Programmazione Java

Rappresentazione dati, Passaggio per valore, Control-flow statements, Array

Davide Di Ruscio
Dipartimento di Informatica
Università degli Studi dell'Aquila

diruscio@di.univaq.it

Sommario

- Rappresentazione dati
- Passaggio per valore
- Control-flow statements
- Array

Rappresentazione dati (1)

» JVM definisce diverse aree di memoria che memorizzano i dati durante l'esecuzione di un programma

» Alcune di tali aree sono create allo start-up e sono distrutte al termine dell'esecuzione della VM

» Altre sono associate ai singoli thread

» Diversi tipi

- Stack
- Heap
- Area metodi
- …

Rappresentazione dati (2): Stack

- VM associa ad ogni thread uno stack
- Pila (FIFO) che memorizza i cosiddetti *Frames*
- Frame
  - Viene creato ogni volta che viene invocato un metodo
  - Viene distrutto al termine dell'esecuzione del metodo
  - Contiene
    - Valori delle variabili locali
    - Valori dei parametri formali
    - ….

Rappresentazione dati (3): Heap

- Condiviso tra tutti i thread della VM
- Contiene tutti le istanze delle classi e gli array che sono allocati
- E’ creato allo start-up della VM
- Garbage Collector è un thread a bassa priorità che elimina gli oggetti non più utilizzati
- Può avere una dimensione fissa oppure essere espanso o diminuito quando necessario

Rappresentazione dati (4): Area Metodi

- Condivisa tra tutti i thread della VM
- Memorizza strutture associate alla classe
  - Runtime constant pool
  - Codice dei metodi e costruttori
  - Codice per inizializzatori statici e di classe
  - **Variabili statiche**
- Logicamente fa parte dell'heap
- La gestione e l'organizzazione è demandata alle singole implementazioni

Rappresentazione dati (5)


```
public class Point {
  private int x = 10
  private int y = 20;

  public Point(int dx, int dy) {
      x = dx;
      y = dy;
  }
}
```

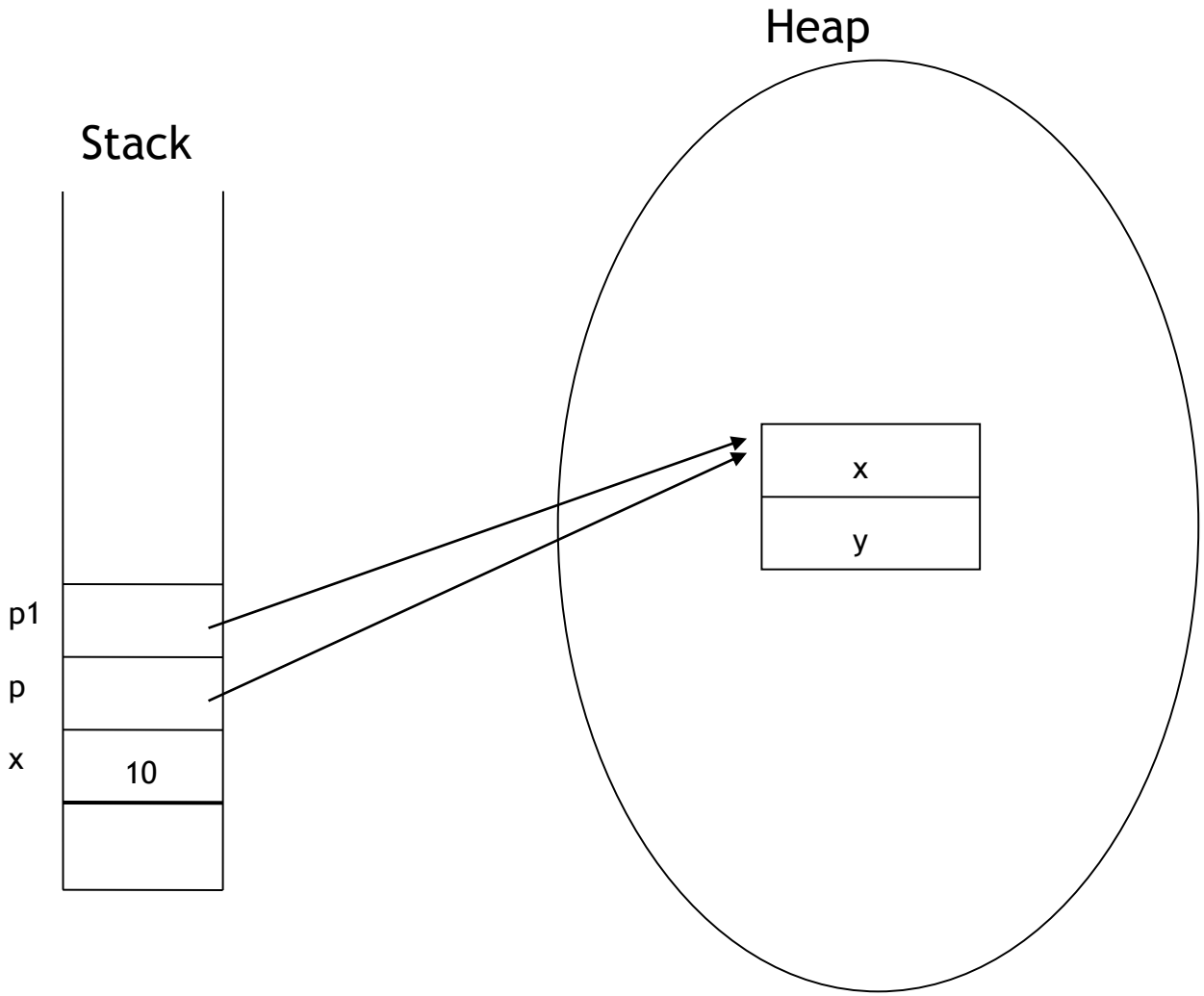
Rappresentazione dati (6)


```
public class Test {
  public static void main(String[] args) {
      int x = 10;
      Point p = new Point( 100, 200 );
      Point p1;
      p1 = p;
  }
}
```

Rappresentazione dati (7)

- Passi eseguiti dalla VM durante la creazione di un oggetto (`new Point(100, 200)`)
  - Viene allocato dello spazio nell'heap per contenere l'oggetto
  - Variabili di istanza vengono inizializzate al loro valore di default
  - Vengono eseguite le esplicite inizializzazioni
  - Viene eseguito il costruttore
  - Variabile viene *assegnata* all'oggetto

Rappresentazione dati (8)



Passaggio per valore (1)

» Java ammette soltanto il passaggio per valore dei parametri nei metodi e costruttori

» Nell'invocazione del metodo viene effettuata una copia (all'interno del frame del metodo nello stack) del valore contenuto nel parametro attuale nel parametro formale

» Modifiche del valore del parametro formale all'interno del metodo non hanno effetto nel parametro attuale

» Nota

– Nel caso di tipi **reference** viene copiato il riferimento → valori *contenuti* nell'oggetto possono essere modificati

Passaggio per valore (2)


```
public static void tripleValue(double x) {

  x = 3 * x;

}

double percent = 10;
tripleValue(percent);
```

Passaggio per valore (3)


```
public class Swap {

  public static void main(String[] args) {
        int x = 100;
        int y = 200;
        System.out.println("x=" + x);
        System.out.println("y=" + y);
        swap( x, y );
        System.out.println("x=" + x);
        System.out.println("y=" + y);

  }

  public static void swap(int x, int y) {
        int temp = x;
        x = y;
        y = temp;
  }
}
```

(Vedi Swap1.java)

Passaggio per valore (4)


```
public class Swap {

  public static void main(String[] args) {
        Point p1 = new Point(100,200);
        Point p2 = new Point(300,400);
        System.out.println("p1=" + p1);
        System.out.println("p2=" + p2);
        swap( p1, p2 );
        System.out.println("p1=" + p1);
        System.out.println("p2=" + p2);

  }

  public static void swap(Point p1, Point p2) {
        Point temp = p1;
        p1 = p2;
        p2 = temp;
  }
}
```

Passaggio per valore (5)


```
public class Swap {

  public static void main(String[] args) {
        Point p1 = new Point(100,200);
        Point p2 = new Point(300,400);
        System.out.println("p1=" + p1);
        System.out.println("p2=" + p2);
        swap( p1, p2 );
        System.out.println("p1=" + p1);
        System.out.println("p2=" + p2);

  }

  public static void swap(Point p1, Point p2) {
        int tempx = p1.x;
        int pempy = p1.y;
        p1.x = p2.x;
        p1.y = p2.y;
        p2.x = tempx;
        p2.y = tempy;
  }
}
```

Flusso di controllo (1)

- Permettono di variare il normale flusso di esecuzione di un programma
- Tipi
  - Condizionali
    - if, if-else, switch-case
  - Cicli
    - while, do-while, for
  - Branching
    - break, continue, label:, return

Flusso di controllo: return

- Sintassi
  - `return <valore ritorno>;`
- Due scopi
  - Specificare il valore restituito da un metodo (a meno che non abbia il valore di ritorno di tipo `void`)
  - Fare in modo che l'esecuzione del metodo termini e quel valore venga immediatamente restituito

Flusso di controllo: if (1)

» Sintassi

```
if (espressione booleana) {
    statement(s)
}

if (espressione booleana) {
    statement(s)
} else {
    statement(s)
}
```

Flusso di controllo: if (2)


```
if (espressione booleana) {
statement(s)
} else if (espressione booleana) {
        statement(s)
} else if (espressione booleana) {
        statement(s)
} else {
        statement(s)
}
```

Nota: Parentesi possono essere sostituite con una singola istruzione

Flusso di controllo: if (3)

» Esempi

```
if (yourSales >= target) {
  performance = “Satisfactory”;
  bonus = 100;
}
```

```
if (yourSales >= target) {
  performance = “Satisfactory”;
  bonus = 100;
} else {
  performance = “Unsatisfactory”;
  bonus = 0;
}
```

Flusso di controllo: if (4)

» Il seguente frammento

```
if (x <= y)
 if (y <= z)
    return z;
else
    return y;
```

Flusso di controllo: if (5)

è interpretato come:

```
  if (x <= y) {
   if (y <= z) {
      return z;
   } else {
      return y;
   }
  }
```

e non come:

```
if (x <= y) {
  if (y <= z) {
    return z;
  }

} else {
  return y;

}
```

Flusso di controllo: if (6)

» Esempio

– Scrivere un metodo che prende in ingresso due interi (testVal e target) e restituisce un intero

- Se testVal > target restituisce 1
- Altrimenti se testVal < target restituisce -1
- altrimenti se testVal = target restituisce 0

Operatore Ternario if-else

- Generalmente viene utilizzato quando è necessario assegnare il valore di una variabile a seconda del verificarsi o meno di una condizione
- Sintassi

```
Condizione ? Espressione1 : Espressione2
```

- Esempio

```
int i = (x > 10) ? 100 : -1;
```

Flusso di controllo: while (1)

- Esegue un'istruzione o un blocco di istruzioni fino a quando una determinata condizione restituisce `true`
- Sintassi

```
while (espressione) {
  statement
}
```

- Espressione deve restituire un valore `boolean`
- Se l'espressione restituisce sempre `false` il blocco non viene mai eseguito

Flusso di controllo: while (2)


```
int i = 1, res = 0;
while (i <= 100){
 res += i;
 i++;
}
```

Alla fine res vale 101*50 = 5050 ($1 + 2 + 3 + \ldots + n = n(n+1)/2$)

```
int i = 1, res = 0;
while (i <= 100) {
 res += i;
 i *= 2;
}
```

Alla fine res vale 1+2+4+8+16+32+64 = 117

Flusso di controllo: while (3)


```
public class Retirement {
   public static void main(String[] args) {
     double goal = Double.parseDouble( args[ 0 ] );
     System.out.println("Obiettivo: " + goal );
     double payment = Double.parseDouble( args[ 1 ] );
     System.out.println("Rata annuale: " + payment);
     double interestRate = Double.parseDouble( args[ 2 ] );
     System.out.println( "Tasso interesse percentuale: " +
                                                  interestRate );
       double balance = 0;
       int years = 0;
       double interest = 0;
       while (balance < goal) {
          balance += payment;
          interest = balance * interestRate / 100;
          balance += interest;
          years++;
       }
       System.out.println("Puoi ritirare in " + years + " anni.");
   }
}
```

Flusso di controllo: do-while

- A differenza del `while` viene eseguito prima il blocco e poi viene valutata l'espressione
- Blocco viene ripetuto fintanto che l'espressione è `true`
- Sintassi

```
do {
statement(s)
} while (expression);
```

Flusso di controllo: do-while


```
int i = 101, res = 0;
while (i <= 100) {
 res += i;
 i++;
}
```

```
int i = 101, res = 0;
do {
 res += i;
 i++;
} while (i <= 100);
```

Flusso di controllo: do-while


```
int i = 101, res = 0;
while (i <= 100) {
 res += i;
 i++;
}
```

Alla fine `res` vale 0. Invece

```
int i = 101, res = 0;
do {
 res += i;
 i++;
} while (i <= 100);
```

Alla fine `res` vale 101.

Flusso di controllo: for (1)

- Costrutto generico che supporta le iterazioni controllate da un contatore o da una variabile analoga aggiornata dopo ciascuna iterazione
- Sintassi

```
for ([inizializzazione]; [condizione terminazione];
                         [incremento]) {
  statement
}
```

Flusso di controllo: for (2)

- Caratterizzato da 3 fasi
  - Inizializzazione
    - Eseguita prima di iniziare la prima iterazione
  - Test sulla condizione per proseguire l'esecuzione del corpo del ciclo
  - Alla fine di ogni interazione qualche forma di avanzamento

Flusso di controllo: for (3)

»Esempi

```
for (int i = 0; i < 10; i++) {
  System.out.println(i);
}


for (int i = 10; i >= 0; i--) {
  System.out.println(i);
}
```

Flusso di controllo: for (4)


```
for (int i = 1, j = i + 10; i < 5; i++, j = i * 2) {
  System.out.println(“i = ” + i + “ j = ” + j);
}
```

Ciclo infinito

```
for (; ;) {
  System.out.println(i);
}
```

Flusso di controllo: for (5)

Alternativi

- ```
  int i,res = 0;
  for (i = 1; i <= 100; i++){
   res += i;
  }
  ```

- ```
  for (int i = 1, res = 0; i <= 100; res += i,i++) {
  }
  ```

- ```
  int res = 0;
  for (int i = 0,j = 10; i < j; i++,j--){
       res += i+j;
  }
  ```

Alla fine res vale 50.
  ```

Flusso di controllo: switch (1)

» Costrutto if/else può risultare scomodo quando si ha a che fare con selezioni multiple che prevedono diverse alternative

» Sintassi

```
switch (espressione intera) {
  case espressione intera: statement(s)
  break;
  ...

  default: statement(s)
  break;
}
```

Flusso di controllo: switch (2)

» Esempio

```
switch (choice) {
  case 1:
       …
  break;
  case 2:
       …
  break;

  case 3:
  case 4:
       …
  break;

  default:
       …
  break;
}
```

Flusso di controllo: switch (3)


```
boolean checkAnswer (char c) {
   boolean res;
   switch (c) {
      case 'y':
         res = true;
      default:
         res = false;
   }
  return res;

}
```

ritorna sempre `false`.

Flusso di controllo: break (1)

- Utilizzato all'interno di `switch`, `for`, `while`, `do-while`
- Termina il flusso di controllo attuale e l'esecuzione passa all'istruzione successiva
- Ha due forme
  - Non etichettata
  - Etichettata
    - Utile quando sono presenti molti flussi di controllo annidati e si vuole uscire dal ciclo più annidato
    - Simula in qualche modo il `goto`

Flusso di controllo: break (2)


```
» boolean checkAnswer (char c) {
   boolean res;
   switch (c) {
      case 'y': {
         res = true;
      break;
      }
      default:
         res = false;
      }
   return res;
  }
```

Flusso di controllo: break (3)


```
int res = 0;
for (int i = 0,j = 10; i<j; i++,j--) {
  if (res >= 30) {
   break;
  }
  res += i+j;
}
```

Flusso di controllo: continue (1)

- Utilizzato all'interno di `for, while, do-while`
- Interrompe il flusso regolare e trasferisce il controllo all'intestazione del ciclo più interno
- Come il break ha due forme
  - Non etichettata
  - Etichettata

Flusso di controllo: continue (2)


```
» int res = 0;
  for (int i = 0,j = 5; i<j; i++,j--) {
   if (i == 3) {
       continue;
   }
   res += i+j;
}
```

Flusso di controllo: break e continue


```
etichetta1:

iterazione-esterna {

  iterazione-interna {

    //…
    break; //1
    //…
    continue; //2
    //…
    continue etichetta1; //3
    //…
    break etichetta1; //4

  }

}
```

(es. LabeledFor.java e LabeledWhile.java)

Flusso di controllo: break e continue


```
etichetta1:

iterazione-esterna {

  iterazione-interna {

    //…
    break; //1
    //…
    continue; //2
    //…
    continue etichetta1; //3
    //…
    break etichetta1; //4

  }

}
```

Fa uscire dall'iterazione per continuare con l'iterazione esterna

(es. LabeledFor.java e LabeledWhile.java)

Flusso di controllo: break e continue


```
etichetta1:

iterazione-esterna {

  iterazione-interna {

    //…
    break; //1
    //…
    continue; //2
    //…
    continue etichetta1; //3
    //…
    break etichetta1; //4

  }

}
```

Fa ritornare all'inizio dell'interazione interna

(es. LabeledFor.java e LabeledWhile.java)

Flusso di controllo: break e continue


```
etichetta1:

iterazione-esterna {

  iterazione-interna {

    //…
    break; //1
    //…
    continue; //2
    //…
    continue etichetta1; //3
    //…
    break etichetta1; //4

  }

}
```

Fa uscire dall'iterazione interna e dall'iterazione esterna facendo tornare l'esecuzione direttamente a etichetta1

(es. LabeledFor.java e LabeledWhile.java)

Flusso di controllo: break e continue


```
etichetta1:

iterazione-esterna {

  iterazione-interna {

    //…
    break; //1
    //…
    continue; //2
    //…
    continue etichetta1; //3
    //…
    break etichetta1; //4

  }

}
```

Fa uscire direttamente a etichetta1, senza ripetere l'iterazione. L'effetto, dunque, è quella di far uscire da entrambe le iterazioni

(es. LabeledFor.java e LabeledWhile.java)

Esercizi

- Scrivere un programma che generi 25 valori casuali di tipo int. Per ciascun valore usate un'istruzione if-else per classificarlo come maggiore, minore o uguale a un secondo valore generato per caso
  - Suggerimento:

```
Random generator = new Random(System.currentTimeMillis());
int threshold = generator.nextInt();
```

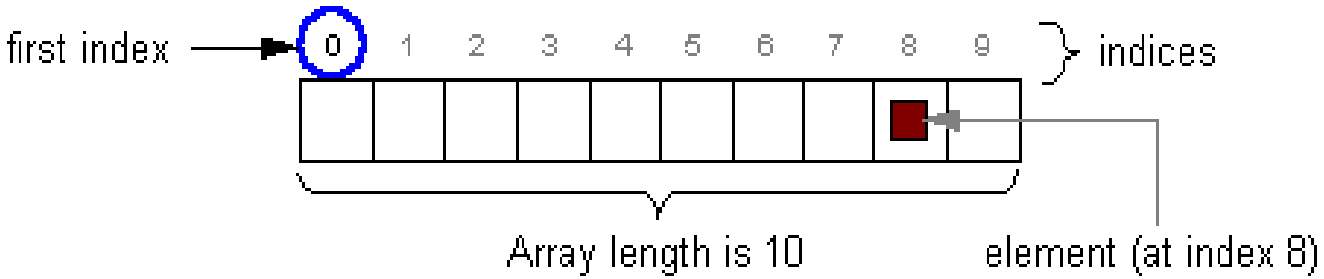
- Scrivere un metodo `isPrime` che, dato un numero n intero, verifica se tale numero è primo, ovvero se è divisibile solo da 1 e da n.
  - Suggerimento: a divide b (b%a==0)

Esercizi

» Scrivere un metodo `isPerfect` che, dato un numero intero, verifica se tale numero è perfetto, ovvero se è uguale alla somma dei suoi divisori (dove il numero stesso non è incluso fra i divisori). Per esempio, 6 è perfetto (6 = 1+2+3).

Array

- Struttura contenente valori dello stesso tipo
- Si accede a ciascun valore dell'array mediante un *indice* intero
- La lunghezza viene stabilita all'atto della sua creazione
- Dopo la creazione la lunghezza non può essere variata
- Per modificare dinamicamente la dimensione è necessario utilizzare un'implementazione di `Collection` (es. `Vector`)



Array : Esempio


```
public class ArrayDemo {

  public static void main(String[] args) {
        int[] anArray;                  // declare an array of integers
        anArray = new int[10];          // create an array of integers

        // assign a value to each array element and print
        for (int i = 0; i < anArray.length; i++) {
                anArray[i] = i;
                System.out.print(anArray[i] + " ");
        }
        System.out.println();
  }
}
```

Array : Dichiarazione

- Possibile dichiarare array di tipo primitivo e di tipo reference (classe, interfacce)
- Due modi per dichiarare un array
  - `char[] s;`
  - `Point[] p;`

  - `char s[];`
  - `Point p[];`
  - `Point p1, p2[];`

  - `Point[][] p;` **`//Array multidimensionali`**

Array : Creazione ed accesso

» Creazione

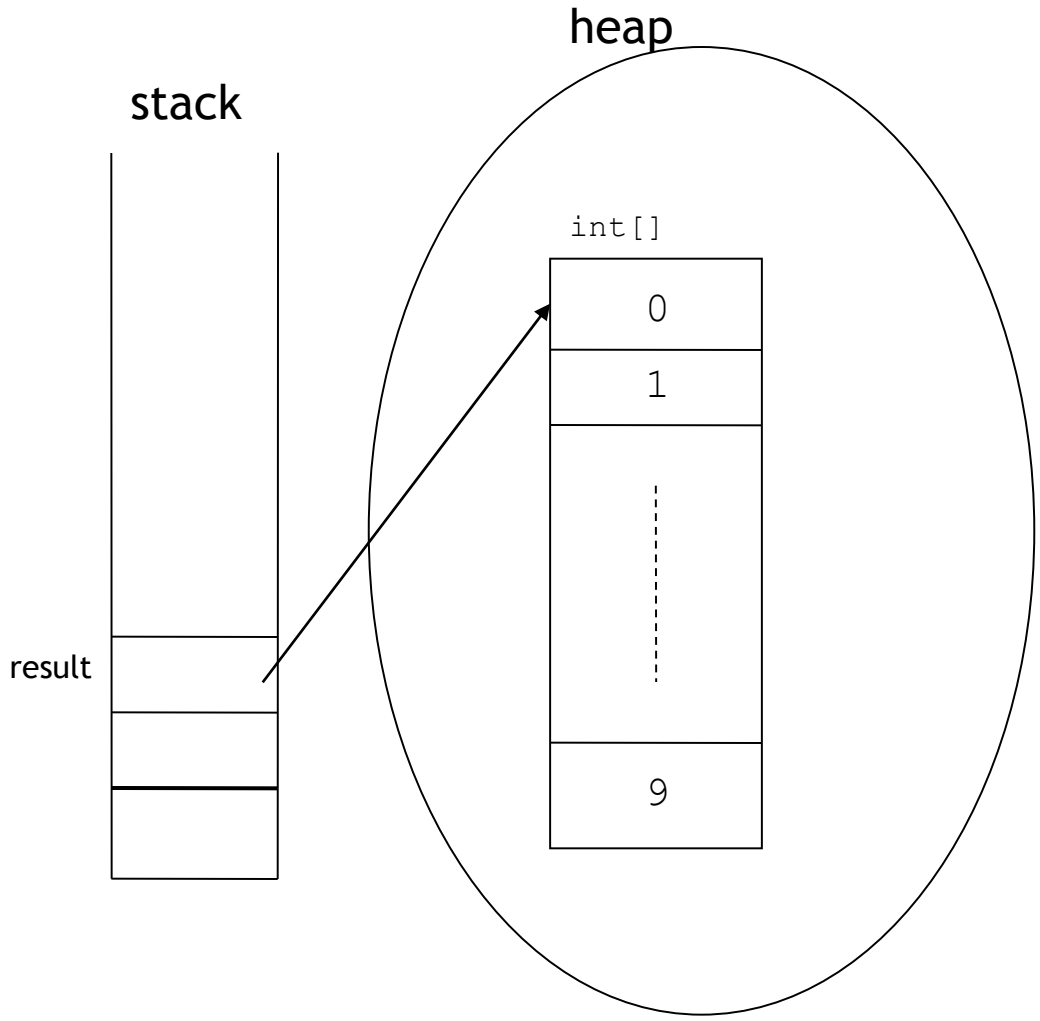
- Un array è un oggetto quindi viene creato con `new`
- Viene creato dello spazio in memoria per contenere il riferimento e il contenuto dell'array

» Accesso

- `anArray[i]`

Array : Rappresentazione


```
…
public int[] getArray() {
   int[] result;
   result = new int[ 10 ];
   for( int i=0; i<result.length;i++) {
         result[i] = i;
   }
   return result;
}
```



Array : Inizializzazione


```
» boolean[] answers = { true, false, true, true, false };

» Point[] points =
              {new Point(0,1), new Point(1,2), new Point(2,3)};

» Point[] points;
  points  = new Point[3];
  points[0] = new Point(0,0);
  points[1] = new Point(1,1);
  points[2] = new Point(2,2);
```

(vedi ArrayDemo.java, ArrayDemo2.java, …)

Array Multidimensionali

» Esempi

- `int[][] a = new int [4][];`
- `a[0] = new int[5];`
- `a[1] = new int[5];`
- `int[][] a = new int [][4]; //illegale`
- `int[][] a = new int[4][5]; //Matrice`

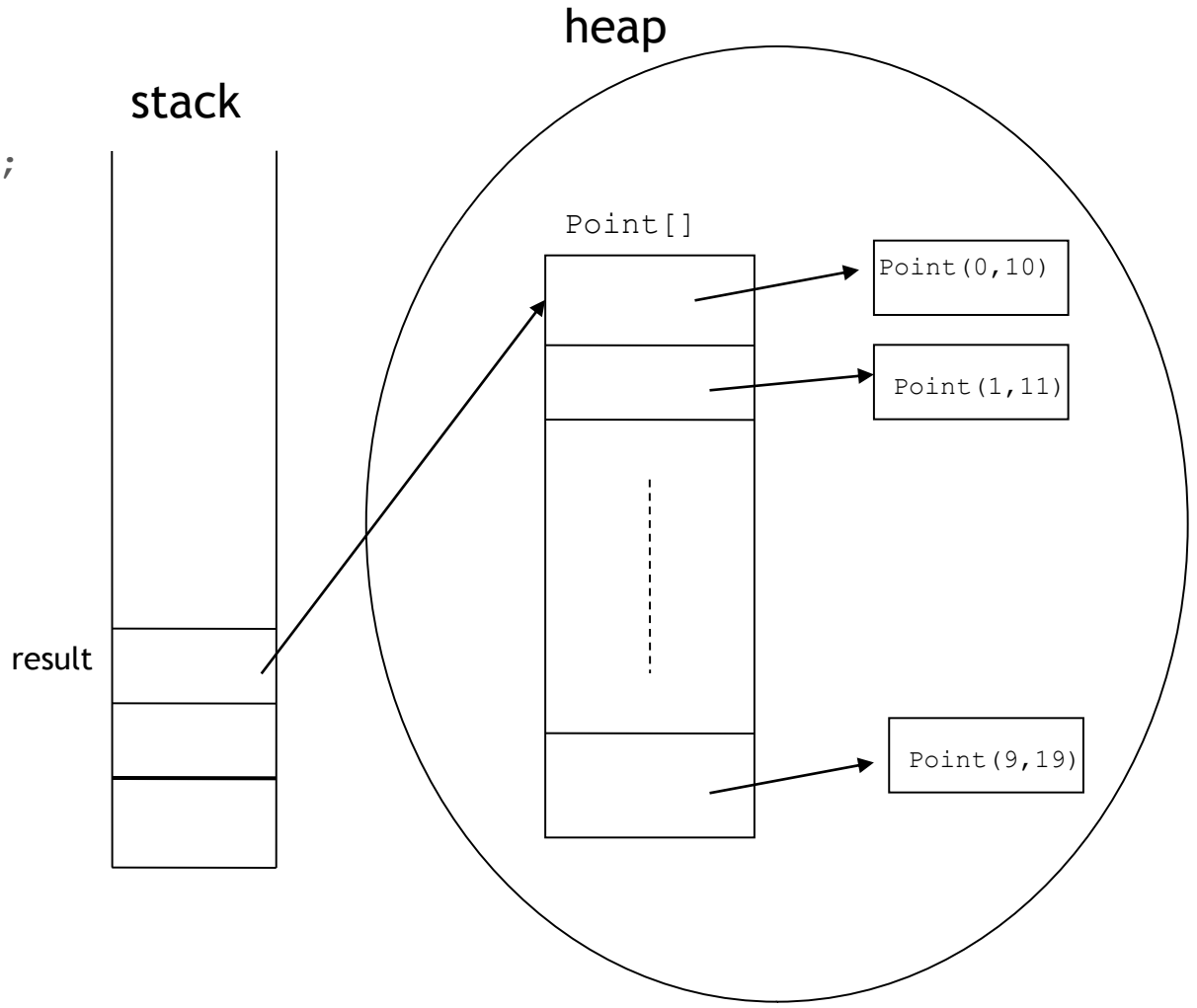
```
public class ArrayOfArraysDemo {
   public static void main(String[] args) {

          String[][] cartoons = {
          { "Flintstones", "Fred", "Wilma", "Pebbles", "Dino" },
          { "Rubbles", "Barney", "Betty", "Bam Bam" },
          { "Jetsons", "George", "Jane", "Elroy", "Judy", "Rosie", "Astro"},
          { "Scooby Doo Gang", "Scooby Doo", "Shaggy", "Velma", "Fred", "Daphne" }
                                  };
          for (int i = 0; i < cartoons.length; i++) {
                    System.out.print(cartoons[i][0] + ": ");
                    for (int j = 1; j < cartoons[i].length; j++) {
                              System.out.print(cartoons[i][j] + " ");
                    } System.out.println();
          }
   }
}
```

(vedi ArrayClassObj.java)

Array Multidimensionali : Rappresentazione


```
public int[] getArray() {
  Point[] result;
  result = new Point[ 10 ];
  for( int i=0; i<result.length;i++) {
        result[i] = new Point(i,i+10);
  }
  return result;
}
```



Array Multidimensionali

» Esempi

- `int[][] a = new int [4][];`
- `a[0] = new int[5];`
- `a[1] = new int[5];`
- `int[][] a = new int [][4]; //illegale`
- `int[][] a = new int[4][5]; //Matrice`

Ciascun vettore può avere una lunghezza qualsiasi

```
public class ArrayOfArraysDemo {
   public static void main(String[] args) {

          String[][] cartoons = {
          { "Flintstones", "Fred", "Wilma", "Pebbles", "Dino" },
          { "Rubbles", "Barney", "Betty", "Bam Bam" },
          { "Jetsons", "George", "Jane", "Elroy", "Judy", "Rosie", "Astro"},
          { "Scooby Doo Gang", "Scooby Doo", "Shaggy", "Velma", "Fred", "Daphne" }
                                  };
          for (int i = 0; i < cartoons.length; i++) {
                    System.out.print(cartoons[i][0] + ": ");
                    for (int j = 1; j < cartoons[i].length; j++) {
                              System.out.print(cartoons[i][j] + " ");
                    } System.out.println();
          }
   }
}
```

(vedi ArrayOfArraysDemo.java)

Array Multidimensionali


```
int[][][] a3 = new int[rand.nextInt(7)][][];

for (int i = 0; i< a3.length; i++){
  a3[i] = new int[rand.nextInt(5)][];
  for (int j=0; j < a3[i].length; j++)
       a3[i][j] = new int[rand.nextInt(5)];
}
```

» Il primo new crea un array con un primo elemento di lunghezza casuale ed il resto indeterminato

» Il secondo new all'interno del ciclo for riempie gli elementi ma lascia il terzo indice indeterminato finchè non si arriva al terzo new

(vedi ArrayOfArraysDemo.java)

Array : Copia


```
int[] a1 = {1, 2, 3, 4, 5};
int[] a2;

a2=a1; //copia il riferimento non il contenuto
```

Esempio

```
public class Arrays {

  public static void main(String[] args) {
   int[] a1 = { 1, 2, 3, 4, 5 };
   int[] a2;
   a2 = a1;

   for(int i = 0; i < a2.length; i++)
      a2[i]++;

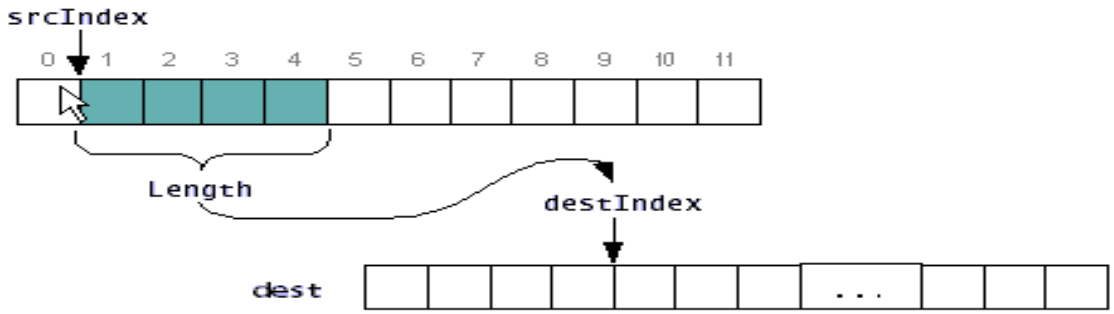
   for(int i = 0; i < a1.length; i++)
      System.out.println("a1[" + i + "] = " + a1[i]);
  }

}
```

Array : Copia

» Per copiare i dati da un array ad un altro si utilizza il metodo statico `arraycopy` della classe `System`

- `public static void arraycopy(Object source, int srcIndex, Object dest, int destIndex, int length)`



Array : Copia


```
public class ArrayCopyDemo {
   public static void main(String[] args) {
          char[] copyFrom =
             { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };
          char[] copyTo = new char[7];
          System.arraycopy(copyFrom, 2, copyTo, 0, 7);
          System.out.println(new String(copyTo));
   }
}
```