Programmazione 2

Esercitazione 9

Livio Pompianu - Simone Porcu

JAR

Jar

E’ un archivio dati compresso come un comune file zip

Al suo interno comprende un sistema di package (ora sapete perché ve li facciamo usare fino alla morte)

Il suo utilizzo comporta dei vantaggi quali:

compressione **->** riduce lo spazio

firma **->** è possibile inserire il nome dell’autore

estensione **->** è visto come un package, quindi può essere espanso

portabilità **->** utilizzabile su tutte le piattaforme JVM

documentazione **->** JavaDoc

Creazione di un JAR

Creare un file di testo **manifest.txt** che contenga la stringa:

- **Main-Class: package.NomeMainClass** (mettere il nome della classe contenente il main)

e che termini con un "a capo" (nuova linea). Quindi si utilizza il comando jar con la seguente sintassi:

- **jar cfm MyJar.jar manifest.txt package/*.class** (se sono stati utilizzati più package, indicarli tutti. Esempio: package1/*.class package2/*.class ...)

Verrà generato il file MyJar.jar eseguibile col seguente comando:

- **java -jar MyJar.jar**

Esercizio guidato JAR - ConvertiEuro

Creare un file di testo **manifest.txt** che contenga la stringa:

- **Main-Class: finanza.test.TestConvertiEuroEccezioni**

e che termini con un "a capo" (nuova linea). Quindi si utilizza il comando jar con la seguente sintassi:

- **jar cfm MyJar.jar manifest.txt finanza/test/TestConvertiEuroEccezioni.class finanza/*.class finanza/eccezioni/*.class**

Verrà generato il file MyJar.jar eseguibile col seguente comando:

- **java -jar MyJar.jar**

Test Driven Development

Test Driven Development

Metodologia di sviluppo codice in maniera legata alla strutturazione di test automatici.

Tre passi definiti “Red-Green-Refactor”:

1. Scrittura di un test che necessariamente fallisce (RED);
2. Implementazione del minimo codice necessario a superare il test (GREEN);
3. Refactoring del codice (REFACTOR).

Test Driven Development

L’approccio descritto permette di sviluppare codice che rispetta le specifiche finali.

I programmatori possono inoltre rimodellare il software certi di ottenere un funzionamento corretto (non si modifica l’interfaccia esterna).

Un software così prodotto è testabile per singoli moduli, semplificando la manutenzione del codice.

JUnit

JUnit

Framework di unit testing per il linguaggio java.

Consente di scrivere dei test sul proprio codice ed ottenere una risposta visuale immediata sul risultato ottenuto.

Una barra verde indica il superamento di tutti i test, una barra rossa indica che un test è fallito: in tal caso è possibile osservare i dettagli.

JUnit

JUnit consiste in un semplice jar, mantiene la portabilità tipica di java e può essere:

- integrato in un ambiente di sviluppo;
- utilizzato con Eclipse;
- utilizzato da terminale.

Le classi di test realizzate utilizzano codice Java con in più specifiche annotazioni e asserzioni.

Stream

Stream

Sono diversi dagli InputStream e OutputStream visti nelle scorse lezioni

Uno stream rappresenta una sequenza di elementi sui quali possiamo eseguire differenti operazioni

Le operazioni possono essere di 2 tipi:

- Intermedie (restituiscono un nuovo stream -> Filter, Map, Sorted)
- Finali (possono essere void o comunque non restituire un risultato -> ForEach)

Stream

Gli stream sono sequenze di elementi.

```
List<String> myList = Arrays.asList("a1", "a2", "b1", "c2", "c1");

myList
    .stream()
    .filter(s -> s.startsWith("c"))
    .map(String::toUpperCase)
    .sorted()
    .forEach(System.out::println);

// C1
// C2
```

Stream

Molte operazioni accettano come parametro una *functional interface* che stabilisce l'esatto comportamento del metodo.

La functional interface dovrebbe avere solo un metodo astratto. Insieme all'unico metodo astratto, si possono avere un numero qualsiasi di metodi predefiniti e statici.

```
@FunctionalInterface
interface MyFunctionalInterface {
    public int sumMethod(int a, int b);
}

public class MyMainClass {
   public static void main(String args[]) {
     // lambda expression
     MyFunctionalInterface myInterface = (a, b) -> a + b;
        System.out.println("Result: " + myInterface.sumMethod(12, 100));
    }
}
```

Stream

Spesso è richiesto che l'operazione abbia le seguenti proprietà:

- Non-Interference: non modifica la sorgente di dati dello stream (nel nostro esempio non devono essere eseguite modifiche aggiungendo o rimuovendo elementi dalla collection).

- Stateless: l'operazione dello stream non dipende da variabili o stati che possano cambiare durante l'esecuzione: l'operazione è deterministica.

Stream

Scaricare il file TestStream.java da Moodle.

Utilizzare le API dell’interfaccia **Stream**.