

Fondamenti di Architettura e Programmazione

13 febbraio 2006

Cognome **Babbi** Nome **Babbo**

Matricola 669966

- Usate il retro dei fogli per scrivere lo svolgimento degli esercizi di programmazione.
 - Quando vi è richiesto di scrivere un programma, potete limitarvi al *corpo* del metodo main, assumendo se necessario che in e out siano due variabili di classe ConsoleInputManager e ConsoleOutputManager (rispettivamente), già dichiarate e inizializzate.
1. Nei seguenti esercizi, è essenziale riportare l'intero procedimento di soluzione.
 - (a) Eseguire la sottrazione $14 - 35$ rappresentando i numeri in *complemento a 2* su 8 bit. Mostrare che il risultato rappresenta effettivamente -21 .
 - (b) Qual è la rappresentazione di -15 in *modulo e segno* su 10 bit?
 - (c) La rappresentazione di 207 in base x è 543. Quanto vale x ?
 - (d) Rappresentare in binario il numero esadecimale ABC.

(Svolgimento sul retro)

(a)

14	00001110
-35	11011101
-21	11101011

(b) 1000001111 (essendo $15_{10} = 1111_2$)

(c) $5x^2 + 4x + 3 = 207$, cioè $5x^2 + 4x - 204 = 0$, che ha come sola soluzione positiva $x = 5$

(d) $10 * 16^2 + 11 * 16 + 12 = 2748$

2. Per ognuna delle seguenti espressioni booleane, fornite la tabella di verità, evidenziando eventuali *tautologie e contraddizioni*.

(a) $x \vee (\neg x \vee y \vee z)$,

(b) $(x \vee y) \wedge (\neg x \vee y)$,

(c) $\neg(x \wedge \neg y) \wedge (\neg x \vee y)$.

3. Dimostrare la seguente equivalenza logica sia mediante tabelle di verità sia mediante semplificazioni algebriche:

$$\neg((x \wedge y) \vee \neg(x \vee \neg y)) = \neg y$$

(Svolgimento sul retro)

2.

x	y	z	$\neg x$	$\neg x \vee y \vee z$	$x \vee (\neg x \vee y \vee z)$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	0	1	1

(a) è una tautologia

x	y	$x \vee y$	$\neg x$	$\neg x \vee y$	$(x \vee y) \wedge (\neg x \vee y)$
0	0	0	1	1	0
0	1	1	1	1	1
1	0	1	0	0	0
1	1	1	0	1	1

(b)

x	y	$\neg x$	$\neg y$	$x \wedge \neg y$	$\neg x \vee y$	$\neg(x \wedge \neg y)$	$\neg(x \wedge \neg y) \wedge (\neg x \vee y)$
0	0	1	1	0	1	1	1
0	1	1	0	0	1	1	1
1	0	0	1	1	0	0	0
1	1	0	0	0	1	1	1

(c)

3.

x	y	$\neg y$	$x \vee \neg y$	$\neg(x \vee \neg y)$	$x \wedge y$	$(x \wedge y) \vee \neg(x \vee \neg y)$	$\neg((x \wedge y) \vee \neg(x \vee \neg y))$	$\neg y$
0	0	1	1	0	0	0	1	1
0	1	0	0	1	0	1	0	0
1	0	1	1	0	0	0	1	1
1	1	0	1	0	1	1	0	0

$$\begin{aligned} \neg((x \wedge y) \vee \neg(x \vee \neg y)) &= \text{(de Morgan)} \\ \neg(x \wedge y) \wedge \neg\neg(x \vee \neg y) &= \text{(doppia negazione)} \\ \neg(x \wedge y) \wedge (x \vee \neg y) &= \text{(de Morgan)} \\ (\neg x \vee \neg y) \wedge (x \vee \neg y) &= \text{(distributiva)} \\ (\neg x \wedge x) \vee (\neg x \wedge \neg y) \vee (\neg y \wedge x) \vee (\neg y \wedge \neg y) &= \text{(idempotenza)} \\ (\neg x \wedge x) \vee (\neg x \wedge \neg y) \vee (\neg y \wedge x) \vee \neg y &= \text{(essendo } \neg x \wedge x = 0) \\ 0 \vee (\neg x \wedge \neg y) \vee (\neg y \wedge x) \vee \neg y &= \text{(el. neutro)} \\ (\neg x \wedge \neg y) \vee (\neg y \wedge x) \vee \neg y &= \text{(distributiva)} \\ (\neg y \wedge (x \vee \neg x)) \vee \neg y &= \text{(essendo } x \vee \neg x = 1) \\ (\neg y \wedge 1) \vee \neg y &= \text{(el. neutro)} \\ \neg y \vee \neg y &= \text{(idempotenza)} \\ \neg y & \end{aligned}$$

4. Scrivete un (frammento di) programma Java che operi come segue:

- legga un intero n da tastiera;
- legga n stringhe da tastiera, memorizzandole in un array A di dimensione n ;
- stampi le stringhe in A che terminano con lettere maiuscole (ricordiamo il metodo statico `isUpperCase(char c)` della classe `Character`).

Ecco un esempio di esecuzione (le parti in grassetto sono state inserite dall'utente):

```
Quante stringhe? 4
```

```
Stringa 0: Nel mezzo del cammin di nostra vita
```

```
Stringa 1: mi ritrovai per una selva oscurA
```

```
Stringa 2: ché la diritta via era smarritA
```

```
Stringa 3: Ah quanto a dir qual era è cosa dura
```

```
mi ritrovai per una selva oscurA
```

```
ché la diritta via era smarritA
```

(Svolgimento sul retro)

```
int n = in.readInt( "Quante stringhe?" );
String a[] = new String[ n ];
for ( int i = 0; i < n; i++ )
    a[ i ] = in.readLine( "Stringa" + i + ":" );
for ( int i = 0; i < n; i++ )
    if ( Character.isUpperCase( a[ i ].charAt( a[ i ].length() - 1 ) ) )
        out.println( a[ i ] );
```

5. Considerate la classe `Frazione` i cui oggetti rappresentano frazioni; questa classe ha un costruttore pubblico

```
public Frazione( int n, int d )
```

che costruisce una frazione di numeratore `n` e denominatore `d`. La classe ha i metodi pubblici

```
public double getNumeratore()
public double getDenominatore()
```

che restituiscono rispettivamente il numeratore ed il denominatore della frazione su cui sono attivati.

Scrivete un programma che operi come segue:

- (a) chieda all'utente di inserire un numero intero, diciamo N ;
- (b) dichiari un array di N frazioni e lo riempi con frazioni chieste all'utente;
- (c) stampi tutte le frazioni il cui numeratore superi il quadrato del denominatore (per la stampa, ricordiamo il metodo `toString()` della classe `Frazione`).

Ecco un esempio di esecuzione (le parti in grassetto sono state inserite dall'utente):

```
Inserisci un intero: 4
```

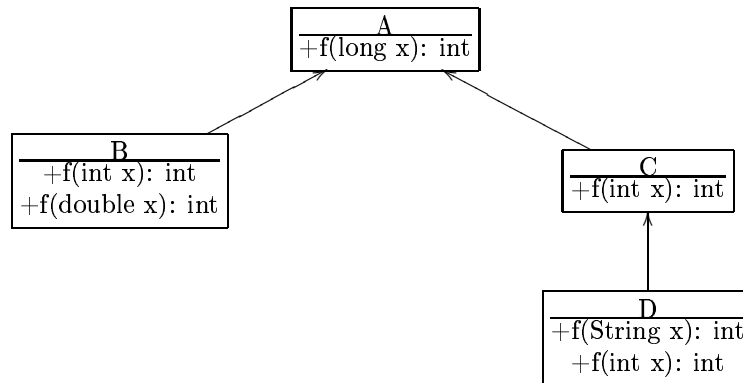
```
Frazione #0, Numeratore = 1
Frazione #0, Denominatore = 2
Frazione #1, Numeratore = 35
Frazione #1, Denominatore = 4
Frazione #2, Numeratore = 44
Frazione #2, Denominatore = 20
Frazione #3, Numeratore = 15
Frazione #3, Denominatore = 3
```

```
35/4
15/3
```

(Svolgimento sul retro)

```
int n = in.readInt( "Inserisci un intero: " );
Frazione a[] = new Frazione[ n ];
for ( int i = 0; i < n; i++ ) {
    int num = in.readInt( "Frazione #" + i + ", Numeratore=" );
    int den = in.readInt( "Frazione #" + i + ", Denominatore=" );
    a[ i ] = new Frazione( num, den );
}
for ( int i = 0; i < n; i++ )
    if ( a[ i ].getNumeratore() > a[ i ].getDenominatore() * a[ i ].getDenominatore() )
        out.println( a[ i ] );
```


6. Considerate la seguente gerarchia di classi (rappresentata mediante un diagramma UML):



e ipotizzate che tutte le classi abbiano un costruttore pubblico senza argomenti. Assumete le seguenti definizioni e inizializzazioni:

```
A a1, a2;
B b;
C c1, c2;
D d;
```

```
a1 = new A(); a2 = new B();
b = new B();
c1 = new C(); c2 = new D();
d = new D();
```

Per ciascuna delle seguenti invocazioni di metodo, dire se essa è consentita e in caso affermativo indicare il nome della classe che contiene il metodo che verrà effettivamente eseguito:

- `a1.f(3+"5")`: **Non**
- `b.f(3+3L)`: **A:f(long)**
- `c1.f(3L)`: **A:f(long)**
- `d.f(3.0)`: **Non consentita**
- `a2.f(3+"ciao")`: **Non consentita**
- `c2.f(3+"ciao")`: **Non consentita**

7. Scrivete il codice di una classe di nome `AtletaOlimpico` le cui istanze rappresentano atleti che partecipano alle olimpiadi. Ogni atleta è caratterizzato dal nome, da una *nazione* di appartenenza e da una *specialità*: questi ultimi due dati sono espressi mediante due istanze di due opportune classi, che dovete supporre pre-esistenti, di nome `Nazione` e `Specialita`.

Dovete fornire i seguenti costruttori e metodi pubblici:

- `AtletaOlimpico(String nome, Nazione naz, Specialita spec)`: crea un nuovo atleta con i dati specificati, predisponendo inoltre tre variabili che tengano il conto delle medaglie d'oro, d'argento e di bronzo vinte dall'atleta (all'atto della costruzione, l'atleta non ha alcuna medaglia);
- `void vinceOro()`: incrementa di una unità il conto delle medaglie d'oro vinte dall'atleta che invoca il metodo;
- `void vinceArgento()`: incrementa di una unità il conto delle medaglie d'argento vinte dall'atleta che invoca il metodo;
- `void vinceBronzo()`: incrementa di una unità il conto delle medaglie di bronzo vinte dall'atleta che invoca il metodo;
- `String medaglie()`: restituisce una stringa che descrive la situazione delle medaglie vinte dall'atleta;
- `boolean eMeglioDi(AtletaOlimpico x)`: confronta l'atleta con un altro atleta `x` sulla base del numero di medaglie vinte; più precisamente, si assume che ogni medaglia d'oro valga 3 punti, ogni medaglia d'argento 2 e ogni medaglia di bronzo 1 punto, e si sommano i punti totali per confrontare i due atleti;
- `boolean stessaSpecialita(AtletaOlimpico x)`: restituisce `true` se i due atleti gareggiano nella stessa specialità (si assume che la classe `Specialita` abbia un metodo `equals` con l'usuale segnatura).

La classe `AtletaOlimpico` va, inoltre, *estesa* da una classe `AtletaCapitano` i cui oggetti rappresentano gli atleti capitani delle rispettive nazioni. Oltre ai parametri che caratterizzano un atleta olimpico, un capitano viene caratterizzato dall'*età*.

Oltre al costruttore, in tale classe va scritto il metodo `getEta()` che restituisce l'età dell'atleta capitano che invoca il metodo.

(Svolgimento sul retro)

```

class AtletaOlimpico {
    private String nome;
    private Nazione naz;
    private Specialita spec;
    private int oro, arg, bronzo;

    public AtletaOlimpico(String nome, Nazione naz, Specialita spec) {
        this.nome = nome;
        this.naz = naz;
        this.spec = spec;
    }

    public void vinceOro() { oro++; }
    public void vinceArgento() { arg++; }
    public void vinceBronzo() { bronzo++; }

    public String medaglie() {
        return "Oro:␣" + oro + ",␣Argenti:␣" + arg
            + ",␣Bronzi:␣" + bronzo;
    }

    public boolean eMeglioDi( AtletaOlimpico x ) {
        int punt = 3*oro+2*arg+bronzo;
        int xpunt = 3*x.oro+2*x.arg+x.bronzo;
        return punt > xpunt;
    }

    public boolean stessaSpecialita( AtletaOlimpico x ) {
        return spec.equals( x.spec );
    }
}

class AtletaCapitano extends AtletaOlimpico {
    private int eta;

    public AtletaCapitano( String nome, Nazione naz, Specialita spec, int eta ) {
        super( nome, naz, spec );
        this.eta = eta;
    }

    public int getEta() {
        return eta;
    }
}

```