Programmazione

Corso di Laurea in Informatica
Corso di Laurea in Informatica per la Comunicazione Digitale

Università di Camerino

Prof. Michele Loreti

Anno Accademico 2025/26

Programmazione

Metodi

- Motivazione
- Il principio di astrazione procedurale
- Coefficiente binomiale con metodo
- Commenti
- Esecuzione del programma con metodi
- Definizione di un metodo
- Invocazione di un metodo
- Metodi predefiniti
- Il comando `return`
- Uso corretto di `return` con tipo di ritorno `void`
- Uso errato di `return` con tipo di ritorno `void`
- Uso corretto di `return` con tipo di ritorno non `void`
- Uso errato di `return` con tipo di ritorno non `void`
- Metodi e variabili locali
- Esercizi

Motivazione

Problema

Scrivere un programma che, dati due numeri n e k tali che $0 \leq k \leq n$, calcoli il coefficiente binomiale "n su k", ovvero la quantità

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Soluzione (con difetto)

```
int n = StdIn.readInt();
int k = StdIn.readInt();
int a = 1;                                  // calcola n!
for (int i = 2; i <= n; i++) a = a * i;
int b = 1;                                  // calcola k!
for (int i = 2; i <= k; i++) b = b * i;
int c = 1;                                  // calcola (n - k)!
for (int i = 2; i <= n - k; i++) c = c * i;
StdOut.println(n + " su " + k + " = " + a / (b * c));
```

Il principio di astrazione procedurale

Difetto della soluzione proposta

- occorre calcolare il fattoriale di **tre numeri diversi**
- il codice nei tre casi è **essenzialmente lo stesso**, cambia solo l'estremo superiore del ciclo `for` e la variabile in cui memorizzare il risultato

Rimedio: usare un metodo

- un **metodo** è un frammento di programma al quale è associato un nome e che svolge un **compito specifico** (es. calcolare il fattoriale di un numero)
- una volta definito, il metodo può essere **invocato** attraverso il suo nome ogniqualvolta è necessario svolgere quel compito
- il metodo può avere degli **argomenti** attraverso i quali forniamo informazioni utili alla sua esecuzione
  (es. il numero del quale vogliamo calcolare il fattoriale)
- il metodo può avere un **valore di ritorno** attraverso il quale il metodo comunica all'esterno il risultato dell'esecuzione
  (es. il fattoriale del numero fornito come argomento)

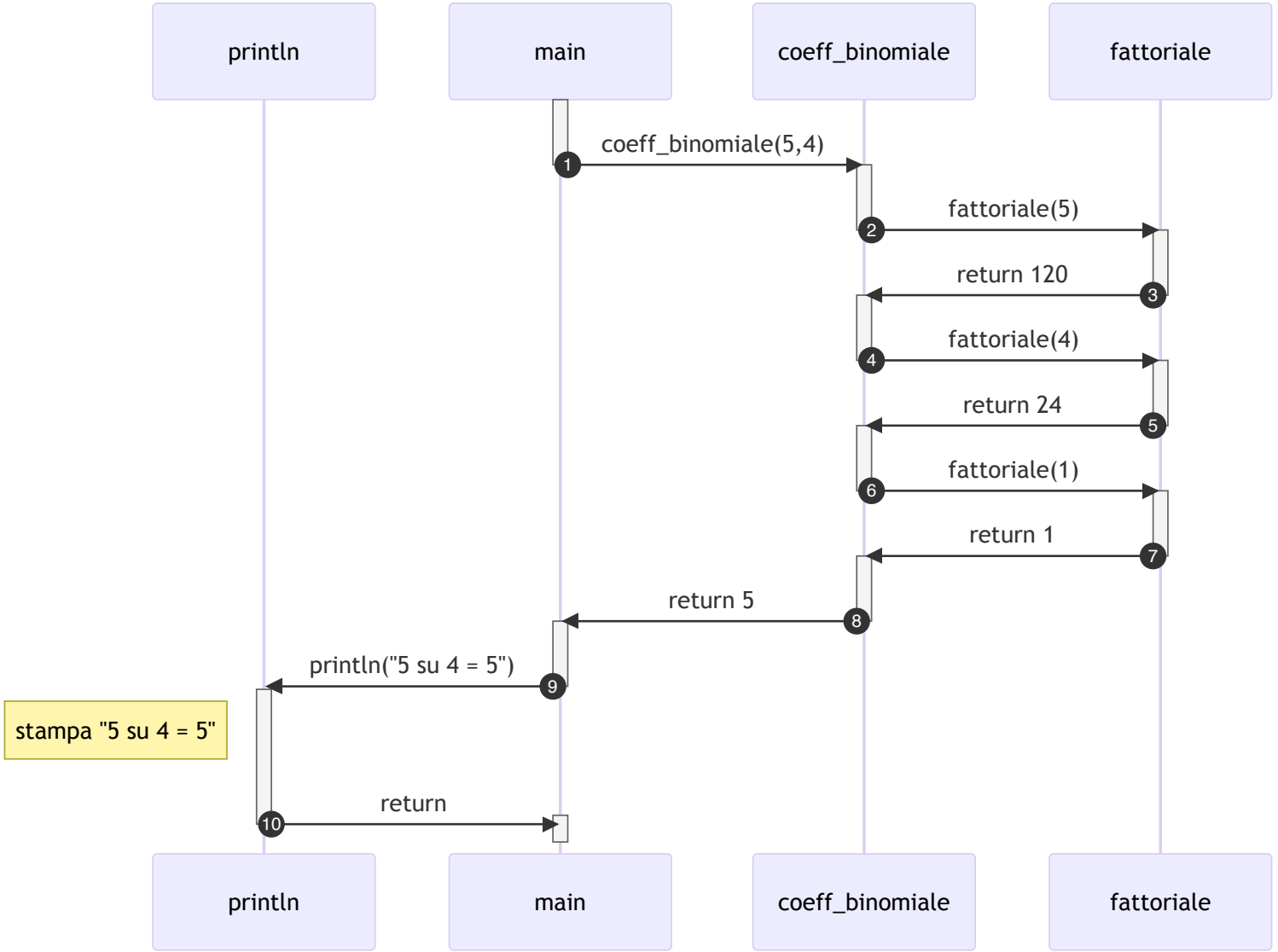
Coefficiente binomiale con metodo

```
public class CoefficienteBinomiale {
    public static int fattoriale(int n) {
  int r = 1;
  for (int i = 2; i <= n; i++) r = r * i;
  return r;
    }

    public static int coeff_binomiale(int n, int k) {
  return fattoriale(n) / (fattoriale(k) * fattoriale(n - k));
    }

    public static void main(String[] args) {
  int n = StdIn.readInt();
  int k = StdIn.readInt();
  int c = coeff_binomiale(n, k);
  StdOut.println(n + " su " + k + " = " + c);
    }
}
```

Commenti

- definisco un metodo `fattoriale`
  - il metodo è “pubblico” e “statico”
  - il metodo ha un argomento n di tipo `int`
  - il metodo ha valore di ritorno di tipo `int`
  - il **corpo del metodo** è scritto **una sola volta** e calcola il fattoriale di n
- definisco un metodo `coeff_binomiale`
  - il metodo è “pubblico” e “statico”
  - il metodo ha due argomenti n e k entrambi di tipo `int`
  - il metodo ha valore di ritorno di tipo `int`
  - il **corpo del metodo** calcola “n su k” **invocando** `fattoriale` **tre volte**
- definisco un metodo `main`
  - il metodo è “pubblico” e “statico”
  - il metodo ha un argomento `args` di tipo `String[]` (da rivedere in futuro)
  - il metodo non restituisce nulla (`void`)
  - il **corpo del metodo** invoca `coeff_binomiale` (e altri metodi)

Esecuzione del programma con metodi



Definizione di un metodo

Sintassi

```
tipo identificatore(lista argomenti) blocco di comandi
```

Significato

- `tipo` è il **tipo di ritorno** del metodo, ovvero il tipo del valore che il metodo restituisce a chi lo invoca nel momento in cui termina
- i metodi che **non restituiscono alcun valore** hanno tipo di ritorno `void`
- `identificatore` è il **nome** del metodo da definire, deve essere descrittivo della funzione del metodo e rispettare le regole di formazione degli identificatori
- `lista argomenti` è una lista di coppie `tipo identificatore` separate da virgole e delimitata da `(...)`. Ogni coppia dichiara un **argomento** di tipo e nome specificati.
- `blocco di comandi` è il **corpo** del metodo, ovvero la sequenza di comandi racchiusi entro `{...}` che vengono eseguiti ogni volta che il metodo viene invocato

Invocazione di un metodo

Sintassi

```
identificatore(lista espressioni)
```

Significato

- `identificatore` è il **nome** del metodo da invocare
- `lista espressioni` è una lista di espressioni separate da virgole e delimitata da `(...)`.
  Ogni espressione calcola il valore dell'argomento corrispondente per questa invocazione particolare del metodo

Note

- se il metodo restituisce un valore, l'invocazione compare all'interno di una espressione (es. `coeff_binomiale`)
- se il metodo non restituisce alcun valore, l'invocazione è di fatto un comando (es. `println`)

Metodi predefiniti

`StdIn.readInt`, `StdOut.println`, `Math.sqrt` (insieme ad altri) sono **metodi** a disposizione del programmatore **definiti in una libreria**

```
public static double sqrt(double x) {
    ...
    // corpo di sqrt
    ...
}

public static int readInt() {
    ...
    // corpo di readInt
    ...
}

public static void println(String messaggio) {
    ...
    // corpo di println
    ...
}
```

Il comando `return`

Sintassi

```
return;             // per metodi con tipo di ritorno void
return espressione; // per tutti gli altri metodi
```

Significato

1. se presente, valuta `espressione`
2. termina l'esecuzione del metodo e restituisce il controllo al chiamante, che riprende l'esecuzione dal punto dell'invocazione
3. se presente, il valore di `espressione` è il risultato dell'invocazione del metodo nel chiamante

Note

- se il tipo di ritorno è `void`, l'uso di `return` è **opzionale** (in mancanza di un `return` esplicito, Java ne inserisce uno alla fine del corpo del metodo)
- se il tipo di ritorno è diverso da `void`, l'uso di `return` è **obbligatorio** (Java non può "indovinare" qual è il valore giusto da restituire al chiamante)

Uso corretto di `return` con tipo di ritorno `void`

`return` presente

```java
public static void saluto(String nome) {
    StdOut.println("Ciao " + nome + "!");
    return;
}
```

È **consentito** l'uso esplicito del comando `return` senza espressione nei metodi con tipo di ritorno `void`

`return` non presente

```java
public static void saluto(String nome) {
    StdOut.println("Ciao " + nome + "!");
}
```

L'uso del comando `return` senza espressione è **opzionale**. Se assente, viene inserito automaticamente dal compilatore Java alla fine del corpo del metodo

Uso errato di `return` con tipo di ritorno `void`

`return` seguito da codice

```
public static void saluto(String nome) {
    return;
    StdOut.println("Ciao " + nome + "!");
}
```

Il codice che segue `return` non viene mai eseguito e per tale motivo è detto **codice morto**, evidentemente c'è qualcosa di sbagliato nel corpo del metodo

`return` con espressione

```
public static void saluto(String nome){
    StdOut.println("Ciao " + nome + "!");
    return true;
}
```

Il tipo di ritorno del metodo è `void`, dunque la forma corretta da utilizzare per il comando `return` è quella **senza espressione**

Uso corretto di `return` con tipo di ritorno non `void`

`return` multipli

```
public static int abs(int n) {
  if (n < 0) return -n;
  else return n;
}
```

È indicato il valore restituito dal metodo alla sua terminazione in **ogni** percorso di esecuzione

`return` unico

```
public static int abs(int n) {
  if (n < 0) n = -n;
  return n;
}
```

È indicato il valore restituito dal metodo alla sua terminazione

Uso errato di `return` con tipo di ritorno non `void`

Esistenza di percorsi senza `return`

```
public static int abs(int n) {
  if (n < 0) return -n;
  // ...e se n >= 0?
}
```

Ogni percorso di esecuzione nel corpo del metodo deve concludersi con un comando `return`, poiché occorre indicare esplicitamente il valore restituito dal metodo alla sua terminazione

`return` senza espressione

```
public static int abs(int n) {
  if (n < 0) return -n;
  else return;
}
```

Quando il tipo di ritorno non è `void`, l'unica forma consentita per il comando `return` è quella con espressione

Metodi e variabili locali

```
public static void A() {
    int i = 0;
}

public static void B() { // NO
    StdOut.println("I = " + i);
}

public static void C() { // OK
    int i = 0;
    StdOut.println("I = " + i);
}
```

- la variabile `i` dichiarata in A **non è visibile** in B
- la variabile `i` dichiarata in A è **diversa** dalla variabile `i` dichiarata in C
- variabili dichiarate in metodi diversi possono avere **tipi diversi**

```
public static void incrementa() {
    int i = 0;
    println("I = " + i);
    i++;
}

incrementa(); // I = 0
incrementa(); // I = 0
incrementa(); // I = 0
incrementa(); // I = 0
incrementa(); // I = 0
incrementa(); // I = 0
```

- il valore delle variabili locali **non si preserva** da una invocazione alla successiva dello stesso metodo
- ogni invocazione di un metodo comporta l'inizializzazione delle sue variabili locali

Esercizi

1. Definire due metodi `mcd` ed `mcm` per calcolare rispettivamente MCD ed MCM di due numeri interi positivi
2. Definire due metodi `pari` e `dispari`, ciascuno con un argomento di tipo `int` e tipo di ritorno `boolean`, per determinare se un numero è, appunto, pari o dispari. Evitare quanto più possibile la duplicazione di codice
3. Definire un metodo `round` con un argomento di tipo `double` e tipo di ritorno `int` che arrotonda il suo argomento al numero intero più vicino
4. Definire un metodo `hex` con un argomento `x` di tipo `int` e tipo di ritorno `String` che converte `x` nella sua rappresentazione in base 16. Tenere in considerazione la possibilità che `x` sia negativo
5. Definire un metodo `primo` con un argomento `x` di tipo `int` e tipo di ritorno `boolean` che determina se `x` è primo
6. Definire un metodo `stampa_primi` con un argomento `n` di tipo `int` che stampa tutti i numeri primi compresi tra 1 ed `n`, estremi inclusi. Utilizzare il metodo `primo` dell'esercizio precedente