/2022

Prof. ENRICO DENTI

Dipartimento di Informatica – Scienza e Ingegneria (DISI)

ARCHIVI JAR: DATI vs. RISORSE

- Gli archivi JAR si usano per distribuire *librerie* o *applicazioni*
- Se tali software devono accedere a file, occorre distinguere:
  - JAR che devono accedere a file ***esterni*** al JAR stesso
  - JAR che devono accedere a file ***interni*** al JAR stesso
- Tipicamente:
  - un'applicazione che elabori file di utente *non* li avrà, ovviamente, inclusi nel JAR stesso
  - un'applicazione che invece carichi dati, icone o suoni per la propria configurazione li avrà al proprio interno
    → si parla più propriamente di ***risorse***

ARCHIVI JAR: DATI vs. RISORSE

- Le classi **`FileReader`** e **`FileInputStream`** possono accedere solo a file *esterni* al JAR stesso
  - motivo: sono pensati per accedere al *file system*
  - *non sono in grado* di accedere a *risorse interne* al JAR
- Per leggere file di *risorse* (testo, immagini, suoni) *interne* al JAR, occorre utilizzare il metodo **`getResourceAsStream`** della classe in cui si fa la lettura
  - **`getResourceAsStream`** restituisce un **`InputStream`**, che nel caso di file di testo potrà essere utilmente incapsulato in un **`InputStreamReader`** per apparire come **`Reader`**
  - più esattamente, **`getResourceAsStream`** è un metodo della (meta) classe **`Class`**, i cui oggetti rappresentano le classi presenti nel sistema a run-time: si usa perciò secondo uno *schema particolare*

getResourceAsStream

- L'oggetto di classe **`Class`** su cui invocare il metodo **`getResourceAsStream`** si ottiene:
  - invocando il metodo **`getClass()`** sull'oggetto corrente, se il metodo in cui si sta lavorando *non è* statico
    ESEMPIO:
    `this.getClass().getResourceAsStream(…)`
  - con la notazione ***`Nomeclasse`*`.class`**, se il metodo in cui si sta lavorando *è* statico
    ESEMPIO:
    `MyClass.class.getResourceAsStream(…)`
  - NB: il percorso nel metodo **`getResourceAsStream`** si riferisce alla *root dello zip file*

ESEMPIO 1: LETTURA DI DATI BINARI

- Si supponga che il file binario **`resource.bin`** contenga valori interi, precedentemente salvati
  - non è dato sapere a priori quanti siano, quindi occorrerà per forza effettuare un ciclo di lettura «alla cieca» fino a EOFException
- Versione classica, con file esterno all'eventuale JAR:

```
try (DataInputStream is = new DataInputStream(new FileInputStream(filename))) {
    while(true) System.out.println(is.readInt());
    // eventually, it will be interrupted by EOF
}
catch(FileNotFoundException e1){
  System.out.println("File not found: " + filename);
  System.exit(2);
}
catch(EOFException e2){
  // normal termination, actually
  System.exit(0);
}
catch (IOException e3){
  System.out.println("Unexpected input error");
  System.exit(3);
}
```

ESEMPIO 1: LETTURA DI DATI BINARI

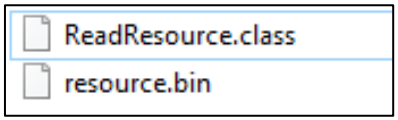
- Se il file binario **`resource.bin`** deve essere incluso in un JAR come risorsa interna, la lettura va riformulata
  - anziché costruire un **`FileInputStream`**, occorre recuperare un input stream come risorsa tramite **`getResourceAsStream`**

```
public class ReadResource {

 public static void main(String args[]){
    String filename = "resource.bin";

    try (DataInputStream is = new DataInputStream(ReadResource.class.getResourceAsStream(filename))) {
        while(true) System.out.println(is.readInt());
        // eventually, it will be interrupted by EOF
    }
    catch(FileNotFoundException e1){
      System.out.println("File not found: " + filename);
      System.exit(2);
    }
    catch(EOFException e2){
      // normal termination, actually
      System.exit(0);
    }
    catch (IOException e3){
      System.out.println("Unexpected input error");
      System.exit(3);
    }
 }
}
```

Usa la notazione
`NomeClasse.class`
perché questo codice si trova in un metodo statico (il main)

ESEMPIO 1: LETTURA DI DATI BINARI

- Dopo aver testato il corretto funzionamento, si può predisporre il JAR autocontenuto




```
>jar cef ReadResource ReadData.jar ReadResource.class resource.bin
```

Nome classe con il main

Nome JAR da produrre

File da includere

- Test del JAR autocontenuto appena creato
  - NB: da fare senza nessun altro file nella cartella!


```
C:\Temp\prova>java -jar ReadData.jar
23
-5
61
18
```

ESEMPIO 2: LETTURA DI UN FILE DI TESTO

- Main originale (statico) che accedeva a un file su disco:

```
try(FileReader rdr =
          new FileReader("LineaMI-BO.txt")) {
   ...
}
```

- Main modificato per accedere a un file interno al JAR:

```
try(InputStreamReader rdr =
          new InputStreamReader(
          MyMain.class.getResourceAsStream(
                "/LineaMI-BO.txt"))) {
   ...
}
```

Usa la notazione
NomeClasse.class
perché questo codice si trova in un metodo statico (il main)

Il percorso **/** si riferisce alla *root dello zip file*

ESEMPIO 2 – VERSIONE ORIGINALE (file da leggere esterno al JAR)

```
>java -jar ferrovia.jar
[Milano Centrale - km 0, Milano Lambrate - km 4, Milano Rogoredo - km 10, Piacen
za - km 72, Fidenza - km 107, Parma - km 129, Reggio Emilia - km 157, Modena - k
m 182, Bologna Centrale - km 219]
Milano Centrale - km 0
Milano Lambrate - km 4
Milano Rogoredo - km 10
Piacenza - km 72
Fidenza - km 107
Parma - km 129
Reggio Emilia - km 157
Modena - km 182
Bologna Centrale - km 219
Milano Centrale - km 0
Milano Lambrate - km 4
Milano Rogoredo - km 10
Piacenza - km 72
Fidenza - km 107
Parma - km 129
Reggio Emilia - km 157
Modena - km 182
Bologna Centrale - km 219
```

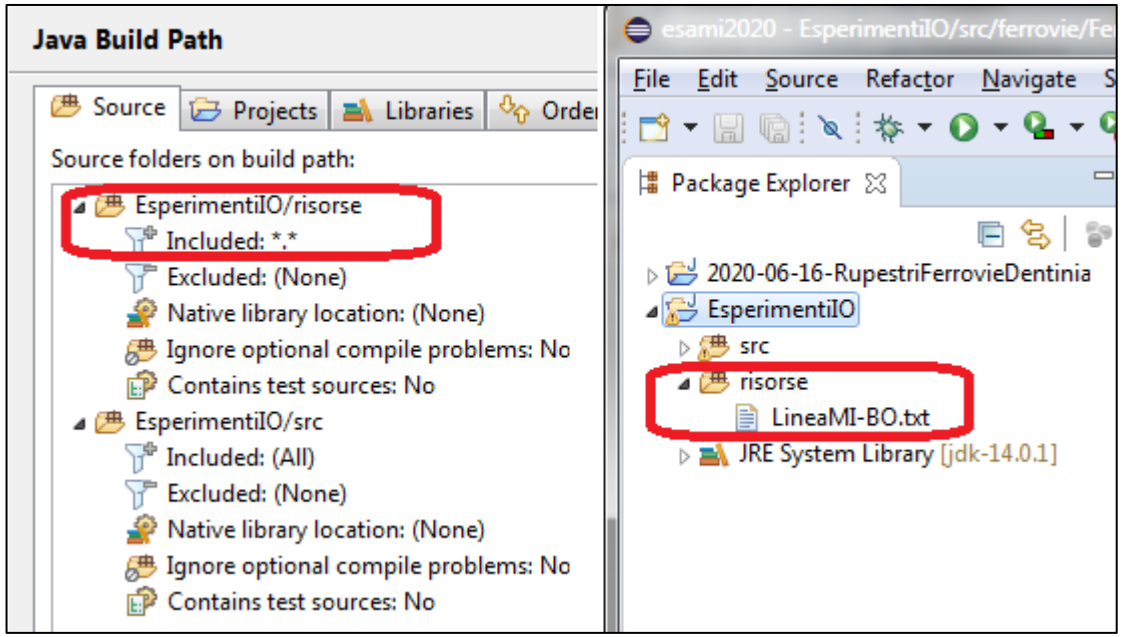
Situazione classica: il file da leggere è esterno al JAR
→ si usa normalmente FileReader

```
>jar -tf ferrovia.jar jar
META-INF/MANIFEST.MF
ferrovie/
ferrovie/Ferrovia.class
ferrovie/BadFileFormatException.class
ferrovie/FerroviaConScanner.class
ferrovie/Stazione.class
ferrovie/FerroviaBinaryPersister.class
persone/
persone/BadFileFormatException.class
persone/Persona.class
persone/LeggiPersone.class
persone/LeggiPersoneConScanner.class
persone/LeggiPersoneConSplit.class
voli/
voli/Flight.class
voli/FlightPos.class
voli/BadFileFormatException.class
voli/LeggiVolo.class
```

Infatti, il file di testo *non* è incluso nel JAR

CREAZIONE DEL JAR CON RISORSE IN ECLIPSE

- Da Eclipse, far includere i file-risorse richiede un po' di cura
  1. creare una cartella apposita, di tipo *source folder,* nel progetto (es. risorse) e *metterci dentro i file necessari*
  2. aggiungerla al *build path*
  3. specificare come *include pattern* di quella cartella il tipo di file da includere (es. tutti: *.*)
  4. Esportare normal-mente il *Runnable jar*



ESEMPIO 2 – VERSIONE MODIFICATA (file da leggere *interno* al JAR)

```
>jar -tf ferrovia2.jar
META-INF/MANIFEST.MF
org/
org/eclipse/
org/eclipse/jdt/
org/eclipse/jdt/internal/
org/eclipse/jdt/internal/jarinjarloader/
org/eclipse/jdt/internal/jarinjarloader/JIJConstants.class
org/eclipse/jdt/internal/jarinjarloader/JarRsrcLoader$ManifestInfo.class
org/eclipse/jdt/internal/jarinjarloader/JarRsrcLoader.class
org/eclipse/jdt/internal/jarinjarloader/RsrcURLConnection.class
org/eclipse/jdt/internal/jarinjarloader/RsrcURLStreamHandler.class
org/eclipse/jdt/internal/jarinjarloader/RsrcURLStreamHandlerFactory.class
ferrovie/
ferrovie/FerroviaMainResource.class
ferrovie/FerroviaMain.class
ferrovie/BadFileFormatException.class
ferrovie/FerroviaConScanner.class
ferrovie/Stazione.class
ferrovie/FerroviaBinaryPersister.class
persone/
persone/BadFileFormatException.class
persone/Persona.class
persone/LeggiPersone.class
persone/LeggiPersoneConScanner.class
persone/LeggiPersoneConSplit.class
voli/
voli/Flight.class
voli/FlightPos.class
voli/BadFileFormatException.class
voli/LeggiVolo.class
LineaMI-BO.txt
```

Conseguenza:
Eclipse include automaticamente la libreria necessaria

Ora il file da leggere è *interno* al JAR, letto da **`getResourceAsStream`**