Università degli Studi di Ferrara

DE: Department of Engineering Ferrara

Esercitazioni di

FONDAMENTI DI INFORMATICA MODULO B

ESERCITAZIONE 9 - JAVA 2

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

Esercizi di modifica incrementale in Java

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**:

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

Come fare per non dover rifare tutto da capo?

- 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

Esercizio ereditarietà

Si crei la seguente classe **Veicolo**:

```
private double velocita;
private double accelerazione;

public Veicolo() { ... }
public Veicolo(double velocita, double accelerazione) { ... }

// GETTERS

public double getVelocita() { ... }
public double getAccelerazione() { ... }
```

Esercizio ereditarietà

```
// SETTERS

public void setVelocita(double velocita) { ... }
public void setAccelerazione(double accelerazione) { ... }

// prende in input un oggetto di classe VEICOLO e ne
// stampa la velocità

static public void printVelocita(Veicolo v){ ... }

// stampa il numero di ruote del veicolo

public void printNRuote() {
    System.out.println("Numero di ruote: " + ...);
}
```

Esercizio ereditarietà

Si crei la seguente classe **Automobile** che estende la classe **Veicolo**:

```
private String targa;
private boolean avviata;

public Automobile(String targa){ ... }

// "accende" e "spegne" la macchina, agendo sul
// boolean avviata

public void avvia() { ... }
public void spegni() { ... }
```

Esercizio ereditarietà

```
// accelera controlla se la macchina è avviata, nel
// qual caso calcola i nuovi valori
// di accelerazione e velocità (vel. Precedente + acc*sec)

public void accelera(double accelerazione, int secondi) { ... }

// ridefinire il metodo printNRuote() e stampare
// il numero di ruote del veicolo => 4

public void printNRuote() { … }
```

Esercizio ereditarietà

Si crei la seguente classe **Bicicletta** che estende la classe **Veicolo**

```
// nel costruttore, impostare velocità e accelerazione a 0

public Bicicletta() { ... }

// pedala imposta i nuovi valori nel seguente
// modo:
// - accelerazione (num. Pedalate / sec ^2)
// - velocità (vel. Precedente + acc*sec)

public void pedala(int numeroPedalate, int secondi) { ... }

// ridefinire il metodo printNRuote() e stampare il numero di ruote del
// veicolo => 2

public void printNRuote() { … }
```

Esercizio ereditarietà

Creare una classe **VeicoloMain** contenente il metodo **main**.

- Nel metodo main definire un oggetto auto istanza di **Automobile** e un oggetto bici istanza di **Bicicletta**.
- Richiamare i metodi pedala e accelera rispettivamente per bici e auto.
- Stampare la velocità dei due veicoli e il loro numero di ruote, usando i metodi **printVelocita**(Veicolo v) e **printNRuote**().

Esercizio sugli oggetti compositi

Dalla precedente esercitazione:

Definire la classe **CounterDec** che inglobi un oggetto c istanza di **Counter** al suo interno: ogni istanza di CounterDec sarà un **oggetto composto**.

La classe CouterDec deve definire **due costruttori** (come la classe Counter), i metodi **inc()**, **reset()**, **getCount()**, **toString()** e **equals()** delegando le operazioni all'oggetto Counter e aggiungere il metodo **dec()** che decrementa il conteggio di uno. Il metodo dec() non può accedere direttamente allo stato del contatore, deve quindi servirsi dei metodi esposti dalla classe Counter.

Creare una classe **CounterDecMain** che espone il metodo **main()**, dove definisce un oggetto cd istanza di CounterDec, ne azzera il valore, lo incrementa due volte, lo stampa a video, lo decrementa e ne stampa nuovamente il valore a video.

Esercizio sugli oggetti compositi

- 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.

Esercizio sull'ereditarietà

Si definisca la classe **ImprovedCounter**, sottoclasse (estensione) di Counter. ImprovedCounter ha **in più** un attributo intero di nome **step**.

Ai due costruttori di Counter (che vanno modificati con l'aggiunta dell'inizializzazione a 1 della variabile step) aggiungere un terzo costruttore, il quale prende in ingresso due interi che andranno a costituire i valori iniziali di count e step.

ImprovedCounter **ridefinisce inc()** in modo che, ad ogni invocazione, sommi a count il valore di step (invece che 1) e offre, inoltre, una **variante di inc(int n)** che, preso in ingresso un intero, lo somma a count.

Infine espone i metodi **getStep()**, che restituisce in uscita il valore di step e **setStep(int n)** che, preso in ingresso un intero, pone a questo valore step.

Esercizio sull'ereditarietà

Definire una classe **ImprovedCounterMain** contenente un metodo **main()** che dichiari due ImprovedCounter iC1 e iC2.

Si inizializzi iC1.count a 0, iC1.step a 1, iC2.count a un valore diverso da 0, iC2.step a un valore diverso da 1.

Realizzare poi un ciclo con indice i da i a N con 5 < N < 20 scelto in maniera casuale. Per generare un numero random, utilizzare il metodo statico **Math.random()** che restituisce un valore double compreso tra 0.0 e 1.0 (per maggiori informazioni cliccare su Math.random()).

Come generare un intero N compreso tra due valori partendo da un numero compreso tra 0 e 1?

All'interno del ciclo, se i è **pari**, invocare **inc()** su iC1 e iC2. Altrimenti, se i è **dispari**, invocare **inc(i)** su iC1 e iC2.

Esercizio sull'ereditarietà

Si stampi infine un messaggio che dica se iC1 e iC2 hanno lo stesso valore di conteggio o no, e il valore di conteggio e di step di entrambi.

N.B. è necessario definire i metodi **equals** e **toString** per le classi **Counter** e **ImprovedCounter**.

Il metodo equals deve essere veramente ridefinito per la classe **ImprovedCounter**?

Esercizio sull'ereditarietà

Affinché una classe sia efficacemente estendibile usando l'ereditarietà i suoi attributi e metodi non possono essere dichiarati **private**, ma è necessario usare la parola chiave **protected** per consentire l'accesso anche ai metodi delle classi derivate.

In alternativa le classi base devono fornire metodi di accesso agli attributi che dichiarano private.