Capitolo 10

Organizzazione della memoria e ricorsione

Sommario: Organizzazione della memoria e ricorsione

Intestazione di un metodo

- Modificatori
  Forniscono informazioni relative al metodo e alla sua fruibilità da parte di altri metodi o classi

Intestazione di un metodo

- Modificatori
  Forniscono informazioni relative al metodo e alla sua fruibilità da parte di altri metodi o classi
- Tipo restituito
  Può essere un tipo primitivo o un tipo riferimento (riferimento a un oggetto oppure `null`)

Intestazione di un metodo

- Modificatori
  Forniscono informazioni relative al metodo e alla sua fruibilità da parte di altri metodi o classi
- Tipo restituito
  Può essere un tipo primitivo o un tipo riferimento (riferimento a un oggetto oppure `null`)

  `void` per indicare che un metodo non restituisce alcun valore

Intestazione di un metodo

- Modificatori
  Forniscono informazioni relative al metodo e alla sua fruibilità da parte di altri metodi o classi
- Tipo restituito
  Può essere un tipo primitivo o un tipo riferimento (riferimento a un oggetto oppure `null`)

  `void` per indicare che un metodo non restituisce alcun valore
- Nome del metodo
  Un identificatore scelto da chi ha scritto il metodo

Intestazione di un metodo

- Modificatori
  Forniscono informazioni relative al metodo e alla sua fruibilità da parte di altri metodi o classi
- Tipo restituito
  Può essere un tipo primitivo o un tipo riferimento (riferimento a un oggetto oppure `null`)

  `void` per indicare che un metodo non restituisce alcun valore
- Nome del metodo
  Un identificatore scelto da chi ha scritto il metodo
- Lista degli argomenti
  Sono identificatori scelti da chi ha scritto il metodo. Ogni identificatore è preceduto dal nome del proprio tipo. I parametri sono separati tra loro da virgole

Intestazione di un metodo

- Modificatori
  Forniscono informazioni relative al metodo e alla sua fruibilità da parte di altri metodi o classi
- Tipo restituito
  Può essere un tipo primitivo o un tipo riferimento (riferimento a un oggetto oppure `null`)

  `void` per indicare che un metodo non restituisce alcun valore
- Nome del metodo
  Un identificatore scelto da chi ha scritto il metodo
- Lista degli argomenti
  Sono identificatori scelti da chi ha scritto il metodo. Ogni identificatore è preceduto dal nome del proprio tipo. I parametri sono separati tra loro da virgole
- Eccezioni

Parametri formali e parametri attuali

- Parametri formali
  - Sono gli identificatori indicati nella lista degli argomenti

Parametri formali e parametri attuali

- Parametri formali
  - Sono gli identificatori indicati nella lista degli argomenti
  - Sono di fatto variabili locali del metodo

Parametri formali e parametri attuali

- Parametri formali
  - Sono gli identificatori indicati nella lista degli argomenti
  - Sono di fatto variabili locali del metodo

```
private static int mcd(int a, int b)
```

Parametri formali e parametri attuali

- Parametri formali
  - Sono gli identificatori indicati nella lista degli argomenti
  - Sono di fatto variabili locali del metodo

```
private static int mcd(int a, int b)
```

- Parametri attuali
  - Sono gli argomenti con cui il metodo viene chiamato

Parametri formali e parametri attuali

- Parametri formali
  - Sono gli identificatori indicati nella lista degli argomenti
  - Sono di fatto variabili locali del metodo

```
private static int mcd(int a, int b)
```

- Parametri attuali
  - Sono gli argomenti con cui il metodo viene chiamato
  - Sono espressioni di tipo compatibile con quelle dei parametri formali corrispondenti

Parametri formali e parametri attuali

- Parametri formali
  - Sono gli identificatori indicati nella lista degli argomenti
  - Sono di fatto variabili locali del metodo

```
private static int mcd(int a, int b)
```

- Parametri attuali
  - Sono gli argomenti con cui il metodo viene chiamato
  - Sono espressioni di tipo compatibile con quelle dei parametri formali corrispondenti

```
x = 3 * mcd(x + y, z) + 1;
```

Invocazione di un metodo

- Passaggio per valore
  Al momento della chiamata, i parametri formali vengono allocati in memoria e inizializzati con il valore dell'argomento corrispondente

Invocazione di un metodo

- Passaggio per valore
  Al momento della chiamata, i parametri formali vengono allocati in memoria e inizializzati con il valore dell'argomento corrispondente
- Esecuzione
  Compiuto il passaggio dei parametri la JVM inizia ad eseguire le istruzioni nel corpo del metodo

Invocazione di un metodo

- Passaggio per valore
  Al momento della chiamata, i parametri formali vengono allocati in memoria e inizializzati con il valore dell'argomento corrispondente
- Esecuzione
  Compiuto il passaggio dei parametri la JVM inizia ad eseguire le istruzioni nel corpo del metodo
- Rientro
  L'esecuzione del metodo termina con l'istruzione `return`:
  - restituisce il risultato
  - determina il ritorno al metodo chiamante

Invocazione di un metodo

- Passaggio per valore
  Al momento della chiamata, i parametri formali vengono allocati in memoria e inizializzati con il valore dell'argomento corrispondente
- Esecuzione
  Compiuto il passaggio dei parametri la JVM inizia ad eseguire le istruzioni nel corpo del metodo
- Rientro
  L'esecuzione del metodo termina con l'istruzione `return`:
  - restituisce il risultato
  - determina il ritorno al metodo chiamante

  Il tipo dell'espressione che segue `return` dev'essere assegnabile al tipo restituito dal metodo

Invocazione di un metodo

- Passaggio per valore
  Al momento della chiamata, i parametri formali vengono allocati in memoria e inizializzati con il valore dell'argomento corrispondente
- Esecuzione
  Compiuto il passaggio dei parametri la JVM inizia ad eseguire le istruzioni nel corpo del metodo
- Rientro
  L'esecuzione del metodo termina con l'istruzione `return`:
  - restituisce il risultato
  - determina il ritorno al metodo chiamante

  Il tipo dell'espressione che segue `return` dev'essere assegnabile al tipo restituito dal metodo
- Distruzione dell'ambiente di esecuzione
  Terminata l'esecuzione viene rilasciata la memoria utilizzata per l'esecuzione del metodo

Memoria utilizzata dalla JVM

Memoria statica

- Utilizzata per i campi statici delle classi

Memoria utilizzata dalla JVM

Memoria statica

- Utilizzata per i campi statici delle classi
- Allocata al momento in cui le classi vengono caricate per l'esecuzione

Memoria utilizzata dalla JVM

Memoria statica

- Utilizzata per i campi statici delle classi
- Allocata al momento in cui le classi vengono caricate per l'esecuzione
- La quantità di memoria statica necessaria per una classe può essere stabilita a priori esaminando esclusivamente il testo della classe

Memoria utilizzata dalla JVM

Memoria statica

- Utilizzata per i campi statici delle classi
- Allocata al momento in cui le classi vengono caricate per l'esecuzione
- La quantità di memoria statica necessaria per una classe può essere stabilita a priori esaminando esclusivamente il testo della classe

Stack

- Per contenere i dati usati dai singoli metodi che vengono man mano eseguiti

Memoria utilizzata dalla JVM

Memoria statica

- Utilizzata per i campi statici delle classi
- Allocata al momento in cui le classi vengono caricate per l'esecuzione
- La quantità di memoria statica necessaria per una classe può essere stabilita a priori esaminando esclusivamente il testo della classe

Stack

- Per contenere i dati usati dai singoli metodi che vengono man mano eseguiti
- La sua struttura evolve dinamicamente durante l'esecuzione in base alle chiamate dei metodi

Memoria utilizzata dalla JVM

Memoria statica

- Utilizzata per i campi statici delle classi
- Allocata al momento in cui le classi vengono caricate per l'esecuzione
- La quantità di memoria statica necessaria per una classe può essere stabilita a priori esaminando esclusivamente il testo della classe

Stack

- Per contenere i dati usati dai singoli metodi che vengono man mano eseguiti
- La sua struttura evolve dinamicamente durante l'esecuzione in base alle chiamate dei metodi

Heap

- Per memorizzare gli oggetti creati dinamicamente durante l'esecuzione

Gestione dello heap

- Gli oggetti vengono creati dinamicamente richiamando i costruttori all'interno di espressioni `new`

Gestione dello heap

- Gli oggetti vengono creati dinamicamente richiamando i costruttori all'interno di espressioni `new`
- Quando un oggetto non è più accessibile l'area di memoria utilizzata dall'oggetto può essere recuperata e riutilizzata successivamente per altri oggetti

Gestione dello heap

- Gli oggetti vengono creati dinamicamente richiamando i costruttori all'interno di espressioni `new`
- Quando un oggetto non è più accessibile l'area di memoria utilizzata dall'oggetto può essere recuperata e riutilizzata successivamente per altri oggetti
- Garbage collector
  - recupera la memoria occupata da oggetti non più referenziati

Gestione dello heap

- Gli oggetti vengono creati dinamicamente richiamando i costruttori all'interno di espressioni `new`
- Quando un oggetto non è più accessibile l'area di memoria utilizzata dall'oggetto può essere recuperata e riutilizzata successivamente per altri oggetti
- Garbage collector
  - recupera la memoria occupata da oggetti non più referenziati
  - riorganizza lo heap rimediando ai problemi di allocazione dovuti alla frammentazione

Lo stack

Prende il nome dalla struttura dati utilizzata per gestirla, lo stack (pila).

Lo stack

Prende il nome dalla struttura dati utilizzata per gestirla, lo stack (pila).

- Struttura LIFO: è possibile aggiungere o eliminare elementi solo in cima alla pila

Lo stack

Prende il nome dalla struttura dati utilizzata per gestirla, lo stack (pila).

- Struttura LIFO: è possibile aggiungere o eliminare elementi solo in cima alla pila

Memoria stack

- È una pila di record di attivazione

Lo stack

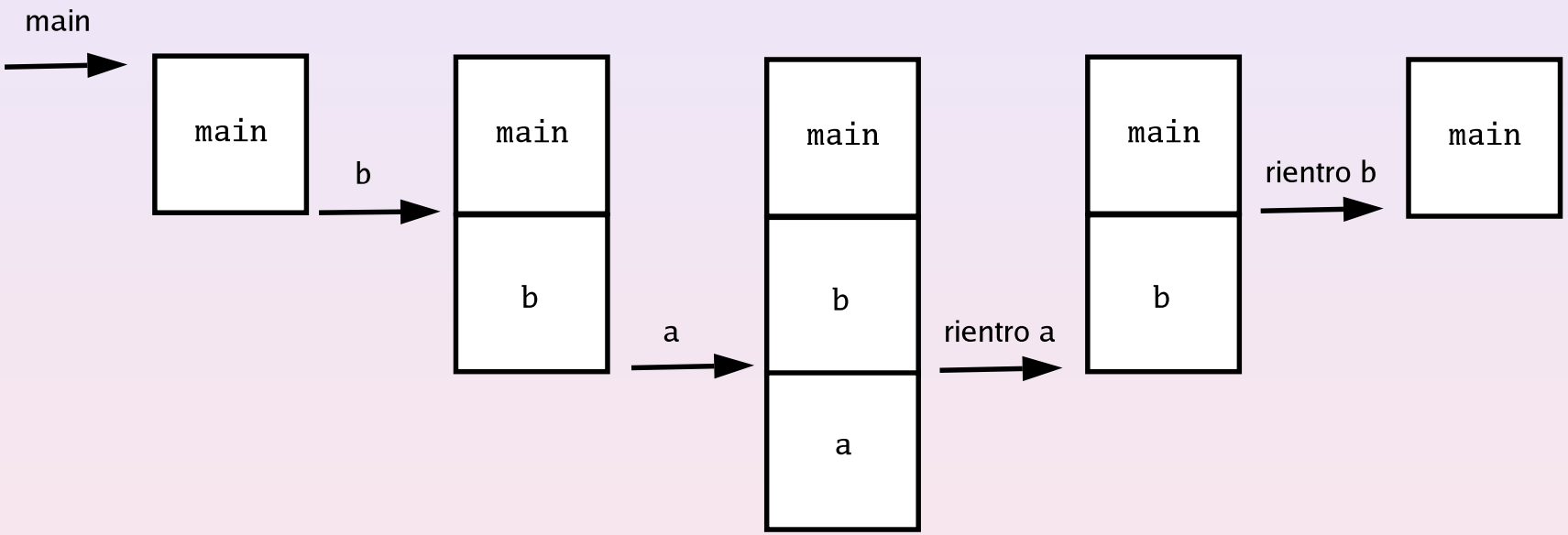
Prende il nome dalla struttura dati utilizzata per gestirla, lo stack (pila).

- Struttura LIFO: è possibile aggiungere o eliminare elementi solo in cima alla pila

Memoria stack

- È una pila di record di attivazione
- Ogni record di attivazione è un'area di memoria locale contenente i dati relativi a ciascun metodo attivato

Esempio



Struttura del record di attivazione

Informazioni di controllo

- per memorizzare i risultati dei metodi richiamati
- per effettuare correttamente il rientro dai metodi richiamati

Struttura del record di attivazione

Informazioni di controllo

- per memorizzare i risultati dei metodi richiamati
- per effettuare correttamente il rientro dai metodi richiamati

Dati del metodo

- variabili locali

Struttura del record di attivazione

Informazioni di controllo

- per memorizzare i risultati dei metodi richiamati
- per effettuare correttamente il rientro dai metodi richiamati

Dati del metodo

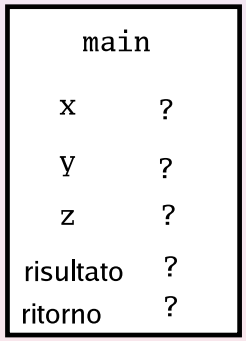
- variabili locali
- parametri formali
  inizializzati con gli argomenti specificati al momento della chiamata

Esempio

```
import prog.io.*;

public class Doppio {
  public static void main(String[] args) {
    ConsoleInputManager in = new ConsoleInputManager();
    ConsoleOutputManager out = new ConsoleOutputManager();

    int x = in.readInt("Inserire un intero ");
    int y = doppio(x);  //1
    int z = doppio(y);  //2
    out.println(y);
    out.println(z);
  }

  public static int doppio(int n) {
    int k = 2 * n;
    return k;
  }
}
```

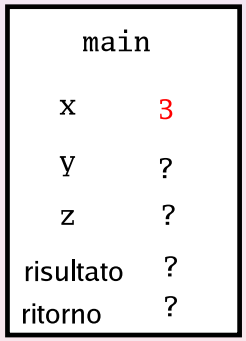
Esempio

`invocazione del main`



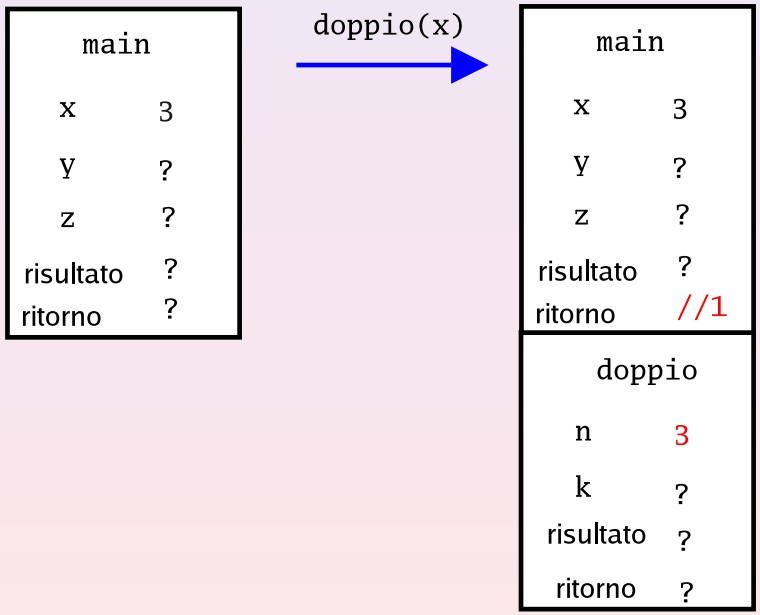
Esempio

```
int x = in.readInt("Inserire un intero ");
```



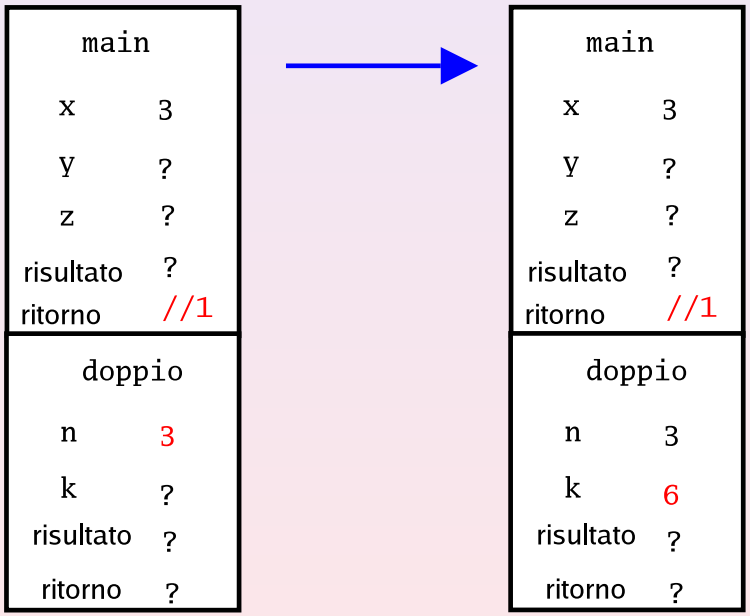
Esempio

```
int y = doppio(x);  //1
```



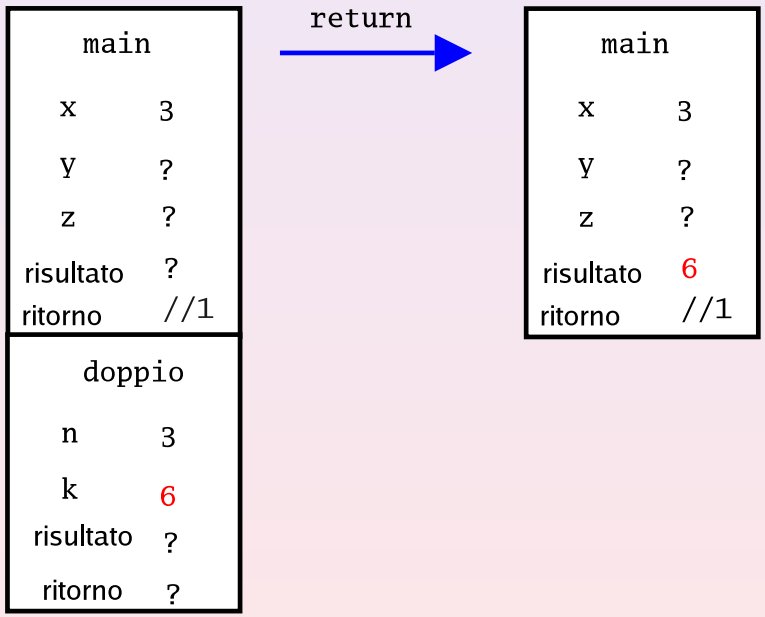
Esempio

```
int k = 2 * n;
```



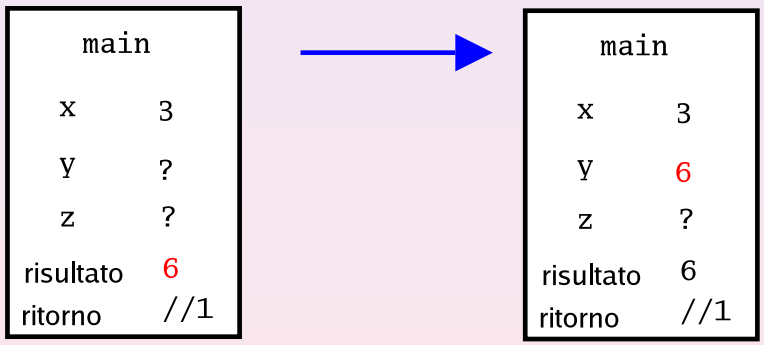
Esempio

```
return k;
```



Esempio

```
int y = doppio(x);  //1
```



Ricorsione

Un'entità è detta ricorsiva se è definita in termini di se stessa.

Ricorsione

Un'entità è detta ricorsiva se è definita in termini di se stessa.

Definizione di funzioni matematiche

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{altrimenti} \end{cases}$$

Ricorsione

Un'entità è detta ricorsiva se è definita in termini di se stessa.

Definizione di funzioni matematiche

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{altrimenti} \end{cases}$$

Grammatiche

espressione ::= variabile | costante |
espressione + espressione |
espressione − espressione |
espressione * espressione |
espressione / espressione |
(espressione)

Metodi ricorsivi

Un metodo ricorsivo è un metodo che richiama se stesso.

Metodi ricorsivi

Un metodo ricorsivo è un metodo che richiama se stesso.

Esempio

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{altrimenti} \end{cases}$$

Metodi ricorsivi

Un metodo ricorsivo è un metodo che richiama se stesso.

Esempio

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{altrimenti} \end{cases}$$

```
public static int fattoriale(int n) {
  if (n == 0)
    return 1;
  else
    return n * fattoriale(n - 1);
}
```