Java: esempi di programmazione

Questo capitolo raccoglie solo alcuni esempi di programmazione, in parte già descritti in altri capitoli. Lo scopo di questi esempi è solo didattico, utilizzando forme non ottimizzate per la velocità di esecuzione.

Problemi elementari di programmazione

«

In questa sezione vengono mostrati alcuni algoritmi elementari portati in Java.

Somma tra due numeri positivi

«

Il problema della somma tra due numeri positivi, attraverso l'incremento unitario, è descritto nella sezione 62.3.1.

```
//
// java SommaApp <x> <y>
// Somma esclusivamente valori positivi.
//
import java.lang.*; // predefinita
//
class SommaApp
{
    //
    static int somma (int x, int y)
    {
        int i;
        int z = x;
        //
        for (i = 1; i <= y; i++)
          {
            z++;
          }
        return z;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        int y;
        //
        x = Integer.valueOf(args[0]).intValue ();
        y = Integer.valueOf(args[1]).intValue ();
        //
        System.out.println (x + "+" + y + "=" + somma (x, y));
    }
}
//
```

In alternativa si può tradurre il ciclo '**`for`**' in un ciclo '**`while`**':

```
    static int somma (int x, int y)
    {
        int z = x;
        int i = 1;
        //
        while (i <= y)
          {
            z++;
            i++;
          }
        return z;
    }
```

Moltiplicazione di due numeri positivi attraverso la somma

«

Il problema della moltiplicazione tra due numeri positivi, attraverso la somma, è descritto nella sezione 62.3.2.

```
//
// java MoltiplicaApp <x> <y>
// Moltiplica esclusivamente valori positivi.
//
import java.lang.*; // predefinita
//
class MoltiplicaApp
{
    //
    static int moltiplica (int x, int y)
    {
        int i;
        int z = 0;
        //
        for (i = 1; i <= y; i++)
          {
            z = z + x;
          }
        return z;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
```

```
        int y;
        //
        x = Integer.valueOf(args[0]).intValue ();
        y = Integer.valueOf(args[1]).intValue ();
        //
        System.out.println (x + "*" + y + "=" + moltiplica (x, y));
    }
}
//
```

In alternativa si può tradurre il ciclo '**`for`**' in un ciclo '**`while`**':

```
    static int moltiplica (int x, int y)
    {
        int z = 0;
        int i = 1;
        //
        while (i <= y)
          {
            z = z + x;
            i++;
          }
        return z;
    }
```

Divisione intera tra due numeri positivi

Il problema della divisione tra due numeri positivi, attraverso la sottrazione, è descritto nella sezione 62.3.3.

```
//
// java DividiApp <x> <y>
// Divide esclusivamente valori positivi.
//
import java.lang.*; // predefinita
//
class DividiApp
{
    //
    static int dividi (int x, int y)
    {
        int z = 0;
        int i = x;
```

```
        //
        while (i >= y)
          {
            i = i - y;
            z++;
          }
        return z;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        int y;
        //
        x = Integer.valueOf(args[0]).intValue ();
        y = Integer.valueOf(args[1]).intValue ();
        //
        System.out.println (x + ":" + y + "=" + dividi (x, y));
    }
}
//
```

Elevamento a potenza

«

Il problema dell'elevamento a potenza tra due numeri positivi, attraverso la moltiplicazione, è descritto nella sezione 62.3.4.

```
//
// java ExpApp <x> <y>
// Elevamento a potenza di valori positivi interi.
//
import java.lang.*; // predefinita
//
class ExpApp
{
    //
    static int exp (int x, int y)
    {
        int z = 1;
        int i;
```

```
        //
        for (i = 1; i <= y; i++)
          {
            z = z * x;
          }
        return z;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        int y;
        //
        x = Integer.valueOf(args[0]).intValue ();
        y = Integer.valueOf(args[1]).intValue ();
        //
        System.out.println (x + "**" + y + "=" + exp (x, y));
    }
}
//
```

In alternativa si può tradurre il ciclo '**`for`**' in un ciclo '**`while`**':

```
    static int exp (int x, int y)
    {
        int z = 1;
        int i = 1;
        //
        while (i <= y)
          {
            z = z * x;
            i++;
          }
        return z;
    }
```

Infine, si può usare anche un algoritmo ricorsivo:

```
    static int exp (int x, int y)
    {
        if (x == 0)
          {
            return 0;
          }
        else if (y == 0)
          {
            return 1;
          }
        else
          {
            return (x * exp (x, y-1));
          }
    }
```

Radice quadrata

«

Il problema della radice quadrata è descritto nella sezione 62.3.5.

```
//
// java RadiceApp <x>
// Estrazione della parte intera della radice quadrata.
//
import java.lang.*; // predefinita
//
class RadiceApp
{
    //
    static int radice (int x)
    {
        int z = 0;
        int t = 0;
        //
        while (true)
          {
            t = z * z;

            if (t > x)
              {
                //
                // È stato superato il valore massimo.
                //
                z--;
```

```
                return z;
              }
            z++;
          }
        //
        // Teoricamente, non dovrebbe mai arrivare qui.
        //
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        //
        x = Integer.valueOf(args[0]).intValue ();
        //
        System.out.println ("radq(" + x + ")=" + radice (x));
    }
}
//
```

Fattoriale

«

Il problema del fattoriale è descritto nella sezione 62.3.6.

```
//
// java FattorialeApp <x>
// Calcola il fattoriale di un valore intero.
//
import java.lang.*; // predefinita
//
class FattorialeApp
{
    //
    static int fattoriale (int x)
    {
        int i = x - 1;
        //
        while (i > 0)
          {
            x = x * i;
            i--;
```

```
            }
        return x;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        //
        x = Integer.valueOf(args[0]).intValue ();
        //
        System.out.println (x + "! = " + fattoriale (x));
    }
}
//
```

In alternativa, l'algoritmo si può tradurre in modo ricorsivo:

```
    static int fattoriale (int x)
    {
        if (x > 1)
          {
            return (x * fattoriale (x - 1));
          }
        else
          {
            return 1;
          }
        //
        // Teoricamente non dovrebbe arrivare qui.
        //
    }
```

Massimo comune divisore

«

Il problema del massimo comune divisore, tra due numeri positivi, è descritto nella sezione 62.3.7.

```
//
// java MCDApp <x> <y>
// Determina il massimo comune divisore tra due numeri interi positivi.
//
```

```
import java.lang.*; // predefinita
//
class MCDApp
{
    //
    static int mcd (int x, int y)
    {
        int i;
        int z = 0;
        //
        while (x != y)
          {
            if (x > y)
              {
                x = x - y;
              }
            else
              {
                y = y - x;
              }
          }
        return x;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        int y;
        //
        x = Integer.valueOf(args[0]).intValue ();
        y = Integer.valueOf(args[1]).intValue ();
        //
        System.out.println ("Il massimo comune divisore tra " + x
                            + " e " + y + " è " + mcd (x, y));
    }
}
//
```

Numero primo

«

Il problema della determinazione se un numero sia primo o meno, è descritto nella sezione 62.3.8.

```
//
// java PrimoApp <x>
// Determina se un numero sia primo o meno.
//
import java.lang.*; // predefinita
//
class PrimoApp
{
    //
    static boolean primo (int x)
    {
        boolean primo = true;
        int i = 2;
        int j;
        //
        while ((i < x) && primo)
          {
            j = x / i;
            j = x - (j * i);
            //
            if (j == 0)
              {
                primo = false;
              }
            else
              {
                i++;
              }
          }
        return primo;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        //
        x = Integer.valueOf(args[0]).intValue ();
```

```
        //
        if (primo (x))
          {
            System.out.println (x + " è un numero primo");
          }
        else
          {
            System.out.println (x + " non è un numero primo");
          }
    }
}
//
```

Scansione di array

In questa sezione vengono mostrati alcuni algoritmi, legati alla scansione degli array, portati in Java.

Ricerca sequenziale

Il problema della ricerca sequenziale all'interno di un array, è descritto nella sezione 62.4.1.

```
//
// java RicercaSeqApp
//
import java.lang.*; // predefinita
//
class RicercaSeqApp
{
    //
    static int ricercaseq (int[] lista, int x, int a, int z)
    {
        int i;
        //
        // Scandisce l'array alla ricerca dell'elemento.
        //
        for (i = a; i <= z; i++)
          {
            if (x == lista[i])
              {
```

```
                return i;
              }
          }
        //
        // La corrispondenza non è stata trovata.
        //
        return -1;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int[] lista = new int[args.length-1];
        int x;
        int i;
        //
        // Conversione degli argomenti della riga di comando in
        // numeri.
        //
        x = Integer.valueOf(args[0]).intValue ();
        //
        for (i = 1; i < args.length; i++)
          {
            lista[i-1] = Integer.valueOf(args[i]).intValue ();
          }
        //
        // Esegue la ricerca.
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento.
        //
        i = ricercaseq (lista, x, 0, lista.length-1);
        //
        // Visualizza il risultato.
        //
        System.out.println (x + " si trova nella posizione "
                            + i + ".");
    }
}
//
```

Esiste anche una soluzione ricorsiva che viene mostrata nella

subroutine seguente:

```
    static int ricercaseq (int[] lista, int x, int a, int z)
    {
        if (a > z)
          {
            //
            // La corrispondenza non è stata trovata.
            //
            return -1;
          }
        else if (x == lista[a])
          {
            return a;
          }
        else
          {
            return ricercaseq (lista, x, a+1, z);
          }
    }
```

Ricerca binaria

«

Il problema della ricerca binaria all'interno di un array, è descritto nella sezione 62.4.2.

```
//
// java RicercaBinApp.java
//
import java.lang.*; // predefinita
//
class RicercaBinApp
{
    //
    static int ricercabin (int[] lista, int x, int a, int z)
    {
        int m;
        //
        // Determina l'elemento centrale.
        //
        m = (a + z) / 2;
        //
        if (m < a)
          {
```

```
            //
            // Non restano elementi da controllare: l'elemento cercato
            // non c'è.
            //
            return -1;
          }
        else if (x < lista[m])
          {
            //
            // Si ripete la ricerca nella parte inferiore.
            //
            return ricercabin (lista, x, a, m-1);
          }
        else if (x > lista[m])
          {
            //
            // Si ripete la ricerca nella parte superiore.
            //
            return ricercabin (lista, x, m+1, z);
          }
        else
          {
            //
            // m rappresenta l'indice dell'elemento cercato.
            //
            return m;
          }
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int[] lista = new int[args.length-1];
        int x;
        int i;
        //
        // Conversione degli argomenti della riga di comando in
        // numeri.
        //
        x = Integer.valueOf(args[0]).intValue ();
        //
        for (i = 1; i < args.length; i++)
          {
```

```
            lista[i-1] = Integer.valueOf(args[i]).intValue ();
          }
        //
        // Esegue la ricerca.
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento.
        //
        i = ricercabin (lista, x, 0, lista.length-1);
        //
        // Visualizza il risultato.
        //
        System.out.println (x + " si trova nella posizione "
                            + i + ".");
    }
}
//
```

Algoritmi tradizionali

In questa sezione vengono mostrati alcuni algoritmi tradizionali portati in Java.

Bubblesort

Il problema del Bubblesort è stato descritto nella sezione 62.5.1. Viene mostrata prima una soluzione iterativa e successivamente il metodo '**bsort**' in versione ricorsiva.

```
//
// java BSortApp
//
import java.lang.*; // predefinita
//
class BSortApp
{
    //
    static int[] bsort (int[] lista, int a, int z)
    {
        int scambio;
        int j;
```

```
        int k;
        //
        // Inizia il ciclo di scansione dell'array.
        //
        for (j = a; j < z; j++)
          {
            //
            // Scansione interna dell'array per collocare nella
            // posizione j l'elemento giusto.
            //
            for (k = j+1; k <= z; k++)
              {
                if (lista[k] < lista[j])
                  {
                    //
                    // Scambia i valori.
                    //
                    scambio = lista[k];
                    lista[k] = lista[j];
                    lista[j] = scambio;
                  }
              }
          }
        //
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento. Qui si restituisce ugualmente un
        // riferimento all'array ordinato.
        //
        return lista;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int[] lista = new int[args.length];
        int i;
        //
        // Conversione degli argomenti della riga di comando in
        // numeri.
        //
        for (i = 0; i < args.length; i++)
          {
            lista[i] = Integer.valueOf(args[i]).intValue ();
```

```
          }
        //
        // Ordina l'array.
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento.
        //
        bsort (lista, 0, args.length-1);
        //
        // Visualizza il risultato.
        //
        for (i = 0; i < lista.length; i++)
          {
            System.out.println ("lista[" + i + "] = "
                                + lista[i]);
          }
    }

}
//
```

Segue il metodo '**`bsort`**' in versione ricorsiva:

```
    static int[] bsort (int[] lista, int a, int z)
    {
        int scambio;
        int k;
        //
        if (a < z)
          {
            //
            // Scansione interna dell'array per collocare nella
            // posizione a l'elemento giusto.
            //
            for (k = a+1; k <= z; k++)
              {
                if (lista[k] < lista[a])
                  {
                    //
                    // Scambia i valori.
                    //
                    scambio = lista[k];
                    lista[k] = lista[a];
                    lista[a] = scambio;
                  }
              }
            bsort (lista, a+1, z);
          }
        return lista;
    }
```

Torre di Hanoi

«

Il problema della torre di Hanoi è descritto nella sezione 62.5.3.

```
//
// java HanoiApp <n-anelli> <piolo-iniziale> <piolo-finale>
//
import java.lang.*; // predefinita
//
class HanoiApp
{
    //
    static void hanoi (int n, int p1, int p2)
    {
        if (n > 0)
          {
```

```
            hanoi (n-1, p1, 6-p1-p2);
            System.out.println ("Muovi l'anello " + n
                                + " dal piolo " + p1
                                + " al piolo " + p2 + ".");
            hanoi (n-1, 6-p1-p2, p2);
          }
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int n;
        int p1;
        int p2;
        //
        n = Integer.valueOf(args[0]).intValue ();
        p1 = Integer.valueOf(args[1]).intValue ();
        p2 = Integer.valueOf(args[2]).intValue ();
        //
        hanoi (n, p1, p2);
    }
}
//
```

Quicksort

«

L'algoritmo del Quicksort è stato descritto nella sezione 62.5.4.

```
//
// java QSortApp
//
import java.lang.*; // predefinita
//
class QSortApp
{
    //
    static int part (int[] lista, int a, int z)
    {
        int scambio;
        //
        // Si assume che a sia inferiore a z.
        //
```

```
        int i = a + 1;
        int cf = z;
        //
        // Inizia il ciclo di scansione dell'array.
        //
        while (true)
          {
            while (true)
              {
                //
                // Sposta i a destra.
                //
                if ((lista[i] > lista[a]) || (i >= cf))
                  {
                    break;
                  }
                else
                  {
                    i++;
                  }
              }
            while (true)
              {
                //
                // Sposta cf a sinistra.
                //
                if (lista[cf] <= lista[a])
                  {
                    break;
                  }
                else
                  {
                    cf--;
                  }
              }
            //
            if (cf <= i)
              {
                //
                // È avvenuto l'incontro tra i e cf.
                //
                break;
              }
            else
```

```
            {
              //
              // Vengono scambiati i valori.
              //
              scambio = lista[cf];
              lista[cf] = lista[i];
              lista[i] = scambio;
              //
              i++;
              cf--;
            }
        }
      //
      // A questo punto lista[a..z] è stata ripartita e cf è la
      // collocazione di lista[a].
      //
      scambio = lista[cf];
      lista[cf] = lista[a];
      lista[a] = scambio;
      //
      // A questo punto, lista[cf] è un elemento (un valore) nella
      // giusta posizione.
      //
      return cf;
  }
  //
  static int[] quicksort (int[] lista, int a, int z)
  {
      int cf;
      //
      if (z > a)
        {
          cf = part (lista, a, z);
          quicksort (lista, a, cf-1);
          quicksort (lista, cf+1, z);
        }
      //
      // In Java, gli array sono oggetti e come tali vengono passati
      // per riferimento. Qui si restituisce ugualmente un
      // riferimento all'array ordinato.
      //
      return lista;
  }
  //
```

```
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int[] lista = new int[args.length];
        int i;
        //
        // Conversione degli argomenti della riga di comando in
        // numeri.
        //
        for (i = 0; i < args.length; i++)
          {
            lista[i] = Integer.valueOf(args[i]).intValue ();
          }
        //
        // Ordina l'array.
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento.
        //
        quicksort (lista, 0, args.length-1);
        //
        // Visualizza il risultato.
        //
        for (i = 0; i < lista.length; i++)
          {
            System.out.println ("lista[" + i + "] = "
                                + lista[i]);
          }
    }
}
//
```

Permutazioni

«

L'algoritmo ricorsivo delle permutazioni è descritto nella sezione 62.5.5.

```
//
// java PermutaApp
//
import java.lang.*; // predefinita
//
```

```
class PermutaApp
{
    //
    static void permuta (int[] lista, int a, int z)
    {
        int scambio;
        int k;
        int i;
        //
        // Se il segmento di array contiene almeno due elementi, si
        // procede.
        //
        if ((z - a) >= 1)
          {
            //
            // Inizia un ciclo di scambi tra l'ultimo elemento e uno
            // degli altri contenuti nel segmento di array.
            //
            for (k = z; k >= a; k--)
              {
                //
                // Scambia i valori.
                //
                scambio = lista[k];
                lista[k] = lista[z];
                lista[z] = scambio;
                //
                // Esegue una chiamata ricorsiva per permutare un
                // segmento più piccolo dell'array.
                //
                permuta (lista, a, z-1);
                //
                // Scambia i valori.
                //
                scambio = lista[k];
                lista[k] = lista[z];
                lista[z] = scambio;
              }
          }
        else
          {
            //
            // Visualizza la situazione attuale dell'array.
            //
```

```
            for (i = 0; i < lista.length; i++)
              {
                System.out.print (" " + lista[i]);
              }
            System.out.println ("");
          }
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int[] lista = new int[args.length];
        int i;
        //
        // Conversione degli argomenti della riga di comando in
        // numeri.
        //
        for (i = 0; i < args.length; i++)
          {
            lista[i] = Integer.valueOf(args[i]).intValue ();
          }
        //
        // Esegue le permutazioni.
        //
        permuta (lista, 0, args.length-1);
    }
}
//
```