Università degli Studi di Ferrara

DE: Department of Engineering Ferrara

Esercitazioni di

FONDAMENTI DI INFORMATICA MODULO B

ESERCITAZIONE 8 - JAVA 1

Bertagnon Alessandro: alessandro.bertagnon@unife.it
Azzolini Damiano: damiano.azzolini@unife.it

Esercizio su Oggetti Composti

Gli oggetti composti **consentono**:

- di ***aggregare*** componenti complessi a partire da componenti più semplici già disponibili.
- di **mantenere** l'unità del componente, assicurandone la protezione e l'incapsulamento.

Possiamo anche usarli per:

- ***specializzare*** *componenti* già esistenti
- ***aggiungere*** *loro nuovi dati* o *metodi*

Ad esempio, un contatore *con decremento.*

Progettazione Incrementale

Spesso si incontrano problemi che richiedono **componenti simili** *ad altri già disponibili,* **ma non identici.**

Altre volte, **l'evoluzione dei requisiti** comporta una corrispondente **modifica dei componenti**:

- necessità di nuovi dati e/o nuovi comportamenti.
- necessità di modificare il comportamento di metodi già presenti.

Come fare per non dover rifare tutto da capo?

Progettazione Incrementale - Approcci

- Ricopiare manualmente il codice della classe esistente e cambiare quel che va cambiato.

- **Creare un oggetto composto** *(e usare delega)*
  - che incapsuli il componente esistente...
  - … gli "inoltri" le operazioni già previste…
  - … e crei, sopra di esso, le nuove operazioni richieste (eventualmente definendo nuovi dati).
  - **sempre che ciò sia possibile!**

- **Specializzare** (per ereditarietà) la classe Counter (lo vedrete …).

Progettazione Incrementale per Delega

Creare un oggetto composto:

- che incapsuli il componente esistente...
- … gli "inoltri" le operazioni già previste…
- … e crei, sopra di esso, le nuove operazioni richieste (eventualmente definendo nuovi dati).
- sempre che ciò sia possibile.

Progettazione Incrementale per Delega

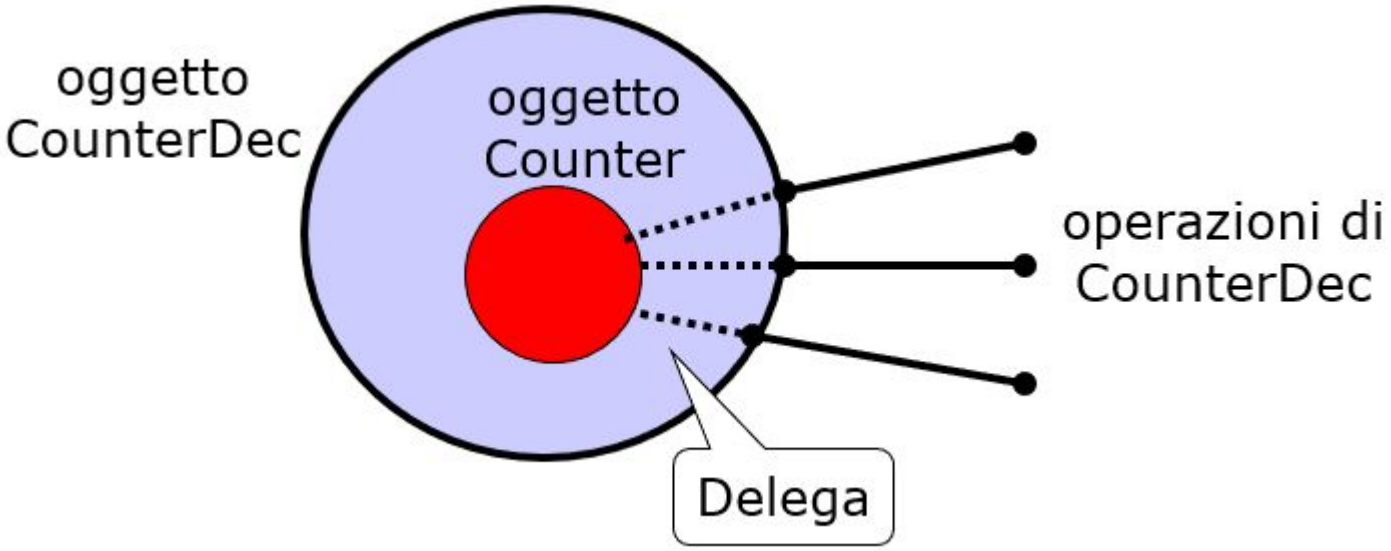
Dal contatore solo in avanti...

```
public class Counter {
    private int val;
    public Counter() { val = 1; }
    public Counter(int v) { val = v; }
    public void reset() { val = 0; }
    public void inc()   { val++; }
    public int getValue() { return val;}
}
```

Progettazione Incrementale per Delega

… al *contatore avanti/indietro (con decremento).*

- Concettualmente, ogni oggetto **CounterDec** **ingloba** un oggetto **Counter** al suo interno.
- Ogni operazione richiesta a **CounterDec** viene ***delegata*** all'oggetto **Counter** interno.



Progettazione Incrementale per Delega

- Aggiungere alla classe **Counter** anche un opportuno metodo **toString** e un metodo **equals.**
- Definire poi l'ADT **CounterDec** che inglobi un oggetto c, istanza di **Counter** al suo interno.
- Ogni istanza di **CounterDec** sarà un oggetto **composto.**
- Nel main definire un oggetto cd, istanza di **CounterDec,** che ne azzeri il valore (reset), lo incrementi due volte, lo stampi a video, lo decrementi e ne ristampi il valore a video.

Progettazione Incrementale per Delega

… al contatore avanti/indietro (con decremento):

```
public class CounterDec {
    private Counter c;
    public CounterDec() { c = new Counter(); }
    public CounterDec(int v){ c = new Counter(v); }
    public void reset() { c.reset(); }
    public void inc()   { c.inc(); }
    public int getValue() { return c.getValue(); }
    public void dec() { ... }
}
```

Delega

Progettazione Incrementale per Delega

Il metodo `dec()` deve:

- recuperare il valore attuale V del contatore.
- resettare il contatore.
- riportarlo, tramite incrementi, al valore V’ = V-1.

Progettazione Incrementale per Delega

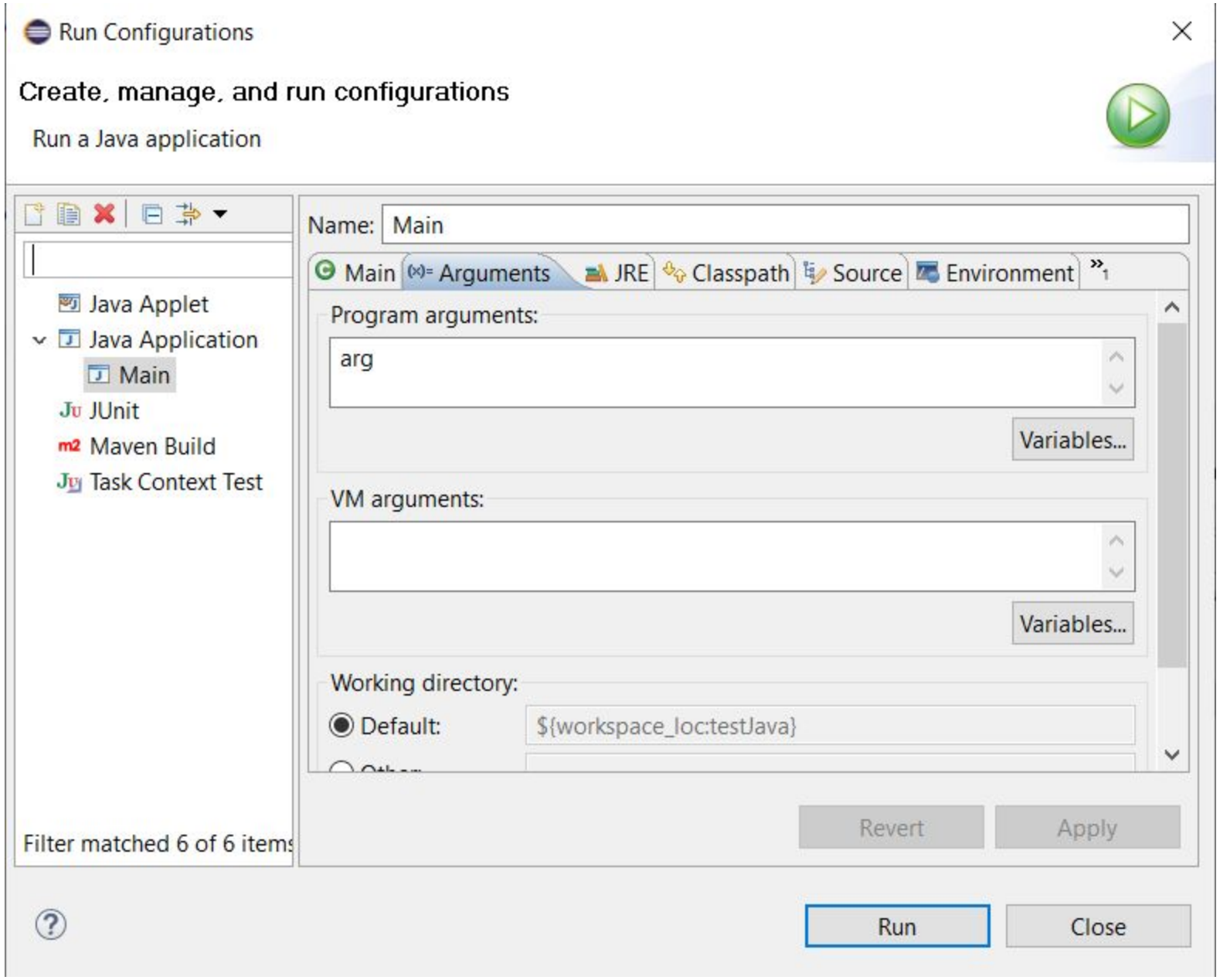
- Poiché i campi privati non sono accessibili, bisogna riscrivere anche tutti i metodi che concettualmente rimangono uguali, procedendo per delega (delegation).

- Non è detto che le operazioni già disponibili consentano di ottenere qualsiasi nuova funzionalità si renda necessaria (potrebbe essere necessario accedere ai dati privati).

- Occorre poter riusare le classi esistenti in modo più flessibile.

Lettura di Argomenti da Linea di Comando

- Per leggere gli argomenti da linea di comando, per eclipse, selezionare il tab run -> run configurations.
- Selezionare poi java application -> classe che contiene il metodo main -> Arguments (vedere slide successiva per dettagli).
- NB: l'opzione è disponibile solamente dopo aver lanciato il programma una volta.


Run Configurations
Create, manage, and run configurations
Run a Java application
Java Applet
Java Application
Main
JUnit
Maven Build
Task Context Test
Filter matched 6 of 6 items
Name: Main
Main
Arguments
JRE
Classpath
Source
Environment
Program arguments:
arg
Variables...
VM arguments:
Variables...
Working directory:
Default:
${workspace_loc:testJava}
Revert
Apply
Run
Close

Esercizio su Stringhe

- Si leggano N stringhe da riga di comando.
- Per ogni stringa si sostituiscano tutte le occorrenze di `old_char` con `new_char` (variabili inizializzate nel codice).
  - Utilizzare la classe `StringBuffer` (e relativi metodi)
    - Stampare l'ID dell'istanza ad ogni modifica dell'oggetto.
  - Utilizzare la classe `String` (e relativi metodi)
    - Stampare l'ID dell'istanza ad ogni modifica della stringa.

Esercizio su Stringhe

Domande:

- Come si stampa l'ID di un'istanza?
  - Metodo `hashcode()`
  - https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode()
- Quali sono i metodi necessari per sostituire un carattere in un'istanza della classe:
  - `StringBuffer`
  - `String`

Esercizio su Stringhe

Domande:

- Data una stringa `s` (definita all'interno del programma), creare una nuova stringa `reverse_s` che rappresenti la stringa `s` invertita e stampare il contenuto a video.

- Utilizzare opportunamente `String` e `Stringbuffer`.
  - `StringBuffer`
  - `String`

```
s = "CIAO"
reverse_s = "OIAC"
```