Informazioni generali

Esercitazioni del corso di Fondamenti di informatica

Tutor: Ing. Rughetti Diego

Esercitazione: Giovedì 14.30-16.00

Tutoraggio: Giovedì 16.15-17.45

Contatto: rughettidiego@tiscali.it

Ringraziamenti

- Prof. Giuseppe de Giacomo, Dipartimento di Informatica e Sistemistica, Università di Roma “La Sapienza”

- Prof. Enrico Denti, Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna

Esercizio 1 – Telefono cellulare

Un telefono cellulare permette l'immissione di testo in due circostanze:

1. composizione di SMS,
2. immissione nuovo nome/numero in rubrica.

Esistono delle regole generali che disciplinano l'immissione del testo, ad es:

- come si cambia fra maiuscole e minuscole,
- come si attiva/disattiva la modalità T9.

Dove mettereste queste regole nel manuale utente?

A. nella sezione sugli SMS, in quanto vengono inviati spesso,
B. nella sezione sulla rubrica, in quanto viene prima nel manuale,
C. in una sezione a parte.

Circostanziare la risposta tenendo conto dei principi di buona modularizzazione.

Esercizio 1 – Soluzione

La risposta giusta è C. Per comprendere ciò, dobbiamo considerare le sezioni del manuale come moduli e gli argomenti trattati come servizi offerti. La persona che vuole imparare le regole è assimilabile ad un modulo cliente. Le soluzioni A e B prefigurano due moduli server, dei quali almeno uno (quello in cui poniamo le regole sull'immissione del testo) con bassa coesione. Di conseguenza ci sarebbe un alto accoppiamento fra le sezioni sugli SMS e sulla rubrica: nel caso A, ad esempio, una persona che volesse imparare ad usare la rubrica telefonica e fosse interessata ad una particolare combinazione di maiuscole/minuscole (ad es., "Alessandro Rossi ufficio") dovrebbe necessariamente leggere la sezione sugli SMS. La soluzione C garantisce anche maggiore estendibilità: se in future versioni del manuale utente saranno previste altre regole, saranno illustrate nella sezione apposita. Sull'information hiding non possiamo fare alcuna osservazione, mentre un alto interfacciamento esplicito potrebbe essere garantito dall'indice degli argomenti.

Esercizio 2 – Ufficio postale

Si consideri un ufficio postale, e lo si descriva in termini di un insieme di moduli. Ogni modulo deve essere caratterizzato da:

- i servizi che offre agli utenti,
- il modo in cui gli utenti possono usufruirne,
- la sua struttura interna,
- il suo rapporto con gli altri moduli.

Esercizio 2 – Soluzione

Modulo A: Servizi postali

Servizi offerti agli utenti:

- S-A1: ritiro corrispondenza/telegrammi/pacchi in giacenza
- S-A2: spedizione pacchi, telegrammi e corrispondenza raccomandata/assicurata
- S-A3: acquisto francobolli

Come gli utenti possono usufruirne:

- S-A1: presentazione avviso e documento del destinatario, oppure di delega, documento del destinatario e documento proprio, contante (per pagamento diritti di giacenza)
- S-A2: presentazione busta/collo da spedire, moduli per spedizione compilati, contante
- S-A3: presentazione contante

Struttura interna:

- n (≥ 1) sportelli con operatore abilitato a tutti i servizi e dotato di terminale, deposito per la modulistica necessaria, deposito contante e deposito moduli compilati
- clienti in fila unica (disciplina FIFO), che accedono al primo sportello libero
- depositi corrispondenza in giacenza, condivisi fra gli operatori

Esercizio 2 – Soluzione (2)

Modulo B: servizi finanziari

Servizi offerti agli utenti:

- S-B1: pagamento bollettino/invio vaglia
- S-B2: ricossione vaglia
- S-B3: apertura conto corrente o libretto postale
- S-B4: versamento su conto corrente o libretto postale

Come gli utenti possono usufruirne:

- S-B1: presentazione contante/bancomat e moduli compilati
- S-B2: presentazione vaglia e documento del beneficiario, oppure di delega, documento del beneficiario e documento proprio
- S-B3: presentazione documento e codice fiscale dell'intestatario (di entrambi gli intestatari se cointestato) e dei moduli compilati
- S-B4: presentazione contante/assegni e documento dell'intestatario, oppure di delega, documento dell'intestatario e documento proprio, e dei moduli compilati

Struttura interna:

- n (≥ 1) sportelli con operatore abilitato a tutti i servizi e dotato di terminale, deposito per la modulistica necessaria, deposito contante e assegni e deposito moduli compilati
- clienti in fila unica (disciplina FIFO), che accedono al primo sportello libero

Esercizio 2 – Soluzione (3)

Modulo C: Gestione investimenti

Servizi offerti agli utenti:

- S-C1: acquisto titoli (ad es., buoni postali) e fondi di investimento

Come gli utenti possono usufruirne:

- S-C1: colloquio con funzionario, seguito da compilazione moduli, seguito da verifica disponibilità finanziaria

Struttura interna:

- 1 sportello con funzionario abilitato a tutti i servizi e dotato di deposito per la modulistica necessaria e deposito moduli compilati
- clienti in fila unica (disciplina FIFO), che accedono allo sportello quando libero

Modulo D: Vendita cancelleria al pubblico

Servizi offerti agli utenti:

- S-D1: acquisto cancelleria

Come gli utenti possono usufruirne:

- S-D1: presentazione contante e richiesta beni desiderati

Struttura interna:

- 1 sportello con operatore abilitato a tutti i servizi e dotato di deposito cancelleria e deposito contante
- clienti in fila unica (disciplina FIFO), che accedono allo sportello quando libero

Esercizio 2 – Soluzione (4)

Modulo E: Logistica

Servizi offerti agli utenti:

- S-E1: Disposizione modulistica negli appositi espositori

Come gli utenti possono usufruirne:

- S-E1: Ritiro moduli da espositori

Servizi agli altri moduli:

- S-E2: Deposito corrispondenza in arrivo (al modulo A)
- S-E3: Ritiro corrispondenza in giacenza e non ritirata entro i termini di legge (al modulo A)
- S-E4: Distribuzione modulistica (ai moduli A-D)
- S-E5: Ritito contante (ai moduli A, B, D) e assegni (al modulo B)
- S-E6: Ritito moduli compilati (ai moduli A-C)

Esercizio 2 – Soluzione (5)

Modulo F: Direzione

Servizi offerti agli utenti:

- Nessun servizio al pubblico

Servizi agli altri moduli:

- Chiarimenti su attuazione norme

NOTA: Per erogare il servizio S-C1, il funzionario deve verificare la disponibilità finanziaria, ma non è dotato di terminale. Si dovrà rivolgere ad un operatore del modulo B (accoppiamento). Si può migliorare la modularizzazione dando al funzionario un terminale.

Esercizio 3 – Funzione java per la ricerca di un elemento in una lista

Consideriamo una funzione booleana per la ricerca di un elemento in un array. La funzione è stata progettata in modo che il modulo chiamante debba porre a 0, prima della invocazione, una variabile globale (ovvero *static*) beta. Analizzare il grado di coesione, di information hiding, di accoppiamento e di interfacciamento della funzione

Esercizio 3 – Soluzione

Nel caso in cui si eliminasse l'uso della variabile globale beta, l'interfacciamento esplicito della funzione aumenterebbe, e il grado di accoppiamento con gli altri moduli diminuirebbe

Esercizio 4 – Funzione java per il calcolo del fattoriale

Consideriamo la seguente funzione Java per il calcolo del fattoriale di un numero:

```
public class Server {
        public static int Fatt(int n) {
        int res=1;
        while (n!=0) {
                res=res*n;
                n--;
                }
        return res;
        }
}
```

Cosa succede se la funzione viene invocata con un valore negativo in ingresso? Cosa si può dire sul grado di coesione e di accoppiamento della funzione Fatt() rispetto ai moduli client? Come si possono migliorare le qualità del modulo?

Esercizio 4 – Soluzione

La funzione entra in un ciclo infinito se viene invocata con un valore negativo per il parametro attuale. Il grado di accoppiamento con altri moduli è alto, proprio perché i moduli clienti, per evitare il rischio del ciclo infinito, sono costretti ad eseguire un test prima di invocare la funzione `Fatt()`, come mostra l'esempio seguente.

```
// File Fattoriale/Cattiva/Client.java

public class Client {
  public static void main(String[] args) {
      System.out.print("Inserisci un intero: ");
      int my_int = InOut.readInt();
      if (my_int >= 0)  // ALTO ACCOPPIAMENTO: il cliente deve
                        // controllare che l'intero non sia negativo
          System.out.print("Il fattoriale di " + my_int + " vale " +
                           Server.Fatt(my_int) + "\n");
  }
}
```

Il grado di coesione della funzione aumenterebbe se la verifica del valore non negativo del parametro fosse eseguita dalla funzione stessa. D'altra parte, inserire la verifica direttamente nella funzione `Fatt()` compromette l'efficienza della funzione, perché ogni chiamata ricorsiva effettuerebbe un test inutile. Ne segue che la qualità del modulo aumenta se la funzione ricorsiva viene nascosta all'esterno rendendola privata, e la funzione che effettua il servizio per i moduli clienti esegue il test e chiama la funzione ricorsiva solo in condizioni corrette:

Esercizio 4 – Soluzione (2)

```
public class Server {

        public static int Fatt(int n) {
        // restituisce -1 se n <0
        if (n<0) return -1;
        int res=1;
        while (n!=0) {
                res=res*n;
                n--;
                }
        return res;
        }
}
```

Proposte di esercizi

Venditori di computer

La ditta HAL vende PC laptop solamente con il sistema operativo Doors preinstallato. La ditta Tony vende PC laptop senza sistemi operativi preinstallati, oppure con Doors preinstallato, oppure con Gynux preinstallato. Analizzare lo scenario dal punto di vista della modularizzazione.

Elenchi telefonici

Spiegare in termini di modularizzazione perché, in generale, è scomodo non possedere l'elenco telefonico cartaceo, ma solo su CD-ROM. In quali condizioni può invece essere vantaggioso?

Dispositivo segnalatore per autobus

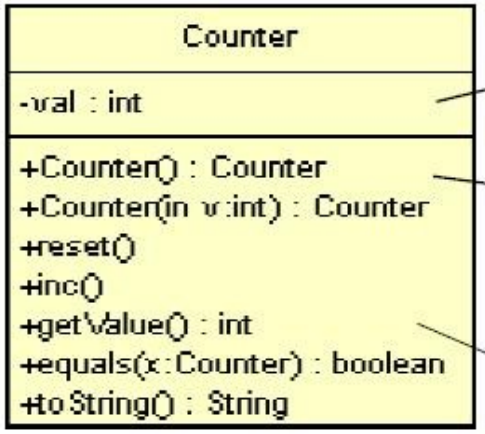
Un sistema per la visualizzazione delle informazioni su un autobus (ad es., numero linea e itinerario) prevede tre display luminosi (davanti, dietro, sul lato) e un'interfaccia per l'autista. Perché è conveniente che l'autista inserisca solo una volta le informazioni, e che il sistema informatico provveda a visualizzarle sui tre display?

Saggezza popolare

Un antico proverbio dell'Italia rurale recita "Il mestolo vicino al secchio, la legna vicino al caminetto". È possibile rifrasarlo in termini di princìpi di buona modularizzazione?

Rappresentazione di una classe

Una classe è rappresentata mediante un disegno come il seguente:



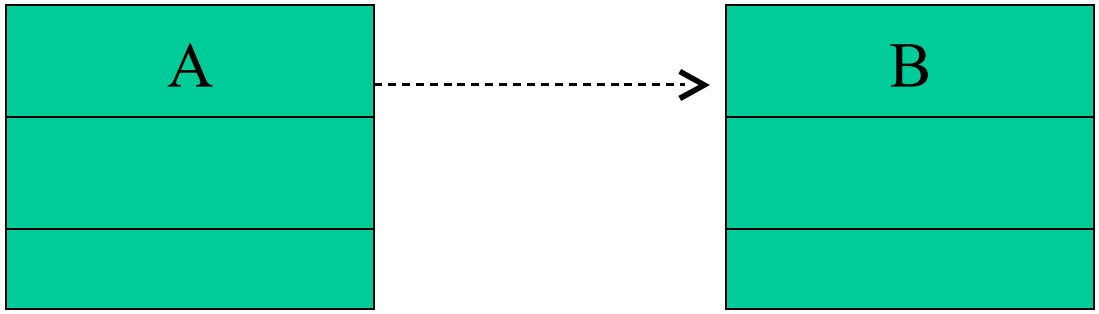
campi dati:
"–" se privati, "+" se pubblici,
 # se package o protetti

costruttori e metodi:
"–" se privati, "+" se pubblici,
 # se package o protetti

Eventuali dati o metodi statici sono tipicamente sottolineati

“Utilizzo”: Relazione client-server tra classi

Si dice che una classe A “utilizza” la classe B quando A sfrutta i servizi messi a disposizione da B per svolgere il proprio lavoro. In questo caso tra A e B si instaura una relazione di tipo client – server con A che ricopre il ruolo di client e B che ricopre il ruolo di server. La relazione di utilizzo viene indicata come mostrato di seguito:

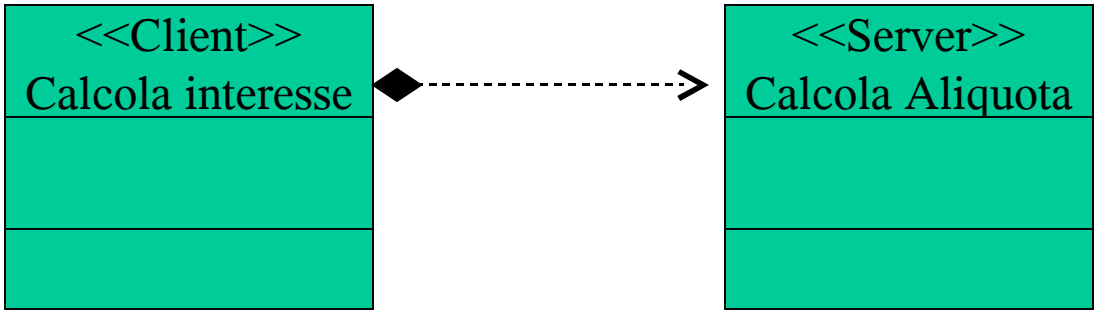


Esempio di relazione “utilizzo”

Voglio realizzare una classe X che calcola gli interessi sull’ammontare di un capitale che la classe riceve in input. La percentuale dell’interesse non è costante ma varia in funzione dell’ammontare del capitale di ingresso e del periodo di investimento del capitale. Anche tale periodo è fornito in input alla classe. Quest’ultima non è a conoscenza delle varie aliquote di interesse e per acquisirle le richiede ad un’altra classe Y, che è in grado di determinare la percentuale di interesse da applicare sul capitale da investire.

Definire le due classi, determinare la relazione di utilizzo specificando i ruoli (client o server) delle classi, indicare la responsabilità della relazione, fornire una bozza del codice delle classi delineando solo le porzioni principali di codice

Soluzione esempio – Schema delle classi



Soluzione esempio (2) – Bozza codice

E' la classe Client CalcolaInteresse che mantiene il riferimento alla classe Server CalcolaAliquota

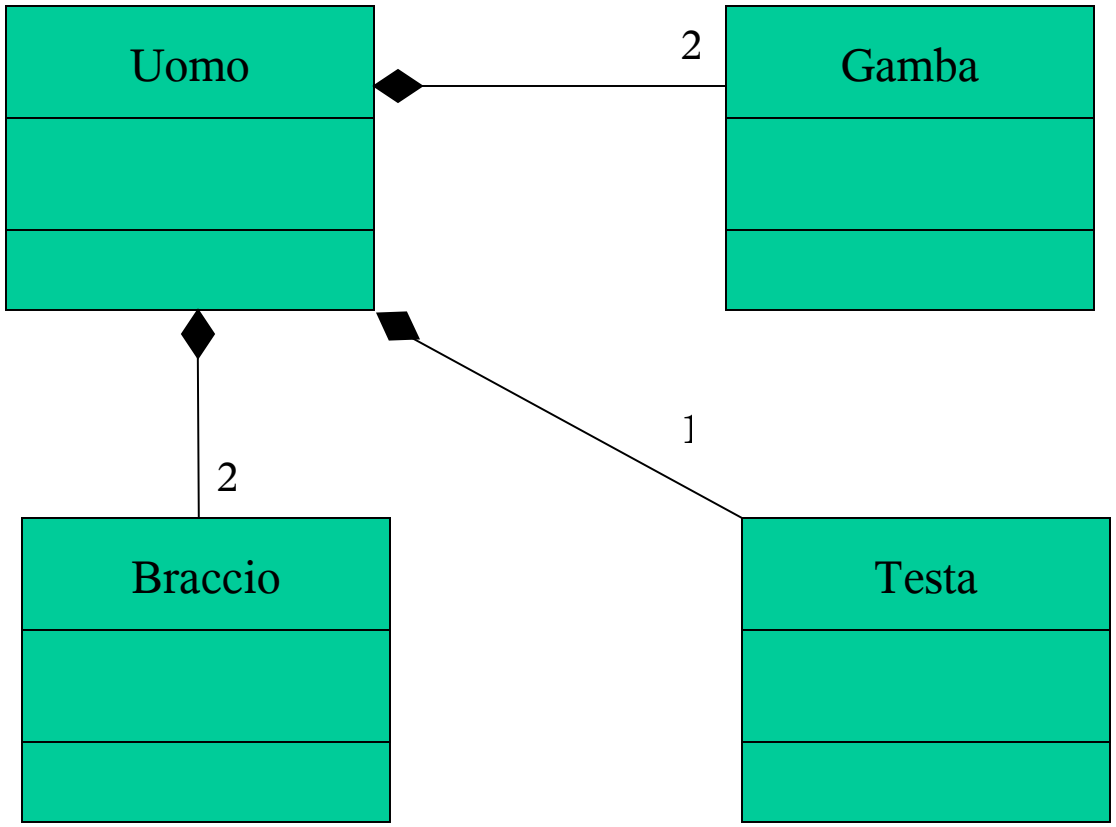
```
public class CalcolaInteresse{
    .....
    private CalcolaAliquota ca;

    public CalcolaInteresse(){
        ........
        this.ca = new CalcolaAliquota();
    }

    public int getAmmontareInteresse(int capitale, int giorni){
        .......
        int i = this.ca.getAliquota(capitale, giorni);
        .......
    }
}

public class CalcolaAliquota{

    public CalcolaAliquota(){
        .......
    }

    public int getAliquota(int cap, int gior){
        ........
    }
}
```

Esempio - Composizione

Ogni Uomo ha una testa, due gambe e due braccia

“Uomo” è un oggetto complesso, caratterizzato dalla presenza di 2 oggetti “gamba”, due oggetti “braccio” e da un oggetto “testa”.

Composizione - rappresentazione



Bozza codice

```
public class Uomo{

    private Gamba[] gambe;
    private Braccio[] braccia;
    private Testa t;

    public Uomo(){
        ......
        this.t = new Testa();
        this.gambe = new Gamba[2];
        this.braccia = new Braccio[2];
        this.gambe[0] = new Gamba();
        this.gambe[1] = new Gamba();
        this.braccia[0] = new Braccio();
        this.braccia[1] = new Braccio();
    }

}
public class Testa{

    .......

    public Testa(){
        ......
    }

    ........

}
```

Array (dimensione non variabile una volta impostata)

Nel costruttore creo e popolo due array di due elementi “. In questo modo realizzo il vincolo sulla cardinalità della relazione di composizione tra le classi Uomo e Gamba e tra Uomo e Braccio

Bozza codice (2)

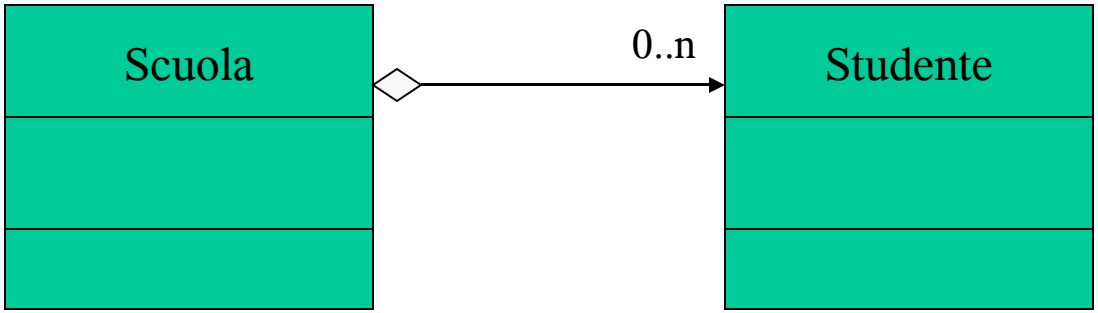
```
public class Braccio{

      .......

     public Braccio(){
          ......
     }

      ........

}
public class Gamba{

      .......

     public Gamba(){
          ......
     }

      ........

}
```

Esempio - Aggregazione

Ogni scuola ha degli studenti

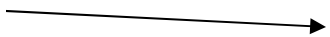
“Scuola” è un oggetto complesso, formato dall’aggregazione di parti, ovvero oggetti “studenti”. Nessuno oggetto studente preso singolarmente è però essenziale per l’esistenza o per il funzionamento dell’oggetto complesso scuola

Aggregazione - rappresentazione



Bozza codice

```
public class Scuola{
    .....
    private Vector studenti;

    public Scuola(){
        ........
        this.studenti = new Vector();
    }

    public void setStudente(Studente s){
        .......
        this.studenti.add(s)
        .......
    }

    public Studente getStudente(.....){
        .....
        return (Studente) this.studenti.get(....);
    }
}
public class Studente{

    .......

    public Studente(....){
        ......
    }

    ........

}
```



N.B.

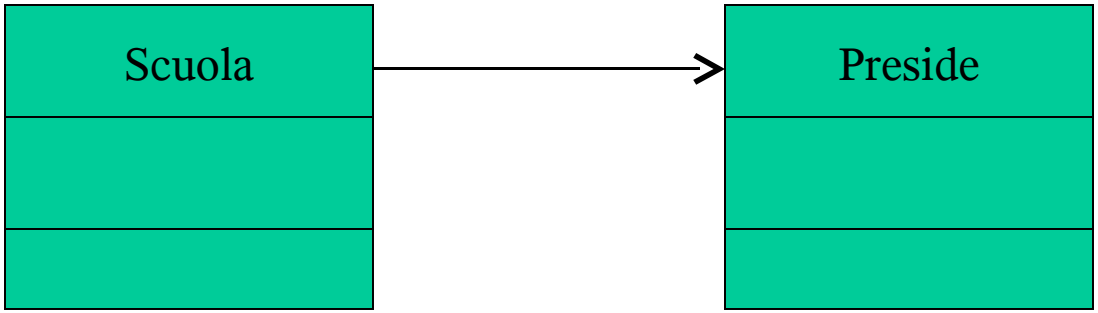
A differenza della relazione di composizione in questo caso non ho vincoli di cardinalità sulla relazione tra le classi. Ciò si riflette sul codice

Esempio - Associazione

Ogni scuola ha un preside.

Un oggetto “scuola” non è un aggregato di oggetti “preside”, inoltre non è neanche composto da oggetti “preside”. La relazione che si ha in questo quindi non è né un’Aggregazione, né una composizione. Ci troviamo nel caso di una relazione di ASSOCIAZIONE

Associazione - rappresentazione



Bozza codice

```
public class Scuola{

    private Preside p;

    public Scuola(){
        .......
    }

    public Scuola(Preside pr){
        this.p = pr;
    }

    public Preside getPreside(){
        return this.p;
    }

    public void setPreside(Preside pr){
        this.p = pr;
    }

}

public class Preside{

    .......

    public Preside(){
        ......
    }

    ........

}
```

Esercizio

Stabilire se si tratta di Composizione, Aggregazione o associazione:

- Una linea ha dei punti
- Una biblioteca ha dei libri
- Un portafoglio di azioni ha dei titoli
- Un battaglione ha un capitano
- Aggiungerne altri