DIPARTIMENTO DI INGEGNERIA E SCIENZE DELL'INFORMAZIONE E MATEMATICA – UNIVAQ

CORSO DI LABORATORIO DI PROGRAMMAZIONE I

ESERCITAZIONE 5

PROF. MONICA NESI - DR. FRANCESCO GALLO

Blocco 0 - francesco.gallo@univaq.it www.di.univaq.it/francesco.gallo

OUTLINE

- Java - Ricorsione

Ricorsione

La ricorsione è una tecnica di programmazione utilizzata per eseguire operazioni che direttamente o indirettamente richiamano se stessi.

Nella ricorsione viene invocato un metodo mentre questo è in esecuzione. Il metodo che ne fa uso si chiama **metodo ricorsivo**.

La ricorsione è connessa al principio di **induzione matematica**

Ricorsione

Quando un metodo ricorsivo invoca se stesso, la macchina virtuale Java esegue le stesse azioni che vengono eseguite quando viene invocato un metodo qualsiasi:

1. sospende l'esecuzione del metodo invocante
2. esegue il metodo invocato fino alla sua terminazione
3. riprende l'esecuzione del metodo invocante dal punto in cui era stata sospeso.

La ricorsione può essere di diversi tipi: **ricorsione in coda** e **ricorsione multipla**.

La ricorsione in coda si presenta quando un metodo invoca se stesso una sola volta mentre **la ricorsione multipla avviene quando un metodo invoca se stesso più volte**.

Ricorsione in coda

La ricorsione in coda[1] (**tail recursion**) si presenta quando il metodo ricorsivo esegue una sola invocazione ricorsiva e tale invocazione è l'ultima azione del metodo.

Tale ricorsione può essere agevolmente eliminata, trasformando il metodo ricorsivo in un metodo che usa un ciclo.

[1] - La ricorsione in coda si presenta meno efficiente del ciclo in quanto viene utilizzata molta più memoria per gestire le invocazione sospese nello stack ma rispetto al ciclo presenta il codice molto più leggibile.

Ricorsione in coda - Esempio

```java
class Ricorsione {

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

public class Fattoriale {

  public static void main(String[] args) {
    int val = 5;
    int risultato;

    risultato = Ricorsione.ricorsione(val);
    String string = ": ";
    String x = "Fattoriale di " + val + string + risultato;
    System.out.println(x);
  }
}
```

Ricorsione in coda

La ricorsione può sempre essere sostituita con l'utilizzo di un ciclo. In questo caso la sostituzione non si presenta particolarmente complicata ed è possibile pertanto creare agevolmente un metodo **non ricorsivo** che permetta il calcolo del nostro numero fattoriale:

```
public static int ciclo(int x) {
    int fattoriale = 1;
    while (x > 0) {
      fattoriale = fattoriale * x;
      x--;
    }
    return fattoriale;
  }
}
```

Ricorsione multipla

Consiste nel caso in cui il metodo invoca se stesso più volte durante la sua esecuzione:

Un esempio di ricorsione multipla è il numero di Fibonacci, che effettua una doppia invocazione.

$$\text{Fib}(n) \begin{cases} n & \text{se } 0 <= n < 2 \\ \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & \text{se } n >= 2 \end{cases}$$

Vengono considerati due casi basi: n=0 e n=1.
Il calcolo viene ottenuto attraverso una doppia invocazione ricorsiva al metodo stesso

Ricorsione multipla

```java
public static int fib(int n) {
    if (n < 2)
      return n;
    else
      return fib(n - 2) + fib(n - 1);
  }
```

Ricorsione multipla - Fibonacci Iterativo

Ricorsione: schema generale

La definizione ricorsiva ricalca la struttura della **definizione induttiva del dominio** sui cui opera la funzione.

1. Uno (o più) casi base, per i quali il risultato può essere determinato direttamente;
2. Uno (o più) casi ricorsivi, per i quali si riconduce il calcolo del risultato al calcolo della stessa funzione su un valore più piccolo/semplice.

definizione induttiva del dominio, cioè prima o poi arriviamo ad uno dei casi base

Somma Ricorsiva

somma(x, y) {

Caso Base

Somma Ricorsiva

somma(x, y) { x Caso Base

Somma Ricorsiva

$$somma(x, y) \begin{cases} x & \text{se } y = 0 \quad \text{Caso Base} \end{cases}$$

Somma Ricorsiva

$$somma(x, y) \begin{cases} x & \text{se } y = 0 \quad \text{Caso Base} \\ & \quad \text{Caso Ricorsivo} \end{cases}$$

Somma Ricorsiva

$$\text{somma}(x, y) \begin{cases} x & \text{se } y = 0 \quad \text{Caso Base} \\ & \text{se } y > 0 \quad \text{Caso Ricorsivo} \end{cases}$$

Somma Ricorsiva

$$\text{somma}(x, y) \begin{cases} x & \text{se } y = 0 \quad \text{Caso Base} \\ 1 + \text{somma}(x, y - 1) & \text{se } y > 0 \quad \text{Caso Ricorsivo} \end{cases}$$

Somma Ricorsiva

$$\text{somma}(x, y) \begin{cases} x & \text{se } y = 0 \quad \text{Caso Base} \\ 1 + \text{somma}(x, y - 1) & \text{se } y > 0 \quad \text{Caso Ricorsivo} \end{cases}$$

```java
public static int sommaRicorsiva (int x, int y) {
    if (y == 0) {
      return x;
    }
    else {
      return 1 + sommaRicorsiva(x, y - 1);
    }
  }
```

Esercizi:

Implementare i metodi:
- moltiplicazione ricorsiva
- potenza ricorsiva

Schema Processamento dati ricorsivo:

Calcolare l'operazione **op** per la collezione di elementi ***C***

```
se C è vuota
      return elemento neutro di op
altrimenti
      return primo elemento op risultato chiamata ricorsiva
```

Schema Processamento dati ricorsivo:

Calcolare l'operazione **op** per la collezione di elementi ***C***

```
se C è vuota
      return elemento neutro di op
altrimenti
      return primo elemento op risultato chiamata ricorsiva
```

```java
public static int sommaRicorsiva (int x, int y) {

    if (y == 0) {
      return x;
    }
    else {
      return 1 + sommaRicorsiva(x, y - 1);
    }
  }
```

Schema Processamento dati ricorsivo:

Calcolare l'operazione **op** per la collezione di elementi ***C***

```
se C è vuota
      return elemento neutro di op
altrimenti
      return primo elemento op risultato chiamata ricorsiva
```

```java
public static int sommaRicorsiva (int x, int y) {

    if (y == 0) {
      return x;
    }
    else {
      return 1 + sommaRicorsiva(x, y - 1);
    }
  }
```

Schema Processamento dati ricorsivo:

Calcolare l'operazione **op** per la collezione di elementi ***C***

```
se C è vuota
      return elemento neutro di op
altrimenti
      return primo elemento op risultato chiamata ricorsiva
```

```java
public static int sommaRicorsiva (int x, int y) {

    if (y == 0) {
      return x;
    }
    else {
      return 1 + sommaRicorsiva(x, y - 1);
    }
  }
```

Schema Processamento dati ricorsivo:

Calcolare l'operazione **op** per la collezione di elementi ***C***

```
se C è vuota
      return elemento neutro di op
altrimenti
      return primo elemento op risultato chiamata ricorsiva
```

```java
public static int sommaRicorsiva (int x, int y) {

    if (y == 0) {
      return x;
    }
    else {
      return 1 + sommaRicorsiva(x, y - 1);
    }
  }
```

Schema Processamento dati ricorsivo:

Calcolare l'operazione **op** per la collezione di elementi ***C***

```
se C è vuota
      return elemento neutro di op
altrimenti
      return primo elemento op risultato chiamata ricorsiva
```

```java
public static int sommaRicorsiva (int x, int y) {

    if (y == 0) {
      return x;
    }
    else {
      return 1 + sommaRicorsiva(x, y - 1);
    }
  }
```

Schema di Ricerca ricorsivo:

Ricerca dell'elemento **x** nell'insieme ***S***

```
se l'insieme è vuoto
      return false
altrimenti {
      return (primo elemento == elemento cercato) or
         Elemento presente resto insieme
```

Schema di Ricerca ricorsivo:

Ricerca dell'elemento **x** nell'insieme ***S***

```
se l'insieme è vuoto
      return false
altrimenti {
      return (primo elemento == elemento cercato) or
         Elemento presente resto insieme
```

```java
public static boolean trovaCarattere(char c, String string) {

    int i = 1;

    if (string.length() == 0) {
      return false;
    }
    else {
      return ((c == string.charAt(0)) || trovaCarattere(c, string.substring(i++)));
    }
  }
```

Schema conteggio ricorsivo:

Conta occorrenze di **x** nel multi insieme ***M***

```
se l'insieme è vuoto
      return 0
altrimenti {
      se il primo elemento è l'elemento cercato
         return 1 + numero occorrenze resto insieme;
      } else {
         Return numero di occorrenze resto insieme
      }
}
```

Esercizi:

Implementare il metodo:
- contaRicorsivoStringa