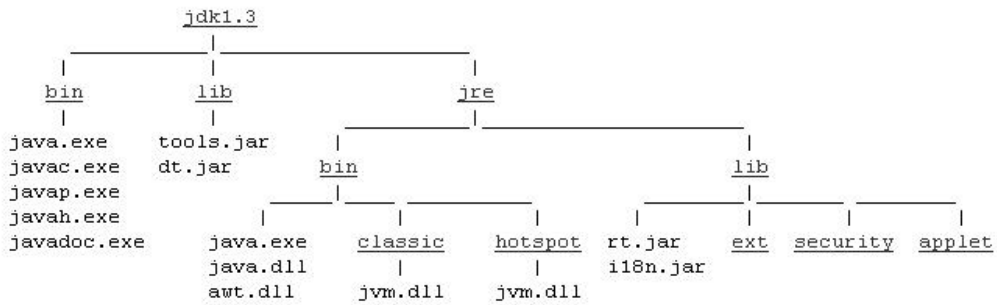
Esercitazione n° 1

Obiettivi

- Introduzione all'utilizzo di ***Java Development Kit*** (**JDK**) versione 1.3
- Sviluppare programmi Java tramite linea di comando
  Es: *javac, java, jdb, javadoc…*
- Primo esempio di programma Java con due classi:
  - variante in un solo file
  - variante in due file distinti

JDK1.3: il direttorio

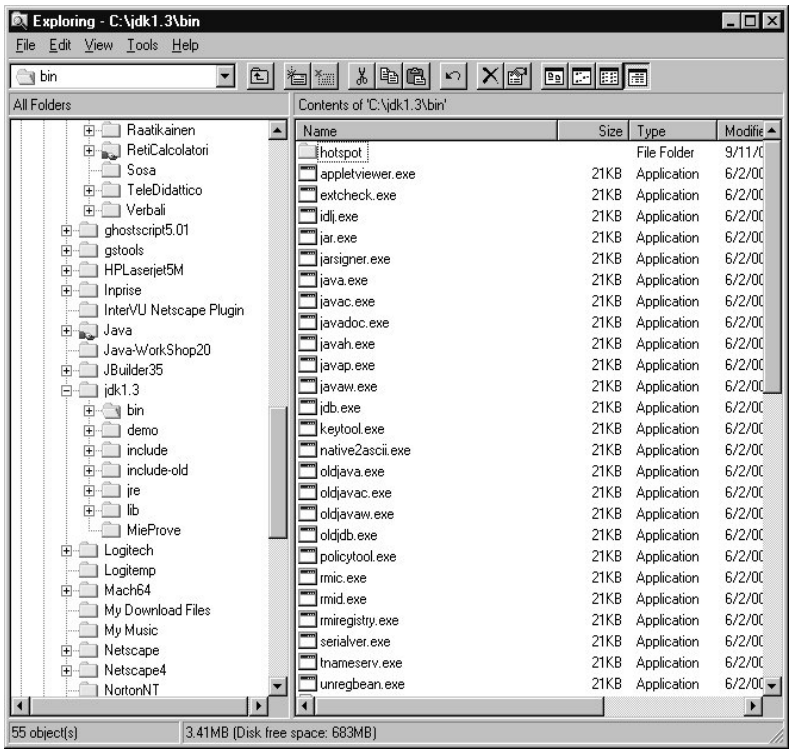


.\jdk1.3\lib Contiene i file usati dai tool di sviluppo

.\jdk1.3\jre Java Runtime Environment. Librerie ed eseguibili per la Java Virtual Machine

.\jdk1.3\bin Contiene i file eseguibili del tool di sviluppo

JDK1.3: il direttorio /bin



Strumenti a linea di comando

- ***Compilatore*** - javac
- ***Macchina virtuale*** - java
- ***Debugger*** - jdb
- Generatore automatico di ***documentazione ipertestuale*** - javadoc
- Altri ***comandi*** - appletviewer, javap, rmic, …

Documentazione completa:
http://www.java.sun.com/j2se/1.3/docs/tooldocs/tools.html

javac

- Legge *sorgenti Java* di classi e interfacce e li ***compila*** in *classfile* in formato *bytecode*
- Permette di compilare ***classi singole*** e ***gruppi di classi***, anche mantenendo direttori separati per i file sorgenti e compilati

javac [opzioni] [sorgenti] [@ElencoSorg]

possibili opzioni:

- **classpath**: classi bootstrap, extension, poi *classpath* di utente (*variabile di ambiente* e *opzione*)

-**d** (direttorio per classi), -**g** (debugging abilitato), -**verbose** (info estese sulla compilazione), **...**

java

- Mette in ***esecuzione*** una applicazione Java
- Avvia una macchina virtuale Java, ***carica*** una classe specificata e invoca il suo metodo ***main*** (*pubblico* e *statico*)

java [opzioni] File.class [parametri]

java [opzioni] -jar File.jar [parametri]

varianti: ***javaw***, ***oldjava***, ***oldjavaw***

opzioni: -**classpath**/-**cp**, -**jar**, -**verbose**, -**version**, -**?**, ...

jdb

- Permette di eseguire il ***debugging*** (esecuzione *passo-passo*, *breakpoint*, osservazione di *variabili*, ...) di un programma Java

  jdb File.class

  – invoca una ***nuova*** macchina virtuale Java che esegue il programma in modalità debugging

  – è anche possibile "agganciare" una macchina virtuale già in esecuzione (opzione -Xdebug di **java**)

- ***Comandi***: *help*/*?*, *run*, *cont*, *print* (mostra oggetti o tipi primitivi), *stop at/in* (posiziona breakpoint), *clear* (rimuove breakpoint), *step* (esecuzione passo-passo)

javadoc

- Analizza le *dichiarazioni* e i *commenti* di un insieme di file sorgenti Java e produce la corrispondente ***documentazione ipertestuale*** (formato html), descrivendo *classi*, *interfacce*, *costruttori*, *metodi* e *campi*
- Può essere invocato su ***singole classi*** o ***interi package***

  javadoc [opzioni][package][sorg][@ElencoS/P]

Vari tag standardizzati per i commenti:
@author, @param, @return, @throws, @see, @version…

ESEMPIO

```
/**
 * This is a <b>doc</b> comment.
 * @see java.lang.Object
*/
```

Esempio: L'EURO-CONVERTITORE (1)

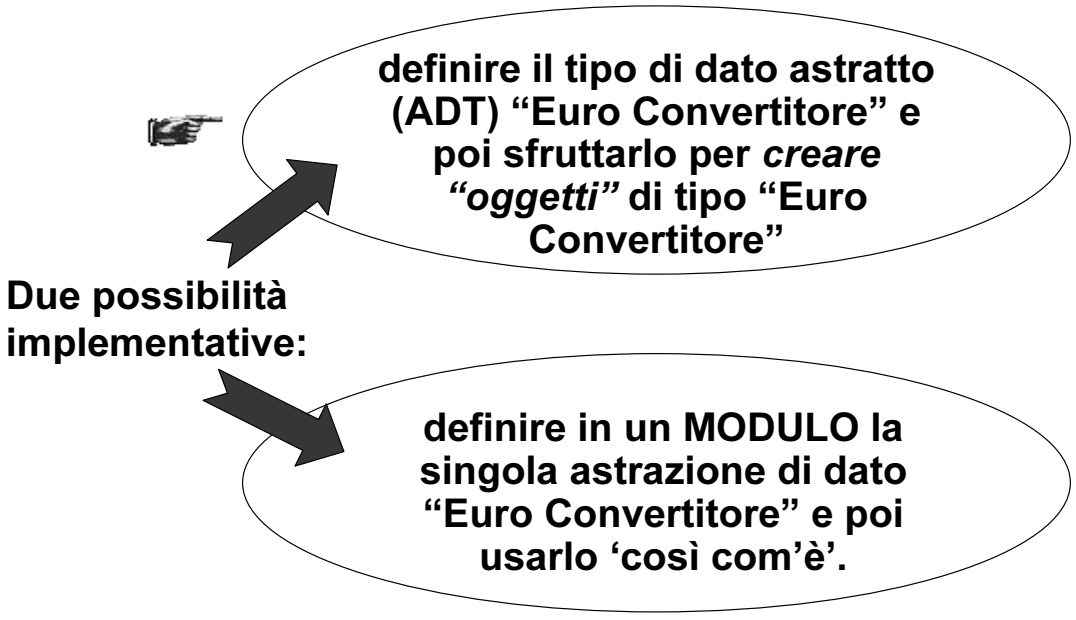
Scopo

Definire una astrazione per un semplice Euro-Convertitore

Specifica

- Componente caratterizzato in ogni istante da un accumulatore che contiene la quantità corrente di Euro da convertire
- Componente a cui si accede tramite le operazioni di:
  - **add**: Somma Euro alla quantità in accumulatore
  - **sub:** Toglie Euro alla quantità in accumulatore
  - **convert:** restituisce il corrispondente valore in Lire della quantità di Euro in accumulatore
  - **currentValue:** restituisce l'attuale valore dell'accumulatore

Esempio: L'EURO-CONVERTITORE (2)



Esercitazione n°1: variante in un file (1)

Come procedere?

1. Scaricare il file **Esempio1.java** dal sito Web del corso e salvarlo in **C:\TEMP**

2. Il file **Esempio1.java** contiene due classi: una classe **public** di nome completo ***Esempio1*** e una classe innestata di nome ***EuroConverter***.

Esercitazione n°1: variante in un file (2)

4. La classe ***EuroConverter*** definisce il tipo di dato astratto 'Euro-Convertitore'.

   4.1 **Parte Nascosta (stato):** definisce in una variabile protetta l'accumulatore

   4.2 **Parte Visibile (operazioni):**

   Costruttore: EuroConverter(double euro)
   Trasformatori: void add(double euro)
   void sub (double euro)
   ***Selettori*:** double currentValue()
   double convert()

Per l'arrotondamento si possono usare i metodi di utilità predefiniti nella classe java.lang.Math di Java

Esercitazione n°1: variante in un file (3)

5 La classe **Esempio1** contiene il metodo ***main()*** nel quale viene utilizzato il tipo di dato astratto **EuroConverter**

5.1 In particolare, crea due oggetti di tipo ***EuroConverter*** referenziati dalle variabili uno e due ed inizializzati al valore iniziale comune di € 100,00.

5.2 Quindi somma 10 Euro al convertitore uno e ne toglie 10 dal convertitore due. [istruzioni "`uno.add(10.00);`" e "`due.dec(10.00);`"].

5.3 Infine stampa a video i valori di conversione in Lire degli accumulatori dei due convertitori usando "`System.out.println`" per la stampa. Il valore nel convertitore `uno` sarà £ 212.990, quello del convertitore `due` £ 174.264.

Esercitazione n°1: variante in un file (4)

Una possibile soluzione. Contenuto del file Esempio1.java:

```
public class Esempio1 {
  public static void main(String[] args) {
    EuroConverter uno = new EuroConverter (100.00);
    EuroConverter due = new EuroConverter (100.00);
    uno.add(10.00);
    due.sub(10.00);
    System.out.println("Uno: Importo Lire " + uno.convert());
    System.out.println("Due: Importo Lire " + due.convert());}
}
public class EuroConverter{
  private double i=0;      //ACCUMULATORE
  public EuroConverter(double j) {i=j;}
  public void add(double j) {i=i+j;} //Somma i di j Euro
  public void sub(double j){i=i-j;}  // Sottrazione...
  public double currentValue(){return i;}
  public double convert() {return Math.rint(i*1936.27);}
 }
```

Esercitazione n°1: variante in un file (5)

Compilazione

C:\TEMP> `javac Esempio1.java`

? *quali e quanti file .class produce? In quale directory?*

C:\TEMP> `javac -g Esempio1.java`

? *cosa cambia?*

L'opzione -g avvia una compilazione con informazione di debugging abilitate

Esecuzione

C:\TEMP> `java Esempio1`

Documentazione

C:\TEMP> `javadoc Esempio1.java`

Le strutture di Controllo

Esistono in Java alcune istruzioni che consentono di saltare in modo controllato da una istruzione all'altra, variando l'ordine implicito di **esecuzione sequenziale** del programma.

- **Istruzioni di Selezione (if/else; switch)**
- **Istruzioni di Iterazione (for ; while; do/while)**
- **Istruzioni di Salto (break ; continue + eccezioni)**

Da usare solo se strettamente necessario (es: in associazione al costrutto switch) poiché possono pregiudicare la leggibilità del codice!

Istruzioni di Selezione

```
if (condizione)          if (condizione){
   istruzione1;             // Istruzioni
else istruzione2;        } else istruzione2;
```

- *Condizione*: qualunque espressione che ritorni un valore boolean (true o false)

Se *condizione* è **true** viene eseguita *istruzione1*, se **false** *istruzione2*. In nessun caso entrambe le istruzioni (o blocchi di istruzioni).

- Parte *else* opzionale
- Possibilità di più *if* annidati

Istruzioni di Iterazione (1)

```
for (inizializzazione; condizione; iterazione) {
   // istruzioni
}
```

- *Inizializzazione*: espressione che imposta il valore della/e variabile/i di controllo del ciclo
- *Condizione*: qualunque espressione che ritorni un valore boolean (true o false)

Se *condizione* è **true** viene eseguito il corpo del ciclo, se **false** il ciclo termina

- *Iterazione*: espressione che incrementa o riduce alla fine di ogni iterazione il valore della/e variabile/i di controllo del ciclo

ES:

```
for (int a=0; a<10; a++)
  System.out.println(" Contantore: " + a);
```

Istruzioni di Iterazione (2)

while *(condizione) {* *// corpo del ciclo* *}*	***do*** *{* *// corpo del ciclo* *}* ***while*** *(condizione);*

- *Condizione*: qualunque espressione che ritorni un valore boolean (true o false)

Viene eseguito il corpo del ciclo finchè (tante volte quante) *condizione* è **true**. Quando *condizione* diventa **false** il controllo passa alla riga di codice immediatamente sucessiva al ciclo.

- Nel ciclo ***do/while*** il corpo con le istruzioni viene eseguito almeno una volta, nel ciclo ***while*** ciò non è sempre vero.

ES:

```
int a = 0;
 while (a<10){
   System.out.println(" Contantore: " + a); a++;
}
```


Esercitazione n°1: variante su due file(1)

Scopo

- **Definire le classi su più file**
- **Esempio di uso delle strutture di controllo**
- **Realizzare un'applicazione che stampa una tabella di conversione**

Specifica

Si vuole produrre in uscita la stampa di una tabella di conversione Euro -> Lire per i valori 1,5,10,15,20,….100.

ES:

```
Euro: 1 Lire: 1936
Euro: 5 Lire: 9681
Euro: 10  …..
  …….
```

Esercitazione n°1: variante su due file(2)

Come procedere?

1. Togliere dal file **Esempio1.java** la classe **EuroConverter** e copiarla in un nuovo file di nome **EuroConverter.java**

```
public class EuroConverter {

   private double i=0;

   public EuroConverter(double j) { i=j;}
   public void add(double j) { i=i+j;}
   public void sub(double j){ i=i-j;}
   public double currentValue(){return i;}
   public double convert() {return Math.rint(i*1936.27);}
}
```

2. Salvare il file sotto la directory C:\TEMP

Ricorda : Il nome del file deve corrispondere al nome della classe!!!!

Esercitazione n°1: variante su due file(3)

3. ***Compilare…***

```
C:\TEMP> javac EuroConverter.java
```

? *quali e quanti file .class produce? In quale directory?*

4. ***Eseguire…***

```
C:\TEMP> java EuroConverter
```

ha senso? ?

5. ***Documentare…***

```
C:\TEMP> javadoc EuroConverter.java
```

? *Qual è il vantaggio di avere una classe EuroConverter a parte?*

Maggiore RIUTILIZZABILITA', tutti i cambiamenti sulla classe **EuroConverter** sono confinati alla classe **EuroConverter** dentro il file **EuroConverter.java**

Esercitazione n°1: variante su due file(4)

⑥ Scaricare anche file **Esempio2.java** e salvare sempre in **C:\TEMP**

```
public class Esempio2 {

   public static void main(String args[]) {
        EuroConverter uno = new EuroConverter(1.00);

        for (int i=0; i<20; i++){
          System.out.println("Euro: " + uno.currentValue() + " Lire: "
                                                    + uno.convert());
          if (uno.currentValue() == 1) uno.add(4.00);
          else uno.add(5);
        }
   }

}
```

⑦ Notare l'uso delle strutture di controllo. Quindi compilare ed eseguire come per **Esempio1.java** .

Proposte di Esercizio

? Si riesce a compilare ed eseguire anche la classe Esempio1.java modificata (senza la classe innestata EuroConverter)?

Sì ! Provare…

? Come realizzare un Abstract Data Type in C ?