# Corso di Laurea Ingegneria Informatica

## Fondamenti di Informatica 1

**Dispensa 2**

# Richiami di Fondamenti di Informatica - 1

**A. Miola**

Febbraio 2008

# Contenuti

- **Alcuni argomenti fondamentali**
  - Tipi primitivi e tipizzazione delle espressioni
  - Istruzioni semplici e strutturate
  - Linguaggi e grammatiche
- **Legame dei parametri**
- **Iterazione**
  - Verifiche esistenziali
  - Verifiche universali
- **Tipi riferimento**
  - Stringhe
  - Array
  - Array di oggetti

# Alcuni argomenti fondamentali

**Per un migliore apprendimento degli argomenti del corso di Fondamenti di Informatica 2 è indispensabile riprendere alcuni capitoli importanti del precedente corso di Fondamenti di Informatica 1**

**In particolare :**

- **Tipi primitivi e tipizzazione delle espressioni**
- **Istruzioni semplici e strutturate**
- **Linguaggi e grammatiche**

# Legame di parametri - primitivi

```
class LegameParametri_1 {
    public static void alfa(int b, int a) {
        System.out.println("2: a vale " +a+ ", b vale "+b);
        a = a - 3;
        b = b + 6;
        System.out.println("3: a vale " +a+ ", b vale "+b);
    }
    public static void main(String[] args) {
        double a, b;
        a = 3.0;
        b = 6.0;
        System.out.println("1: a vale " +a+ ", b vale "+b);
        alfa( (int) (a/b + 2.1), (int) (b/a - 1.2));
        System.out.println("4: a vale " +a+ ", b vale "+b);
    }   }
```

```
1: a vale  3.0   , b vale  6.0
2: a vale  0   , b vale  2
3: a vale  -3   , b vale  8
4: a vale  3.0   , b vale  6.0
```

# Legame di parametri - primitivi

```
class LegameParametri_2 {
public static void alfa(double b, double a) {
        System.out.println("2: a vale " +a+ ", b vale "+b);
        a = a/5+1;
        b = b*2+2;
        System.out.println("3: a vale " +a+ ", b vale "+b);
    }
    public static void main(String[] args) {
        int a, b;
        a = 2;
        b = 5;
        System.out.println("1: a vale " +a+ ", b vale "+b);
        alfa(a/b-1,b/a+2);
        System.out.println("4: a vale " +a+ ", b vale "+b);
    }   }
```

```
1: a vale  2   , b vale  5
2: a vale  4.0   , b vale  -1.0
3: a vale  1.8   , b vale  0.0
4: a vale 2   , b vale  5
```

# Legame di parametri - riferimento

```
class LegameParametri_3 {
public static void alfa(TriangoloEquilatero q, TriangoloEquilatero p) {
        System.out.println("2: Il lato di p vale " + p.getLato()+
                                        " , Il lato di q vale " + q.getLato() );
        p = new TriangoloEquilatero(2);
        q.setLato(4);
        System.out.println("3: Il lato di p vale " + p.getLato()+
                                        " , Il lato di q vale " + q.getLato() );    }
public static void main(String[] args) {
        TriangoloEquilatero p = new TriangoloEquilatero (3);
        TriangoloEquilatero q = new TriangoloEquilatero (5);
        System.out.println("1: Il lato di p vale " + p.getLato()+
                                        " , Il lato di q vale " + q.getLato());
        alfa(p,q);
        System.out.println("4: Il lato di p vale " + p.getLato() +
                                        " , Il lato di q vale " + q.getLato());      } }
```

```
1: Il lato di p  3  , Il lato di q  5
2: Il lato di p  5  , Il lato di q  3
3: Il lato di p  2  , Il lato di q  4
4: Il lato di p  4  , Il lato di q  5
```

# Legame di parametri - riferimento

```
class LegameParametri_4 {
public static void alfa(Quadrato q, Quadrato p) {
        System.out.println("2: Il lato di p vale " + p.getLato()+
                                        " , Il lato di q vale " + q.getLato() );
        p = new Quadrato(5);
        q.setLato(7);
        System.out.println("3: Il lato di p vale " + p.getLato()+
                                        " , Il lato di q vale " + q.getLato() );     }
public static void main(String[] args) {
        Quadrato p = new Quadrato(4);
        Quadrato q = new Quadrato(6);
        System.out.println("1: Il lato di p vale " + p.getLato()+
                                        " , Il lato di q vale " + q.getLato());
        alfa(p,q);
        System.out.println("4: Il lato di p vale " + p.getLato() +
                                        " , Il lato di q vale " + q.getLato());    } }
```

```
1: Il lato di p  4  , Il lato di q  6
2: Il lato di p  6  , Il lato di q  4
3: Il lato di p  5  , Il lato di q  7
4: Il lato di p  7  , Il lato di q  6
```

# Iterazione

**Bisogna anche riprendere tutti gli argomenti e gli esercizi relativi ai problemi di iterazione:**

- **Verifiche esistenziali**
- **Verifiche universali**
- **Ricerca**

# Verifica esistenziale

- Bisogna determinare se una sequenza di elementi contiene almeno un elemento che soddisfa una certa proprietà

- in altre parole:

si vuole verificare che in una sequenza di elementi $a_1,\ldots, a_n$ esiste almeno un elemento che verifica una data proprietà $p$ cioè che

$$\exists\, i \in \{1,..,n\},\ p(a_i) = \text{true}$$

N.B. Una sequenza può essere ad esempio una stringa o un array

# Verifica esistenziale: schema risolutivo …

- Viene usata una variabile booleana che indica se la sequenza contiene almeno un elemento che soddisfa la proprietà
- Inizialmente si assegna alla variabile booleana un valore che indica convenzionalmente che la sequenza non contiene nessun elemento che soddisfa la proprietà (false)

# … schema risolutivo …

- A partire dal primo elemento della sequenza si verifica se l'elemento corrente soddisfa la proprietà
  - se l'elemento corrente soddisfa la proprietà, allora si assegna alla variabile booleana un valore che indica convenzionalmente che la sequenza contiene almeno un elemento che soddisfa la proprietà (**true**)
- Quando si trova un elemento che soddisfa la proprietà ci si ferma (non ha senso esaminare oltre perché il problema è risolto)

# … schema risolutivo

```
boolean proprietaSoddisfatta;   // almeno un elemento
                                // soddisfa la proprietà

proprietaSoddisfatta = false;
  ... altre inizializzazioni ...

  finche' non trovo un elemento che soddisfa la
   proprieta'
  while (!proprietaSoddisfatta && sequenza non
                                        terminata) {
      ... accedi al prossimo elemento ...
      if (l'elemento corrente soddisfa la proprietà)
          proprietaSoddisfatta = true;
  }
  ... altre elaborazioni ...
```

usare un nome opportuno per la variabile booleana

# Un errore comune

## Un errore comune nella verifica esistenziale

- ri-assegnare alla variabile booleana usata per la verifica esistenziale il valore che gli è stato assegnato nella sua inizializzazione

```
contieneZero = false;
  while (!Lettore.in.eoln()&& !contieneZero) {
      /* legge il prossimo elemento della sequenza */
      numero = Lettore.in.leggiInt();
      /* se numero vale zero, allora la sequenza
       * contiene almeno uno zero */
      if (numero==0)
          contieneZero = true;
      else
          contieneZero = false;
  }
```

- **Se nella condizione del while non compare `!contieneZero` il programma verifica se l'ultimo elemento della sequenza letta vale zero**
- **Se nella condizione del while compare `!contieneZero` l'ultimo else e' inutile**

# Verifica se una sequenza contiene almeno uno zero

```
... legge una sequenza di numeri interi e
      verifica se contiene almeno uno zero ...
  int numero;         // elemento corrente della sequenza
  boolean contieneZero;    // la sequenza contiene almeno
                           // un elemento uguale a zero

  ... visualizza un messaggio ...
  /* legge la sequenza e verifica
   * se contiene almeno uno zero */
  contieneZero = false;
  while (!Lettore.in.eoln() && !contieneZero) {
      /* legge il prossimo elemento della sequenza */
      numero = Lettore.in.leggiInt();
      /* se numero vale zero, allora la sequenza
       * contiene almeno uno zero */
      if (numero==0)
          contieneZero = true;
  }
  ... il risultato è contieneZero ...
```

# Verifica universale

- **Un problema di verifica universale consiste nel verificare se tutti gli elementi di una sequenza $a_1,\ldots, a_n$ soddisfano una certa proprietà *p***
  - una variante (duale) dei problemi di verifica esistenziale
- **In altre parole:**
  - Un problema di verifica universale è soddisfatto, se tutti gli elementi verificano una data proprietà *p*:

$$\forall\, i \in \{1,..,n\},\ \ p(a_i) = \text{true}$$

  - Oppure un problema di verifica universale non è soddisfatto se esiste almeno un elemento che non verifica *p:*

$$\exists\, i \in \{1,..,n\},\ \ p(a_i) = \text{false}$$

***La verifica universale si può sempre ricondurre alla verifica esistenziale***

# Verifica universale: schema risolutivo

- **Un problema di verifica universale può essere sempre ricondotto a un problema di verifica esistenziale**
  - il problema diventa quello di verificare se **non** esiste nessun elemento della sequenza che **non** soddisfa la proprietà
  - inizialmente si assegna alla variabile booleana un valore che indica convenzionalmente che tutti gli elementi della sequenza soddisfano la proprietà (**true**)
  - per ogni elemento della sequenza, si verifica se l'elemento corrente **non** soddisfa la proprietà
    - **se l'elemento corrente non soddisfa la proprietà, allora si assegna alla variabile booleana un valore che indica convenzionalmente che non tutti gli elementi della sequenza soddisfano la proprietà (false)**

# Verifica universale:schema risolutivo

```
boolean proprietaSoddisfatta;

/* assumo che tutti gli elementi soddisfano
   la proprieta' */
   proprietaSoddisfatta = true;

... altre inizializzazioni ...

finche' non trovo un elemento che non soddisfa
la proprieta
   while (proprietaSoddisfatta &&
                          la sequenza non e' finita){
      ... accedi al prossimo elemento ...
      if (l'elemento corrente non soddisfa la
                                          proprietà)
          proprietàSoddisfatta = false;
}

... altre elaborazioni ...
```

**usare un nome opportuno per la variabile booleana**

# Verifica se una sequenza di dieci elementi è crescente

```
... legge una sequenza di dieci numeri interi e
      verifica se è crescente ...
  int numero;          // elemento corrente della sequenza
  int precedente;      // elemento che precede numero
                       // nella sequenza
  int i;               // per contare i numeri letti
  boolean crescente;   // la sequenza è crescente

  /* il primo elemento della sequenza è il
   * precedente del prossimo che sarà letto */
  precedente = Lettore.in.leggiInt();
  /* finora la sequenza letta è crescente */
  crescente = true;   i=0;
  /* legge e elabora gli altri elementi della sequenza */
  while(i<10 && crescente) {
      /* legge il prossimo numero e verifica che
       * la sequenza sia ancora crescente */
      numero = Lettore.in.leggiInt();
      if (precedente>=numero)
          crescente = false; i++;
      /* prepara la prossima iterazione */
      precedente = numero;
  } ... il risultato della verifica è crescente ...
```

# Verifica l'uguaglianza tra stringhe

- **Scrivere un metodo boolean uguali(String s, String t) che verifica se le stringhe s e t sono uguali**
  - si comporta come **s.equals(t)**
    - **ad esempio, uguali("alfa", "alfa") deve restituire true, mentre uguali("alfa", "beta") deve restituire false**
  - due stringhe s e t sono uguali se
    - **s e t hanno la stessa lunghezza**
    - **ciascun carattere della stringa s è uguale al carattere della stringa t che occupa la stessa posizione**

# Uguaglianza tra stringhe

## Algoritmo:

- **continua il confronto tra le stringhe solo se finora non sono state riscontrate differenze**

```
 /* verifica se s e t sono uguali */
if (s.length()==t.length()) {
    /* s e t hanno la stessa lunghezza: s e t
    /* possono essere uguali, ma sono diverse
     * se contengono almeno un carattere diverso */
    uguali = true;
    i = 0;
    while (uguali && i<s.length()) {
        if (s.charAt(i) != t.charAt(i))
            uguali = false;
        i = i+1;
    }
else /*s e t hanno lunghezza diversa, quindi sono diverse */
   uguali = false;
```

# Verifica esistenziale e verifica universale

## ❑ Verifica esistenziale:

- Si vuole verificare che in una sequenza di elementi

  $a_1,\ldots, a_n$ esiste almeno un elemento che verifica una data proprietà $p$ cioè che

$$\exists\ i \in \{1,..,n\},\ p(a_i) = \text{true}$$

## ❑ Verifica universale:

- Si vuole verificare che in una sequenza di elementi

  $a_1,\ldots, a_n$ tutti gli elementi della sequenza verificano una data proprietà $p$ cioè che

$$\forall\ i \in \{1,..,n\},\ p(a_i) = \text{true}$$

- Oppure si può verificare che esiste almeno un elemento che non verifica la proprietà $p$: in tal caso la verifica universale non è soddisfatta

$$\exists\ i \in \{1,..,n\},\ p(a_i) = \text{false}$$

# Altri argomenti fondamentali

**Per un migliore apprendimento degli argomenti del corso di Fondamenti di Informatica 2 è indispensabile riprendere anche altri capitoli importanti del precedente corso di Fondamenti di Informatica 1**

**In particolare quelli relativi ai Tipi riferimento :**

- **Stringhe**
- **Array**
- **Array di oggetti**

# Ancora sulla classe libro

- **Data la classe Libro per rappresentare oggetti libro con il nome dell'autore, il titolo e il numero di pagine e con i relativi metodi d'istanza**
  - **Costruire un array di oggetti di tipo Libro: elencoLibri (già fatto)**
  - **scrivere un metodo che, ricevendo come parametro un array di stringhe (selezioneLibri) che rappresentano ciascuna il nome di un autore, calcola e restituisce un array di libri formato da tutti e soli i libri degli autori indicati e presenti nell'elenco elencoLibri**

# Esempio

- **Dato il seguente elenco di libri :**

```
elencoLibri = new Libro[10];
elencoLibri[0] = new Libro("Esopo","Le storie dell'asino",20);
elencoLibri[1] = new Libro("Italo Calvino", "Il visconte
                                          dimezzato",158);
elencoLibri[2] = new Libro("Esopo","Le storie del cane",20);
elencoLibri[3] = new Libro("Esopo","Le favole piu' belle",98);
elencoLibri[4] = new Libro("Gianni Rodari","Filastrocche
                                 lunghe e corte",80);
elencoLibri[5] = new Libro("Italo Calvino","le
                                       cosmicomiche",250);
elencoLibri[6] = new Libro("Gianni Rodari","Enciclopedia della
                                          favola",1120);
elencoLibri[7] = new Libro("Italo Calvino","Il barone
                                         rampante",135);
elencoLibri[8] = new Libro("Fratelli Grimm","Raperonzolo",36);
elencoLibri[9] = new Libro("Italo Calvino","Il cavaliere
                                       inesistente",124);
```

# Esempio

- E dato l'array di autori {"Fratelli Grimm", "Gianni Rodari"}, viene creato un nuovo array selezioneLibri formato da:

```
selezioneLibri[0] = Libro("Gianni
Rodari","Filastrocche
       lunghe e corte",80);
selezioneLibri[1] = Libro("Gianni
Rodari","Enciclopedia della
                favola",1120);
selezioneLibri[2] = Libro("Fratelli
Grimm","Raperonzolo",36);
```

# L’algoritmo per selezionare l’elenco

- Utilizzo un array di booleani ok (inizializzato tutto a false) parallelo ad elenco per “*spuntare*” i libri degli autori indicati (autori) e allo stesso tempo conto il numero di elementi spuntati (dim), che corrisponderà alla dimensione del nuovo array
- Creo il nuovo array res di dimensione dim
- Esamino l’array di booleani ed inserisco nel nuovo array solo i libri il cui corrispondente valore booleano nell’array ok è true

# Il metodo seleziona…

## Dichiarazioni e inizializzazioni

```
public static Libro[] seleziona(Libro[] elenco, String[]
                                                    autori){
   int i,j; // per scorrere gli array
   int k; // per scorrere l'array risultato da creare
   boolean[] ok; // per tenere nota dei libri da
                 // memorizzare nel risultato
   int dim;      // per calcolare la dimensione
                 // del nuovo array da creare
   Libro[] res;
   String app;

   /* inizializzo l'array di booleani */
   ok = new boolean[elenco.length];
   for (i=0; i< elenco.length; i++)
       ok[i] = false;
```

# …Il metodo seleziona…

- **Spunta dei libri degli autori indicati nell'array autori**

```
dim = 0;
/* i scorre gli autori,
   j scorre tutto l'elenco di libri e l'array
  (parallelo) di booleani */
for (i=0; i<autori.length; i++){
    app = autori[i]; //seleziono l'autore da cercare
    for (j=0; j<elenco.length; j++)
          if (elenco[j].getAutore().equals(app)){
                ok[j] = true;
                dim++;
          }
}
```

# …Il metodo seleziona…

- Creazione dell'array risultato **res**

```
    /* creo il nuovo array */
    res = new Libro[dim];
    k=0;
    /* scorro l'array di booleani per vedere gli
       autori da inserire */
    for (i=0; i<elenco.length; i++)
          if (ok[i] == true){
                res[k] = elenco[i];
                k++;
          }
    return res;
}
```

# Esempio d’esecuzione

```
==================================
========== ELENCO LIBRI ============
==================================
AUTORE:    Esopo
TITOLO:    Le storie dell'asino
N. PAGINE: 20

AUTORE:    Italo Calvino
TITOLO:    Il visconte dimezzato
N. PAGINE: 158

AUTORE:    Esopo
TITOLO:    Le storie del cane
N. PAGINE: 20

AUTORE:    Esopo
TITOLO:    Le favole piu' belle
N. PAGINE: 98

AUTORE:    Gianni Rodari
TITOLO:    Filastrocche lunghe e corte
N. PAGINE: 80

AUTORE:    Italo Calvino
TITOLO:    le cosmicomiche
N. PAGINE: 250

AUTORE:    Gianni Rodari
TITOLO:    Enciclopedia della favola
N. PAGINE: 1120

AUTORE:    Italo Calvino
TITOLO:    Il barone rampante
N. PAGINE: 135

AUTORE:    Fratelli Grimm
TITOLO:    Raperonzolo
N. PAGINE: 36

AUTORE:    Italo Calvino
TITOLO:    Il cavaliere inesistente
N. PAGINE: 124


*******
******* ELENCO SELEZIONATO ********
==================================
========== ELENCO LIBRI ============
==================================
AUTORE:    Italo Calvino
TITOLO:    Il visconte dimezzato
N. PAGINE: 158

AUTORE:    Italo Calvino
TITOLO:    le cosmicomiche
N. PAGINE: 250

AUTORE:    Italo Calvino
TITOLO:    Il barone rampante
N. PAGINE: 135

AUTORE:    Fratelli Grimm
TITOLO:    Raperonzolo
N. PAGINE: 36

AUTORE:    Italo Calvino
TITOLO:    Il cavaliere inesistente
N. PAGINE: 124


Press any key to continue . . .
```

# Considerazioni

- Per ogni autore da selezionare devo esaminare tutto l'elenco dei libri: è realistico?
- Immaginiamo un vero catalogo di una biblioteca

- Se l'array fosse già ordinato, per esempio per autore e se anche la lista di autori fosse ordinata, come cambierebbe l'algoritmo?