# Programmazione in Java
# Parte I: Fondamenti

## Lezione 2

**Dott. Marco Faella**

- In Java, gli array sono oggetti a tutti gli effetti
- Non esistono variabili che contengono (direttamente) array, ma solo riferimenti ad array

- Ad esempio, la dichiarazione

```
int[] a;
```

introduce un riferimento ad un array, ma non alloca memoria per un array di interi

- Per allocare un array vero e proprio, è necessario utilizzare new, come per gli oggetti:

```
a = new int[10];
```

- Le celle degli array vengono inizializzate automaticamente con un valore di default
- Per i tipi numerici, tale valore è zero

# Gli array multi-dimensionali

- Gli array multi-dimensionale sono trattati come array di array

```
int[][] a = new int[10][5];
```

- Posso anche allocare una riga alla volta
- Di conseguenza, ogni riga può avere lunghezza diversa dall'altra
- In questo caso, si parla di *array imperfetti*
- Ad esempio:

```
int[][] a;
a = new int[4][];
a[0] = new int[10];
a[1] = new int[5];

a[0][8] = 5;
```

- a è un array di 4 riferimenti ad array
- a[0] è un riferimento che punta ad un array di 10 interi
- a[1] è un riferimento che punta ad un array di 5 interi
- a[2] e a[3] sono null

Array.java

# Il memory layout degli array

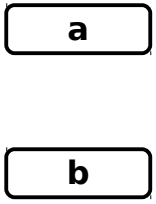
**Stralcio di Array.java:**

```
int[] a, b;
a = new int[10];
b = a;
b[2] = 5;


int[][] c = new int[5][10];

c[0][3] = 7;
c[1] = c[0];
```

**Memory layout:**



# Il memory layout degli array

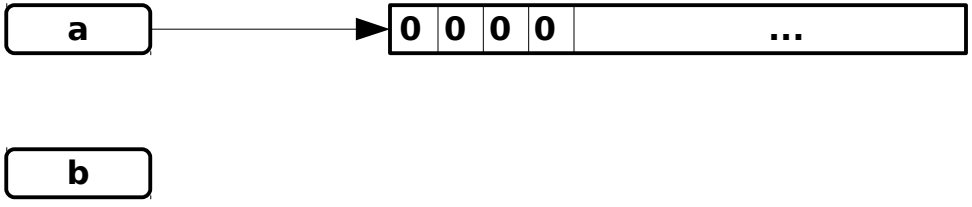
**Stralcio di Array.java:**

```
int[] a, b;
a = new int[10];
b = a;
b[2] = 5;


int[][] c = new int[5][10];

c[0][3] = 7;
c[1] = c[0];
```

**Memory layout:**



# Il memory layout degli array

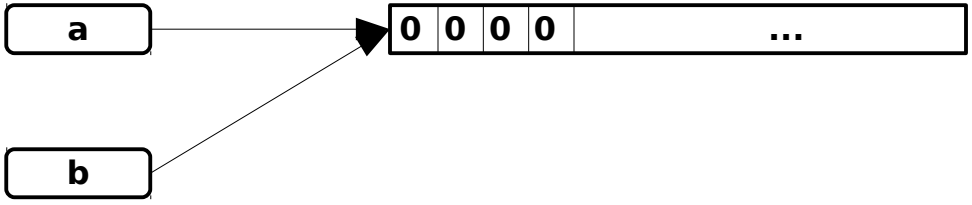
**Stralcio di Array.java:**

```
int[] a, b;
a = new int[10];
b = a;
b[2] = 5;


int[][] c = new int[5][10];

c[0][3] = 7;
c[1] = c[0];
```

**Memory layout:**



# Il memory layout degli array

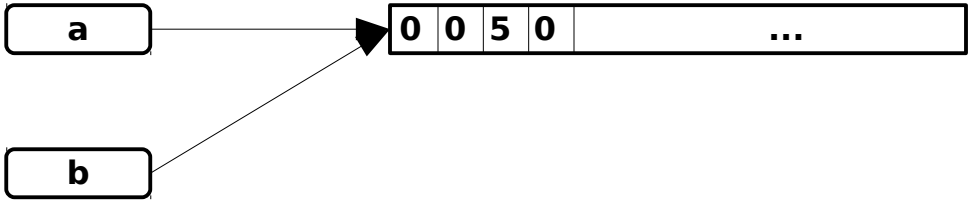
**Stralcio di Array.java:**

```
int[] a, b;
a = new int[10];
b = a;
b[2] = 5;


int[][] c = new int[5][10];

c[0][3] = 7;
c[1] = c[0];
```

**Memory layout:**



# Il memory layout degli array

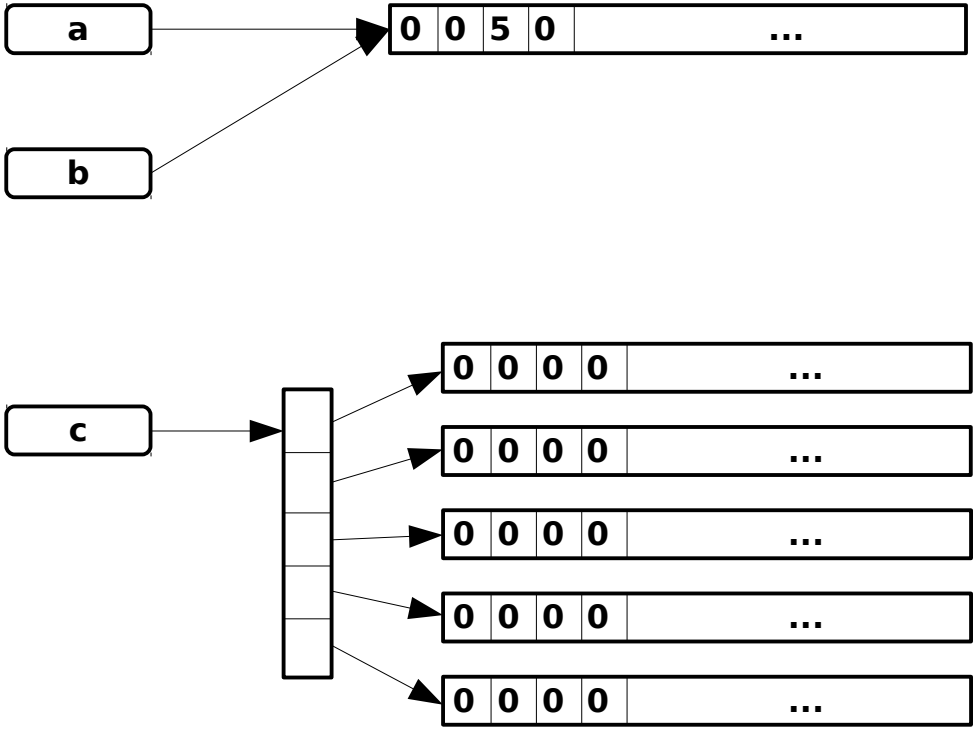
**Stralcio di Array.java:**

```
int[] a, b;
a = new int[10];
b = a;
b[2] = 5;


int[][] c = new int[5][10];

c[0][3] = 7;
c[1] = c[0];
```

**Memory layout:**



# Il memory layout degli array

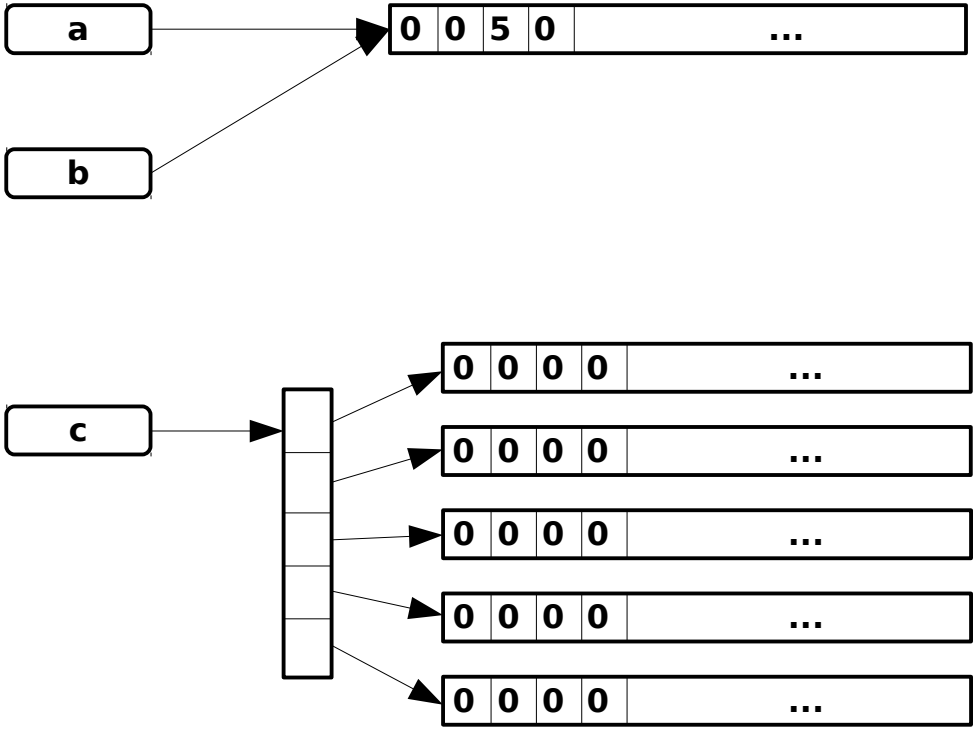
**Stralcio di Array.java:**

```
int[] a, b;
a = new int[10];
b = a;
b[2] = 5;


int[][] c = new int[5][10];

c[0][3] = 7;
c[1] = c[0];
```

**Memory layout:**



# Il memory layout degli array

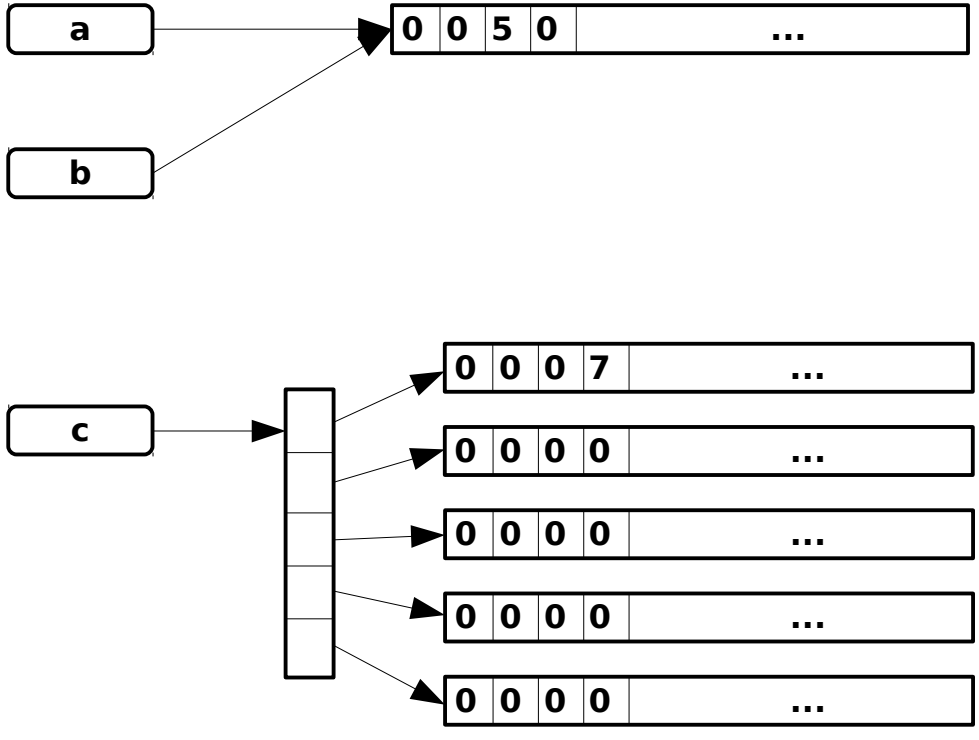
**Stralcio di Array.java:**

```
int[] a, b;
a = new int[10];
b = a;
b[2] = 5;


int[][] c = new int[5][10];

c[0][3] = 7;
c[1] = c[0];
```

**Memory layout:**



# Il memory layout degli array

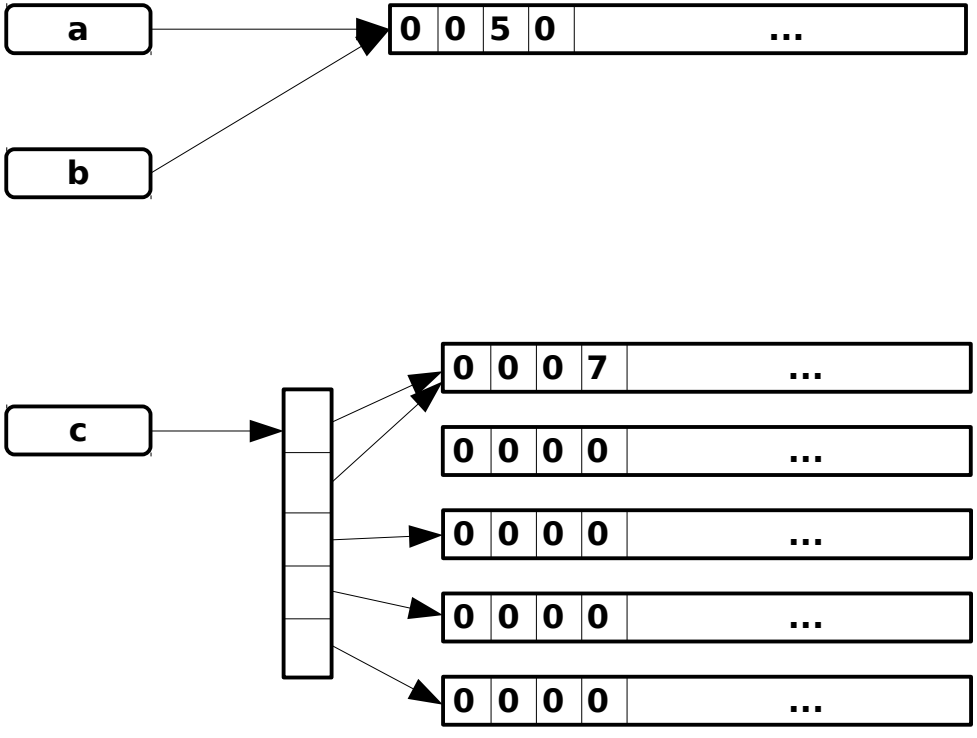
**Stralcio di Array.java:**

```
int[] a, b;
a = new int[10];
b = a;
b[2] = 5;


int[][] c = new int[5][10];

c[0][3] = 7;
c[1] = c[0];
```

**Memory layout:**



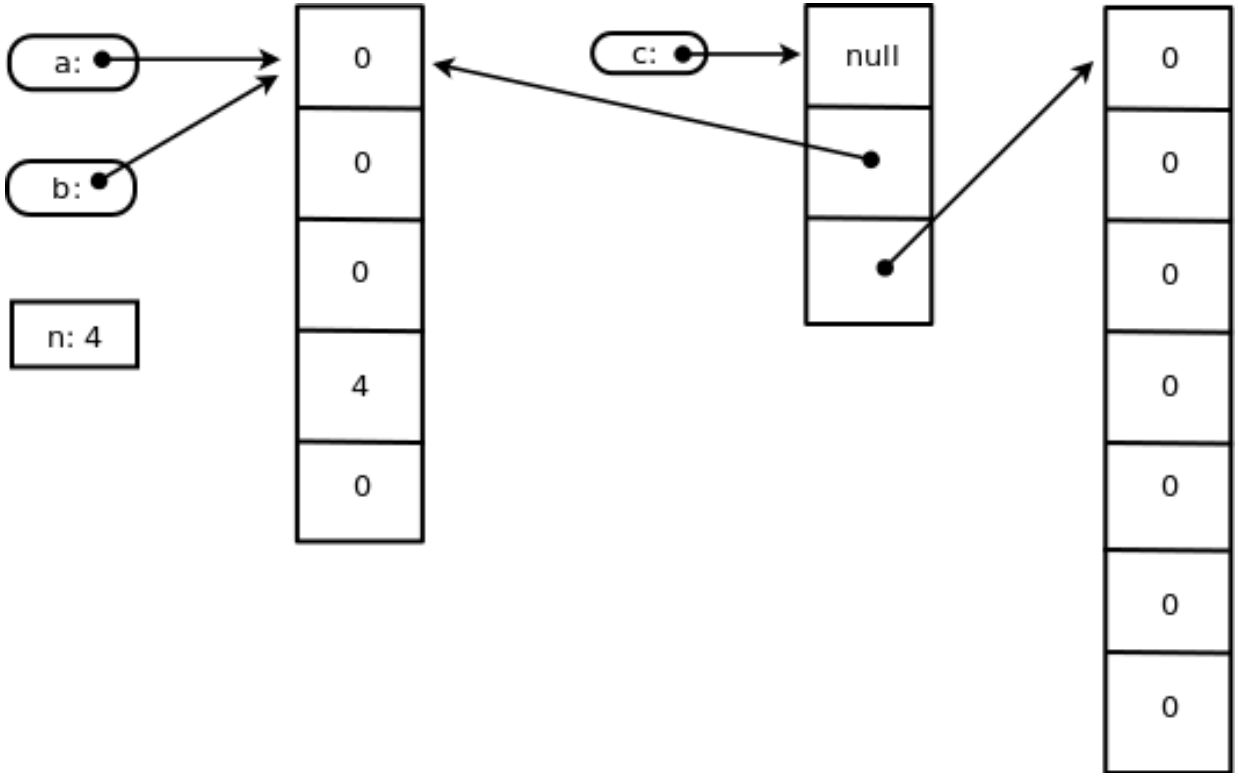
- Disegnare il memory layout del seguente frammento di programma:

```
int[] a = new int[5];
int[] b = a;
int n = 4;
a[3] = n;

int[][] c = new int[3][];
c[1] = b;
c[2] = new int[7];
```

```
int[] a = new int[5];
int[] b = a;
int n = 4;
a[3] = n;

int[][] c = new int[3][];
c[1] = b;
c[2] = new int[7];
```



# Incapsulamento e visibilità

- Ciascun membro di una classe appartiene ad una delle quattro **categorie di visibilità**
- Le categorie di visibilità definiscono i limiti entro i quali il membro di classe è accessibile

- In ordine di crescente “apertura”, le categorie sono: **`private`**, **`protected`**, **default** e **`public`**
- Le categorie **`private`**, **`protected`** e **`public`** sono introdotte dalla keyword omonima, chiamata “**specificatore di visibilità**”
- La categoria **default**, invece, si applica quando non viene indicato alcuno specificatore di visibilità

- Le regole di ciascuna categoria sono le seguenti
  - i membri **`private`** sono accessibili solo all'interno della classe in cui sono definiti
  - i membri default sono accessibili solo nel pacchetto in cui sono definiti
  - i membri `protected` sono accessibili solo nel pacchetto in cui sono definiti ed inoltre nelle sottoclassi della classe in cui sono definiti
  - i membri **`public`** sono accessibili ovunque

- Nota: la regola relativa a protected è stata presentata in una forma leggermente semplificata

Data3.java

# Esercizio: giorni dall'inizio dell'anno

- In una classe, più metodi possono avere lo stesso nome
- In questo caso, è necessario che abbiano numero oppure tipo di argomenti diverso
- Questa caratteristica del linguaggio prende il nome di *overloading*
- Ad esempio, se una classe contiene il metodo

```
public int f(int n, String s)
```

- I seguenti metodi possono essere presenti

```
public int f()
public void f(double d)
public int f(String s, int n)
```

- Mentre i seguenti metodi non possono essere presenti (causerebbero un errore in compilazione)

```
private int f(int n, String s)
public int f(int m, String tmp)
public void f(int n, String s)
private void f(int n, String s)
```

- La stessa regola vale per i costruttori
  - due costruttori devono differire per numero e/o tipo di parametri accettati